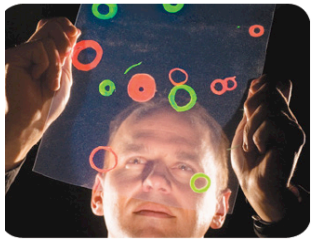
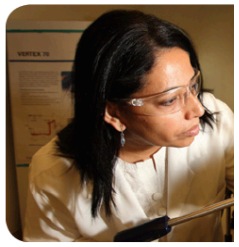




Sandia National Laboratories

Exceptional service in the national interest

9. Stochastic Programming

PYOMO

PySP

U.S. DEPARTMENT OF ENERGY

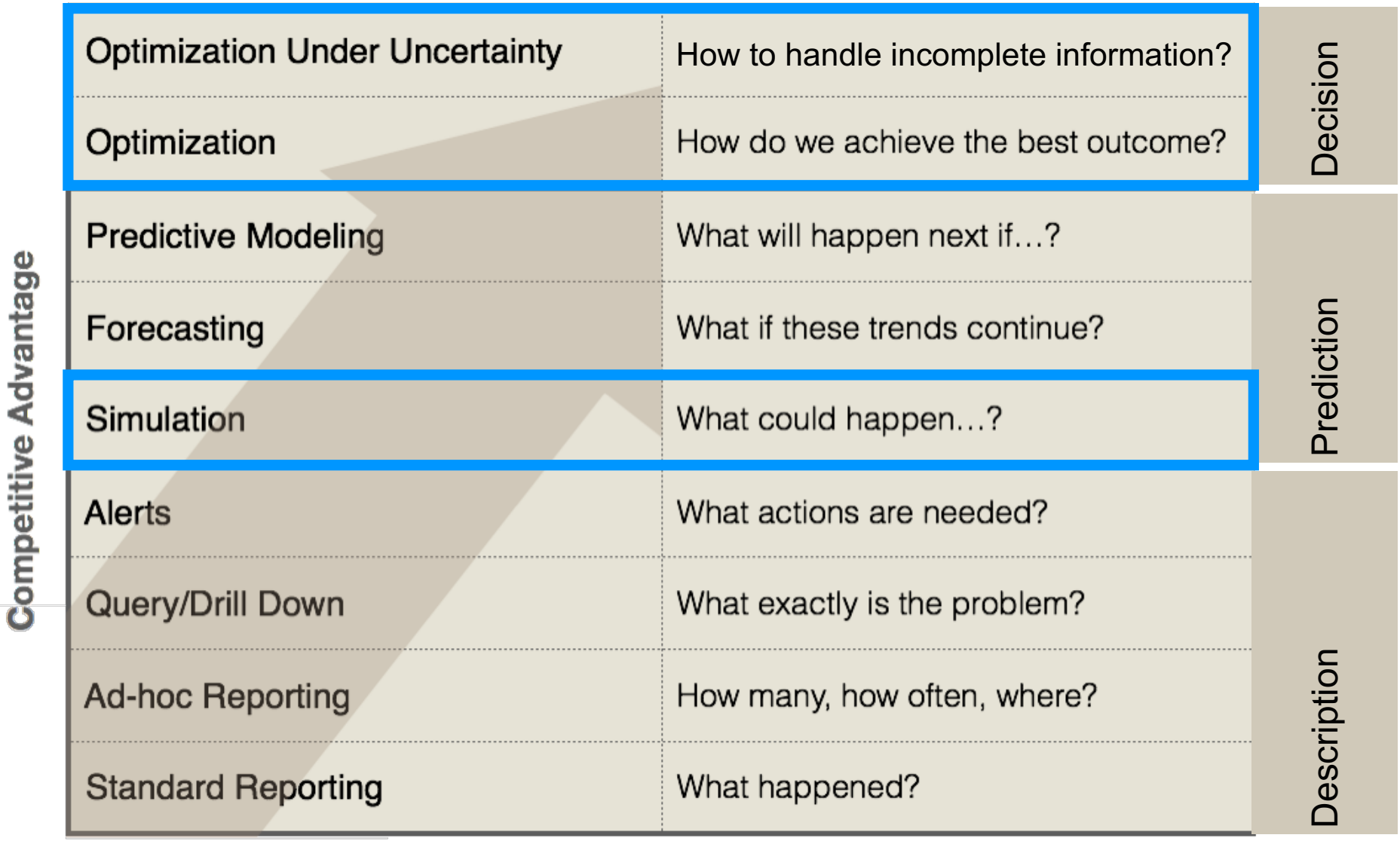
National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Competing on Analytics

Sandia National Laboratories



Degree of Complexity

Source: Adapted from IBM, Based on Davenport and Harris [2007]

CCR Center for Computing Research

PYOMO

COIN-OR

Optimization Under Uncertainty

Sandia National Laboratories

- At its core, Optimization Under Uncertainty (OUU) is decision-making in the face of incomplete information

- (Some) Sources of uncertainty:
  - **Models**: "early" system decisions, un-modeled inputs, missing phenomena, parameter values
  - **Product performance**: manufacturing variability
  - **Usage**: boundary conditions, demands, weather

- How do we deal with uncertainty:
  1. Ignore it (the Flaw of Averages)
  2. Safety margins (leaving money on the table)
  3. Careful case study selection (cannot remove inherent randomness)
  4. Sensitivity analysis (challenging for larger problems)
  5. Integrate it into our decision-making (e.g., OUU)

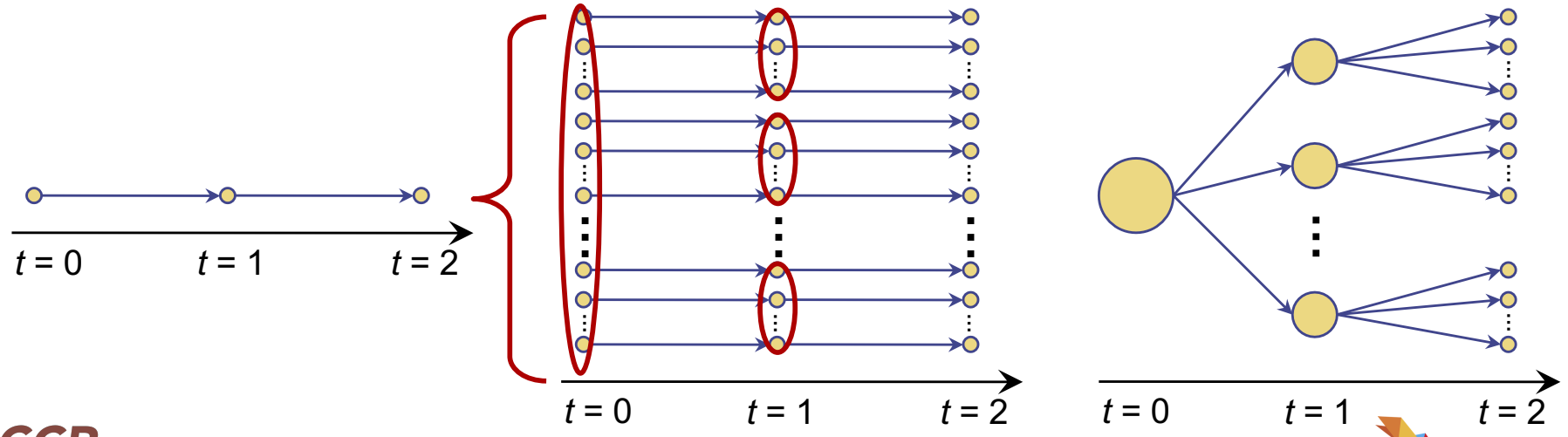
Approaches to OUU

- Computational systems are essentially *deterministic*
  - OUU must be recast into a finite computable problem

- Common approaches:
  - Control the uncertainty
    - Feedback, feedforward, model predictive control (MPC), nonlinear MPC
    - Goal is to adapt the system (in real time) as the uncertainty is revealed
  - Bound the uncertainty
    - Robust optimization, Adjustable robust optimization
    - Bound the uncertainty space and then optimize for the worst case
  - Work backwards
    - Dynamic programming, Approximate dynamic programming
    - Compute a value for all possible states, working back from the final state
  - Sample the uncertainty
    - Sample average approximation, **Stochastic programming**
    - Sample the uncertain space and then optimize over the samples

Stochastic Programming

- Given a potentially infinite uncertainty space
  - That may not be characterized by distributions
  - That may be heavily correlated (in space, time, or function)
- *Approximate* the uncertainty by selecting a finite set of *scenarios*
  - Scenarios may not all be equally likely
  - We could optimize each scenario separately (compute the *value of perfect information*)
    - But I must implement something now… what should I do?
  - Solve a single large problem with scenario decisions constrained to be *implementable*;
    - For a given decision, scenarios that are indistinguishable must all choose the same decision.
    - *Nonanticipativity constraints*



Using PySP: Overview

- PySP simplifies the process of formulating Stochastic Programs by automating the bookkeeping and model construction process.

- This generally reduces to the following four steps:
  1. Formulate the deterministic model
  2. Specify the deterministic model data
     - (necessary to *debug* the deterministic model!)
  3. Specify the scenario tree
  4. Specify the scenario instance data
- Important:
  - We are only talking about instance specification – not solvers

Example: Production Planning

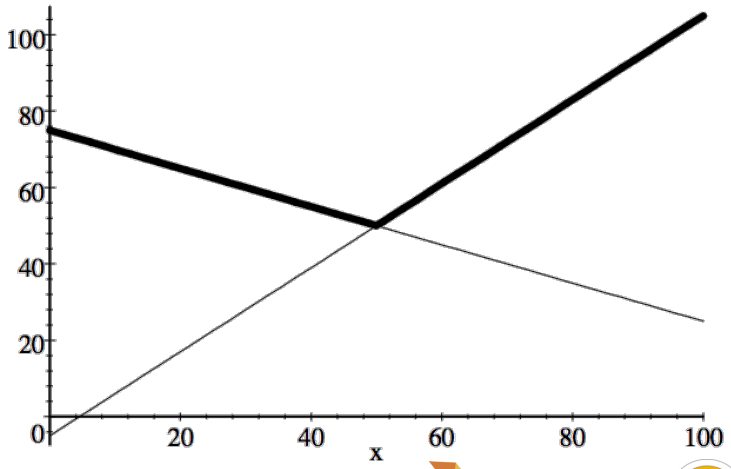
Example from Alexander Shapiro and Andy Philpott (2007)

A company has decided to order a quantity *x* of a product to satisfy demand *d*. The per-unit cost of ordering is *c*. If demand *d* is greater than *x*, then the back-order penalty is *b* per unit. If demand is less than the order quantify, i.e., d<x, then a holding cost *h* is incurred for unused product.

The objective is to minimize the total cost:

$$\max\{(c-b)x+bd,(c+h)x-hd\}$$

For example: c=1, b=1.5, h=0.1, d=50



Deterministic Formulation

In general, the ordering decision is made before a realization of the demand is known.

The deterministic formulation corresponds to a single scenario taken with probability one.

$$\begin{aligned} \min_{x,t} \quad & t \\ \text{s.t.} \quad & t \geq (c-b)x + bd, \\ & t \geq (c+h)x - hd, \\ & x \geq 0. \end{aligned}$$

We can formulate a two-stage stochastic program where the first stage has zero cost and the second stage has cost t.

Deterministic Formulation in Pyomo

```
from pyomo.environ import *

c, b, h = 1.0, 1.5, 0.1

model = AbstractModel()
model.d = Param()
model.t = Var()
model.x = Var(within=NonNegativeReals, bounds=(0,100))

@model.Constraint()
def c1(model):
    return model.t >= (c-b)*model.x + b*model.d

@model.Constraint()
def c2(model):
    return model.t >= (c+h)*model.x - h*model.d

model.FirstStageCost = Expression(expr=0)
model.SecondStageCost = Expression(expr=model.t)
model.obj = Objective(expr=model.FirstStageCost + model.SecondStageCost)
```

Deterministic Model Data

- We used native Python data for the values of *c*, *b,* and *h*
  - These parameters don't vary across scenarios
- We can put the scenario-dependent parameters into a distinct .dat file (for example) that contains the single line:
  - param d := 50 ;
- For pedagogical purposes, we won't use a concrete model quite yet…
  - Avoids the need for scenario-specific data files

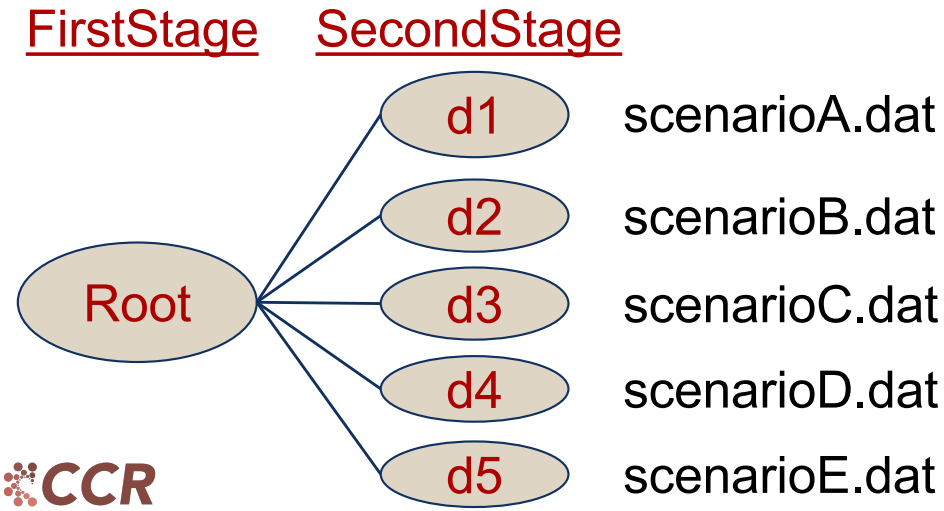
Specifying the Scenario Tree: *names*

- Under the hood, a *Pyomo Model* is used to specify a Scenario tree
  - The scenario tree can be specified through a standard DAT file

```
model = AbstractModel()

model.Stages = Set(ordered=True)

model.Nodes = Set(ordered=True)
```



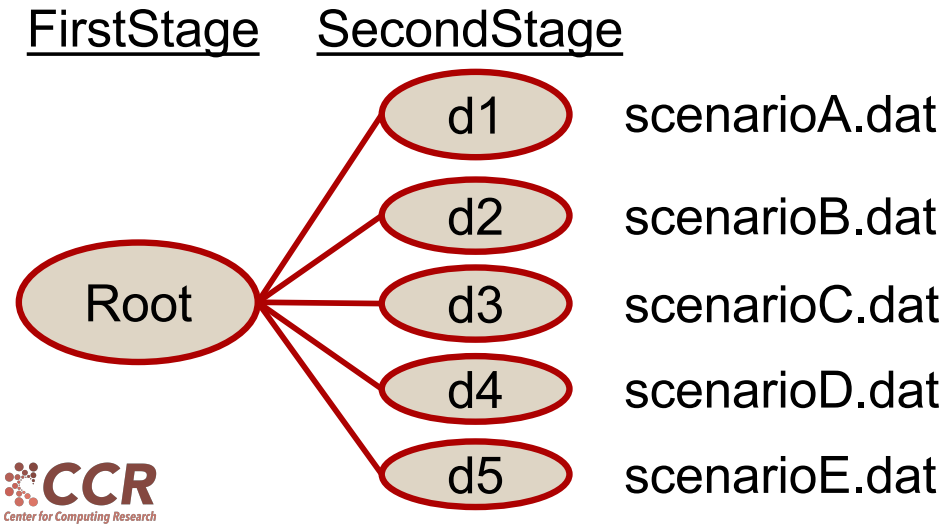

```
set Stages := FirstStage
              SecondStage;

set Nodes := Root
             d1 d2 d3 d4 d5 ;
```

Specifying the Scenario Tree: *structure*

```
model.NodeStage = Param(
        model.Nodes,
        within=model.Stages )

model.Children = Set(
        model.Nodes,
        within=model.Nodes,
        ordered=True )
```



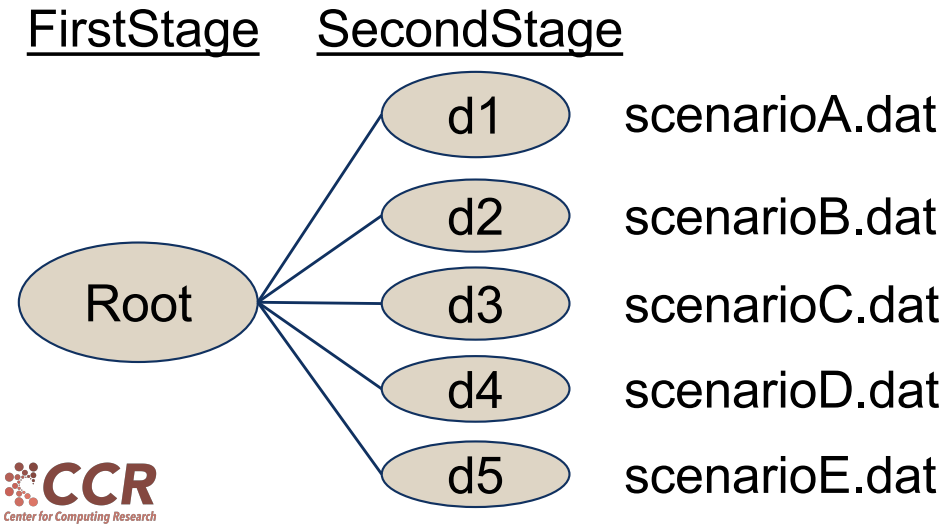

```
param NodeStage :=
   Root  FirstStage
   d1    SecondStage
   d2    SecondStage
   d3    SecondStage
   d4    SecondStage
   d5    SecondStage ;

set Children[Root] :=
   d1 d2 d3 d4 d5 ;
```

Specifying the Scenario Tree: *probabilities*

```
model.ConditionalProbability =
        Param(model.Nodes)
```

```
param
ConditionalProbability :=
   Root  1.0
   d1    0.2
   d2    0.2
   d3    0.2
   d4    0.2
   d5    0.2 ;
```



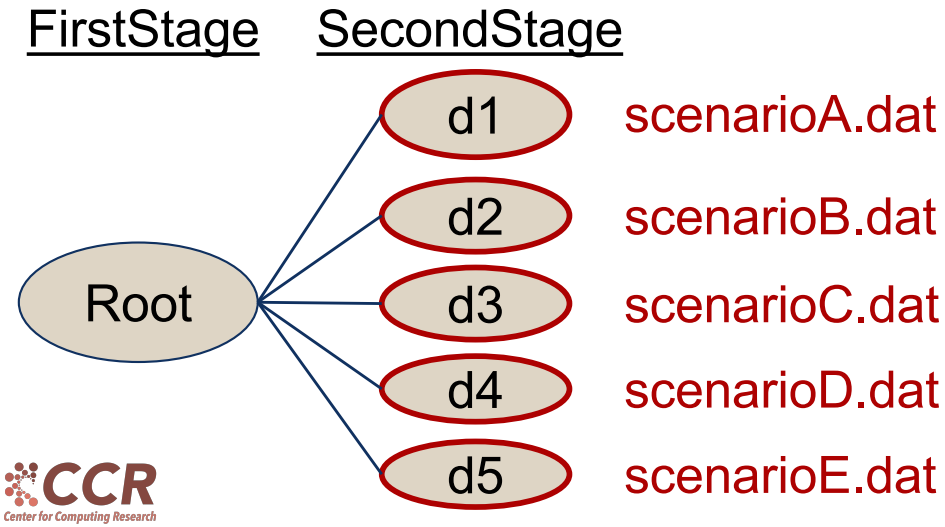
Specifying the Scenario Tree: *scenarios*

```
model.Scenarios = Set(
        ordered=True)

model.ScenarioLeafNode = Param(
        model.Scenarios,
        within=model.Nodes)
```

```
set Scenarios :=
      scenarioA
      scenarioB
      scenarioC
      scenarioD
      scenarioE ;

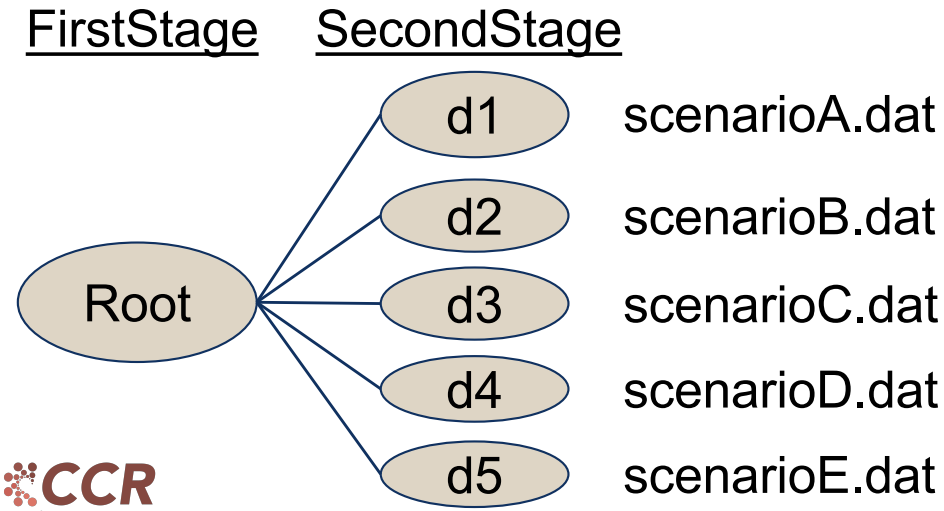
set ScenarioLeafNode :=
      scenarioA   d1
      scenarioB   d2
      scenarioC   d3
      scenarioD   d4
      scenarioE   d5 ;
```



Specifying the Scenario Tree: *variables*

- PySP needs to know which variables belong to which stages
  - For forming the nonanticipativity constraints
  - Can use slicing and wildcards (e.g., "x[*]" to indicate all indices of "x")

```
model.StageVariables = Set(
        model.Stages,
        ordered=True )
```




```
set
StageVariables[FirstStage]
      := x ;

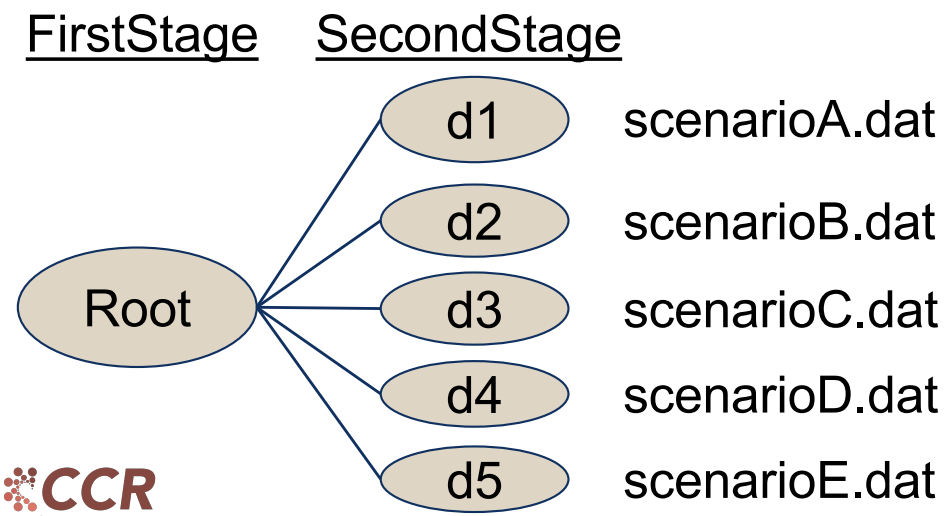
set
StageVariables[SecondStage]
      := t ;
```

Specifying the Scenario Tree: o*bjective*

- PySP needs to know the cost of each stage
  - Frequently a stage cost may be “0” (as it is in the newsvendor)
  - The stage costs are frequently Expressions

```
model.StageCost = Param(
        model.Stages )
```

```
param StageCost :=
  FirstStage FirstStageCost
  SecondStage SecondStageCost
```



Specifying the Scenario Tree: *advanced topics*

```
model.NodeVariables = Set(
    model.Nodes, ordered=True)

model.StageDerivedVariables = Set(
    model.Stages, ordered=True)

model.NodeDerivedVariables = Set(
    model.Nodes, ordered=True)

model.ScenarioBasedData = Param(
    within=Boolean, default=True)

model.Bundling = Param(
    within=Boolean, default=False)

model.Bundles = Set(ordered=True)

model.BundleScenarios = Set(
    model.Bundles, ordered=True)
```

- You can specify variable locations by node as well as stage
- Frequently a subset of stage variables are "derived" – that is, completely calculable from other stage variables. This avoids unnecessary NACs.
- Data can be provided "per node" instead of "per scenario"
- Scenarios may be bundled together by
  - Specifying the bundle names
  - Assigning scenarios to bundles
- Bundles can also be created by randomly assigning scenarios

Scenario Data in PySP

- Two methods are available to specify scenario-specific data
  - Scenario-based
  - Node-based
- In the scenario-based approach, a single and complete .dat file is specified for each individual scenario
  - Redundant, but straightforward if computer-generated
- In the node-based approach, a single .dat file is specified for each node in the scenario tree
  - Maximally compact, but requires some book-keeping
- This example uses scenario-based data, with data files that simply define *d*.

Solving the Extensive Form (1)

- In PySP, the runef script is provided to create, write, and solve the extensive form of a stochastic programming model

- The basic command-line:
  - `runef -m models -i scenarios --solve \ --solver=glpk`
  - Assumes that our deterministic model is in a file called `ReferenceModel.py` in the `models` sub-directory
  - Assumes scenario-specific .dat files are in the `scenarios` sub-directory

- NOTE: Even commercial solvers often have difficulty solving EFs
- NOTE: The `runef` script is agnostic to the details of the EF
  - Integers, Binaries, DAEs, disjunctions, bi-level, …

Solving the Extensive Form (2)

- After the solution, runef reports information about scenario tree solution
  - Variable values
  - Costs
- Also accessible via an API
  - That is not presently documented…

```
Tree Nodes:

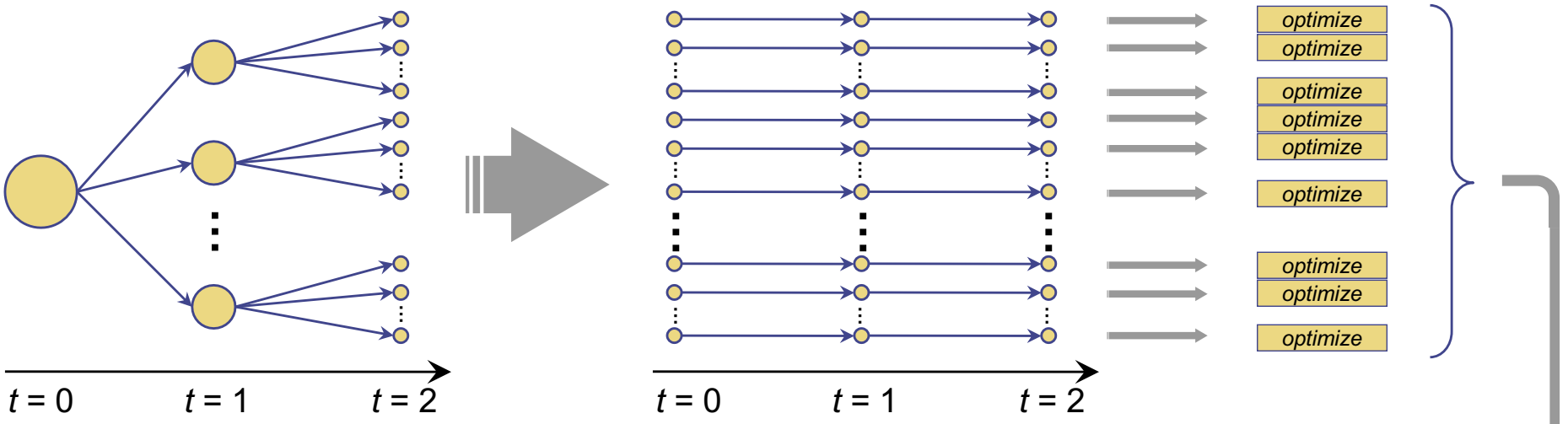
        Name=RootNode
        Stage=FirstStage
        Parent=None
        Conditional probability=1.0000
        Children:
                d1
                d2
                d3
                d4
                d5
        Scenarios:
                s1
                s2
                s3
                s4
                s5
        Expected cost of (sub)tree rooted at node=   76.5000

        Name=d1
        Stage=SecondStage
        Parent=RootNode
        Conditional probability=0.2000
        Children:
                None
        Scenarios:
                s1
        Expected cost of (sub)tree rooted at node=   64.5000

        Name=d2
        Stage=SecondStage
        Parent=RootNode
        Conditional probability=0.2000
        Children:
                None
```

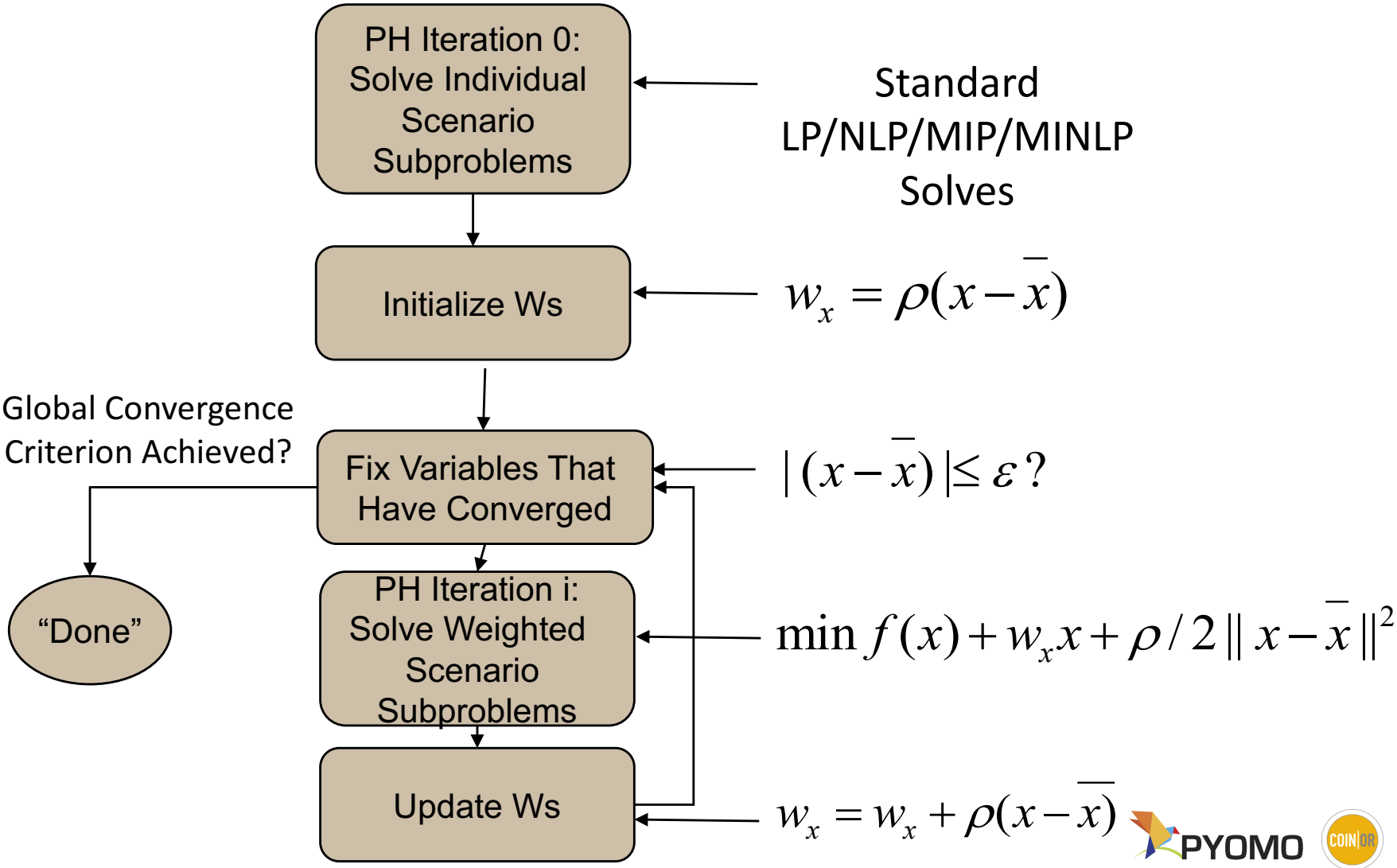
What if the Problem is Too Difficult?

- Use problem decomposition!
  - Stage-wise (e.g., Benders decomposition [Benders, 1962])
    - Master problem (1st stage), independent (2nd stage) subproblems
    - Master problem grows with cuts from subproblems
  - Scenario-based (e.g., Progressive Hedging [Rockafellar & Wets, 1991])
    - *No* master problem
    - Iteratively converge NACs by penalizing deviation from consensus



Progressive Hedging Algorithm

Idea: Use scenario-based decomposition



Progressive Hedging (1)

- The execution of PH requires the specification of the penalty parameter (rho)
- A global rho value can be easily specified:
  - `runph -m models -i scenarios --default-rho=0.05`
- The quadratic penalty term in PH can be computationally problematic
  - E.g., quadratic MIP solvers can be 10x or slower than MIP solvers
  - Open-source quadratic solvers are (almost) non-existent
- PySP provides automatic, generic linearization mechanisms
  - Requires specification of variable lower and upper bounds
  - Specify number of breakpoints, distribution strategy
  - ```
    runph -m models -i scenarios \
       --solver=glpk --default-rho=0.05 \
       --linearize-nonbinary-penalty-terms=100
    ```
    ```

Progressive Hedging (2)

- In the presence of integers, PH is no longer guaranteed to converge
  - Cycling behavior
  - Stagnation behavior
- To facilitate PH convergence for mixed-integer stochastic programs, PySP provides various configurable mechanisms
  - "Watson-Woodruff" Extensions
    - Computational Management Science (2009)
  - Implemented via a generic plug-in callback framework
- Capabilities include:
  - Variable fixing
  - Cycle detection
  - Cycle breaking
  - Slamming

Parallelism for Free in PySP / PH

- PH is widely executed in parallel, on medium-scale clusters and many-core SMP workstations
- Parallelism in Pyomo (and therefore PySP) is based on Pyro
  - Straightforward master-slave or RPC-style communication
- Requires the following "helper" processes
  - `pyomo_ns` – the name server
  - `dispatch_srvr` – routes requests to perform work to workers
  - `phsolversever` – responsible for manipulation and solution of specific scenario sub-problems
- Command-line is then as follows (in the SMP case):
  - `mpiexec –np 1 pyomo_ns : -np 1 dispatch_srvr : -np X phsolverserver : -np 1 runph --solver-manager=phpyro …`

Model Construction via Callbacks in PySP

- Scenario models can also be specified via callbacks in the model file supplied to PySP scripts (e.g., `runef` and `runph`)
  - Allows for the use of `ConcreteModel` alternatives
  - Avoids the need for scenario-specific .dat files

- Callback signature is as follows:

```
def pysp_instance_creation_callback(scenario_name, node_names):
    # load the data and build some_model…
    return some_model
```

- From the scenario and node names, the end-user can construct and return an arbitrary scenario model

Scenario Tree Definition via Callbacks (1)

- Scenario trees can also be specified via callbacks in the model file supplied to PySP scripts (e.g., `runef` and `runph`)
  - Signature:

```
def pysp_scenario_tree_model_callback():
    # load data and build scenario_tree_model…
    return scenario_tree_model
```

  - The simplest approach is to import the Abstract model and create an instance:

```
from pyomo.pysp.scenariotree.tree_structure_model \
    import CreateAbstractScenarioTreeModel
def pysp_scenario_tree_model_callback():
    st = CreateAbstractScenarioTreeModel().create_instance()
    # ...
    return st
```

Scenario Tree Definition via Callbacks (2)

- Constructing the scenario tree for the News Vendor example:

```
from pyomo.pysp.scenariotree.tree_structure_model import CreateAbstractScenarioTreeModel
def pysp_scenario_tree_model_callback():
    m = CreateAbstractScenarioTreeModel().create_instance()
    m.Stages.add('FirstStage')
    m.Stages.add('SecondStage')
    m.Nodes.add('Root')
    m.NodeStage['Root'] = 'FirstStage'
    m.ConditionalProbability['Root'] = 1
    for i in range(5):
        node = 'd%s' % (i+1,)
        m.Nodes.add(node)
        m.NodeStage[node] = 'SecondStage'
        m.Children['Root'].add(node)
        m.ConditionalProbability[node] = 1./N
        scenario = 'scenario%s' % (chr(ord('A')+i),)
        m.Scenarios.add(scenario)
        m.ScenarioLeafNode[scenario] = node
    m.StageVariables['FirstStage'].add('x')
    m.StageVariables['SecondStage'].add('t')
    m.StageCost['FirstStage'] = 'FirstStageCost'
    m.StageCost['SecondStage'] = 'SecondStageCost,
    return m
```

Scenario Tree definition via callbacks (3)

- Two-stage models are so common that we provide a helper function:

```
from pyomo.pysp.scenariotree.tree_structure_model import \
    CreateConcreteTwoStageScenarioTreeModel
def pysp_scenario_tree_model_callback():
    m = CreateConcreteTwoStageScenarioTreeModel(5)
    m.StageVariables[m.Stages.first()].add('x')
    m.StageVariables[m.Stages.last()].add('t')
    m.StageCost[m.Stages.first()] = 'FirstStageCost'
    m.StageCost[m.Stages.last()] = 'SecondStageCost,
    return m
```

- Note that the Scenarios and Nodes are named differently relative to our initial example
  - E.g., scenarios are now named “Scenario1”, “Scenario2”, …

The PySP Solver Zoo

- We only regularly advertise two solvers in PySP
  - `runef` and `runph`
- But, we also have implementations and/or interfaces to a wide variety of alternatives
  - Benders / L-shaped
    - `$ python -m pyomo.pysp.solvers.benders`
  - BBPH (Branch-and-Bound / PH hybrid)
    - `$ python -m pyomo.pysp.solvers.bbph`
  - DDSIP (Dual Decomposition)
    - `$ python -m pyomo.pysp.solvers.ddsip`
  - Stochastic Decomposition (Sen et al.)
    - `$ python -m pyomo.pysp.solvers.sd`
  - SchurIpopt (Laird et al.)
    - `$ python -m pyomo.pysp.solvers.schuripopt`

The End

Sandia National Laboratories

Questions?

CCR Center for Computing Research

PYOMO COIN-OR

Power Grid: A Rich Application Space

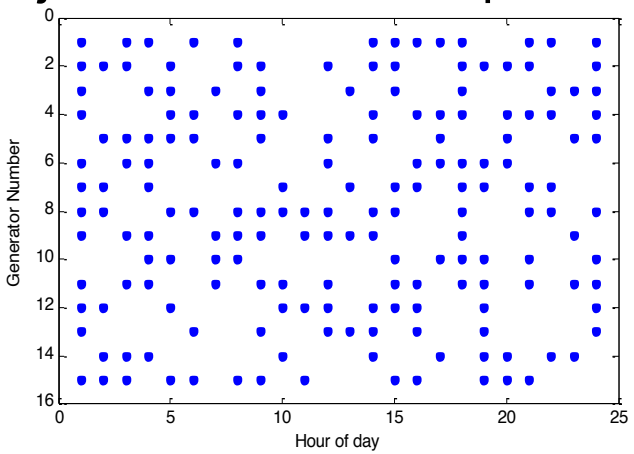
Sandia National Laboratories

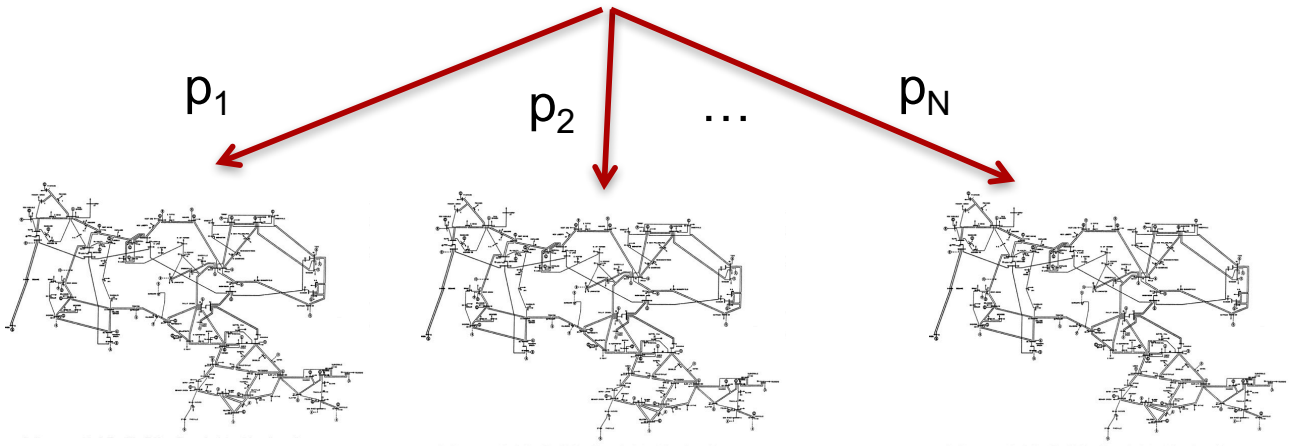
- Power grid operations with high-penetration renewables
  - Stochastic economic dispatch
  - Stochastic unit commitment
  - Contingency analysis
- Transmission / generation expansion planning

- Common theme: operate a large scale system with discrete decisions with high reliability in the face of uncertain {demands, generation, configurations} and at low cost.

Stochastic Unit Commitment (At Scale)

Objective: Minimize expected cost





Scenario 1 Scenario 2 … Scenario N

First stage variables:

- Unit On / Off

Nature resolves uncertainty

- Load
- Renewables output
- Forced outages

Second stage variables (*per time period*):

- Generation levels
- Power flows
- Voltage angles
- …

Solving the Stochastic Extensive Form

Sandia National Laboratories

- Reliability Unit Commitment (RUC) Test Instance: WECC-240++
- J.E. Price, Reduced Network Modeling of WECC as a Market Design Prototype, 2011 IEEE PES General Meeting
- Changes necessary to create viable RUC test case
  - Addition of realistic ramping rates and min up/down time constraints
- Results

Table 3 Solution quality statistics for the extensive form of the *WECC-240-r1* instance, given 4 hours of run time.

# Scenarios	Objective Value	MIP Lower Bound	Gap %	Run Time (s)
3	64278.20	63797.72	0.75	14491
5	62740.67	62180.86	0.89	14723
10	61563.10	60835.45	1.18	14630
25	61455.55	59963.78	2.36	14960
50	61911.74	59540.87	3.83	15480
100	62388.85	59548.23	4.51	16562

Progressive Hedging Results: WECC-240++

Sandia National Laboratories

Table 7 Solve time (in seconds) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 6$, and $\gamma = 0.025$

# Scenarios	Convergence Metric	Obj. Value	PH L.B.	# Vars Fx.	Time	
		64-Core Workstation Results				Latest…
3	0.0 (20 iters)	64213.397	63235.381	4080	508	166
5	0.0 (in 18 iters)	62642.531	61767.253	4079	674	119
10	0.0 (in 35 iters)	61396.553	60476.604	4066	648	167
25	0.0 (in 22 iters)	60935.040	59992.622	4066	761	212
50	0.0 (in 15 iters)	60625.149	59631.839	4034	1076	280
100	0.0 (in 25 iters)	61155.387	60014.571	4080	1735	315
		Red Sky Results				
50	0.0 (in 16 iters)	60623.343	59779.813	4007	404	
100	0.0 (in 25 iters)	61120.943	60275.744	4080	549	

ISO-NE results are obtained on Red Sky on average in 10 minutes, 20 minutes in the worst case (with 100 scenarios)

CCR Center for Computing Research

PYOMO COIN-OR

Improved UC Formulations?

- Morales-Espana et al. (2013)
  - Extends prior tight formulation by Ostrowski et al.
- Shows off advantage of PH, in that improved deterministic models immediately impact stochastic solve times
- Results

Table 10 Solve time (in seconds) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 3$, and the MTR deterministic UC model.

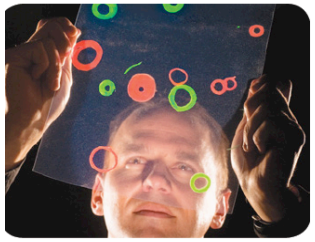
# Scenarios	Convergence Metric	Obj. Value	PH L.B.	# Vars Fx.	Time
		64-Core Workstation Results			
3	0.0 (in 36 iters)	64141.771	64109.021	4080	237
5	0.0 (in 23 iters)	62628.532	62499.212	4080	161
10	0.0 (in 26 iters)	61384.016	61327.734	4080	215
25	0.0 (in 41 iters)	60927.903	60850.717	4080	366
50	0.0 (in 11 iters)	60617.311	60470.956	4044	318

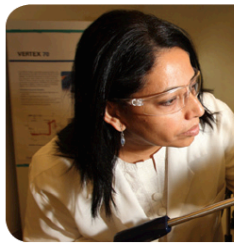
- ISO-NE results drop to 15 minutes maximum (10 average)

Parallelization and Bundling

- Progressive Hedging is, at least conceptually, easily parallelized
  - Scenario sub-problem solves are clearly independent
  - Advantage over Benders, in that "bloat" is distributed
    - Critical in low-memory-per-node cluster environments
  - Parallel efficiency drops rapidly as the number of processors increases
    - But: Relaxing barrier synchronization does not impact PH convergence
  - Bundling scenarios might help with parallel scaling
    - May increase number of iterations required

- PH can provide bounds!
  - Now comes with (rather tight) lower bounds
  - See "Obtaining Lower Bounds from the Progressive Hedging Algorithm for Stochastic Mixed-Integer Programs"





Sandia National Laboratories

Exceptional service in the national interest

11. Solving Bilevel Problems

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Overview of Bilevel Programming

A *bilevel program* is a mathematical program in which a subset of decision variables is constrained to take values associated with an optimal solution of a distinct, "lower" level mathematical program.

General formulation:

$$\begin{array}{ll} \min_{x \in X} & F(x, y) \\ \text{s.t.} & G(x, y) \leq 0 \\ & y \in P(x) \end{array}$$

Upper-level problem

where

$$P(x) = \begin{array}{ll} \operatorname{argmin}_{y \in Y} & f(x, y) \\ \text{s.t.} & g(x, y) \leq 0 \end{array}$$

Lower-level problem

Example: Modeling Security Problems

- Opponents must anticipate each other's moves
- Strategy should account for how opponent (best) responds

Extremely complex

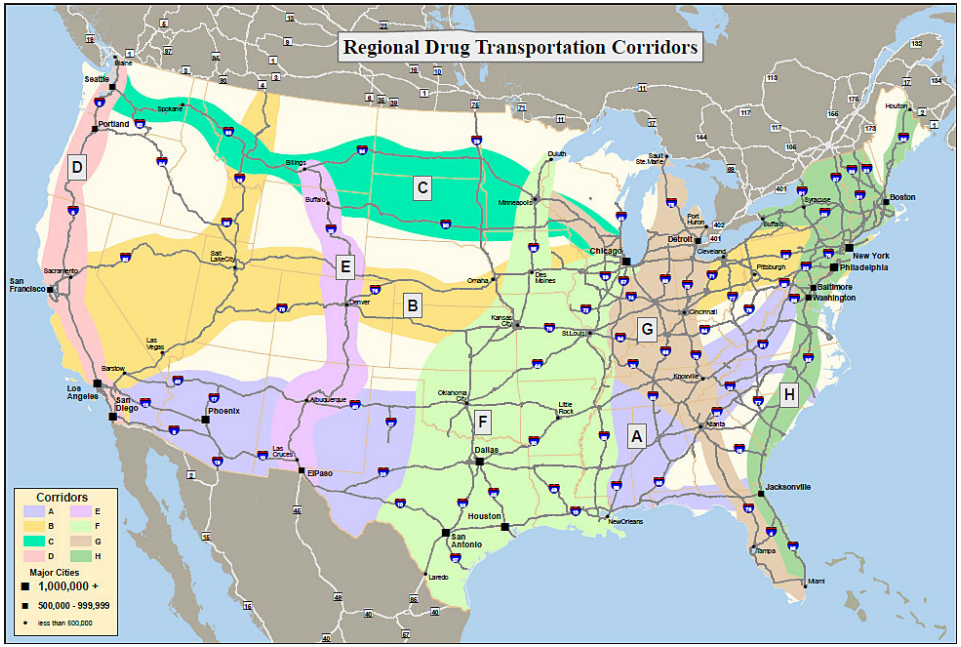
- Impossible to enumerate the set of all states in the game

Stackelberg games – bilevel programs

Example: Smuggling Interdiction

Interdictor minimizes the potential for a smuggler to evade detection

- Interdictor installs defenses (x) to minimize smuggler's evasion probability
- Smuggler traverses path (y) that maximizes the probability of evasion



Origin-destination nodes maybe unknown

- *F(x,y)* – evasion probability
- *X* – interdictor's constraints (e.g. resource, budget)
- *P(x)* – feasible paths given *x*

Modeling Bilevel Programs

No algebraic modeling language currently provides an *intuitive* syntax for expressing the structure of bilevel programs!

MPEC formulations are supported in several AMLs

- AMPL, AIMMS, GAMS, …
- MacMPEC includes bilevel programs that are reformulated as single-level programs using optimality conditions

Bilevel problems can be expressed in several modeling languages:

- GAMS, YALMIP
- Explicitly pass variables and constraints to a bilevel solver

A Simple Bilevel Example

Practical Bilevel Optimization: Algorithms and Applications

Jonathan Bard

Example 5.1.1

$$
\begin{array}{lll}
\min_{x \geq 0} & x - 4y & \\
\text{s.t.} & \min_{y \geq 0} & y \\
 & \text{s.t.} & -x - y \leq -3 \\
 & & -2x + y \leq 0 \\
 & & 2x + y \leq 12 \\
 & & -3x + 2y \leq -4
\end{array}
$$

Modeling Example 5.1.1 with YALMIP

```
sdpvar x, y;

OO = x – 4y;
CO = [x] >= 0;

OI = y;
CI = [[y] >= 0,
       -x -  y <= -3,
      -2x +  y <=  0,
       2x +  y <= 12,
      -3x + 2y <= -4];

solvebilevel(CO,OO,CI,OI,[y])
```

- Solve a bilevel problem using a simple branching strategy.
- Upper level problem defined by `CO` and `OO`
- Lower level problem defined by `CI` and `OI`, with decision variable `y`.

Note: This example adapted from the YALMIP bilevel documentation.

Modeling Example 5.1.1 with GAMS

```
positive variables x,y; variables objout,objin;
equations defout,defin,e1,e2,e3,e4;

defout.. objout =e= x - 4*y;
defin..  objin  =e= y;

e1..     - x -   y =l= -3;
e2..   -2*x +   y =l=  0;
e3..    2*x +   y =l= 12;
e4..    3*x - 2*y =l=  4;

model bard / all /;

$echo bilevel x min objin * defin e1 e2 e3 e4 > "%emp.info%"

solve bard us emp min objout;
```

Writes an "empinfo" file that tells the solver that this is a bilevel problem with a lower level problem that minimizes objective *objin* with variables *y* subject to the constraints (*defin*), *e1*, *e2*, *e3* and *e4*.

Modeling Example 5.1.1 with Pyomo

```
from pyomo.environ import *
from pyomo.bilevel import *

M = ConcreteModel()
M.x = Var(bounds=(0,None))
M.y = Var(bounds=(0,None))
M.o = Objective(expr=M.x - 4*M.y)

M.sub = SubModel(fixed=M.x)
M.sub.o = Objective(expr=M.y)
M.sub.c1 = Constraint(expr=-  M.x -   M.y <= -3)
M.sub.c2 = Constraint(expr=-2*M.x +   M.y <=  0)
M.sub.c3 = Constraint(expr= 2*M.x +   M.y <= 12)
M.sub.c4 = Constraint(expr=-3*M.x + 2*M.y <= -4)
```

- Lower level problem is declared with a *SubModel* component.
- The *var* argument indicates the lower level variables.
- Objectives, variables and constraints for the lower level problem are declared within this component.

Pyomo Extensions for Bilevel Programs

Modeling extensions

- Modeling components (pyomo.bilevel)

Model transformations

- Can be applied automatically

Custom solvers

- *Solvers tailored for specific classes of bilevel problems*

Solving Bilevel Problems

Goal: Enable solution of bilevel problems with standard solvers

Process:

- Model problem with SubModel components
- Transform the problem to a standard form
  - LP, MIP, etc
- Apply a suitable solver

Reformulations for linear bilevel programming (BLP)

A. BLP with continuous variables
B. Quadratic minimax with continuous lower-level variables

(A) BLP with Continuous Variables

Problem:

$$
\begin{aligned}
\min_{x \geq 0} \quad & c_1^T x + d_1^T y \\
\text{s.t.} \quad & A_1 x + B_1 y \leq b_1 \\
& \min_{y \geq 0} \quad c_2^T x + d_2^T y \\
& \qquad\qquad A_2 x + B_2 y \leq b_2
\end{aligned}
$$

Reformulation: Replace lower-level problem with corresponding optimality conditions

$$
\begin{aligned}
\min \quad & c_1^T x + d_1^T y \\
\text{s.t.} \quad & A_1 x + B_1 y \leq b_1 \\
& d_2 + B_2^T u - v = 0 \\
& b_2 - A_2 x - B_2 y \geq 0 \perp u \geq 0 \\
& y \geq 0 \perp v \geq 0 \\
& x \geq 0, y \geq 0
\end{aligned}
$$

(A) BLP with Continuous Variables (cont'd)

Idea: Analyze the MPEC reformulation

Example:

- Use a custom solver that considers complementarity conditions (Bard, 1998)

Example:

- Chain reformulations: BLP -> MPEC -> GDP -> MIP
- Provide "BigM" values for unbounded variables
- Apply standard MIP solver (Fortuny-Amat and McCarl, 1981)

Example:

- Reformulate the complementarity conditions with nonlinear constraints (Ferris and Dirkse, 2005)

(B) Quadratic Min/Max

Problem:

- Upper level constraints do not constrain *y*

$$X = \{x | A_1 x \leq b_1, x \geq 0\}$$

- The upper decision variables may binary

$$\begin{aligned} \min_{x \in X} \quad \max_{y \geq 0} \quad & c_1^T x + d_1^T y + x^T Q y \\ \text{s.t.} \quad & A_2 x + B_2 y \leq b_2 \end{aligned}$$

Reformulation: Replace lower-level problem with the linear dual

$$\begin{aligned} \min \quad & c_1^T x + (b_2 - A_2 x)^T v \\ \text{s.t.} \quad & B_2^T v \geq d_1 + Q^T x \\ & A_1 x \leq b_1 \\ & x \geq 0, v \geq 0 \end{aligned}$$

(B) Quadratic Min/Max (cont'd)

Case 1:

- $A_2 \equiv 0$
- The reformulation is a simple LP (or a MIP if x are binary)

Case 2:

- The upper-level decision variables *x* are binary
- Reformulate the bilinear objective terms as disjunctions:

$$
\begin{aligned}
\min \quad & c_1^T x + b_2^T v - 1^T z \\
\text{s.t.} \quad & B_2^T v \geq d_1 + Q^T x \\
& \begin{bmatrix} x_i = 0 \\ z_i = 0 \end{bmatrix} \veebar \begin{bmatrix} x_i = 1 \\ z_i = A_2^T(i,*)v \end{bmatrix} \\
& x \in X, v \geq 0
\end{aligned}
$$

- Reformulate this GDP -> MIP using "BigM" values

Solving Bilevel Programs in Pyomo

Python Script:

- Formulate the model
- Apply desired model reformulations
- Apply a suitable optimizer

  OR
- Directly analyze the model within Python
  - (e.g. using Pyomo's algebraic structure)

Pyomo Command:

- Execute a command that executes a Pyomo meta-solvers
  - Performs suitable reformulations
  - Applies a suitable optimizer
  - Maps the solution to the original problem

```
pyomo solve --solver=bilevel_ld model.py
```

Pyomo Capabilities

Relevant Pyomo Transformations

- core.linear_dual
- bilevel.linear_dual
- bilevel.linear_mpec
- gdp.bigm
- gdp.bilinear
- gdp.chull
- mpec.simple_disjunction
- mpec.simple_nonlinear

Relevant Pyomo Meta-Solvers

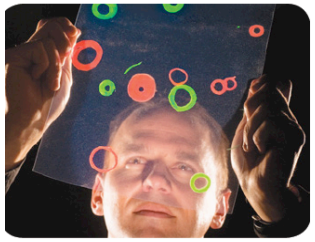
- bilevel_ld

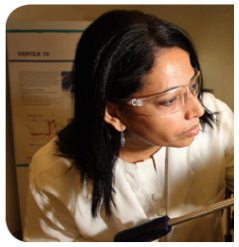
Ongoing Work …

- Generalization and maturation of transformations
  - E.g. Working with general BLP models
- Automatic recognition of bilevel structure
  - Can we automate the application of reformulations?
- Parameterizing transformations
  - How can we flexibly specify transformation options?
  - E.g. Specifying big-M values for specific complementarity conditions
- Additional meta-solvers
  - E.g. bilevel_blp

Acknowledgements

- Richard L. Chen
- William E. Hart
- John D. Siirola
- Jean-Paul Watson





Sandia National Laboratories

Exceptional service in the national interest

12. Mathematical Programming with Equilibrium Conditions

PYOMO

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Overview

Sandia National Laboratories

- Mathematical Programming with Equilibrium Constraints (MPEC)
  - Engineering design, economic equilibrium, multilevel games
  - Feasible region may be nonconvex and disconnected

- Equilibrium Constraints
  - Variational inequalities
  - Complementarity conditions
  - Optimality conditions (for bilevel problems)

MPEC formulations

- General MPEC models can be expressed as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0 \\ & a_i \leq w_i(x) \leq b_i \perp v_i(x) \quad i = 1 \ldots m \end{aligned}$$

- The last set of constraints are generalized mixed complementarity conditions (Ferris, Fourer, and Gay, ‘06), which have the form

$$\begin{aligned} \text{either } w_i(x) = a_i \quad & \text{and} \quad v_i(x) \geq 0 \\ \text{or } w_i(x) = b_i \quad & \text{and} \quad v_i(x) \leq 0 \\ \text{or } a_i < w_i(x) < b_i \quad & \text{and} \quad v_i(x) = 0 \end{aligned}$$

Modeling languages support MPECs

- AMPL
  - The `complements` keyword is used to denote complementarity between two constraints, expressions or variables
- GAMS
  - The `complements` keyword is used to denote complementarity between two constraints, expressions or variables
- AIMMS
  - Express mixed complementarity conditions by declaring complementarity variables along with associated constraints
- YALMIP
  - The `complements` function declares a constraint that reflects a mixed complementarity condition.

➢ Common challenge: lack of control over how the complementarity constraints are exposed to the solver

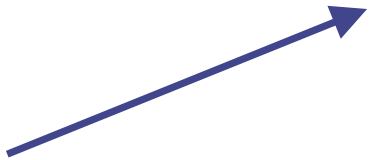
Complementarity conditions in Pyomo

Sandia National Laboratories

```
from pyomo.environ import *
from pyomo.mpec import Complementarity

M = ConcreteModel()
M.x = Var(bounds=(-1,2))
M.y = Var()

M.c3 = Complementarity(expr=(M.y - M.x**2 + 1 >= 0, M.y >= 0))
```



- The `Complementarity` component declares a complementarity condition
- The tuple argument specifies the two constraints, expressions, or variables in the complementarity condition.

This model definition is solver agnostic!

A simple nonlinear reformulation

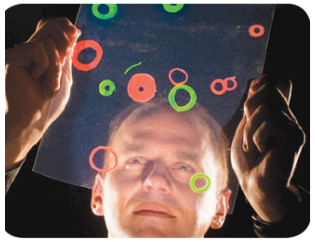
$$
\begin{aligned}
\min \quad & f(x) \\
s.t. \quad & h(x) = 0 \\
& a_i \leq \omega_i \leq b_i && i = 1 \ldots m \\
& \omega_i = w_i(x) && i = 1 \ldots m \\
& (\omega_i - a_i) v_i(x) \leq 0 && i = 1 \ldots m \\
& (\omega_i - b_i) v_i(x) \leq 0 && i = 1 \ldots m
\end{aligned}
$$

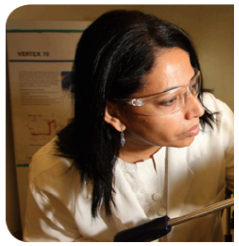
- NOTE: There are serious difficulties with solving this formulation as standard stability assumptions are not met.
  - But other nonlinear transformations exist!

A simple disjunctive reformulation

Sandia National Laboratories

$$
\begin{aligned}
\min \quad & f(x) \\
\text{s.t.} \quad & h(x) = 0 \\
& \begin{bmatrix} y_{1,i} \\ w_i(x) = a_i \\ v_i(x) \geq 0 \end{bmatrix} \vee \begin{bmatrix} y_{2,i} \\ w_i(x) = b_i \\ v_i(x) \leq 0 \end{bmatrix} \vee \begin{bmatrix} y_{3,i} \\ a_i < w_i(x) < b_i \\ v_i(x) = 0 \end{bmatrix} && i = 1 \dots m \\
& y_{1,i} + y_{2,i} + y_{3,i} = 1 && i = 1 \dots m \\
& y_{1,i}, y_{2,i}, y_{3,i} \in \{0,1\} && i = 1 \dots m
\end{aligned}
$$





Sandia National Laboratories

Exceptional service in the national interest

Persistent Solver Interfaces

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Who should consider persistent solvers?

- Are you solving the same problem multiple times with small changes?
- If so, you should consider persistent solvers!

Why Persistent Solvers?

- Performance

Why Not Persistent Solvers (Disclaimer)?

- Slightly more complex code is needed
- Introducing bugs in your code is quite easy

An Example: Warehouse Location

- Suppose we want to know how the cost changes as we increase the number of warehouses built.

An example: Warehouse Location

Sandia National Laboratories

How does the objective change as P increases?

$$\sum_{w \in W} y_w \leq P$$

```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
    . . .
    ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

An Example: Warehouse Location

Sandia National Laboratories

First we will slightly modify the problem:

1. Make the problem larger (data is generated randomly for simplicity)
2. Make P a mutable parameter

```
N_locations = 50
N_customers = 50
W = list(range(N_locations))
C = list(range(N_customers))
numpy.random.seed(0)
d = np.random.randint(low=1, high=100,
size=(N_locations,N_customers))

model.P = Param(mutable=True, initialize=1)
```

An Example: Warehouse Location

Create lists to store the number of warehouses and the cost

```
opt = SolverFactory('gurobi_direct')
t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    model.P.value = p
    res = opt.solve(model, load_solutions=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)

plt.plot(P_list, obj_list)
plt.show()
```

An Example: Warehouse Location

Loop through the values of P, change the value of the mutable Param, solve the problem, and record the cost

```
opt = SolverFactory('gurobi_direct')
t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    model.P.value = p
    res = opt.solve(model, load_solutions=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)

plt.plot(P_list, obj_list)
plt.show()
```

An Example: Warehouse Location

Print the time required to execute the loop

```
opt = SolverFactory('gurobi_direct')
t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    model.P.value = p
    res = opt.solve(model, load_solutions=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)

plt.plot(P_list, obj_list)
plt.show()
```

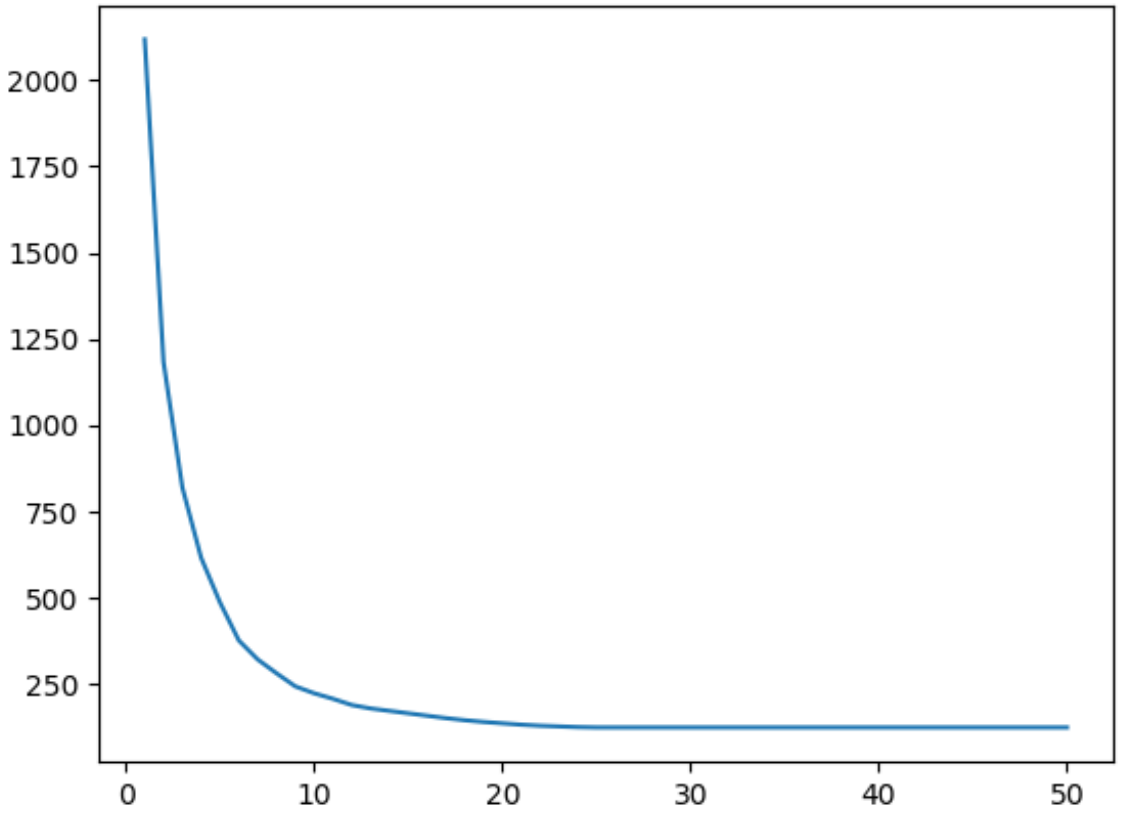
An Example: Warehouse Location

```
opt = SolverFactory('gurobi_direct')
t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    model.P.value = p
    res = opt.solve(model, load_solutions=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)
```

Plot the results

```
plt.plot(P_list, obj_list)
plt.show()
```

An Example: Warehouse Location



Execution time: 15.4 seconds

An Example: Warehouse Location

The model must be provided to the solver with the "set_instance" method.

```
opt = SolverFactory('gurobi_persistent')
opt.set_instance(model)
```

```
t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    opt.remove_constraint(model.num_warehouses)
    model.P.value = p
    opt.add_constraint(model.num_warehouses)
    res = opt.solve(load_solutions=False,
                    save_results=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)

plt.plot(P_list, obj_list)
plt.show()
```

An Example: Warehouse Location

Remove the constraint from the solver, change the value of P, and add the constraint back to the solver

```
opt = SolverFactory('gurobi_persistent')
opt.set_instance(model)

t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    opt.remove_constraint(model.num_warehouses)
    model.P.value = p
    opt.add_constraint(model.num_warehouses)
    res = opt.solve(load_solutions=False,
                    save_results=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)

plt.plot(P_list, obj_list)
plt.show()
```

An Example: Warehouse Location

Sandia National Laboratories

Do not pass the model to the solver when calling solve

```
opt = SolverFactory('gurobi_persistent')
opt.set_instance(model)

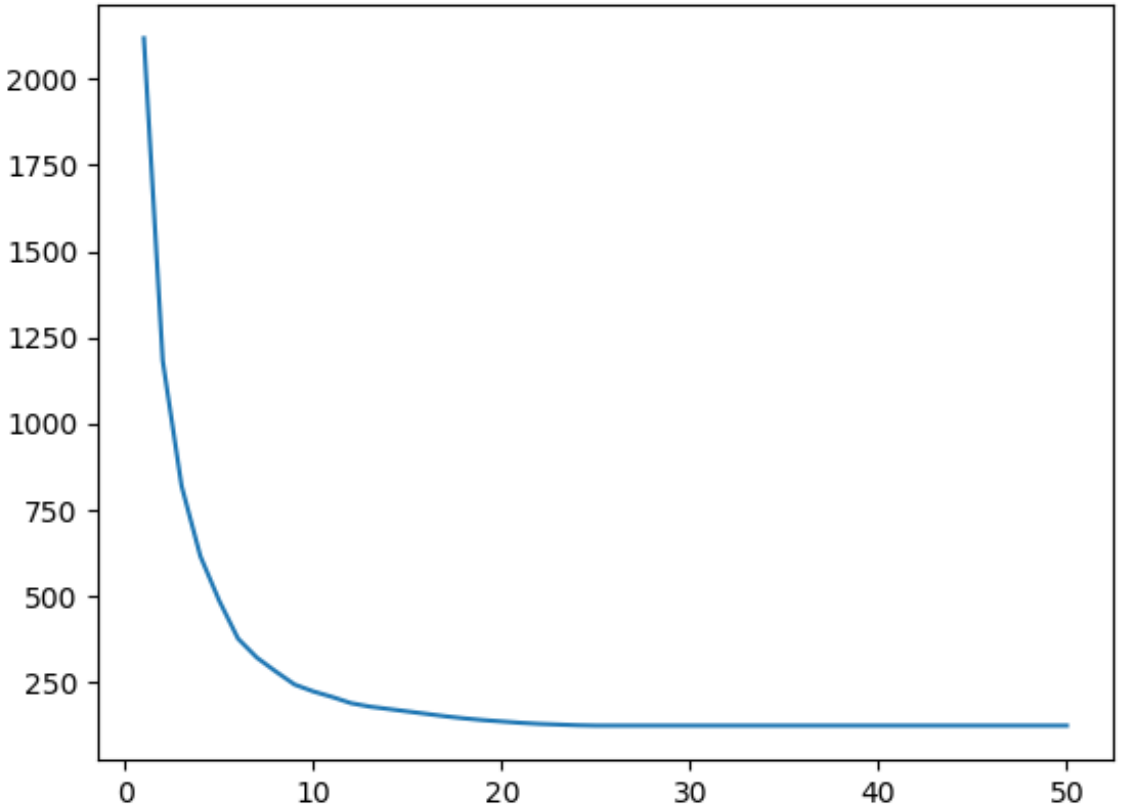
t0 = time.time()
P_list = list(range(1, N_locations+1))
obj_list = []
for p in P_list:
    opt.remove_constraint(model.num_warehouses)
    model.P.value = p
    opt.add_constraint(model.num_warehouses)
    res = opt.solve(load_solutions=False,
                    save_results=False)
    obj_list.append(res.problem.upper_bound)
t1 = time.time()
print(t1-t0)

plt.plot(P_list, obj_list)
plt.show()
```

CCR Center for Computing Research

PYOMO COIN-OR

An Example: Warehouse Location



Execution time: 4.5 seconds
(3 times faster)

Methods/Features

- add_block
- add_constraint
- set_objective
- add_var
- remove_block
- remove_constraint
- remove_var
- set_instance
- update_var
- write
- load_vars

Methods/Features in development

- Callbacks
- Lazy constraints
- Solution pools

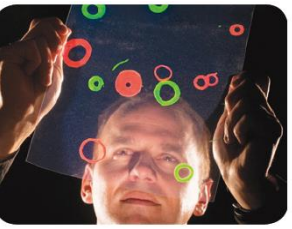
Other Considerations

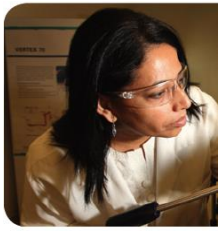
- It is the user's responsibility to keep the model and the solver in sync!
- Currently, persistent solver interfaces only exist for Cplex and Gurobi
- Persistent solvers provide the most benefit for problems that solve quickly relative to the time required to write problem files and read solution files (I have seen 10x speedup with LPs and essentially no speedup with difficult MILPs)

Understand where time is being spent before investing in using a persistent solver!

More info…

- http://pyomo.readthedocs.io/en/latest/solvers/persistent_solvers.html
- http://pyomo.readthedocs.io/en/latest/library_reference/solvers/index.html





Sandia National Laboratories

Exceptional service in the national interest

A Framework for Modeling and Optimizing Complex, Structured Problems

Bethany Nicholson

John D. Siirola

Center for Computing Research
Sandia National Laboratories
Albuquerque, NM

AIChE Annual Meeting October 29 – November 3, 2017

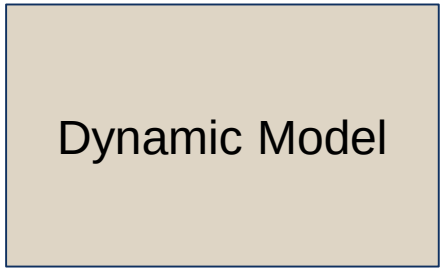
U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration | CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND2017-11920 C

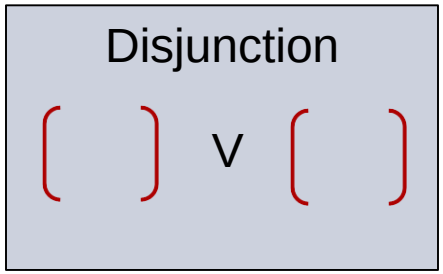
Implement this…

Sandia National Laboratories

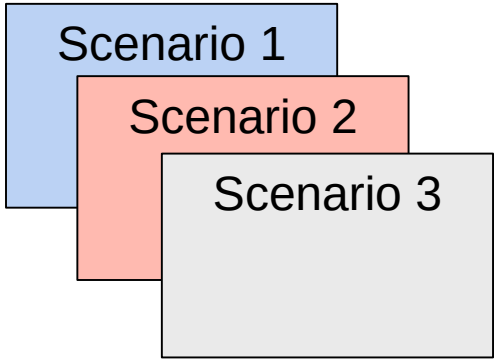
Dynamic Optimization

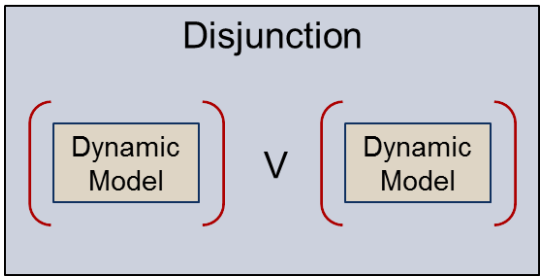


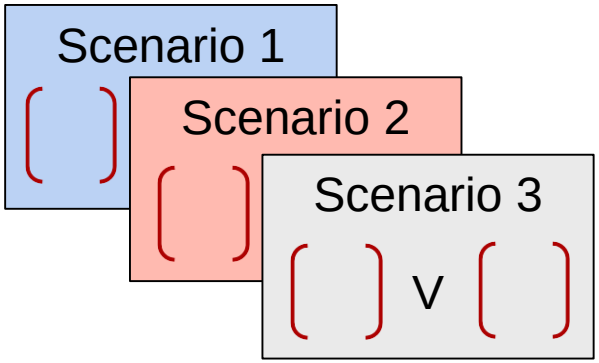
Generalized Disjuntive Programming

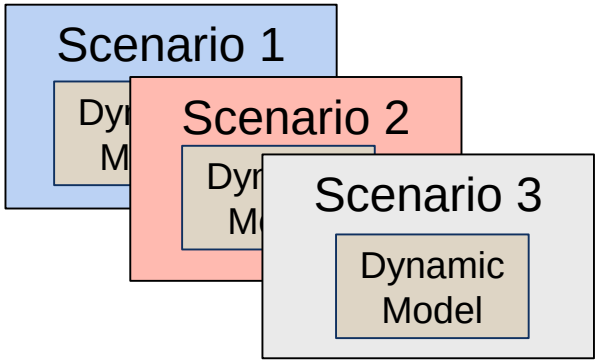


Stochastic Programming





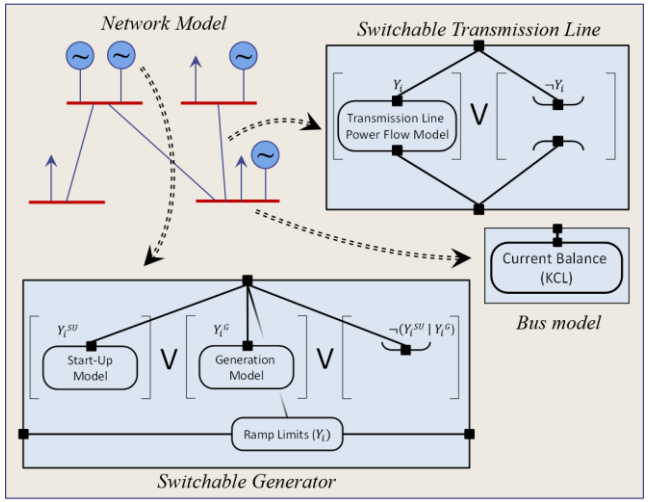


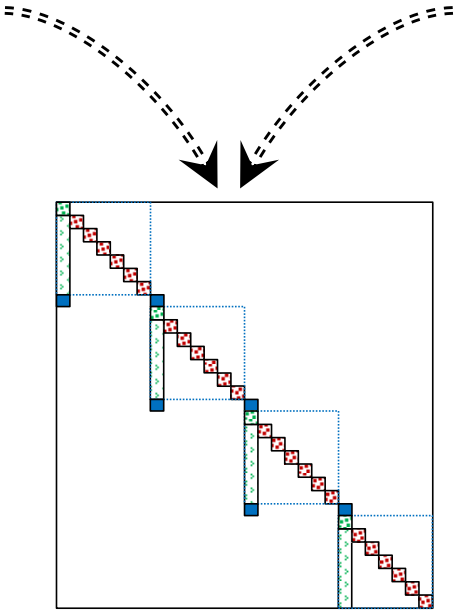


CCR Center for Computing Research

Modeling Structure and Structured Modeling

- Object-based model
  - Explicit structure
  - High-level constructs for more intuitive modeling





Compiled model sent to optimization solver

- Standard algebraic model
  - Flat representation
  - Implicit structure

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

$$\text{S.t.}$$

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

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

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

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

$$\forall\, n, \text{ generator contingency states } c, t$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[1] 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

Software platform

Sandia National Laboratories

- Pyomo: Python Optimization Modeling Objects
- Formulate optimization models within Python


```
from pyomo.environ import *

m = ConcreteModel()

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

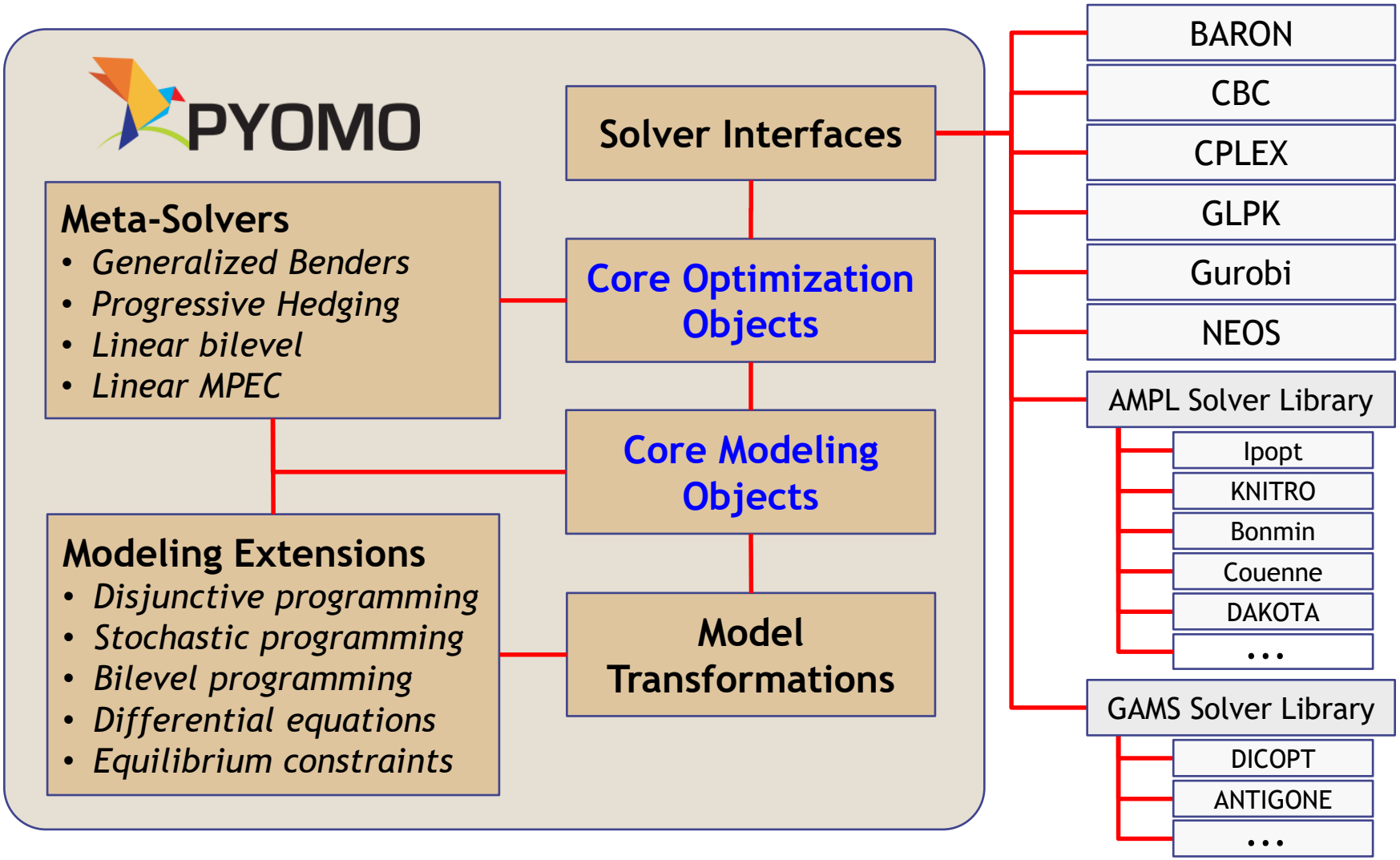
m.obj = Objective(sense = minimize,
    expr = m.x1**2 + (m.x2*m.x3)**4 + m.x1*m.x3
     + m.x2 + m.x2*sin(m.x1+m.x3) )
```

- Utilize high-level programming language to write scripts and manipulate model objects
- Leverage third-party Python libraries

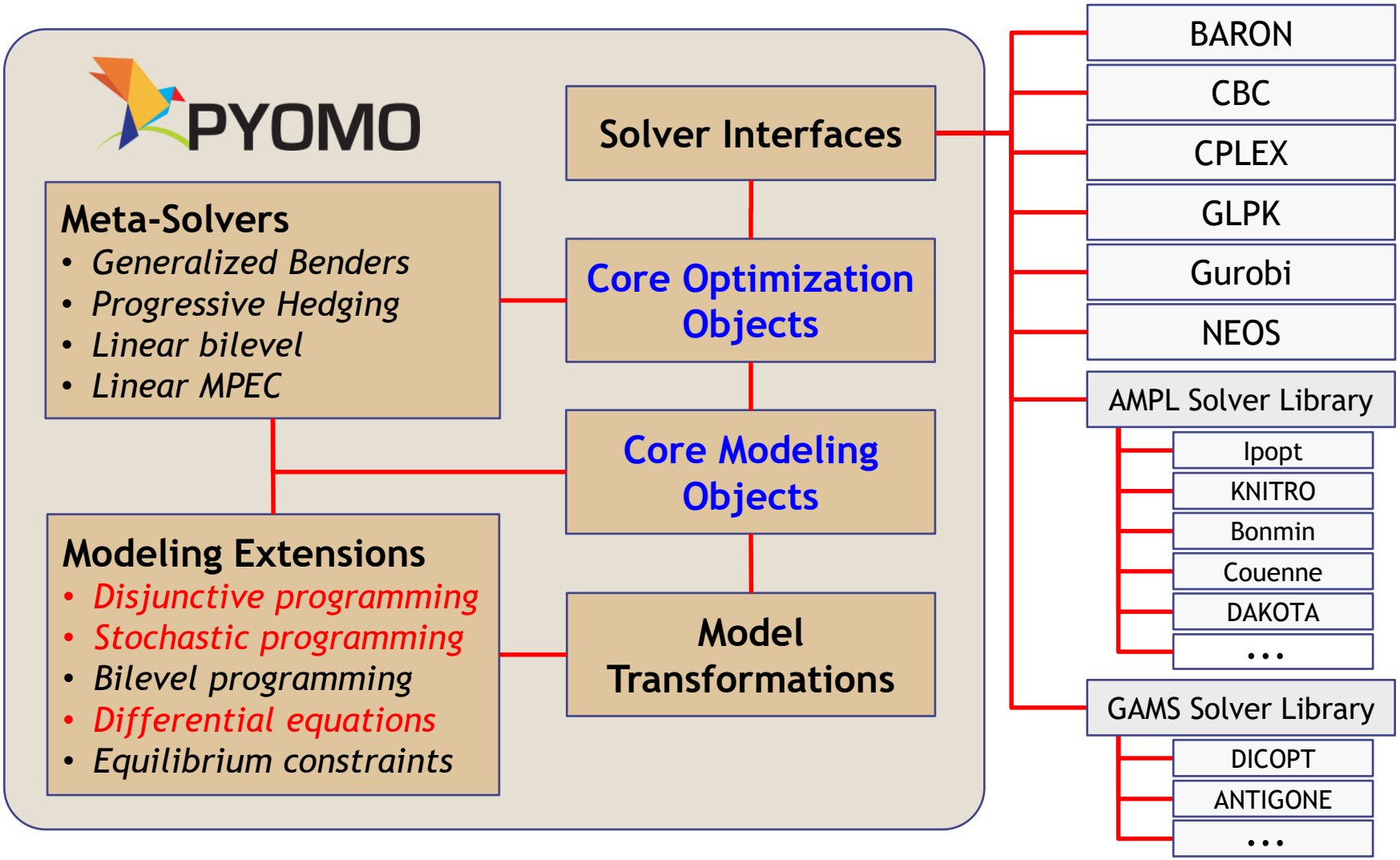
    e.g. SciPy, NumPy, MatPlotLib, Pandas

CCR Center for Computing Research

Pyomo at a Glance



Pyomo at a Glance



Pyomo.GDP

- Represent logical relationships:
  - Sequencing decisions
  - Switching decisions
  - Alternative selection
- Modeling components
  - Disjunct
    - Block of Pyomo components
    - Boolean indicator variable (determines if block is enforced)
  - Disjunction
    - Enforces logical XOR across set of Disjunct indicator variables
- Available relaxations
  - Big-M
  - Convex Hull

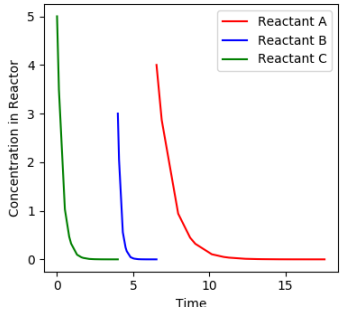
$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \le o \\ c_k = \gamma_{ik} \end{bmatrix}$$
$$\Omega(Y) = true$$

- Framework for representing stochastic programming models, only requiring:
  - deterministic base model
  - scenario tree defining the problem stages and uncertain parameters
- PySP provides two primary solution strategies
  - build and solve the deterministic equivalent (extensive form)
  - Progressive Hedging
  - (plus beta implementations of others, including 2-stage Benders and an interface to DDSIP)
- Parallel infrastructure for generating and solving subproblems on parallel (distributed) computing platforms

- Extend Pyomo syntax to represent:
  - Continuous domains
  - Ordinary or partial differential equations
  - Systems of differential algebraic equations (DAEs)
  - Higher order differential equations and mixed partial derivatives
- Available discretization schemes:
  - Finite difference methods (Backward/Forward/Central)
  - Collocation (Lagrange polynomials with Radau or Legendre roots)
- Extensible framework
  - Write general implementations of custom discretization schemes
  - Build frameworks/meta-algorithms including dynamic optimization
- Interface with numerical simulators
  - Scipy for simulating ODEs
  - CasADi for simulating ODEs and DAEs

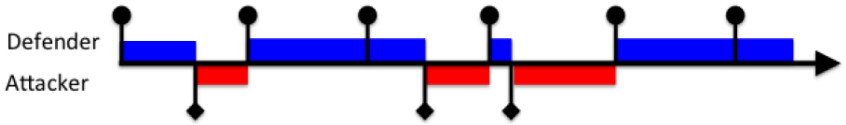
Illustrative examples

Pyomo.DAE & Pyomo.GDP



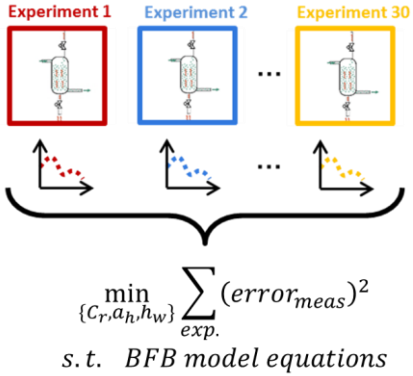
Scheduling incorporating dynamics

Pyomo.GDP & PySP

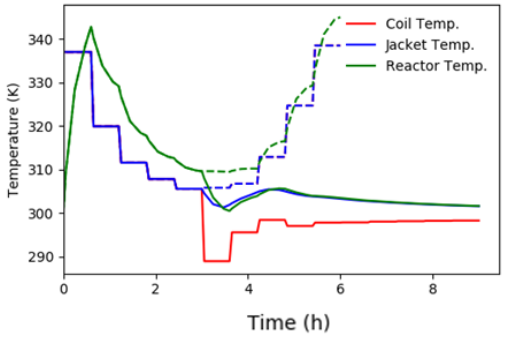


Moving target defense for cyber security

Pyomo.DAE & PySP

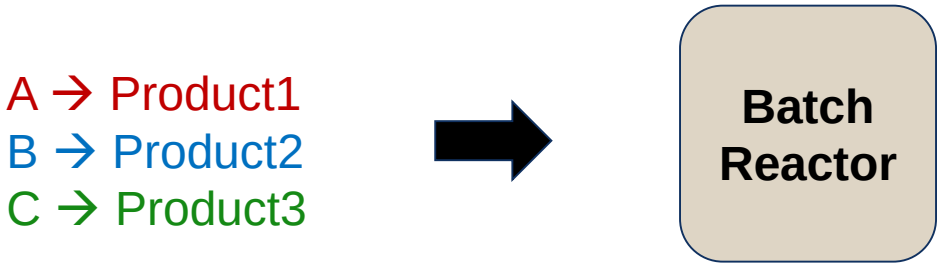


Parameter estimation



Stochastic optimal control

Scheduling incorporating process dynamics



- Scheduling decisions
  - What order to run the reactions?
  - How long to run each reaction?
- Given:
  - Reaction rates and starting concentrations of reactants
  - Deadlines for products
  - Upper bound on exit concentration of reactants
  - Cost function related to purity of product (final concentration of reactants)

Scheduling incorporating process dynamics

Sandia National Laboratories

First order reaction kinetics

$$A \xrightarrow{k} P$$

```
m.products = Set(initialize=['A','B','C'], ordered=True)
m.tau = ContinuousSet(bounds=[0,1]) # Unscaled Time
m.c = Var(m.products, m.tau, bounds=(0,None), initialize=4.0)
m.dc = DerivativeVar(m.c, wrt=m.tau)
# Reaction kinetics
def _diffeq(m,p,t):
    return m.dc[p,t] == -m.tproc[p]*m.k[p]*m.c[p,t]
m.diffeq = Constraint(m.products, m.tau, rule=_diffeq)
```

No clash conditions

```
# No clash disjuncts
def _noclash(disjunct, I, K, IthenK):
    model = disjunct.model()
    if IthenK:
        e = model.tstart[I]+model.tproc[I] <= model.tstart[K]
        disjunct.c = Constraint(expr=e)
    else:
        e = model.tstart[K]+model.tproc[K] <= model.tstart[I]
        disjunct.c = Constraint(expr=e)
m.noclash = Disjunct(m.L, m.IthenK, rule=_noclash)

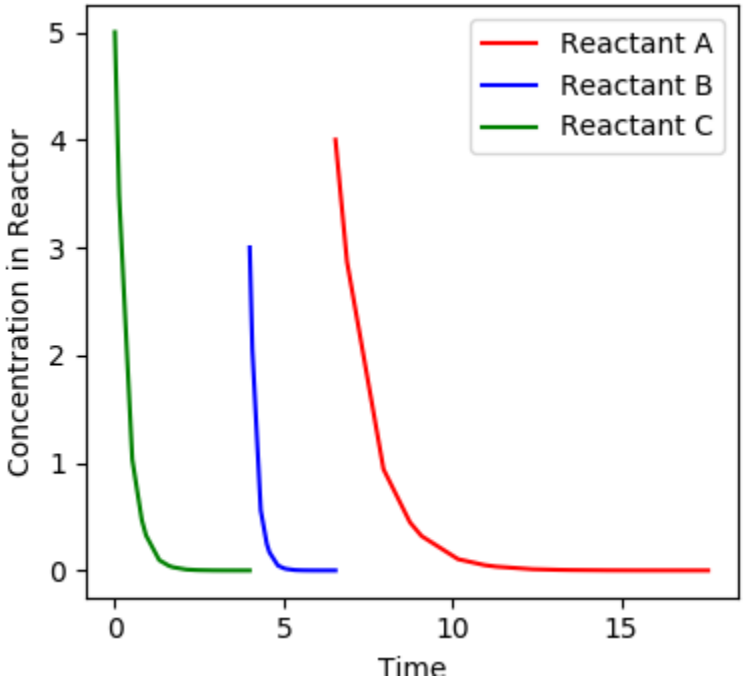
# Define the disjunctions: either reaction I occurs before K or K before I
def _disj(model, I, K):
    return [model.noclash[I, K, IthenK] for IthenK in model.IthenK]
m.disj = Disjunction(m.L, rule=_disj)
```

CCR Center for Computing Research

Scheduling incorporating process dynamics

```
# Discretize model
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(m, nfe=5, ncp=3)

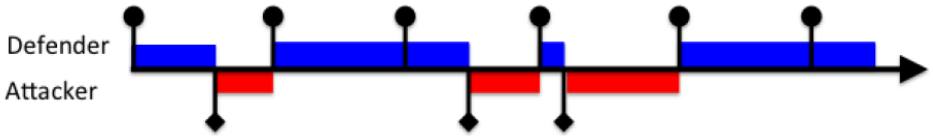
# Reformulate Disjunctions
gdp_relax = TransformationFactory('gdp.bigm')
gdp_relax.apply_to(m, default_bigM=50.0)
```



**Full model implementation can be found at https://github.com/Pyomo*

Moving target defense analysis [2]

- FlipIt game



- Find the defender's best moves given a set of attack scenarios

$$\max \quad |T|^{-1}|S|^{-1} \sum_{s\in S, t\in T} \rho_{s,t} - c_{take} \sum_{t\in T} d_t$$

$$s.t. \quad \begin{bmatrix} Y_{1,s,t} \\ a_{s,t} = 0 \\ d_t = 0 \\ \rho_{s,t} = \rho_{s,t-1} \end{bmatrix} \bigvee \begin{bmatrix} Y_{2,s,t} \\ a_{s,t} = 1 \\ d_t = 0 \\ \rho_{s,t} = 0 \end{bmatrix} \bigvee \begin{bmatrix} Y_{3,s,t} \\ \\ d_t = 1 \\ \rho_{s,t} = 1 \end{bmatrix}$$

At each time step:
Neither attacker nor defender move
Attacker moves but not defender
Defender moves

$$\forall s \in S, \{t | t \in T, t > 0\}$$

$$Y_{1,s,t} + Y_{2,s,t} + Y_{3,s,t} = 1 \qquad \forall s \in S, \{t | t \in T, t > 0\}$$

$$\rho_{s,0} = 1 \qquad \forall s \in S$$

$$d_t \in \{0, 1\} \qquad \forall t \in T$$

$$\rho_{s,t} \in \{0, 1\} \qquad \forall s \in S, t \in T$$

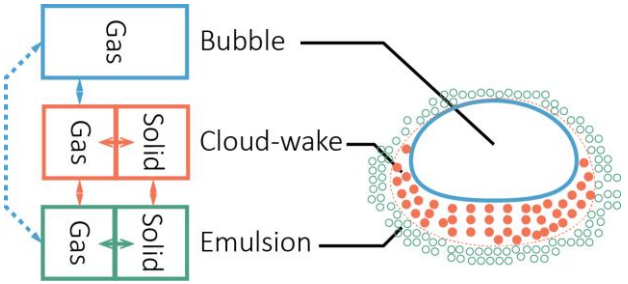
$$Y_{i,s,t} \in \{0, 1\} \qquad \forall i \in \{1, 2, 3\}, s \in S, t \in T$$

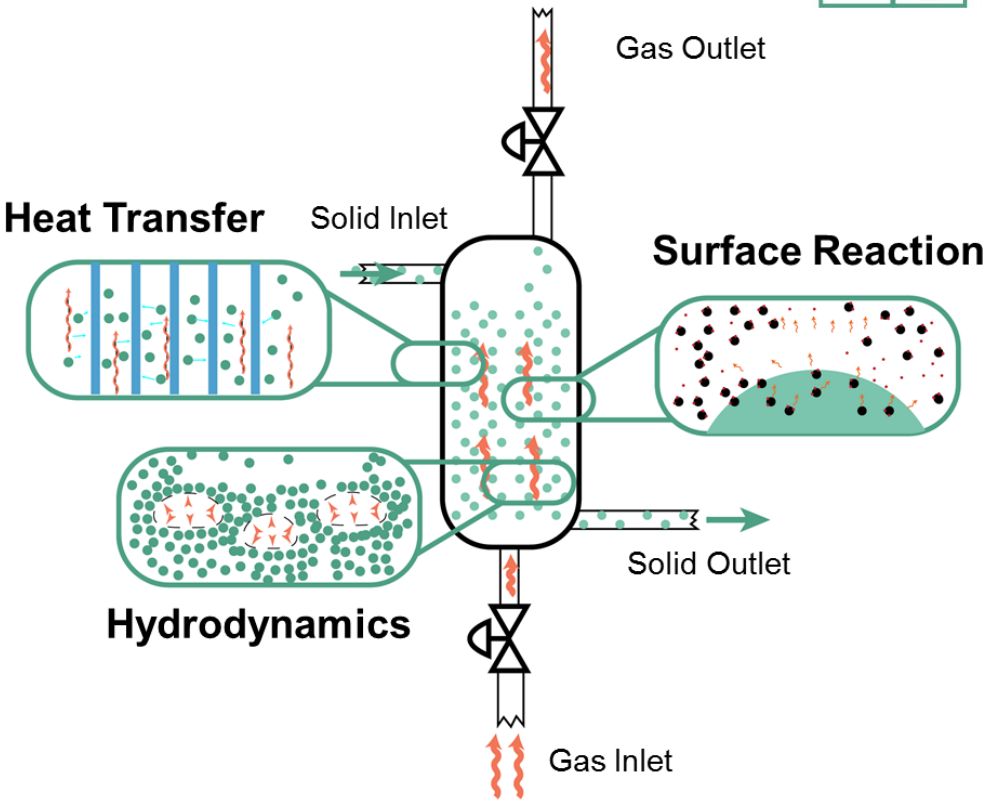
CCR Center for Computing Research

[2] Jones et. al., 2015, Evaluating moving target defense with PLADD. Technical report, SNL

Bubbling Fluidized Bed (BFB) Model

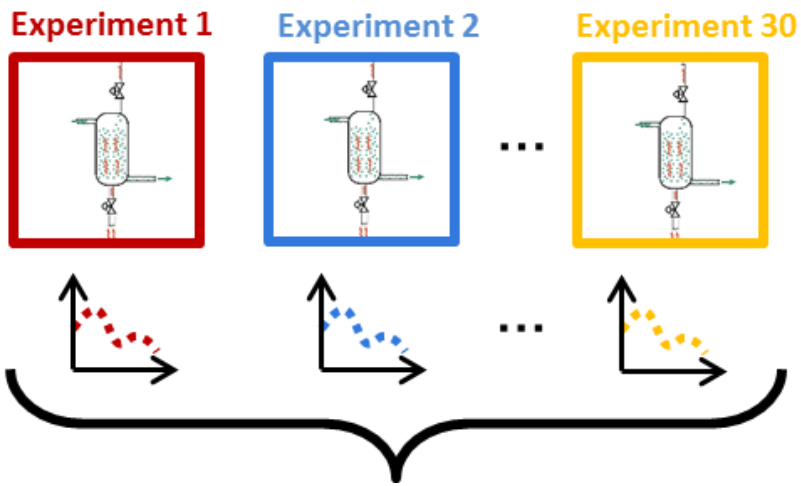
- Gas-solid, 3 region model [3]
- Steady state model with 1-D spatial variation





[3] Lee and Miller, 2013, Ind. Eng. Chem. Res.

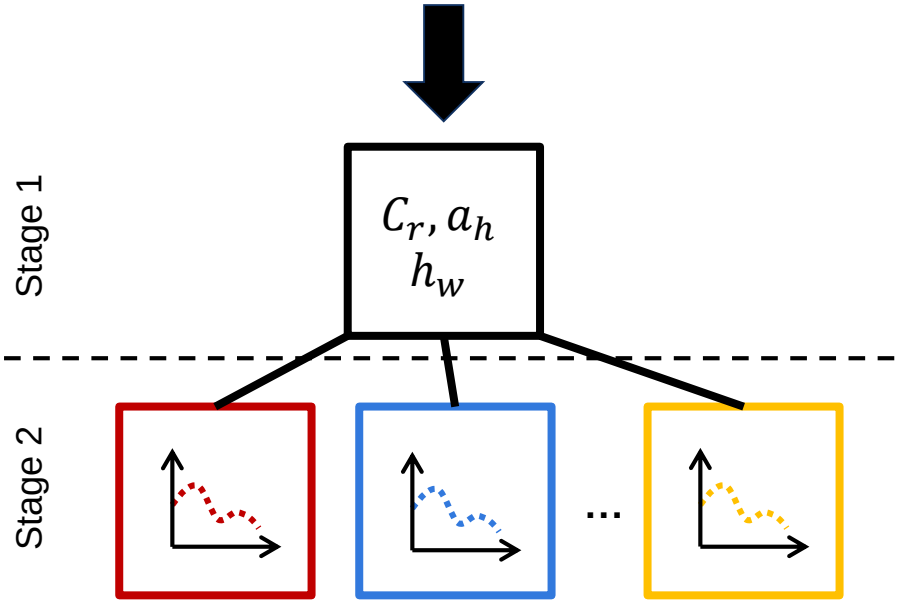
BFB Parameter Estimation



Heat Exchanger Model Parameters	
C_r	Average correction factor for tube model
a_h	Empirical factor for tube model
h_w	Heat transfer coefficient of tube walls

$$\min_{\{C_r, a_h, h_w\}} \sum_{exp.} (error_{meas})^2$$

$$s.t. \quad BFB\ model\ equations$$



Stochastic structure implementation in PySP

Sandia National Laboratories

```
def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model \
        import CreateConcreteTwoStageScenarioTreeModel

    st_model = CreateConcreteTwoStageScenarioTreeModel(scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()
    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('cr')
    st_model.StageVariables[first_stage].add('ah')
    st_model.StageVariables[first_stage].add('hw')
    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'

    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):
    experiment = int(scenario_name.replace('Scenario',''))
    explist = [1,2,3] # Different step changes in control inputs

    experiment = explist[experiment-1]
    instance = generate_model_paramest(experiment)

    return instance
```

CCR Center for Computing Research

BFB Parameter Estimation

- Create and solve extensive form

```
runef --solve --solver ipopt --output-solver-log -m bfb_param.py
```

- Solve using progressive hedging

```
runph --solver ipopt --output-solver-log -m bfb_param.py --default-rho=.25
```

- Solve using progressive hedging in parallel

```
mpirun -np 1 pyomo_ns : -np 1 dispatch_srvr : -np 30 phsolverserver : \
    -np 1 runph --solver-manager=phpyro --shutdown-pyro \
    -m bfb_param.py --solver=ipopt --default-rho=0.25
```

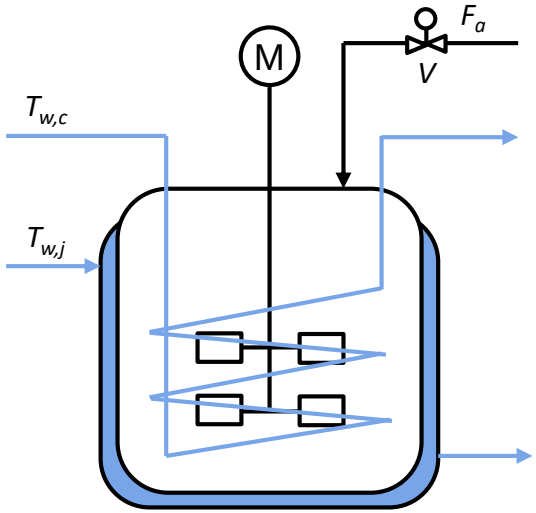
	C_r	a_h	h_w	Solve Time (s)
Actual	1.0	0.8	1500.0	-
Extensive Form	1.016	0.51	1450.35	604.45
Progressive Hedging (15 processors)	0.9824	0.7850	1501.74	610.98
Progressive Hedging (30 processors)	0.9824	0.7850	1501.74	459.10

Extensive form problem size ~400,000 variables and constraints
PH subproblem size ~13,000 variables and constraints

Stochastic optimal control

- Semibatch reactor[4]

$$A \rightarrow B \rightarrow C$$



$$\dot{C}_a = \frac{F_a}{V_r} - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a$$

$$\dot{C}_b = k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a - k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b$$

$$\dot{C}_c = k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b$$

$$\dot{V}_r = \frac{F_a M_{W_a}}{\rho_r}$$

$$(\rho_r c_{p_r})\dot{T}_r = \frac{F_a M_{W_a} c_{p_r}}{V_r}(T_f - T_r) - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a \Delta H_1 - k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \Delta H_2 + \alpha_{w,j}\frac{A_j}{V_{r,0}}(T_{w,j} - T_r) + \alpha_{w,c}\frac{A_c}{V_{r,0}}(T_{w,c} - T_r)$$

Model inputs (control variables)

F_a: Inlet flow rate of A

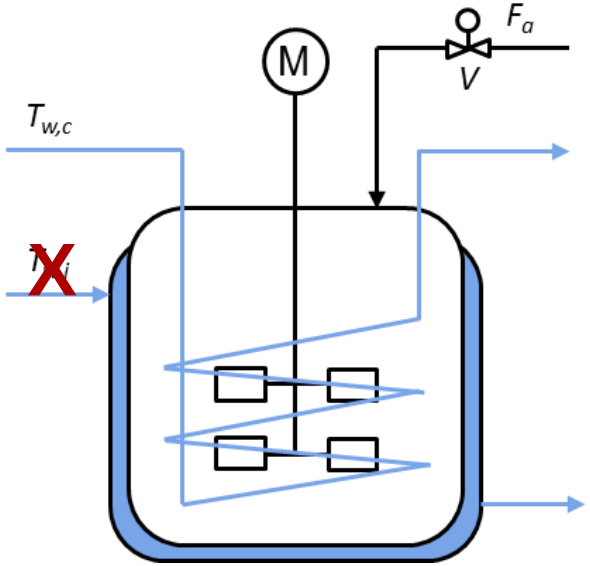
$T_{w,c}$: Water temperature in cooling coil

$T_{w,j}$: Water temperature in cooling jacket

[4] Abel and Marquardt, AIChE Journal (2000)

Optimal control

- Find the nominal control profiles such that the batch can be ‘saved’ given a partial cooling system failure at any point during the batch time[4]

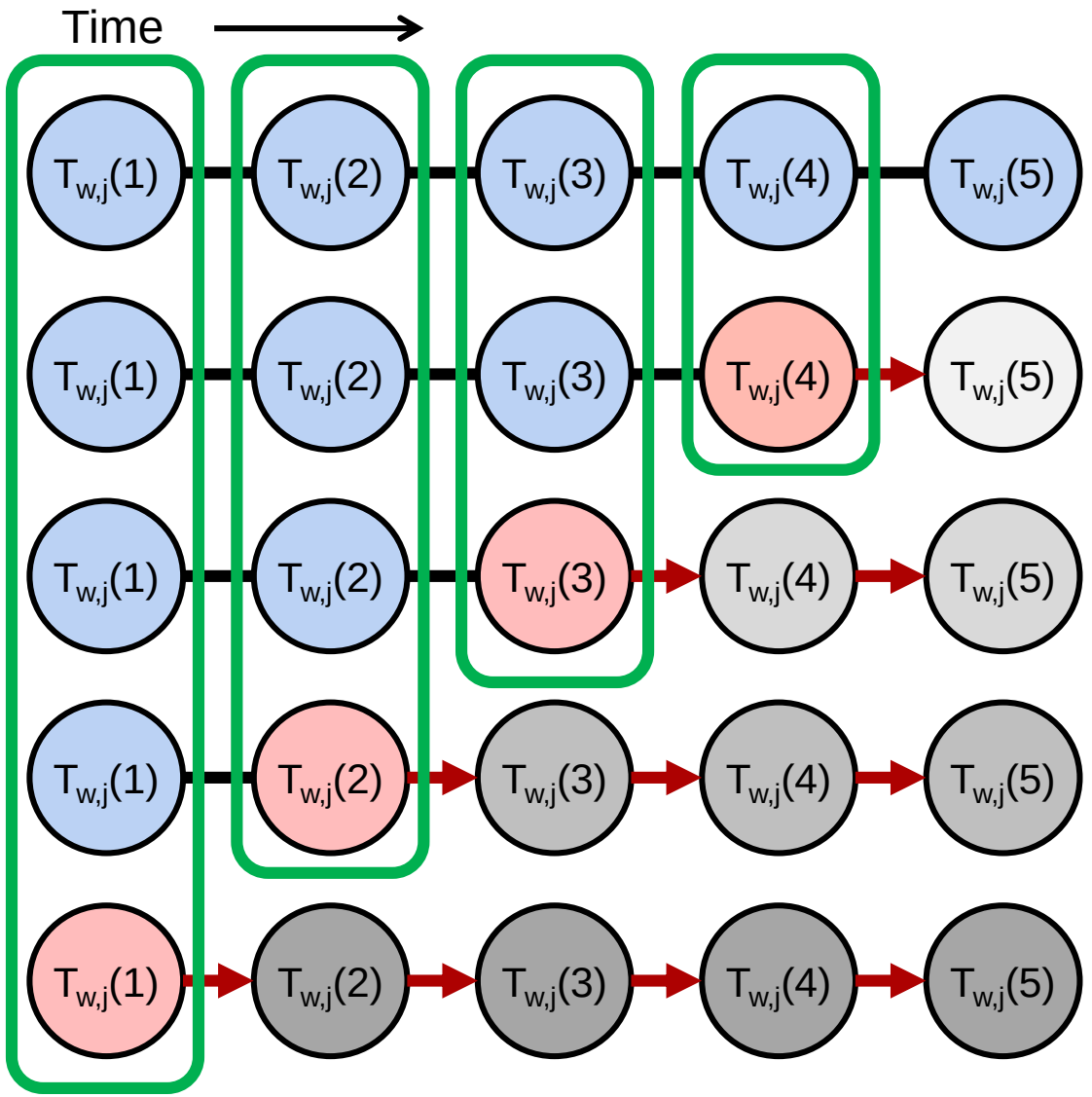


$$\left.\begin{aligned} T_{w,j}(t) &= T_{w,c}(t) && t \le t_{fail} \\ \left(\rho_w C_{p_w} V_j\right)\dot{T}_{w,j} &= \alpha_{w,js} A_j \frac{V_r}{V_{r,0}}\left(T_{w,j} - T_r\right) && t > t_{fail} \end{aligned}\right\}$$

[4] Abel and Marquardt, AIChE Journal (2000)

Optimal control scenarios

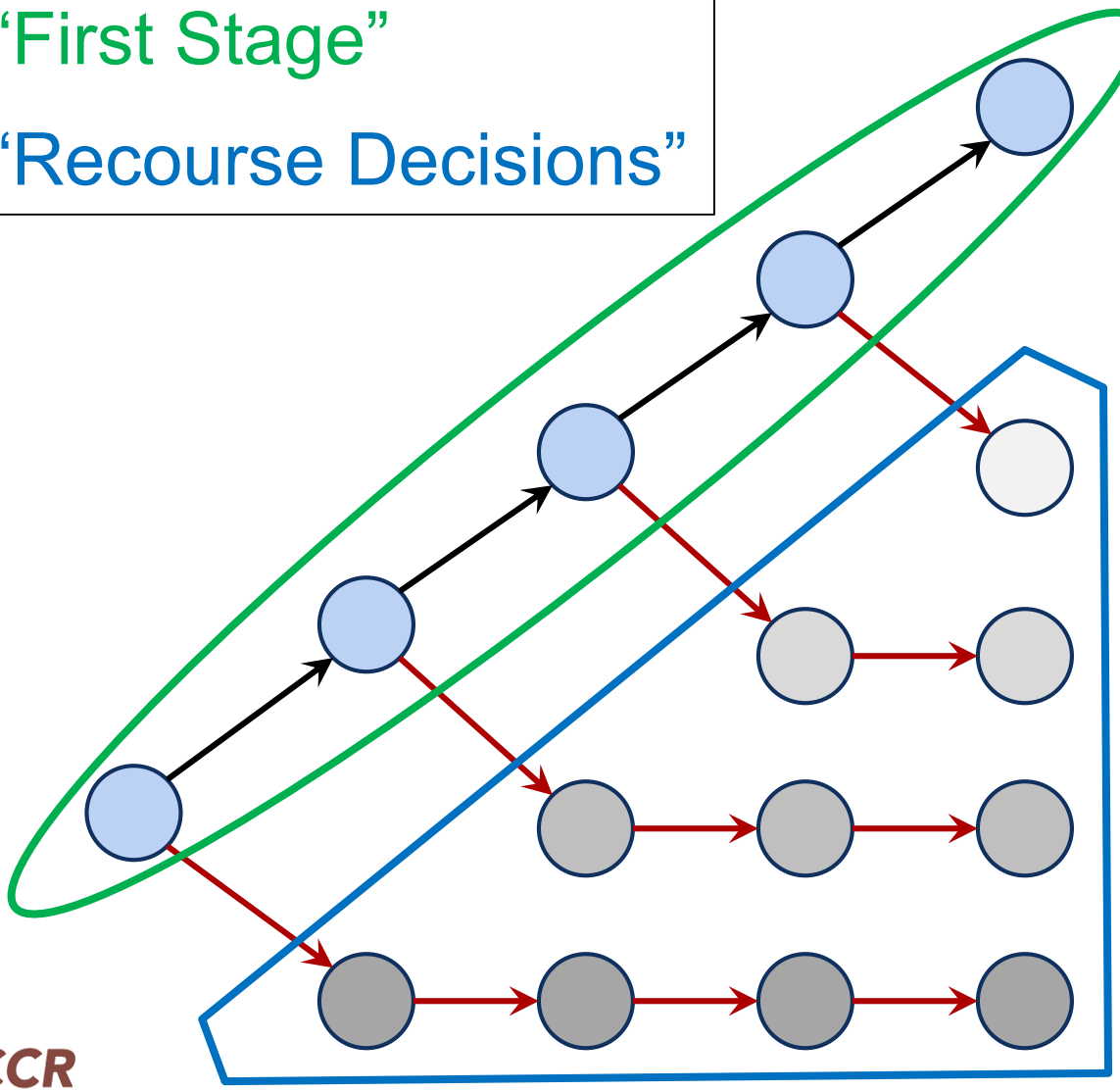
Nonanticipativity Constraints



n-Stage SP as a 2-stage problem

"First Stage"

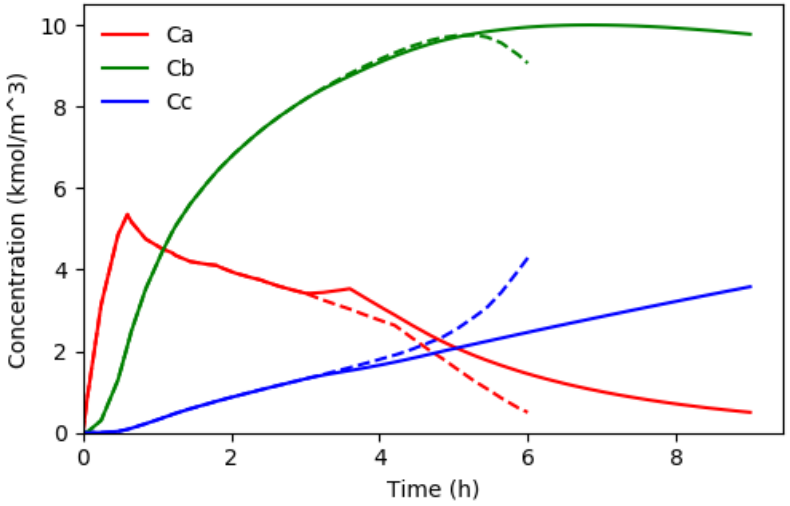
"Recourse Decisions"

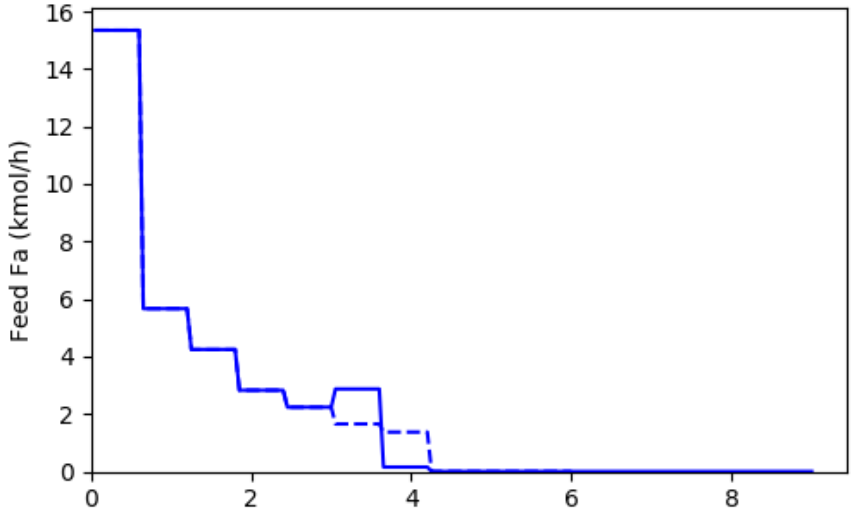


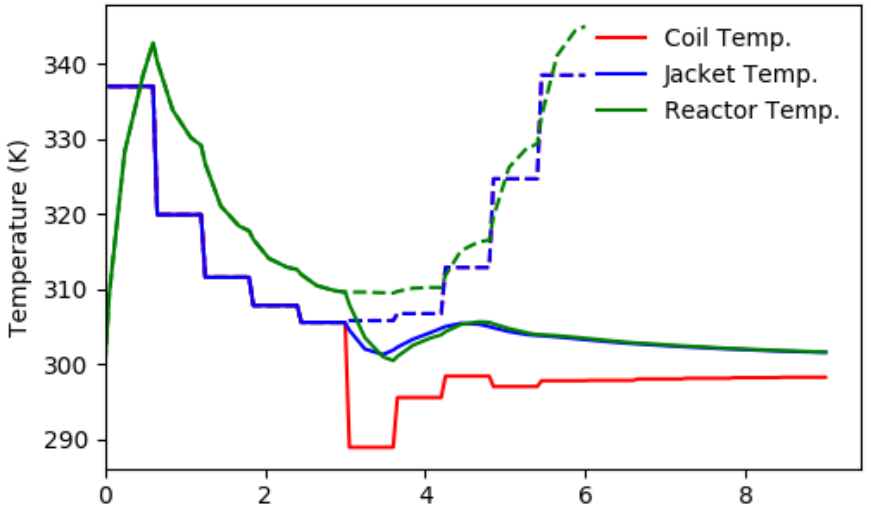
Nominal Scenario

Failure scenarios

Optimal control: t_{fail} = 3 h







Optimal control implementation

- 15 hours (including debugging)
- 300 lines of code
  - (60%)Deterministic dynamic model specification
  - (2%) Discretization
  - (18%) Stochastic problem formulation
  - (20%) Result plotting

```
from pyomo.environ import *
from pyomo.dae import *

def generate_semibatch_model(tstepfail):

    m = ConcreteModel()

    # Parameters for semi-batch reactor model
    m.k1 = Param(initialize=15.01) # 1/s
    m.k2 = Param(initialize=85.01) # 1/s
    m.E1 = Param(initialize=30000.0) # kJ/kmol
    m.E2 = Param(initialize=60000.0) # kJ/kmol
    m.R = Param(initialize=8.314) # kJ/kmol/K
    m.Mwa = Param(initialize=50.0) # kg/kmol
    m.rhor = Param(initialize=1000.0) # kg/m^3
    m.cpr = Param(initialize=3.9) # kJ/kg/K
    m.Tf = Param(initialize=300) # K
    m.deltaH1 = Param(initialize=-40000.0) # kJ/kmol
    m.deltaH2 = Param(initialize=-50000.0) # kJ/kmol
    m.alphaj = Param(initialize=0.8) # kJ/s/m^2/K
    m.alphac = Param(initialize=0.7) # kJ/s/m^2/K
    m.Aj = Param(initialize=5.0) # m^2
    m.Ac = Param(initialize=5.0) # m^2
    m.Vj = Param(initialize=0.9) # m^3
    m.Vc = Param(initialize=0.07) # m^3
    m.rhow = Param(initialize=700.0) # kg/m^3
    m.cpw = Param(initialize=3.1) # kJ/kg/K
    m.Ca0 = Param(initialize=0.0) # kmol/m^3)
    m.Cb0 = Param(initialize=0.0) # kmol/m^3)
    m.Cc0 = Param(initialize=0.0) # kmol/m^3)
    m.Tr0 = Param(initialize=300.0) # K
    m.Vr0 = Param(initialize=1.0) # m^3

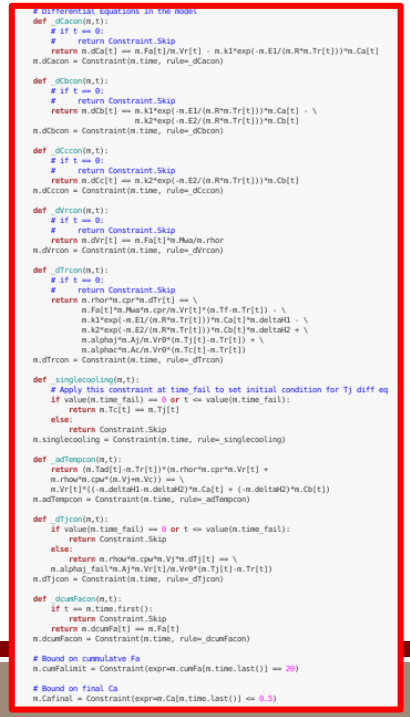
    # Cooling jacket failure parameters
    m.all_fail_times = Set(initialize=[2160*i for i in range(0,11)])
    m.alphaj_fail = Param(initialize=0.3) # kJ/s/m^2/K
    m.time_fail = Param(initialize=2160.0*tstepfail) # s

    # Time dependent variables
    initial_times = [2160.0 * i for i in range(1, 21) if 2160 * i < 21600 + value(m.time_fail)]
    m.time = ContinuousSet(bounds=(0, 21600 + m.time_fail), initialize=initial_times)  # Time in seconds

    m.Ca = Var(m.time, initialize=m.Ca0, bounds=(0,15))
    m.Cb = Var(m.time, initialize=m.Cb0, bounds=(0,15))
    m.Cc = Var(m.time, initialize=m.Cc0, bounds=(0,15))
    m.Vr = Var(m.time, initialize=m.Vr0)
    m.Tr = Var(m.time, initialize=m.Tr0)
    m.Tad = Var(m.time, initialize=300.0, bounds=(288,453))
    m.Tc = Var(m.time, initialize=310.0, bounds=(288,432)) # Cooling coil temp, control input
    m.Tj = Var(m.time, initialize=310.0, bounds=(288,None)) # Cooling jacket temp, follows coil temp until failure
    m.Fa = Var(m.time, initialize=0.0, bounds=(0,0.05)) # Inlet flow rate, control input
    m.cumFa = Var(m.time, initialize=0) # cumulative Fa, used to specify total amount of A used

    # Derivatives in the model
    m.dCa = DerivativeVar(m.Ca)
    m.dCb = DerivativeVar(m.Cb)
    m.dCc = DerivativeVar(m.Cc)
    m.dVr = DerivativeVar(m.Vr)
    m.dTr = DerivativeVar(m.Tr)
    m.dTj = DerivativeVar(m.Tj) # Only used after jacket failure
    m.dcumFa = DerivativeVar(m.cumFa)

    # Extra components so that PySP can be used to enforce the nonanticipativity constraints
    m.Tc_nonant = Var(m.all_fail_times, initialize=310.0, bounds=(288,432))
    m.Fa_nonant = Var(m.all_fail_times, initialize=0.0, bounds=(0,0.05))
```



```
    # Differential Equations in the model
    def _dCacon(m,t):
        # if t == 0:
        #     return Constraint.Skip
        return m.dCa[t] == m.Fa[t]/m.Vr[t] - m.k1*exp(-m.E1/(m.R*m.Tr[t]))*m.Ca[t]
    m.dCacon = Constraint(m.time, rule=_dCacon)

    def _dCbcon(m,t):
        # if t == 0:
        #     return Constraint.Skip
        return m.dCb[t] == m.k1*exp(-m.E1/(m.R*m.Tr[t]))*m.Ca[t] - \
                           m.k2*exp(-m.E2/(m.R*m.Tr[t]))*m.Cb[t]
    m.dCbcon = Constraint(m.time, rule=_dCbcon)

    def _dCccon(m,t):
        # if t == 0:
        #     return Constraint.Skip
        return m.dCc[t] == m.k2*exp(-m.E2/(m.R*m.Tr[t]))*m.Cb[t]
    m.dCccon = Constraint(m.time, rule=_dCccon)

    def _dVrcon(m,t):
        # if t == 0:
        #     return Constraint.Skip
        return m.dVr[t] == m.Fa[t]*m.Mwa/m.rhor
    m.dVrcon = Constraint(m.time, rule=_dVrcon)

    def _dTrcon(m,t):
        if t == 0:
            return Constraint.Skip
        return m.rhor*m.cpr*m.dTr[t] == \
               (m.Fa[t]*m.Mwa*m.cpr/m.Vr[t]*(m.Tf-m.Tr[t]) + \
                m.k1*exp(-m.E1/(m.R*m.Tr[t]))*m.Ca[t]*m.deltaH1 + \
                m.k2*exp(-m.E2/(m.R*m.Tr[t]))*m.Cb[t]*m.deltaH2 + \
                m.alphaj*m.Aj/m.Vr0*(m.Tj[t]-m.Tr[t]) + \
                m.alphac*m.Ac/m.Vr0*(m.Tc[t]-m.Tr[t]))
    m.dTrcon = Constraint(m.time, rule=_dTrcon)

    def _singlecooling(m,t):
        # Apply this constraint at time_fail to set initial condition for Tj diff eq
        if value(m.time_fail) == 0 or t <= value(m.time_fail):
            return m.Tc[t] == m.Tj[t]
        else:
            return Constraint.Skip
    m.singlecooling = Constraint(m.time, rule=_singlecooling)

    def _adTempcon(m,t):
        return (m.Tad[t]-m.Tr[t])*(m.rhor*m.cpr*m.Vr[t] +
        m.rhow*m.cpw*(m.Vj+m.Vc)) == \
            m.Vr[t]*((-m.deltaH1-m.deltaH2)*m.Ca[t] + (-m.deltaH2)*m.Cb[t])
    m.adTempcon = Constraint(m.time, rule=_adTempcon)

    def _dTjcon(m,t):
        if value(m.time_fail) == 0 or t <= value(m.time_fail):
            return Constraint.Skip
        else:
            return m.rhow*m.cpw*m.Vj*m.dTj[t] == \
        m.alphaj_fail*m.Aj*m.Vr[t]/m.Vr0*(m.Tj[t]-m.Tr[t])
    m.dTjcon = Constraint(m.time, rule=_dTjcon)

    def _dcumFacon(m,t):
        if t == m.time.first():
            return Constraint.Skip
        return m.dcumFa[t] == m.Fa[t]
    m.dcumFacon = Constraint(m.time, rule=_dcumFacon)

    # Bound on cumulative Fa
    m.cumFalimit = Constraint(expr=m.cumFa[m.time.last()] <= 20)

    # Bound on final Ca
    m.Cafinal = Constraint(expr=m.Ca[m.time.last()] <= 0.5)
```

```
    # Initial Conditions
    def _initcon(m):
        yield m.Ca[m.time.first()] == m.Ca0
        yield m.Cb[m.time.first()] == m.Cb0
        yield m.Cc[m.time.first()] == m.Cc0
        yield m.Vr[m.time.first()] == m.Vr0
        yield m.Tr[m.time.first()] == m.Tr0
        yield m.cumFa[m.time.first()] == 0
    m.initcon = ConstraintList(rule=_initcon)

    # Helper constraints for enforcing nonanticipativity constraints
    def _Tc_nonantcon(m,t):
        if value(m.time_fail) == 0 or t<= value(m.time_fail):
            return m.Tc_nonant[t] == m.Tc[t]
        return Constraint.Skip
    m.Tc_nonantcon = Constraint(m.all_fail_times, rule=_Tc_nonantcon)

    def _Fa_nonantcon(m,t):
        if value(m.time_fail) == 0 or t <= value(m.time_fail):
            return m.Fa_nonant[t] == m.Fa[t]
        return Constraint.Skip
    m.Fa_nonantcon = Constraint(m.all_fail_times, rule=_Fa_nonantcon)

    # Stage-specific cost computations
    def ComputeFirstStageCost_rule(model):
        return 0
    m.FirstStageCost = Expression(rule=ComputeFirstStageCost_rule)

    def ComputeSecondStageCost_rule(model):
        return -m.Cb[m.time.last()]
    m.SecondStageCost = Expression(rule=ComputeSecondStageCost_rule)

    def total_cost_rule(model):
        return model.FirstStageCost + model.SecondStageCost
    m.Total_Cost_Objective = Objective(rule=total_cost_rule, sense=minimize)
```

```
    # Discretize model. Number of finite elements depends on the cooling failure time
    numfe = m.time.last()/2160.0
    disc = TransformationFactory('dae.collocation')
    disc.apply_to(m, nfe=numfe, ncp=4)
    disc.reduce_collocation_points(m, var=m.Fa, ncp=1, contset=m.time)
    disc.reduce_collocation_points(m, var=m.Tc, ncp=1, contset=m.time)
    return m
```

```
# Number of scenarios
scenarios = 10

def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model \
        import CreateConcreteTwoStageScenarioTreeModel

    st_model = CreateConcreteTwoStageScenarioTreeModel(scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()

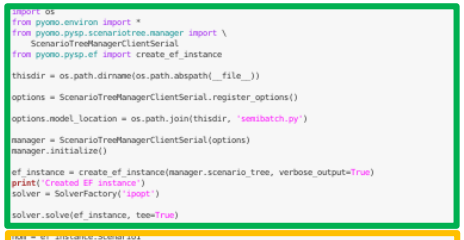
    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('Fa_nonant[*]')
    st_model.StageVariables[first_stage].add('Tc_nonant[*]')

    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'

    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):
    failtimestep = int(scenario_name.replace('Scenario',''))
    instance = generate_semibatch_model(failtimestep-1)

    return instance
```



```
import os
from pyomo.environ import *
from pyomo.pysp.scenariotree.manager import \
    ScenarioTreeManagerClientSerial
from pyomo.pysp.ef import create_ef_instance

thisdir = os.path.dirname(os.path.abspath(__file__))

options = ScenarioTreeManagerClientSerial.register_options()

options.model_location = os.path.join(thisdir, 'semibatch.py')

manager = ScenarioTreeManagerClientSerial(options)
manager.initialize()

ef_instance = create_ef_instance(manager.scenario_tree, verbose_output=True)
print('Created EF instance')
solver = SolverFactory('ipopt')

solver.solve(ef_instance, tee=True)
```

```
n = ef_instance.Scenario6

nomtime = [i/3600.0 for i in nom.time]
time = [i/3600.0 for i in m.time]

import matplotlib.pyplot as plt

grey1 = '0.7'
grey2 = '0.4'
grey3 = '0'

plt.subplot(313)
plt.plot(nomtime,[value(nom.Ca[t]) for t in nom.time],'--',color=grey1)
plt.plot(nomtime,[value(nom.Cb[t]) for t in nom.time],'--',color=grey2)
plt.plot(nomtime,[value(nom.Cc[t]) for t in nom.time],'--',color=grey3)
plt.plot(time,[value(m.Ca[t]) for t in m.time],label='Ca',color=grey1)
plt.plot(time,[value(m.Cb[t]) for t in m.time],label='Cb',color=grey2)
plt.plot(time,[value(m.Cc[t]) for t in m.time],label='Cc',color=grey3)
plt.legend(loc='best')
plt.xlabel('Time (h)')
plt.ylabel('Concentration (kmol/m^3)')
plt.xlim(xmin=0)
plt.ylim(ymin=0)

plt.subplot(311)
plt.plot(time[1:],[value(m.Fa[t])*3600 for t in m.time][1:],color=grey2)
plt.plot(nomtime[1:],[value(nom.Fa[t])*3600 for t in nom.time][1:],'--',color=grey2)
plt.legend(loc='best')
plt.ylabel('Feed Fa (kmol/h)')
plt.xlim(xmin=0)
plt.ylim(ymin=0)

plt.subplot(312)
plt.plot(nomtime[1:],[value(nom.Tc[t]) for t in nom.time][1:],'--',color=grey1)
plt.plot(nomtime[1:],[value(nom.Tj[t]) for t in nom.time][1:],'--',color=grey2)
plt.plot(nomtime,[value(nom.Tr[t]) for t in nom.time],'--',color=grey3)
plt.plot(time[1:],[value(m.Tc[t]) for t in m.time][1:],label='Coil Temp.',color=grey1)
plt.plot(time[1:],[value(m.Tj[t]) for t in m.time][1:],label='Jacket Temp.',color=grey2)
plt.plot(time,[value(m.Tr[t]) for t in m.time],label='Reactor Temp.',color=grey3)
plt.legend(loc='best')
plt.ylabel('Temperature (K)')
plt.xlim(xmin=0)
plt.show()
```

Summary

- Explicitly capturing high-level structure leads to significantly easier, faster, and more flexible implementations
- Pyomo provides high-level modeling constructs that can be easily combined to solve complex, structured optimization problems. (www.pyomo.org)

<u>New challenges and open questions</u>

- general implementations of meta-solvers to exploit layered/nested structure
- scalability of these techniques

Questions?

- Acknowledgements
  - This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Office of Fossil Energy, Cross-Cutting Research, U.S. Department of Energy

IDAES
Institute for the Design of Advanced Energy Systems

NATIONAL ENERGY TECHNOLOGY LABORATORY

BERKELEY LAB

Sandia National Laboratories

Carnegie Mellon
West Virginia University

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND2017-11920 C