SAND2011-4958C

PySP: Modeling and Solving Stochastic Linear and Mixed-Integer Programs in Python

Jean-Paul Watson
jwatson@sandia.gov
Discrete Math and Complex Systems Department
Sandia National Laboratories USA

David L. Woodruff
dlwoodruff@ucdavis.edu
Graduate School of Management
University of California, Davis USA

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

Motivation

- Numerous stochastic programming extensions to Algebraic Modeling Languages (AMLs) have over been proposed over the last decade
  - Useful and necessary, especially for creating extensive forms
- Modeling is not our objective here, but rather a necessary pre-requisite
- Our goals

1. Break down the barrier between modeling languages and solvers
2. Provide model-agnostic stochastic (integer) programming algorithms
3. Facilitate rapid prototyping, development, and extension of algorithms

Our Problem-Solving Objective

- Our "prime directive" is to solve large-scale stochastic programs
  - Multiple stages (>=2)
  - Integer decision variables in *any* stage
  - *Lots* of scenarios (thousands to millions)
- Optimality is nice, but not realistic given these constraints and the scale of problem we are interested in tackling
  - Our goal is to design practical, scalable, and high-performance heuristics for the class of general stochastic program

Why Python? Why Open-Source?

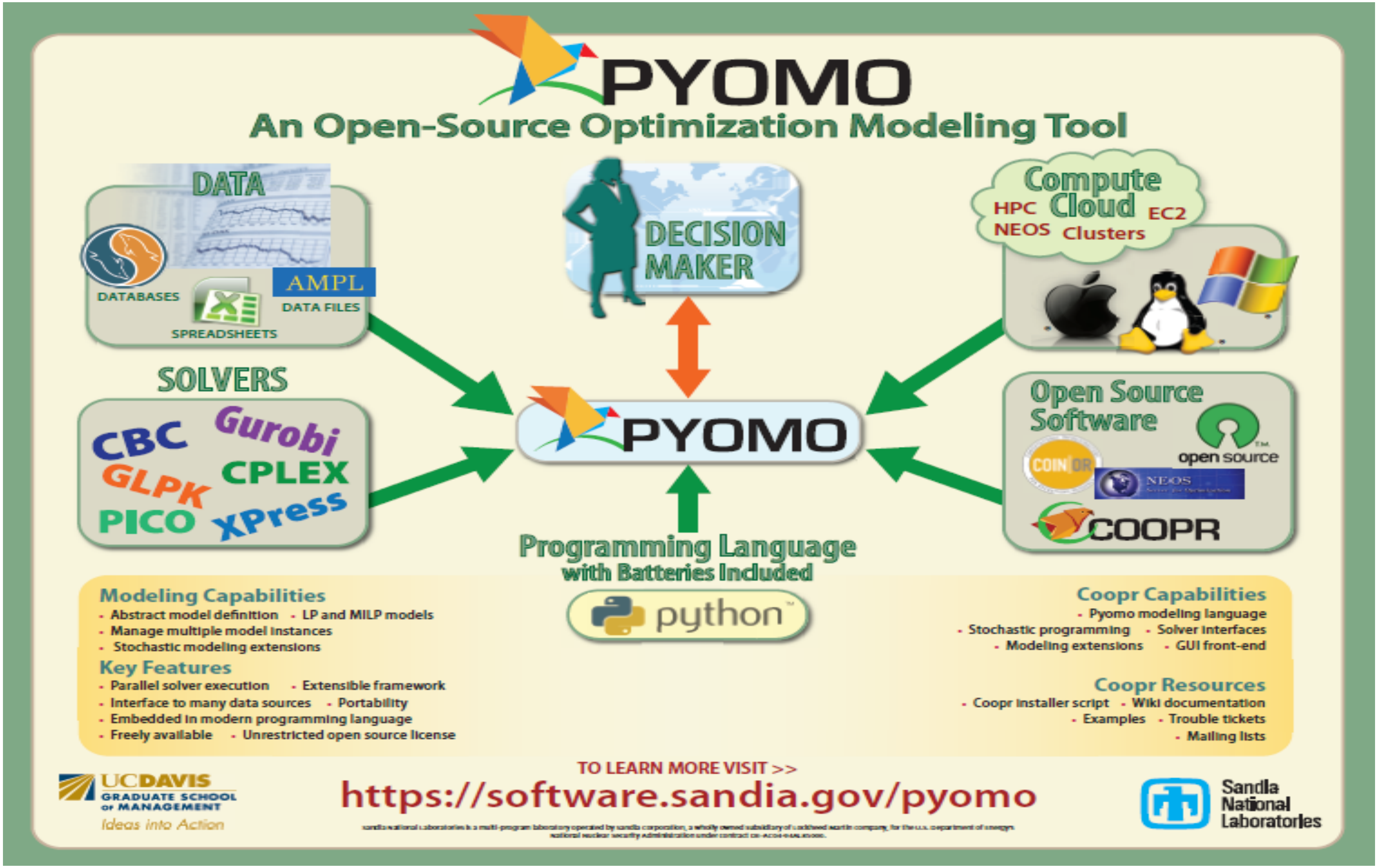
- Python facilitates rapid prototyping and doesn't require a CS degree
  - Important for modelers, OR grads, and general productivity
- Python ships with a huge number of very useful libraries, including
  - Serialization, distributed computation, db/Excel interfaces, …
  - SciPy and NumPy
- Python introspection facilitates the development of generic algorithms
  - If you don't know what this means, I can't tell you in 20 minutes
  - But trust me – it's important!
- Why (noninfectious) open-source?
  - We want the community to contribute, and we have customers that are license-phobic and don't want to pay for third-party tools

Why ***Not*** Python?

- Reasons do exist, but not really good ones
  - If it's good enough for quantum chemistry, it's good enough for operations research

- A great discussion topic for a break or the conference banquet

PYOMO: PYthon Optimization Modeling Objects



Step #1: Formulate the Deterministic Model (1)

```
from coopr.pyomo import *
model=Model()

# Parameters
model.CROPS=Set()
model.TOTAL_ACREAGE=Param(within=PositiveReals)
model.PriceQuota=Param(model.CROPS, within=PositiveReals)
model.SubQuotaSellingPrice=Param(model.CROPS, within=PositiveReals)
model.SuperQuotaSellingPrice=Param(model.CROPS)
model.CattleFeedRequirement=Param(model.CROPS, \
                                  within=NonNegativeReals)
model.PurchasePrice=Param(model.CROPS, within=PositiveReals)
model.PlantingCostPerAcre=Param(model.CROPS, within=PositiveReals)
model.Yield=Param(model.CROPS, within=NonNegativeReals)

# Variables
model.DevotedAcreage=Var(model.CROPS, \
                         bounds=(0.0, model.TOTAL_ACREAGE))

model.QuantitySubQuotaSold=Var(model.CROPS, bounds=(0.0, None))
model.QuantitySuperQuotaSold=Var(model.CROPS, bounds=(0.0, None))

model.QuantityPurchased=Var(model.CROPS, bounds=(0.0, None))

model.FirstStageCost=Var()
model.SecondStageCost=Var()
```

ReferenceModel.py

Birge and Louveaux's (1997) Farmer Example

Sandia National Laboratories

Step #1: Formulate the Deterministic Model (2)

```
# Constraints
def total_acreage_rule(model):
    return summation(model.DevotedAcreage) <= model.TOTAL_ACREAGE
model.ConstrainTotalAcreage=Constraint(rule=total_acreage_rule)

def cattle_feed_rule(i, model):
    return model.CattleFeedRequirement[i] <= \
           (model.Yield[i] * model.DevotedAcreage[i]) + \
            model.QuantityPurchased[i] - \
            model.QuantitySubQuotaSold[i] - \
            model.QuantitySuperQuotaSold[i]
model.EnforceCattleFeedRequirement=Constraint(model.CROPS, \
                                              rule=cattle_feed_rule)

def limit_amount_sold_rule(i, model):
    return model.QuantitySubQuotaSold[i] + \
           model.QuantitySuperQuotaSold[i] <= \
           (model.Yield[i] * model.DevotedAcreage[i])
model.LimitAmountSold=Constraint(model.CROPS, \
                                 rule=limit_amount_sold_rule)

def enforce_quotas_rule(i, model):
    return (0.0, model.QuantitySubQuotaSold[i], model.PriceQuota[i])
model.EnforceQuotas=Constraint(model.CROPS, \
                               rule=enforce_quotas_rule)
```


ReferenceModel.py

Step #1: Formulate the Deterministic Model (3)

```
# Stage-specific cost computations
def first_stage_cost_rule(model):
    return model.FirstStageCost == \
        summation(model.PlantingCostPerAcre, model.DevotedAcreage)
model.ComputeFirstStageCost=Constraint(rule=first_stage_cost_rule)

def second_stage_cost_rule(model):
    expr=summation(model.PurchasePrice, model.QuantityPurchased)
    expr -= summation(model.SubQuotaSellingPrice, \
                      model.QuantitySubQuotaSold)
    expr -= summation(model.SuperQuotaSellingPrice, \
                      model.QuantitySuperQuotaSold)
    return (model.SecondStageCost - expr) == 0.0
model.ComputeSecondStageCost=Constraint(rule=second_stage_cost_rule)

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

ReferenceModel.py

Step #2: Specify the Deterministic Model Data

```
set CROPS := WHEAT CORN SUGAR_BEETS ;

param TOTAL_ACREAGE := 500 ;

param PriceQuota := WHEAT 100000 CORN 100000 SUGAR_BEETS 6000 ;

param SubQuotaSellingPrice := WHEAT 170 CORN 150 SUGAR_BEETS 36 ;

param SuperQuotaSellingPrice := WHEAT 0 CORN 0 SUGAR_BEETS 10 ;

param CattleFeedRequirement := WHEAT 200 CORN 240 SUGAR_BEETS 0 ;

param PurchasePrice := WHEAT 238 CORN 210 SUGAR_BEETS 100000 ;

param PlantingCostPerAcre := WHEAT 150 CORN 230 SUGAR_BEETS 260 ;

param Yield := WHEAT 3.0 CORN 3.6 SUGAR_BEETS 24 ;
```

ReferenceModel.dat

- Can initialize an instance from
  1. An AMPL .dat file
  2. Excel
  3. Raw Python

Step #3: Specify the Scenario Tree

```
set Stages := FirstStage SecondStage ;

set Nodes := RootNode
             BelowAverageNode
             AverageNode
             AboveAverageNode ;

param NodeStage := RootNode          FirstStage
                   BelowAverageNode  SecondStage
                   AverageNode       SecondStage
                   AboveAverageNode  SecondStage ;

set Children[RootNode] := BelowAverageNode
                          AverageNode
                          AboveAverageNode ;

param ConditionalProbability := RootNode          1.0
                                BelowAverageNode  0.33333333
                                AverageNode       0.33333334
                                AboveAverageNode  0.33333333 ;

set Scenarios := BelowAverageScenario
                 AverageScenario
                 AboveAverageScenario ;

param ScenarioLeafNode := BelowAverageScenario BelowAverageNode
                          AverageScenario      AverageNode
                          AboveAverageScenario AboveAverageNode ;

set StageVariables[FirstStage] :=  DevotedAcreage[*] ;
set StageVariables[SecondStage] := QuantitySubQuotaSold[*]
                                   QuantitySuperQuotaSold[*]
                                   QuantityPurchased[*] ;

param StageCostVariable := FirstStage  FirstStageCost
                           SecondStage SecondStageCost ;
```


ScenarioStructure.dat

Step #4: Specify the Scenario Instance Data

- 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

Writing and Solving the Extensive Form (1)

- Now that you have a stochastic programming model in PySP…

- Step #1: Write the extensive form and pray that CPLEX can solve it
  - Fantastic if it works
  - But often it doesn't

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

- The basic command-line:

```
runef --model-directory=models \\
      --instance-directory=scenariodata \\
      --solve
```


Sandia National Laboratories

Writing and Solving the Extensive Form (2)

- After solution, you get (in addition to other information):

```
Tree Nodes:

        Name=RootNode
        Stage=FirstStage
        Variables:
                DevotedAcreage[CORN]=80.0
                DevotedAcreage[SUGAR_BEETS]=250.0
                DevotedAcreage[WHEAT]=170.0

        Name=AboveAverageNode
        Stage=SecondStage
        Variables:
                QuantitySubQuotaSold[CORN]=48.0
                QuantitySubQuotaSold[SUGAR_BEETS]=6000.0
                QuantitySubQuotaSold[WHEAT]=310.0

        Name=AverageNode
        Stage=SecondStage
        Variables:
                QuantitySubQuotaSold[SUGAR_BEETS]=5000.0
                QuantitySubQuotaSold[WHEAT]=225.0

        Name=BelowAverageNode
        Stage=SecondStage
        Variables:
                QuantitySubQuotaSold[SUGAR_BEETS]=4000.0
```

What Happens if the Extensive Form is Too Difficult?

- *We use decomposition!*

Progressive Hedging: A Review and/or Introduction

1. $k := 0$
2. For all $s \in \mathcal{S}$, $x_s^{(k)} := \text{argmin}_x (c \cdot x + f_s \cdot y_s) : (x, y_s) \in \mathcal{Q}_s$
3. $\bar{x}^k := (\sum_{s \in \mathcal{S}} p_s \, d_s x_s^{(k)}) / \sum_{s \in \mathcal{S}} p_s \, d_s$
4. For all $s \in \mathcal{S}$, $w_s^{(k)} := \rho(x_s^{(k)} - \bar{x}^{(k)})$
5. $k := k + 1$
6. For all $s \in \mathcal{S}$, $x_s^{(k)} := \text{argmin}_x (c \cdot x + w_s^{(k-1)} x + \rho/2 \left\| x - \bar{x}^{(k-1)} \right\|^2 + f_s \cdot y_s)$ $: (x, y_s) \in \mathcal{Q}_s$
7. $\bar{x}^{(k)} := (\sum_{s \in \mathcal{S}} p_s \, d_s x_s^{(k)}) / \sum_{s \in \mathcal{S}} p_s \, d_s$
8. For all $s \in \mathcal{S}$, $w_s^{(k)} := w_s^{(k-1)} + \rho \left(x_s^{(k)} - \bar{x}^{(k)} \right)$
9. $g^{(k)} := \frac{(1-\alpha)|\mathcal{S}|}{\sum_{s \in \mathcal{S}} p_s \, d_s} \sum_{s \in \mathcal{S}} \left\| x^{(k)} - \bar{x}^{(k)} \right\|$
10. If $g^{(k)} < \epsilon$, then go to step 5. Otherwise, terminate.

Rockafellar and Wets (1991)

PySP: Generic Progressive Hedging (1)

- If you don't care about the value of the penalty parameter ρ, you are willing to take chances, and/or you have time to kill:

```
runph --model-directory=models --instance-directory=scenariodata
```

- If you think a global value of the penalty parameter will work:
  - Add the argument "--default-rho=your-favorite-value"
- More likely, you want to implement variable-specific strategies:
  - Add the argument "--rho-cfgfile=myrhostrategy.cfg"

myrhostrategy.cfg:

```
model_instance = self._model_instance # syntatic sugar

for i in model_instance.ProductSizes:
   self.setRhoAllScenarios(model_instance.ProduceSizeFirstStage[i], \\
                           model_instance.SetupCosts[i] * 0.001)
   self.setRhoAllScenarios(model_instance.NumProducedFirstStage[i], \\
                           model_instance.UnitProductionCosts[i] * 0.001)
   for j in model_instance.ProductSizes:
      if j <= i:
         self.setRhoAllScenarios(model_instance.NumUnitsCutFirstStage[i,j], \\
                                 model_instance.UnitReductionCost * 0.001)
```

PySP: Generic Progressive Hedging (2)

- The quadratic penalty term in PH is computationally problematic
  - 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
- PySP provides for various termination mechanisms
  - Scenario solution homogeneity (various metrics)
  - Number of converged variables
  - Hybrids

PySP: Generic Progressive Hedging (3)

- 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 (To appear)*
  - Implemented via a generic plug-in callback framework
- Capabilities include:
  - Variable fixing
  - Cycle detection
  - Cycle breaking
  - Slamming

Under the Hood: Facilitating Capabilities in Python

- Cool Python Feature #1
  - Ability to add attributes to objects on-the-fly
  - E.g., my_var.my_personal_attribute = 1234
  - AMPL-ish in the ability to define on-the-fly suffixes
  - Important: The objects don't need to "know" about these attributes
    - Facilitates augmentation of Pyomo models with algorithmic data

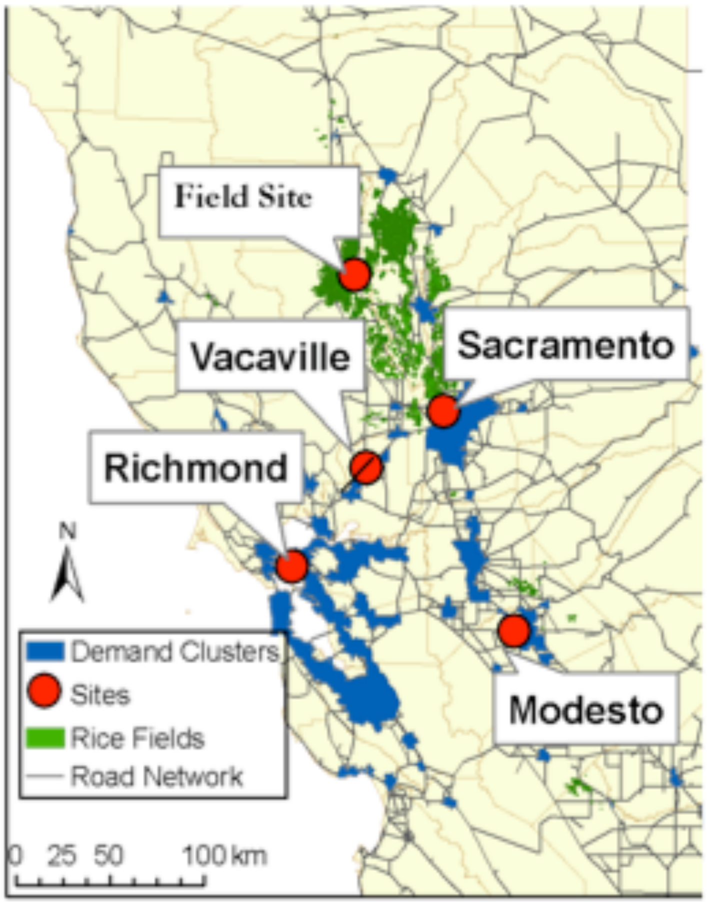
- Cool Python Feature #2
  - By-name access to object attributes
  - E.g., a_var=getattr(my_model, "VariableOfInterest")
  - E.g., setattr(my_model,"VariableOfInterest",modified_variable)
  - Facilitates linkage of user-specified string data to Pyomo model objects

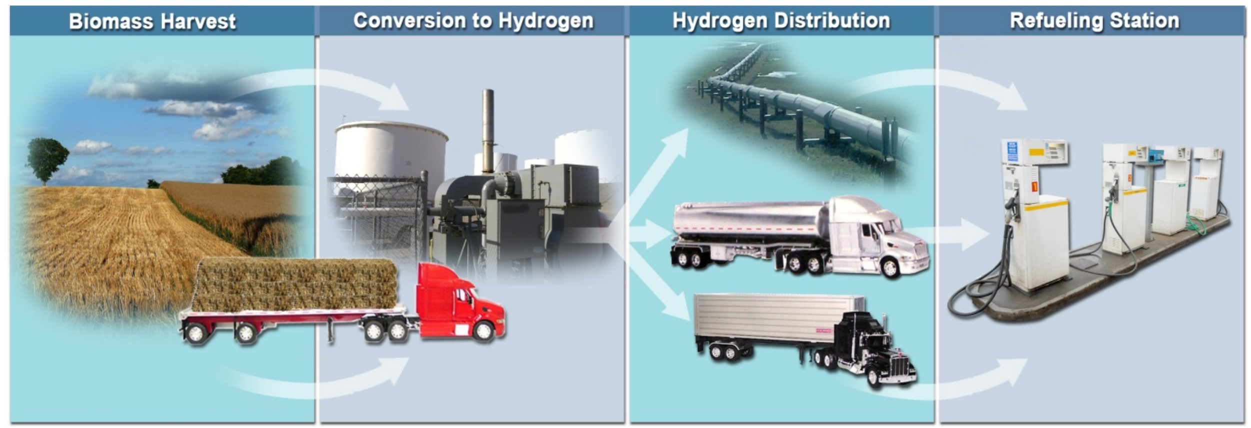
- Also very cool: You can serialize any object in Python, including PH

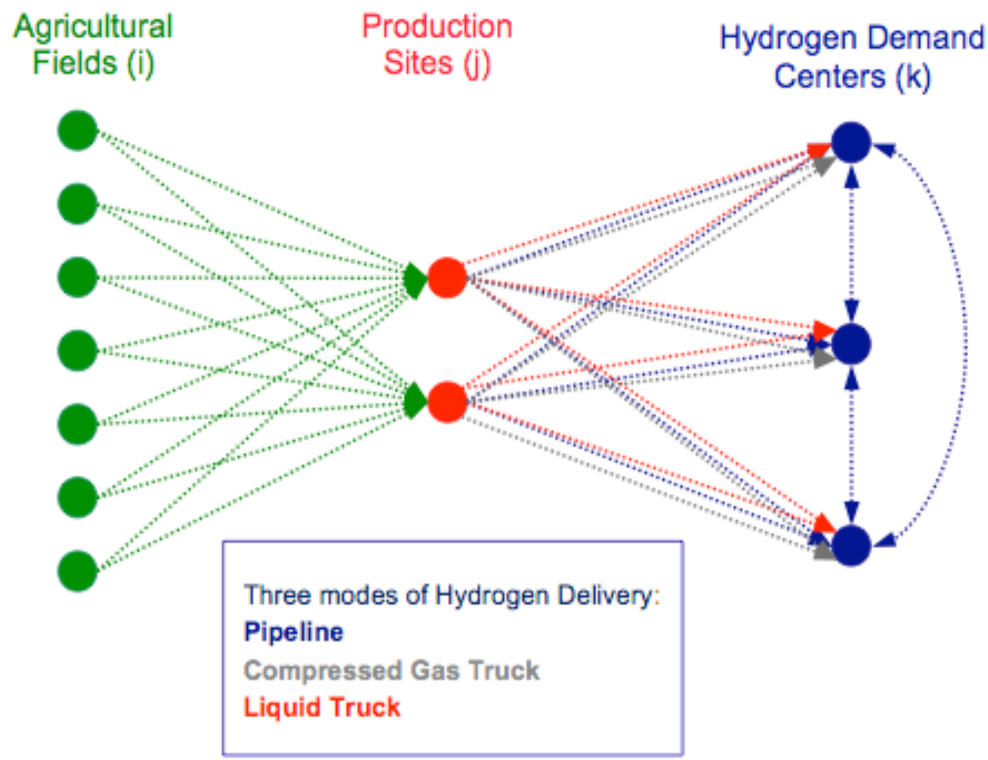
PySP: Benchmark Problems and R&D Models

- Currently available (with corresponding and validated Pyomo models)
  - Birge and Louveaux's farmer problem (continuous 2-stage)
  - SIZES (2-stage with integer variables)
  - Stochastic network design (2-stage with integer variables)
  - Forestry harvesting problem (4-stage with integer variables)
- Available upon request
  - Wind farm network design
  - Stochastic unit commitment
  - Biofuel network design
  - Grid generation capacity expansion
  - Numerous others in the works…

The Impact of PySP: Biofuel Infrastructure and Logistics Planning







<u>Example of PH Impact:</u>

- Extensive form solve time: >20K seconds
- PH solve time: 2K seconds

Slide courtesy of Professor YueYue Fan (UC Davis)

PySP, Distributed Computation, and Progressive Hedging

- Decomposition algorithms for solving multi-stage stochastic mixed-integer programs are "naturally" parallelizable
  - L-shaped method and Progressive Hedging are particularly amenable
- PySP supports simple master-slave parallelism
  - Python pickle module for serialization
  - PYRO: Python Remote Objects
- Scalability to *O*(1000) scenarios and processors
  - Academics don't have commercial solver license issues!
  - For non-academics, prototype EC2/Gurobi deployment

Sandia National Laboratories

Scenario Sampling: How Many is Enough?

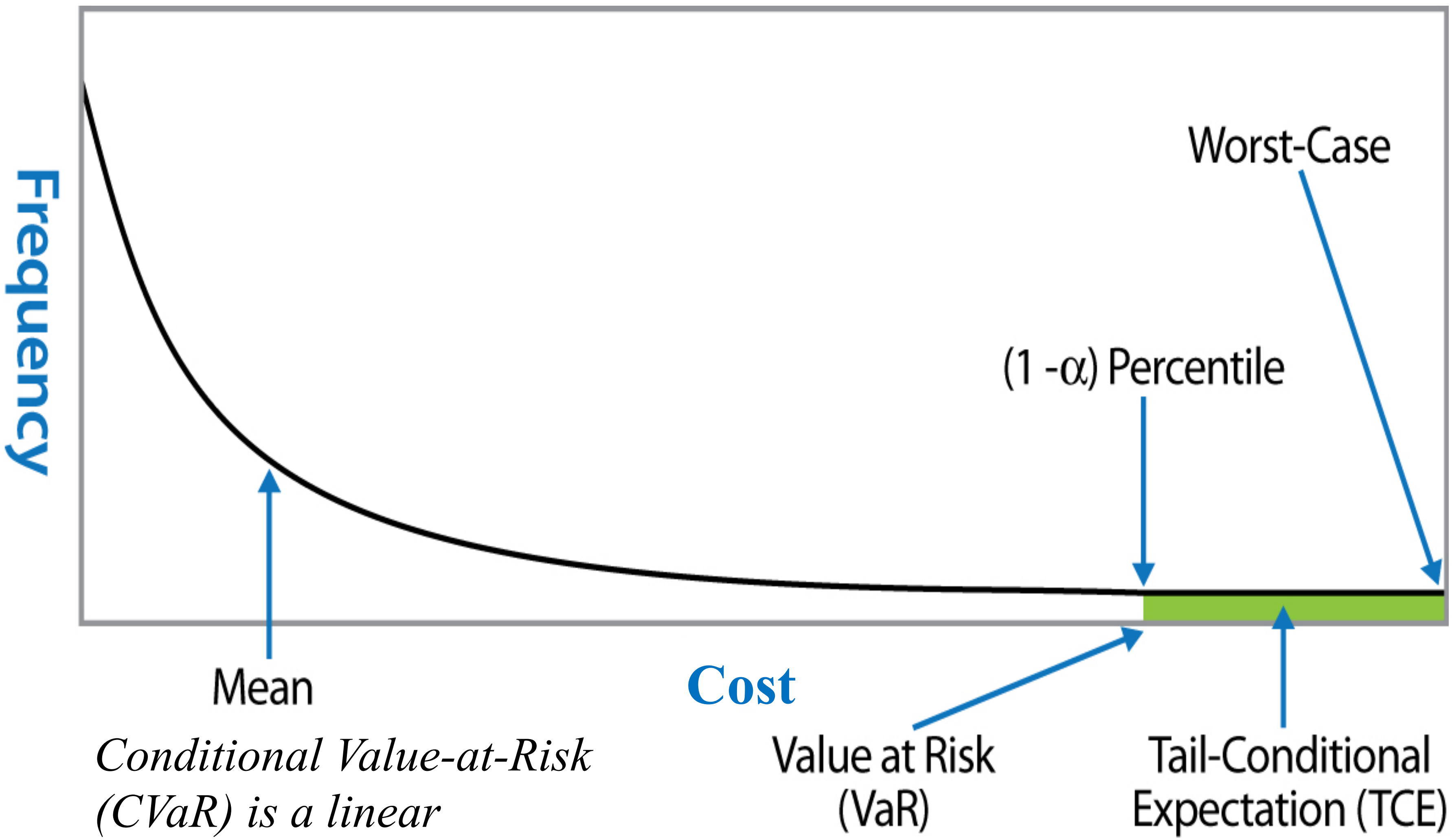
- Discretization of the scenario tree is "standard" in stochastic programming
  - With few exceptions, no mention of solution or objective stability
  - *Don't trust anyone who doesn't show you a confidence interval*

- Two general approaches in the literature
  - Has the solution converged? (Sample Average Approximation)
  - Has the objective converged? (Multiple Replication Procedure)

- Formal question we are concerned with
  - What is the probability that $\hat{x}$'s objective function value is suboptimal by more than α%?

- Initial *generic* implementation of MRP available in PySP
  - Has already identified disturbing results, in both the "too few samples" and "way too many samples" directions

Mean versus Risk? A Matter of Taste!



Conditional Value-at-Risk (CVaR) is a linear approximation of TCE

Sandia National Laboratories

Progressive Hedging and Conditional Value-at-Risk

- Scenario-based decomposition of Conditional Value-at-Risk models is conceptually straightforward (Schultz and Tiedemann 2006)

Proposition 5.1. *Assume that μ is discrete with finitely many scenarios $h_1, \ldots, h_J$ and corresponding probabilities $\pi_1, \ldots, \pi_J$. Let $\alpha \in (0, 1)$. Then the stochastic program*

$$\min\{Q_{CVaR_\alpha}(x) \ : \ x \in X\} \tag{11}$$

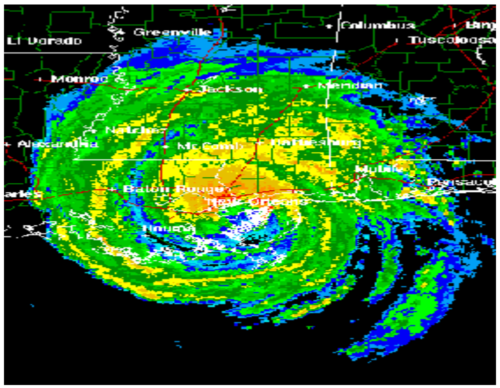
can be equivalently restated as

$$\min_{x,y,y',v,\eta} \Bigg\{ \eta + \frac{1}{1-\alpha} \sum_{j=1}^{J} \pi_j v_j \ : \ W y_j + W' y'_j = h_j - T x, \\ v_j \geq c^\top x + q^\top y_j + q'^\top y'_j - \eta, \\ x \in X, \quad \eta \in \mathbb{R}, \quad y_j \in \mathbb{Z}_+^{\bar{m}}, \\ y'_j \in \mathbb{R}_+^{m'}, \ v_j \in \mathbb{R}_+, \quad j = 1, \ldots, J \Bigg\}. \tag{12}$$

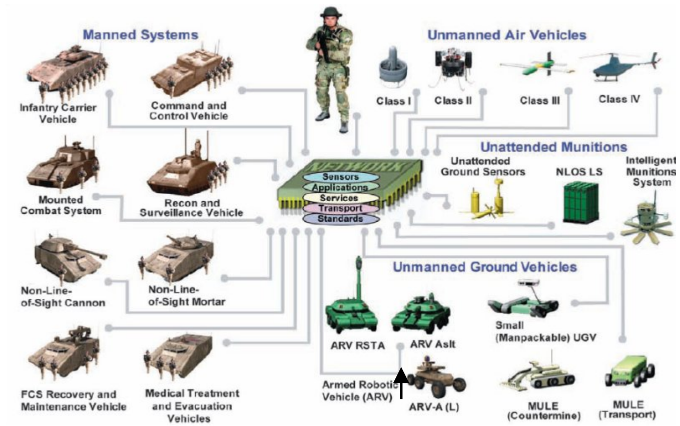
- But
  - Computational issues are largely unexplored

Selecting Scenarios to Ignore in Stochastic Optimization: Advances in Probabilistic Integer Programming Solvers

Ignoring the 100-year Flood
(Infrastructure Planning)



Capacitated Storage
(US Army Future Combat Systems)



Force-on-Force "Anomalies"
(Mission Planning)

Central Theme: The Need to Ignore a Small Fraction α of Scenarios During Optimization

$$
\begin{aligned}
\text{minimize} \quad & c \cdot x + \textstyle\sum_{s \in \mathcal{S}} p_s (f_s \cdot y_s) \qquad \text{(E)} \\
\text{subject to:} \quad & (x, y_s) \in \mathcal{Q}_s, \quad \forall s \in \{\mathcal{S} : d_s = 1\} \\
& \textstyle\sum_{s \in \mathcal{S}} p_s \, d_s \geq (1 - \alpha) \\
& d_s \in \{0, 1\}, \quad \forall s \in \mathcal{S}
\end{aligned}
$$

Results for network design:

- 2-8% better solutions than CPLEX, 1440m versus ~10m

Impact: Excellent heuristic for solving probabilistic integer programs
Key demonstration on large-scale, real-world problems

Scalable Heuristics for Stochastic Programming with Scenario Selection
Watson/Wets/Woodruff – INFORMS JOC (To appear)

PySP: Licensing, Availability, and Distribution

- Open-Source, BSD licensing
  - Non-infectious, use-at-will

- Dependencies
  - Subversion (not required, but rather useful)
  - Python! (2.5, 2.6, or 2.7)

- To get started, visit:
  - https://software.sandia.gov/trac/coopr

- Any questions?
  - Contact us:
    - jwatson@sandia.gov
    - dlwoodruff@ucdavis.edu

Conclusions

- We believe there are significant benefits to breaking down the barrier between modeling languages and solvers
  - Facilitated by various Python language features

- PySP provides a "case-study" illustrating that *generic* stochastic integer programming solvers case be rapidly prototyped and modified
  - Works for continuous cases as well, but that isn't as interesting

- Software is open-source, freely available, use-how-you-want-to.
- But:
  - We would like to work with people to integrate enhancements
  - And expand our suite of algorithms and test problems

Questions?

- We would like to formally acknowledge assistance from:
  - Bill Hart (Sandia)
  - Carl Laird and his research group (Texas A&M)
  - Patrick Steele (William and Mary)
  - Kevin Hunter (North Carolina State University)
  - Andres Weintraub and his research group (University of Chile)
  - Yueyue Fan and her research group (University of California Davis)
  - Roger Wets (University of California Davis)

PySP: For More Information…!