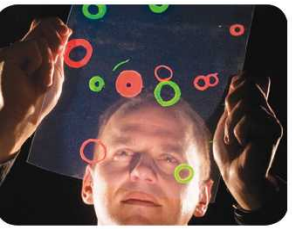
SAND2016-12156C



Recent developments in Pyomo

John D. Siirola, Carl D. Laird, Bethany L. Nicholson, Jean-Paul Watson, William E. Hart

Discrete Math & Optimization (1464)
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM USA

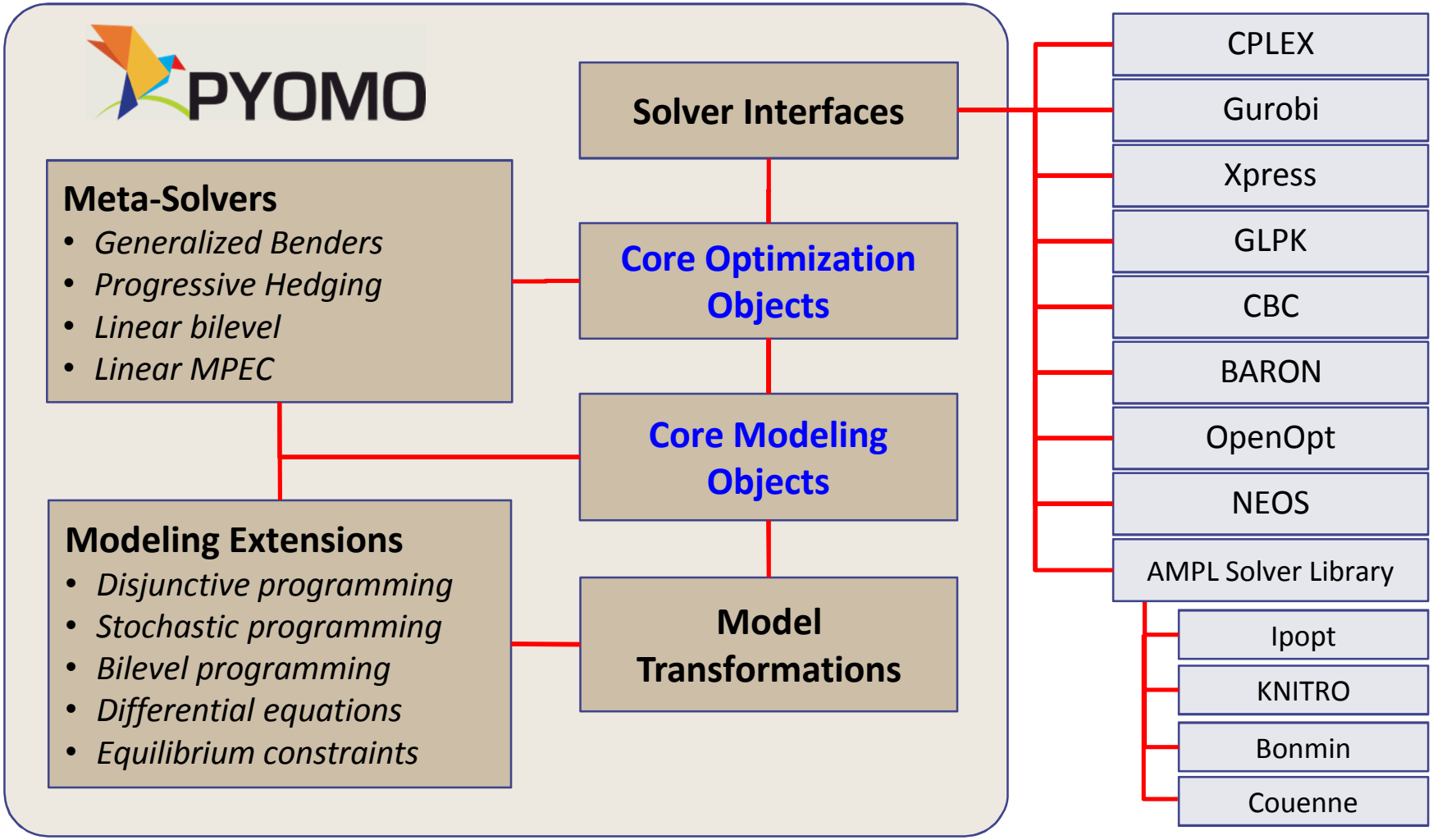
2016 AIChE Annual Meeting
15 November 2016

Sandia National Laboratories

Exceptional service in the national interest

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

Pyomo: *Python Optimization Modeling Objects*



A Quick Tour of Pyomo

PYOMO

Sandia National Laboratories

Idea: a Pythonic framework for formulating optimization models

- Provides a natural syntax to describe mathematical models
- Leverages an extensible optimization object model
- Formulates large models with a concise syntax
- Separates modeling and data declarations
- Enables data import and export in commonly used formats

Highlights:

- Python provides a clean, readable syntax
- Python scripts provide a flexible context for exploring the structure of Pyomo models

```
from pyomo.environ import *

model = ConcreteModel()

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

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

CCR Center for Computing Research

More than *just* mathematical modeling

Sandia National Laboratories

Scripting

- Construct models using native Python data
- Iterative analysis of models leveraging Python functionality
- Data analysis and visualization of optimization results

Model transformations (a.k.a. reformulations)

- Automate generation of one model from another
- Leverage Pyomo's object model to apply transformations sequentially
- E.g.: relax integrality, GDP -> Big M

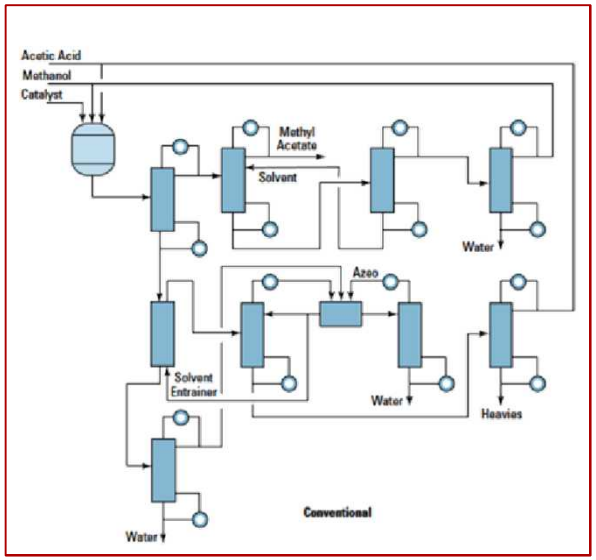
Meta-solvers

- Integrate scripting and/or transformations into optimization solver
- Leverage power of Python to build "generic" capabilities
- E.g.: progressive hedging, SP extensive form -> MIP

CCR Center for Computing Research

Motivation

- Institute for the Design of Advanced Energy Systems (IDAES)
  - Multi-year National Lab + University collaboration
    - Develop and utilize multi-scale, simulation-based computational tools and models to support the design, analysis, optimization, scale-up and troubleshooting of innovative, advanced fossil energy systems with carbon capture.
  - Next generation modeling and optimization platform
    - Flexible and open model
    - Complete provenance information
    - Supports advanced solvers and computer architecture
    - Intrusive UQ
    - Process Synthesis, Integration, and Intensification
    - Process Control and Dynamics
  - Built on Pyomo
    - IDAES has driven 2016 core Pyomo development



What are the “big” developments

Sandia National Laboratories

- Updated modeling
  - Improved Connector modeling component
  - Explicit unit annotation and verification
  - “Template” expressions
  - Support for efficiently slicing / iterating over hierarchical models
- New analysis
  - Ties to ODE simulators
- New algorithm / transformation support
  - Symbolic differentiation
  - Acceleration strategies for Progressive Hedging
- General
  - Improved installation support (pip)
  - “conda” channel for getting pyomo + solvers

CCR Center for Computing Research

What are the “big” developments

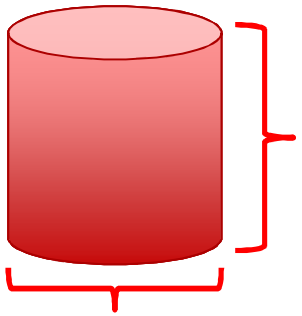
- Updated modeling
  - Improved Connector modeling component
  - Explicit unit annotation and verification
  - “Template” expressions
  - Support for efficiently slicing / iterating over hierarchical models
- New analysis
  - Ties to ODE simulators
- New algorithm / transformation support
  - Symbolic differentiation
  - Acceleration strategies for Progressive Hedging
- General
  - Improved installation support (pip)
  - “conda” channel for getting pyomo + solvers

A trivial "Soda Can Example"

Problem: minimize the metal used in a soda can

minimize $2 \pi r (r + h)$ *# Surface Area*

with $\pi h r^2 = 355$ *# Volume*

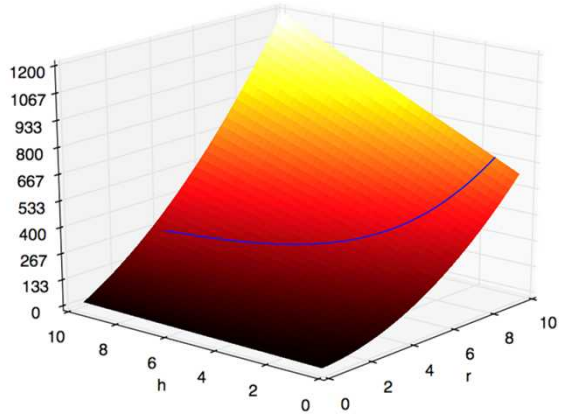



```
# sodacan.py
from pyomo.environ import *
from math import pi

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0)
M.h = Var(bounds=(0,None), initialize=15.0)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= pi*M.h*M.r**2 == 355)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
      % (value(model.r), value(model.h), value(model.surf_area)))
```



“Soda Can” with units

Sandia National Laboratories

```
# sodacan.py
from pyomo.environ import *
from math import pi
um = UnitsManager()

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0, units=um.cm)
M.h = Var(bounds=(0,None), initialize=15.0, units=um.cm)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= pi*M.h*M.r**2 == 355)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
      % (value(model.r), value(model.h), value(model.surf_area)))
```

CCR Center for Computing Research

Check the "Soda Can" units

```
# sodacan.py
from pyomo.environ import *
from math import pi
um = UnitsManager(enable_units_checking=True)

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0, units=um.cm)
M.h = Var(bounds=(0,None), initialize=15.0, units=um.cm)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= pi*M.h*M.r**2 == 355)

um.check_units(model)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
      % (value(model.r), value(model.h), value(model.surf_area)))
```

Fix the "Soda Can" units

Sandia National Laboratories

```
# sodacan.py
from pyomo.environ import *
from math import pi
um = UnitsManager(enable_units_checking=True)

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0, units=um.cm)
M.h = Var(bounds=(0,None), initialize=15.0, units=um.cm)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= pi*M.h*M.r**2 == 355*um.cm**3)

um.check_units(model)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
      % (value(model.r), value(model.h), value(model.surf_area)))
```

CCR Center for Computing Research

We *do not* do automatic conversion

Sandia National Laboratories

```
# sodacan.py
from pyomo.environ import *
from math import pi
um = UnitsManager(enable_units_checking=True)

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0, units=um.cm)
M.h = Var(bounds=(0,None), initialize=15.0, units=um.cm)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= pi*M.h*M.r**2 == 12*um.fluid_ounce)

um.check_units(model)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
      % (value(model.r), value(model.h), value(model.surf_area)))
```

CCR Center for Computing Research

…but allow explicit conversion

```
# sodacan.py
from pyomo.environ import *
from math import pi
um = UnitsManager(enable_units_checking=True)

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0, units=um.cm)
M.h = Var(bounds=(0,None), initialize=15.0, units=um.cm)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= um.convert(pi*M.h*M.r**2, um.fluid_ounce)
                            == 12*um.fluid_ounce)
um.check_units(model)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
      % (value(model.r), value(model.h), value(model.surf_area)))
```

…and support custom units

```
# sodacan.py
from pyomo.environ import *
from math import pi
um = UnitsManager(enable_units_checking=True)
um.create_new_unit('can_of_coke', 'fluid_ounces', 101)

M = ConcreteModel()
M.r = Var(bounds=(0,None), initialize=5.0, units=um.cm)
M.h = Var(bounds=(0,None), initialize=15.0, units=um.cm)

M.surf_area = Objective(expr=  2*pi*M.r*(M.r + M.h))
M.volume = Constraint(expr= um.convert(pi*M.h*M.r**2, um.can_of_coke)
                                == 1*um.can_of_coke)
um.check_units(model)

SolverFactory('ipopt').solve(model)
print('-- optimal solution --\nradius: %0.2f\nheight: %0.2f\narea: %0.2f'
  % (value(model.r), value(model.h), value(model.surf_area)))
```

Checking offset units is … *nuanced*

- Offset units (units without a common origin) are challenging:
  - $27°\mathrm{C} + 27°\mathrm{C} = 54°\mathrm{C}$

Checking offset units is … *nuanced*

- Offset units (units without a common origin) are challenging:
  - $27°\mathrm{C} + 27°\mathrm{C} = 54°\mathrm{C}$
- Addition of two offset units does not make sense!
  - "Proof": converting terms to other units shouldn't change truth
    - $300\mathrm{K} + 300\mathrm{K} = 600\mathrm{K}, \quad 600\mathrm{K} \neq 54°\mathrm{C}$
  - Corrected: "delta units"
    - $27°\mathrm{C} + 27\Delta°\mathrm{C} = 54°\mathrm{C}$
- Subtraction of offset units yields delta units
  - $50°C - 30°C = 20\Delta°\mathrm{C}$
- This does not "play well" with non-units aware model manipulation.
  - For the time being, offset units are not allowed in expressions
    - But explicit conversion is OK.

pyomo.dae recent developments

- Interface to ODE integrators in Scipy for:
  - Simulating systems of ODEs
  - Initializing discretized dynamic optimization problems
  - Verifying dynamic optimization solutions

- Demonstrated compatibility with other Pyomo extensions like PySP for stochastic programming
  - Dynamic systems under uncertainty
    - Parameter estimation
    - Optimal control

Interfacing with Scipy integrators

$$\frac{d\theta}{dt} = \omega$$

$$\frac{d\omega}{dt} = -b * \omega - c * \sin\theta$$

$$\theta(0) = \pi - 0.1, \quad \omega(0) = 0$$

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

model = ConcreteModel()

model.t = ContinuousSet(bounds=(0.0, 10.0))

model.b = Param(initialize=0.25)
model.c = Param(initialize=5.0)

model.omega = Var(model.t)
model.theta = Var(model.t)

model.domegadt = DerivativeVar(model.omega, wrt=model.t)
model.dthetadt = DerivativeVar(model.theta, wrt=model.t)

model.omega[0] = 0.0
model.theta[0] = 3.14 - 0.1

def _diffeq1(m,t):
    return m.dthetadt[t] == m.omega[t]
model.diffeq1 = Constraint(model.t, rule=_diffeq1)

def _diffeq2(m,t):
    return m.domegadt[t] == -m.b*m.omega[t] \
        - m.c*sin(m.theta[t])
model.diffeq2 = Constraint(model.t, rule=_diffeq2)
```

- ODE model simulation

```
mysim = Simulator(model, package='scipy')
mysim.simulate()
```

- Model initialization

```
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(model, nfe=8, ncp=5)

mysim.initialize_model()
```

Interface with Scipy integrators

Sandia National Laboratories

System of ODEs

$$\frac{d\theta}{dt} = \omega$$
$$\frac{d\omega}{dt} = -b * \omega - c * \sin\theta$$
$$\theta(0) = \pi - 0.1, \quad \omega(0) = 0$$

Pyomo Model

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

m = ConcreteModel()

m.t = ContinuousSet(bounds=(0.0, 10.0))

m.b = Param(initialize=0.25)
m.c = Param(initialize=5.0)

m.omega = Var(m.t)
m.theta = Var(m.t)

m.domegadt = DerivativeVar(m.omega, wrt=m.t)
m.dthetadt = DerivativeVar(m.theta, wrt=m.t)

m.omega[0] = 0.0
m.theta[0] = 3.14 - 0.1

def _diffeq1(m,t):
    return m.dthetadt[t] == m.omega[t]
m.diffeq1 = Constraint(m.t, rule=_diffeq1)

def _diffeq2(m,t):
    return m.domegadt[t] == -m.b*m.omega[t] \
        - m.c*sin(m.theta[t])
m.diffeq2 = Constraint(m.t, rule=_diffeq2)
```

Simulate and/or initialize optimization model

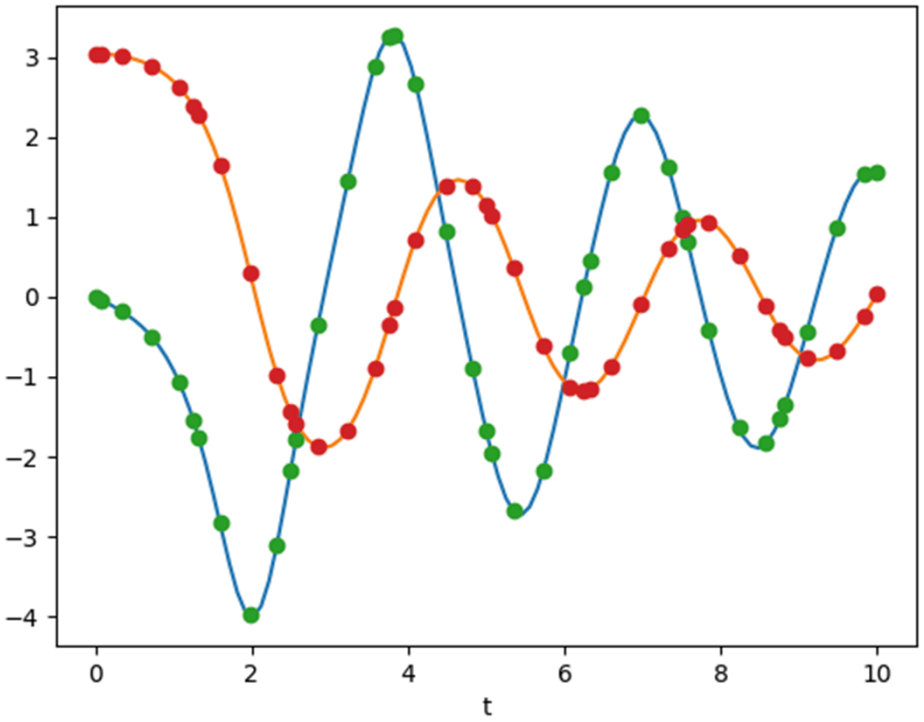
- ODE model simulation

```
mysim = Simulator(m, package='scipy')
mysim.simulate()
```

- Model initialization

```
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(m, nfe=8, ncp=5)

mysim.initialize_model()
```



Omega simulated

Theta simulated

Omega initialized at discretization points

Theta initialized at discretization points

Use the same Pyomo model for simulation and optimization!

Stochastic programming in PySP

Sandia National Laboratories

- PySP provides a convenient framework for describing and generating stochastic programming problems in Pyomo
- Our workhorse algorithm is Progressive Hedging
  - Scenario-based decomposition
    - "No" master problem
    - One subproblem per scenario
    - Relax nonanticipativity constraints
  - Iteratively converge the stage nonanticipativity constraints
    - Penalize decision variable value by weight $w^T x$
    - Penalize deviation from average, $\rho \, \|x - \bar{x}\|^2$
    - Update w using ρ

CCR Center for Computing Research

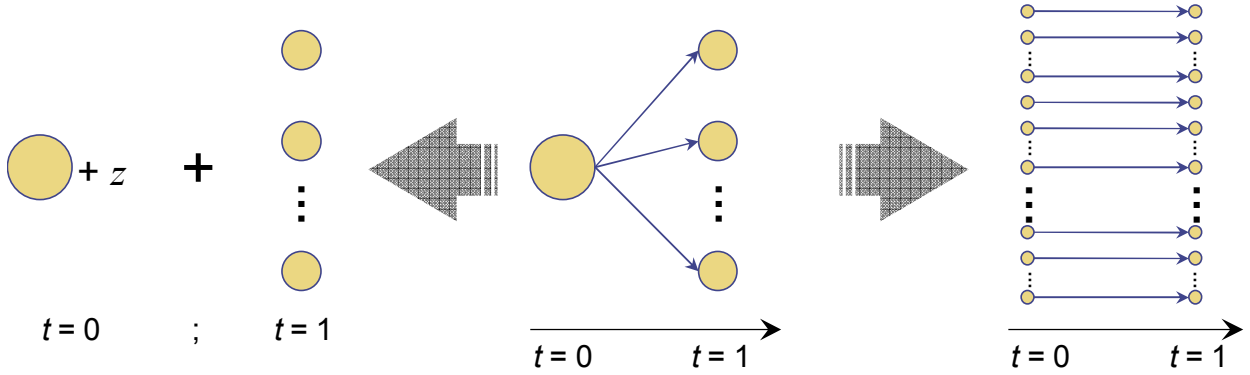
PH: Benefits and open challenges

- PH avoids many of the challenges experienced by Benders
  - No restriction on stage variable domains (discrete 2nd stage OK)
  - "Trivially" extends to multistage problems
  - No master problem, no cuts generated: no problem "bloat"

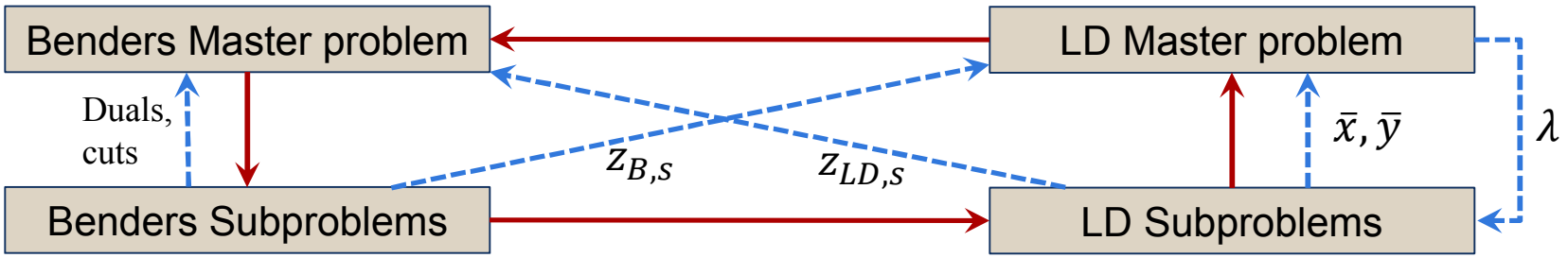
- BUT…
  - Not provably convergent for the discrete case
    - …although bounds exist on the quality of solution you get
  - "Infeasible path" algorithm
  - Discrete variable cycling
  - Slow convergence due to "holdout" scenarios
  - How to choose ρ???

We will attempt to address these challenges

PH + Benders: orthogonal decompositions

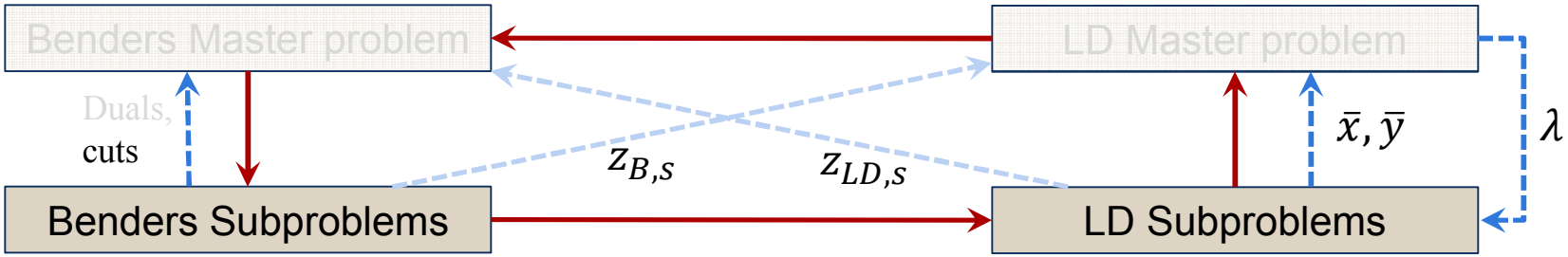


- Can we leverage ideas from each to accelerate the other?
  - Related: Cross Decomposition (Lagrangean Decomposition + Benders)
    - [Van Roy 1983; Holmberg, 1990; Mitra,et al. 2016]

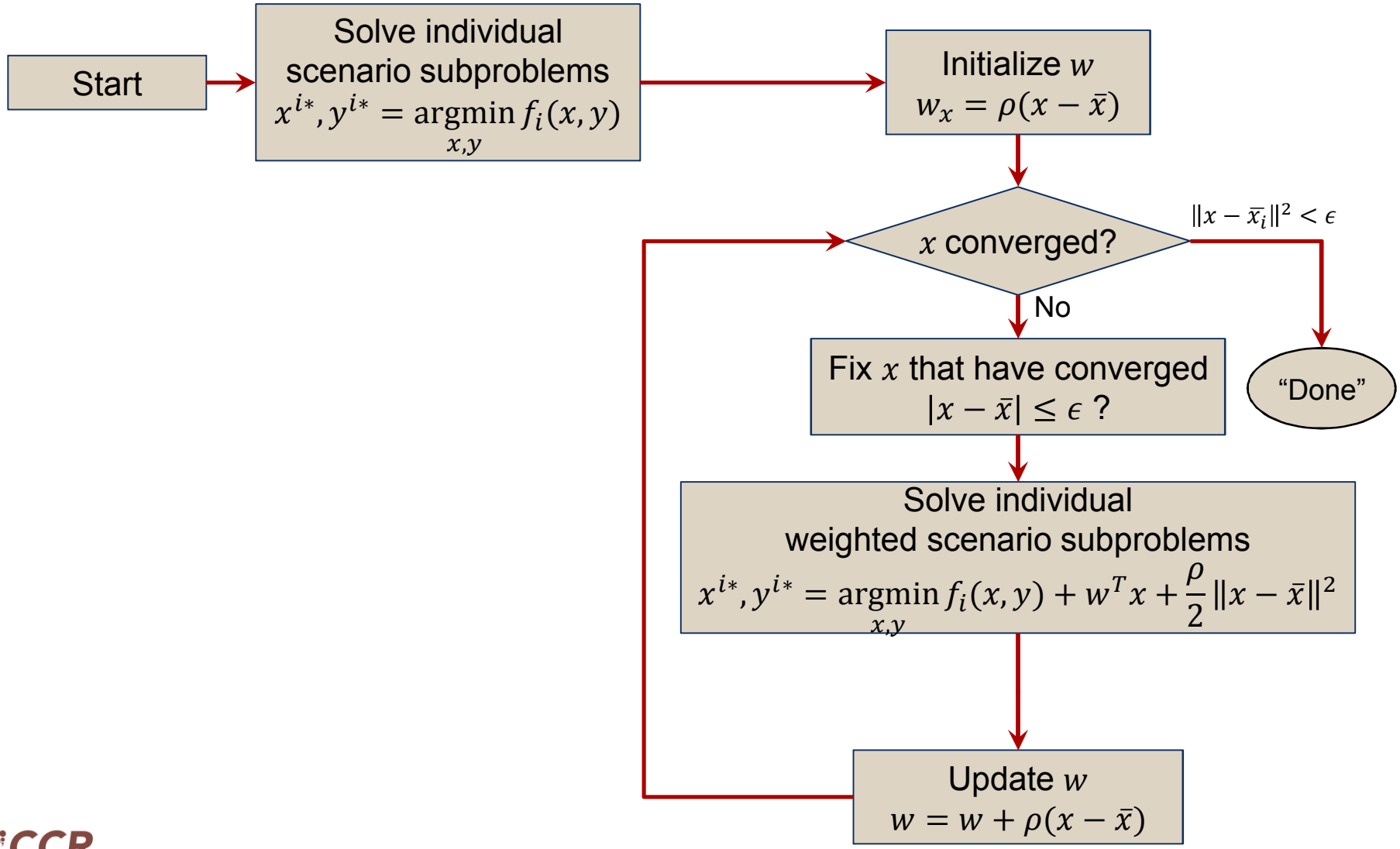


Cross Decomposition for PH?

- Challenges with naïve Cross Decomposition for PH
  - No LD Master problem
  - Discrete second stage variables
  - PH subproblems do not provide proper Lagrangean bounds
    - Except at *iteration 0*



Progressive Hedging: the algorithm



Improving PH: borrowing from Benders

- Consider the Benders "feasibility" cut:
  - Given x^* computed by the RMP, if (dual) subproblem is unbounded, add a cut determined by an extreme ray in the dual space to the RMP
- In PH, a similar operation would be fix the values of x in subproblem f_j to the values computed by subproblem f_i:

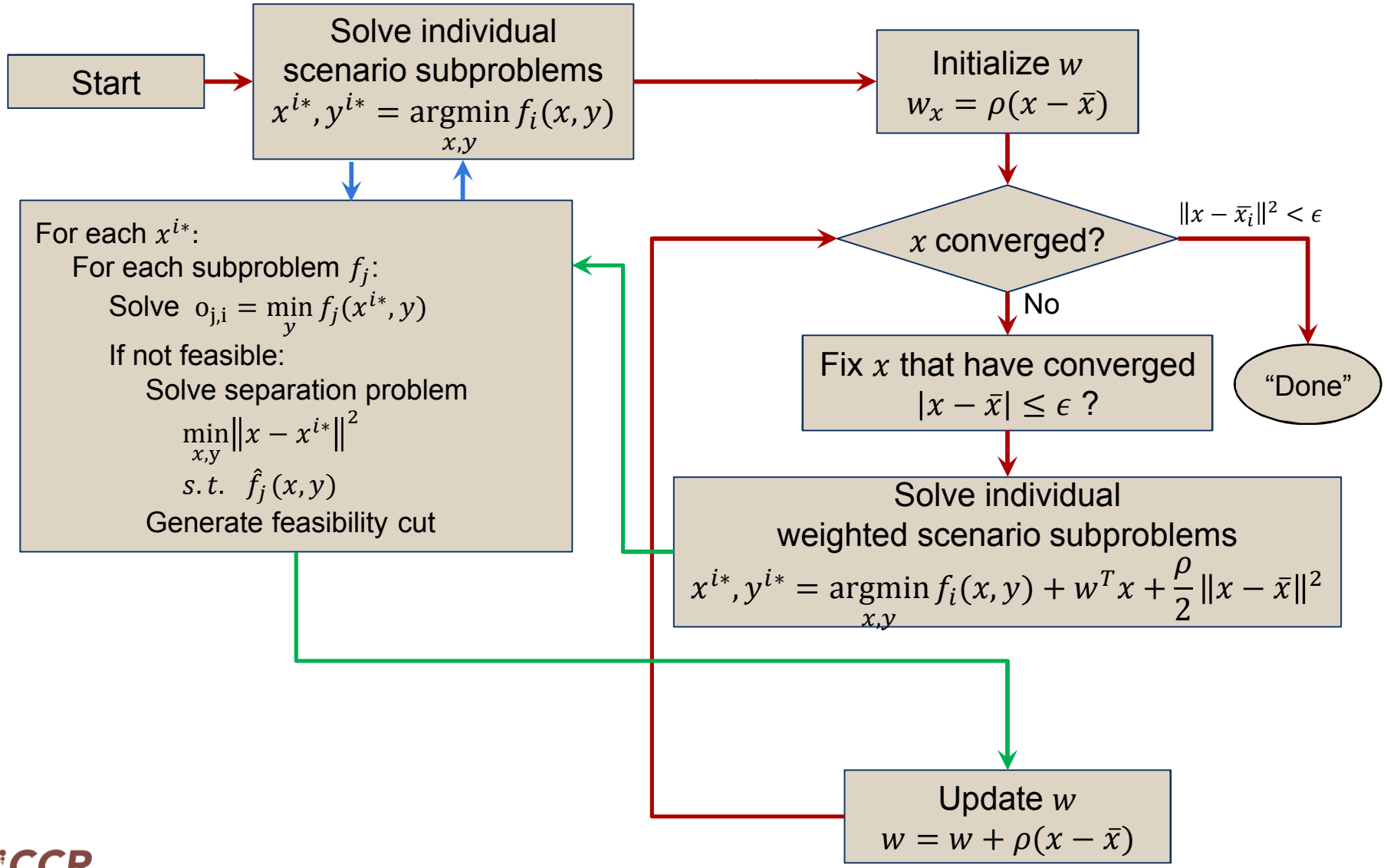
$$\min_y f_j(x^{i*}, y)$$

  - If the problem is infeasible, then we can solve a separation problem (in the primal space) to determine a valid cut in the 1st-stage variables:

$$\min_{x,y} \|x - x^{i*}\|^2$$
$$s.t. \quad \hat{f}_j(x, y)$$

  - Notes:
    - $\hat{f}_j$ is the continuous relaxation of f_j → not guaranteed to generate a cut
    - The resulting cut is valid for *all* scenario subproblems
    - $\sum p_i f_i(x^{i*}, y^{i*})$ for initial scenario solves ($w = 0$) gives Lagrangian bound

PH + feasibility



Case study: stochastic network flow

- 2-stage stochastic network flow from [Watson & Woodruff, 2011]:
  - 1st stage variables:
    - $x_a \; \forall \; a \in Arcs; \; \{x_a |\; x_a \in R, 0 \leq x_a \leq x_a^{UB}\}$ Capacity of Arc a
    - $b_a^0 \; \forall \; a \in Arcs; \quad \mathrm{b_a^0} \in \{0,1\}$ Arc a is available
  - 2nd stage variables:
    - $y_a^s \; \forall \; a \in Arcs, s \in Scenarios; \; \{y_a^s \mid y_a^s \in R, 0 \leq y_a^s \leq x_a\}$ Flow across Arc a in scenario s
    - $b_a^s \; \forall \; a \in Arcs, s \in Scenarios; \quad \mathrm{b_a^s} \in \{0, b_a^s\}$ Arc a is in use for scenario s
- PH ($\rho = 100$, with "Watson-Woodruff" extensions)
  - Significant cycling (no convergence after 1000 iterations)
- PH ($\rho = 100$, with WW extensions, with feasibility cuts)
  - Converges in 42 iterations (objective: 164426, 2726 seconds)
  - 6 preliminary cut passes: raises Lagrangian LB 135085 → 148656

Case study: UC + *N-1* + switching

- 2-stage unit commitment model for the electric power grid
  - 24-hour horizon, 1-hour commitment intervals
  - Explicitly include *N-1* analysis (loss of any 1 generator / non-radial line)
    - Each contingency modeled as a no-cost recourse scenario
    - Line switching (opening /closing a line) in 1st and 2nd stages

- Case 1: 5 busses, 7 generators (13 scenarios):
  - Optimal solution (extensive form): 19.9756
  - Default PH ($\rho = 1$):
    - 17 iterations, objective = 22.9997, total time 123 seconds
  - PH ($\rho = 1$) + Feasibility cuts:
    - 3 feasibility cut iterations at PH iteration 0
      - Improved Lagrangean bound from 19.7 to 19.88
    - 12 iterations, objective = 20.11, total time 568 seconds

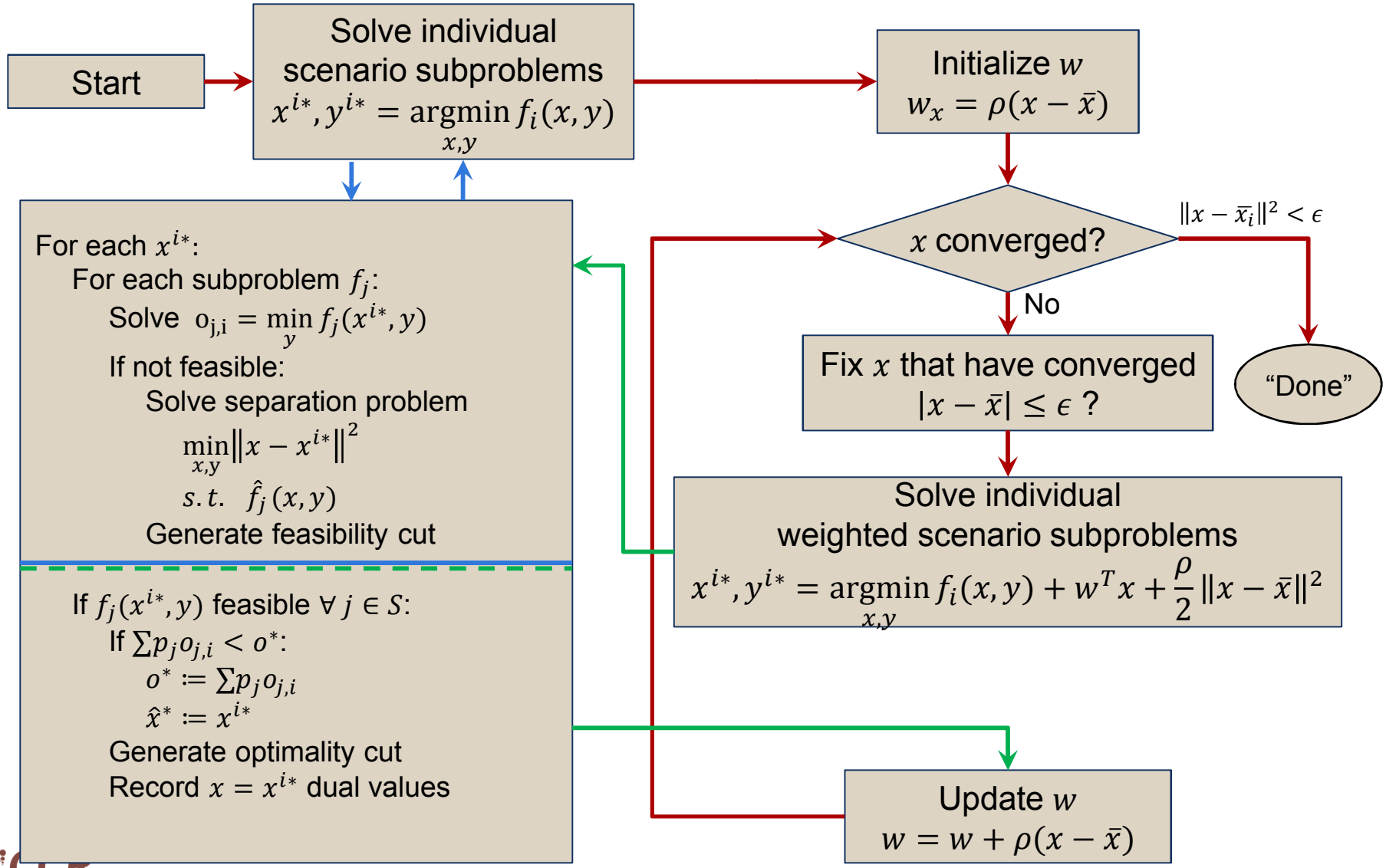
Improving PH: borrowing from Benders

- What happens if the subproblem $\min_y f_i(x^{j*}, y)$ is feasible?
  - The 1st stage decisions are valid in this scenario
  - If x^{j*} is valid in ALL scenarios, then x^{j*} satisfies nonanticipativity and the expectation of the subproblems forms a valid upper bound

$$E[f(x^*, y)] \leq E\left[f\left(x^{j*}, y\right)\right]$$

- If the first stage variables are all discrete
  - Repeating this process for all x^{i*} and identify additional nonanticipative solutions, then the upper bound can be used to generate *optimality cuts* to exclude sub-optimal solutions
  - These cuts are also valid on all scenario subproblems

PH + feasibility + optimality cuts



Case Studies: *farmer*

- 3-scenario farmer problem [Birge & Louveaux]
  - Integer acreage allocations
  - Cplex 12.5
    - PH ($\rho = 1$): 297 iterations, 21.55 seconds: objective = -108390
    - PH + optimality: 47 iterations, 8.75 seconds: objective = -108390

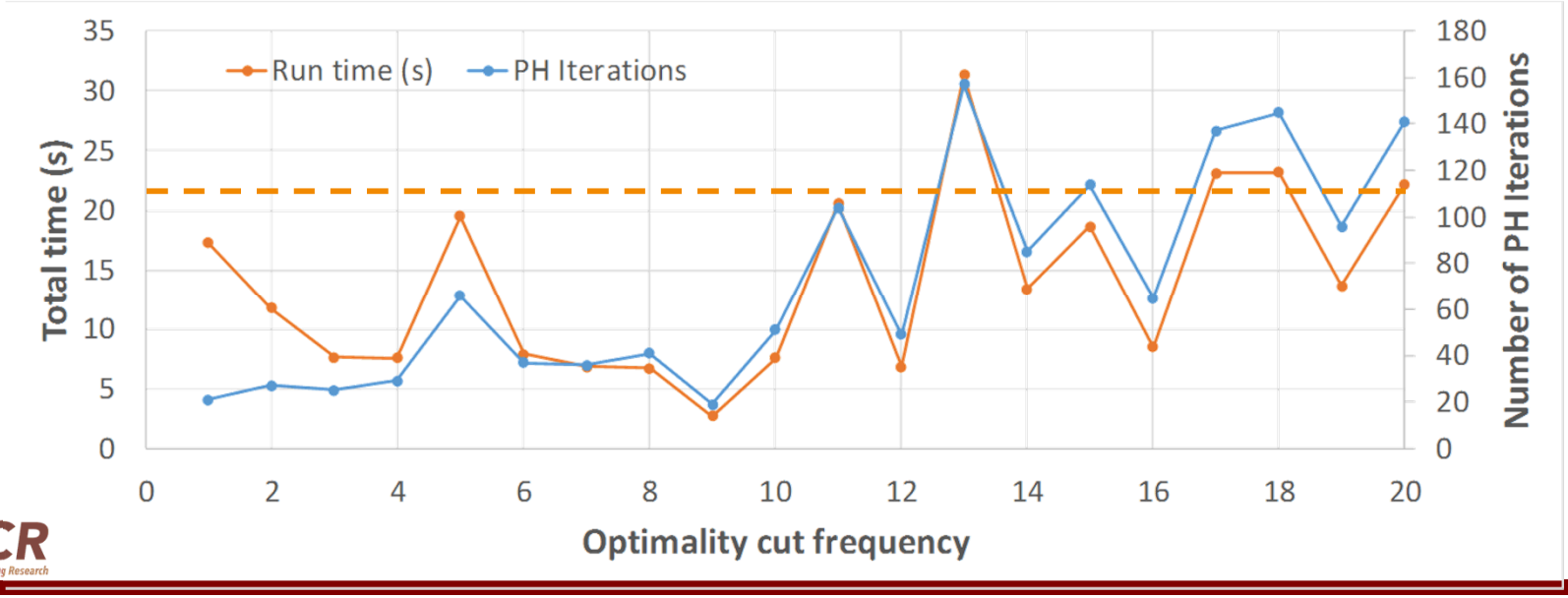
Case Studies: *farmer*

- 3-scenario farmer problem [Birge & Louveaux]
  - Integer acreage allocations
  - Cplex 12.5
    - PH ($\rho = 1$): 297 iterations, 21.55 seconds: objective = -108390
    - PH + optimality: 47 iterations, 8.75 seconds: objective = -108390
- Very sensitive to the solver!
  - Gurobi 6.0.4
    - PH ($\rho = 1$): 39 iterations, 1.09 seconds: objective = -108390
    - PH + optimality: 154 iterations, 16.4 seconds: objective = -108390

Case Studies: *farmer*

- 3-scenario farmer problem [Birge & Louveaux]
  - Integer acreage allocations
  - Cplex 12.5
    - PH ($\rho = 1$): 297 iterations, 21.55 seconds: objective = -108390
    - PH + optimality: 47 iterations, 8.75 seconds: objective = -108390

- Very sensitive to the strategy!



Case study: Stochastic Unit Commitment

- Unit commitment under demand uncertainty
  - 24 hour horizon, 1-hour intervals

- Case 1: 5 busses, 7 generators, 3 load scenarios
  - Optimal solution (extensive form): 348.98
  - Default PH ($\rho = 10$):
    - Significant cycling (no convergence after 100 iterations)
  - PH ($\rho = 10$) + Optimality cuts:
    - 20 iterations, objective = 348.9835

Improving PH: setting ρ

- How do we get "good" values of ρ?
  - Currently: experimentation
  - Challenge: ρ is problem dependent
    - Too low and PH never converges
    - Too high and PH rapidly converges to suboptimal solution
  - Hint: scale relative to cost of each variable [Watson & Woodruff, 2011]

$$\rho_i \propto \frac{C_i}{\left|x_i^{min} - x_i^{max} + 1\right|}$$

- We can get good cost estimates from the subproblem duals
  - When we evaluate $f_j(x^{i*}, y)$, record the duals for $x = x^{i*}$
  - Compute average duals weighted relative to scenario probability

Case Studies: *farmer*

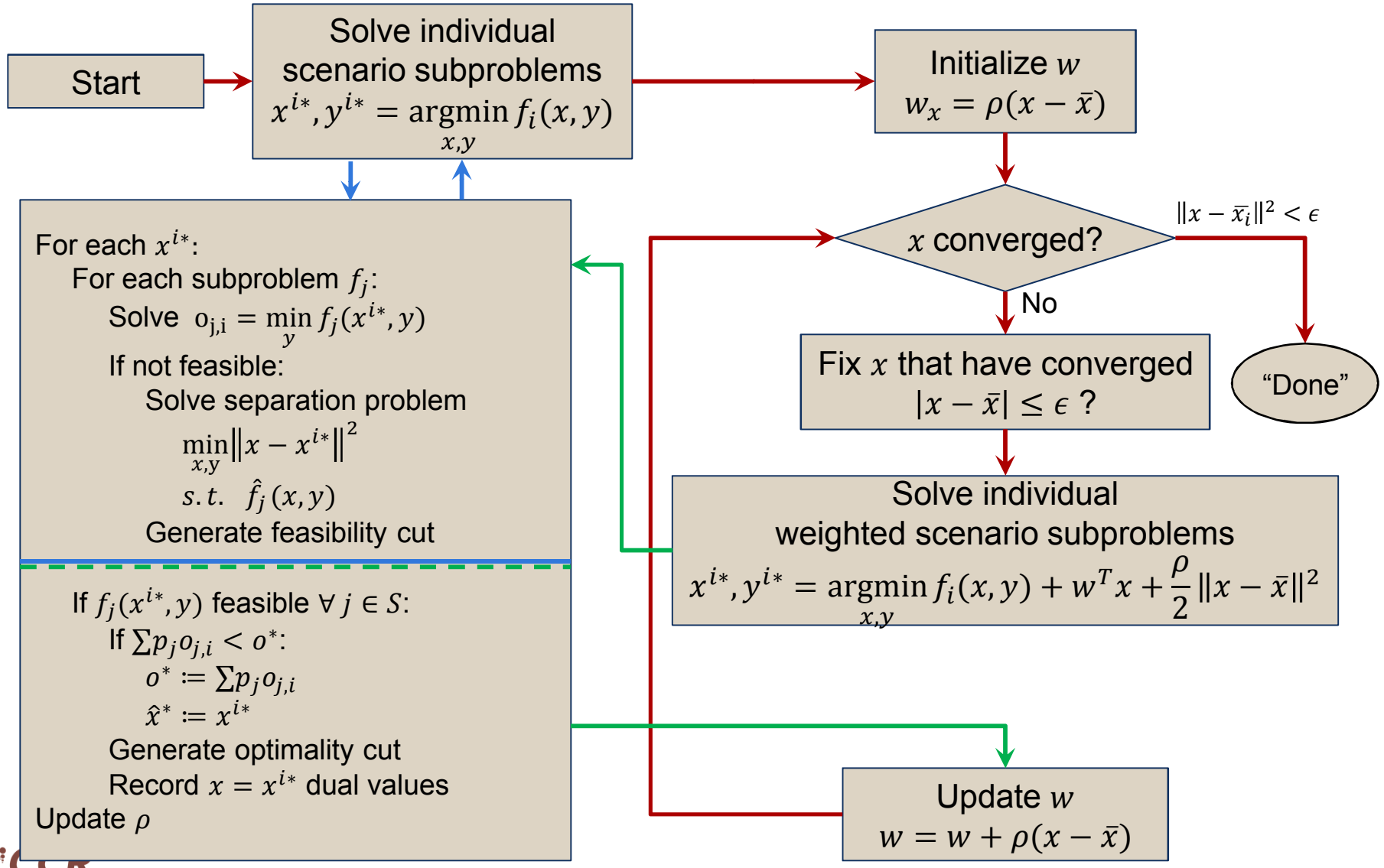
- 3-scenario farmer problem [Birge & Louveaux]
  - Continuous acreage allocations
    - PH ($\rho = 1$):
      - 33 iterations, 1.06 seconds: objective = -108388.7726
    - PH + ρ setter:
      - 34 iterations, 1.34 seconds: objective = -108389.7811
  - Discrete acreage allocations
    - PH ($\rho = 1$):
      - 39 iterations, 1.09 seconds: objective = -108390
    - PH + optimality cuts:
      - 154 iterations, 16.4 seconds: objective = -108390
    - PH + optimality cuts + ρ setter:
      - 40 iterations, 4.42 seconds: objective = -108390

Case studies: UC + *N-1* analysis

- Recall:
  - Optimal solution (extensive form): 19.9756
  - Default PH ($\rho = 1$):
    - 17 iterations, objective = 22.9997, total time 123 seconds
  - PH ($\rho = 1$) + Feasibility cuts:
    - 3 feasibility cut iterations at PH iteration 0
      - Improved Lagrangian bound from 19.7 to 19.88
    - 12 iterations, objective = 20.11, total time 568 seconds

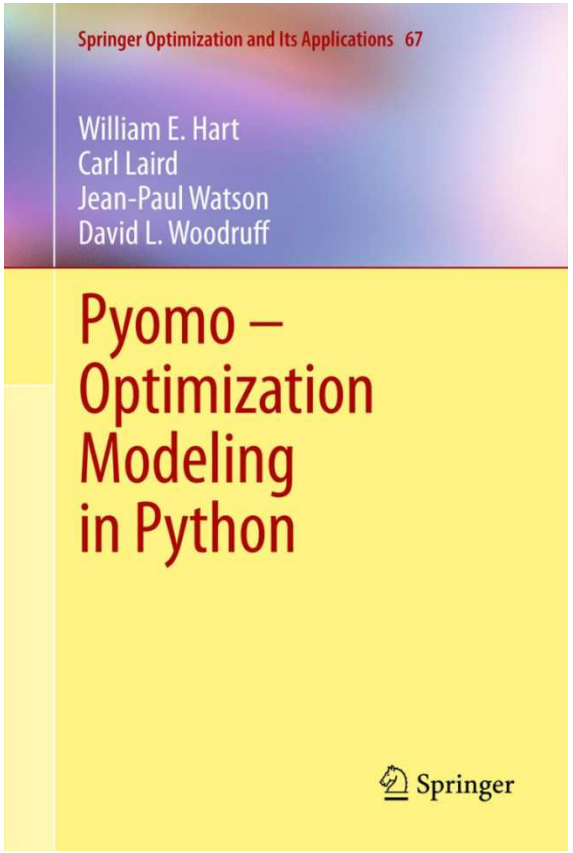
- Now:
  - PH + Feasibility cuts + ρ setter
    - 20 iterations, objective = 19.9808, total time 494 seconds

PH + feasibility + optimality cuts+ ρ



For more information…

- Project homepage
  - http://www.pyomo.org
  - https://github.com/Pyomo/pyomo
- Mailing lists
  - "pyomo-forum" Google Group
  - "pyomo-developers" Google Group
- "The Book"
  - Second edition going to press in O(days)



- Mathematical Programming Computation paper:
  - Pyomo: Modeling and Solving Mathematical Programs in Python (3(3), 2011)

Acknowledgements

- Portions of this work were supported by
  - 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
  - The U.S. Department of Energy Office of Science Advanced Scientific Computing Research program
  - The Sandia National Laboratories' Laboratory Directed Research & Development (LDRD) Program