Plasmo Short Course

Modeling & Optimization of Natural Gas Networks

Jordan Jalving Victor Zavala

Department of Chemical and Biological Engineering
University of Wisconsin-Madison

Mathematics and Computer Science Division
Argonne National Laboratory

Argonne National Laboratory
June 2017

Getting Started

- We will use `Julia` notebooks that will be executed on `JuliaBox`
- Access `http://juliabox.com`
- Click on Sync tab and clone GIT repo `https://github.com/zavalab/JuliaBox.git`
- Click on Jupyter tab and access notebooks under JuliaBox/PlasmoShortCourse

Optimization Overview

Standard Nonlinear Programming (NLP) Problem

$$\begin{aligned} \min_{x} \; & f(x) && \text{Objective function (e.g., economics)} \\ s.t. \; & c(x) = 0 && \text{Equality constraints (e.g. physical equations)} \\ & g(x) \geq 0 && \text{Inequality constraints (e.g. physical limits)} \end{aligned}$$

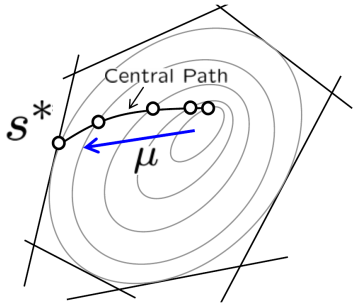
This tutorial considers PDE-constrained optimization problems, which we will cast as standard NLPs.

Example NLP

$$\begin{aligned} \min_{x_1, x_2} \; & x_1^2 + x_2^2 \\ s.t. \; & x_1 + x_2 = 5 \\ & e^{x_1} \leq 4 \end{aligned}$$

Ipopt - Interior Point OPTimizer

- To solve NLPs we will use the interior-point NLP solver `Ipopt`.
- `Ipopt` uses a logarithmic barrier to transform the NLP into an equality-constrained subproblem:



$$\min_x \phi^\mu(x) := f(x) - \mu \sum_{j=1}^{m} ln(g_j(x))$$

$$s.t.\ c(x) = 0$$

- The barrier subproblem is solved for decreasing values of μ to reach a solution of the original NLP. This is done by solving:

$$\nabla_x \mathcal{L}(x,\lambda) = \nabla_x \phi^\mu(x) + \nabla_x c(x)\lambda = 0 \qquad c(x) = 0 \qquad \Longrightarrow \begin{bmatrix} H(x,\lambda) & \nabla_x c(x) \\ \nabla_x c(x)^T & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ c(x) \end{bmatrix}$$

- Highly efficient sparse linear algebra and globalization strategies enable solution of complex NLPs.
- `Ipopt` can solve NLPs with *millions* of variables and constraints.

Julia and JuMP

`Julia`

- Language for high-performance scientific computing
- Extensive library of tools for optimization, statistics, plotting, graphs,...
- Performance comparable to C
- Designed to enable parallelism

`JuMP`

- Algebraic modeling language for optimization written in `Julia`
- Syntax close to natural mathematical expressions
- Processes problems at speeds comparable to `GAMS` and `AMPL`
- Support for open source & commercial solvers (e.g. `Ipopt`, `Gurobi`)
- Automatic computation of derivatives

Solving Optimization Problems with JuMP

Consider our previous NLP example:

$$\begin{aligned} \min_{x_1,x_2} \quad & x_1^2 + x_2^2 \\ s.t. \quad & x_1 + x_2 = 5 \\ & e^{x_1} \leq 4 \end{aligned}$$

```
using JuMP
using Ipopt
m = Model()
@variable(m, x1)
@variable(m, x2)
@objective(m, Min, x1^2 + x2^2)
@constraint(m, x1+x2 == 5)
@NLconstraint(m, exp(x1) <= 4)
solve(m)
```

- `Ipopt` internally solves a sequence of barrier problems of the form:

$$\begin{aligned} \min_{x_1,x_2} \quad & x_1^2 + x_2^2 - \mu \log(4 - e^{x_1}) \\ s.t. \quad & x_1 + x_2 = 5 \end{aligned}$$

- Example implemented in Julia notebook `simple_model.ipynb`

Solving Optimization Problems with JuMP

`JuMP` also enables compact syntax expressions:

$$\min_{x_1,x_2} \sum_{j\in\{1,2\}} x_j^2$$
$$s.t. \sum_{j\in\{1,2\}} x_j = 5$$
$$e^{x_1} \leq 4$$

```
using JuMP
using Ipopt
m = Model()
S = collect(1:2)
@objective(m, Min, sum(x[j]^2 for j in S))
@variable(m, x[S])
@constraint(m, sum(x[j] for j in S) == 5)
@NLconstraint(m, exp(x[1]) <= 4)
solve(m)
```

Example implemented in Julia notebook `simple_model_set.ipynb`

Gas Pipeline Modeling

We begin with the isothermal Euler equations for a single pipeline:

- Mass Conservation

$$\frac{\partial \rho(t,x)}{\partial t} + \frac{\partial (\rho(t,x)v(t,x))}{\partial x} = 0$$

- Momentum Conservation

$$\frac{\partial (\rho(t,x)v(t,x))}{\partial t} + \frac{\partial (\rho(t,x)v(t,x)^2 + p(t,x))}{\partial x} = -\frac{\lambda}{2D}\rho(t,x)v(t,x)\,|v(t,x)|$$

- We can express conservation equations in terms of pressure and flow by using:

$$p(t,x) = c^2\rho(t,x), \quad v(t,x) = \frac{V(t,x)}{A}, \quad f(t,x) = \rho(t,x)V(t,x)$$

- The gas speed of sound c can be computed from the ideal gas law:

$$c^2 = zRT$$

- The Euler equations are highly nonlinear hyperbolic PDEs

Gas Pipeline Modeling

The following procedure describes how to pose the Euler equations in terms of flow and pressure:

Mass Conservation

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho v)}{\partial x} = 0$$
$$\frac{\partial(\frac{p}{c^2})}{\partial t} + \frac{\partial(\rho \frac{V}{A})}{\partial x} = 0$$
$$\frac{1}{c^2}\frac{\partial p}{\partial t} + \frac{1}{A}\frac{\partial(\rho V)}{\partial x} = 0$$
$$\frac{\partial p}{\partial t} + \frac{c^2}{A}\frac{\partial f}{\partial x} = 0$$

Momentum Conservation

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho v^2) + p}{\partial x} = -\frac{\lambda}{D}\rho v\,|v|$$
$$\frac{\partial(\rho \frac{V}{A})}{\partial t} + v^2\frac{\partial \rho}{\partial x} + 2\rho v\frac{\partial v}{\partial x} + \frac{\partial p}{\partial x} = -\frac{\lambda}{2D}\rho v\,|v|$$
$$\frac{1}{A}\frac{\partial f}{\partial t} + \frac{c^4}{A^2c^2}\frac{f^2}{p^2}\frac{\partial p}{\partial x} + \frac{2c^2 f}{A^2}\frac{\partial(\frac{f}{p})}{\partial x} + \frac{\partial p}{\partial x} = -\frac{\lambda}{2D}\frac{c^2 f}{A^2 p}\,|f|$$

Note that : $\frac{\partial(\frac{f}{p})}{\partial x} = \frac{1}{p}\frac{\partial f}{\partial x} - \frac{f}{p^2}\frac{\partial p}{\partial x}$

$$\frac{1}{A}\frac{\partial f}{\partial t} + \frac{2c^2 f}{A^2 p}\frac{\partial f}{\partial x} - \frac{c^2 f^2}{A^2 p^2}\frac{\partial p}{\partial x} + \frac{\partial p}{\partial x} = -\frac{\lambda}{2D}\frac{c^2 f}{A^2 p}\,|f|$$

We thus obtain the final form:

$$\frac{\partial p(t,x)}{\partial t} + \frac{c^2}{A}\frac{\partial f(t,x)}{\partial x} = 0$$

$$\frac{\partial f(t,x)}{\partial t} + \frac{2c^2 f(t,x)}{Ap(t,x)}\frac{\partial f(t,x)}{\partial x} - \frac{c^2 f(t,x)^2}{Ap(t,x)^2}\frac{\partial p(t,x)}{\partial x} + \frac{A\partial p(t,x)}{\partial x} = -\frac{8c^2\lambda A}{\pi^2 D^5}\frac{f(t,x)|f(t,x)|}{p(t,x)}$$

Gas Compression Optimization

Euler Equations

$$\frac{\partial p(t,x)}{\partial t} + \frac{c^2}{A}\frac{\partial f(t,x)}{\partial x} = 0$$

$$\frac{\partial f(t,x)}{\partial t} + \frac{2c^2 f(t,x)}{Ap(t,x)}\frac{\partial f(t,x)}{\partial x} - \frac{c^2 f(t,x)^2}{Ap(t,x)^2}\frac{\partial p(t,x)}{\partial x} + \frac{A\partial p(t,x)}{\partial x} = -\frac{8c^2\lambda A}{\pi^2 D^5}\frac{f(t,x)|f(t,x)|}{p(t,x)}$$

Compressor Power

$$P(t) = c_p \cdot T \cdot f_{in}(t) \cdot \left(\left(\frac{p_{in}(t)+\Delta p(t)}{p_{in}(t)}\right)^{\frac{\gamma-1}{\gamma}} - 1\right)$$

Gas Delivery & Target

$$f_{out}(t) = d_{deliver}(t)$$

$$d_{deliver}(t) \leq d_{target}(t)$$

Boundary Conditions

$$p(0,t) = p_{in}(t) + \Delta p(t)$$

$$p(L,t) = p_{out}(t)$$

$$f(0,t) = f_{in}(t)$$

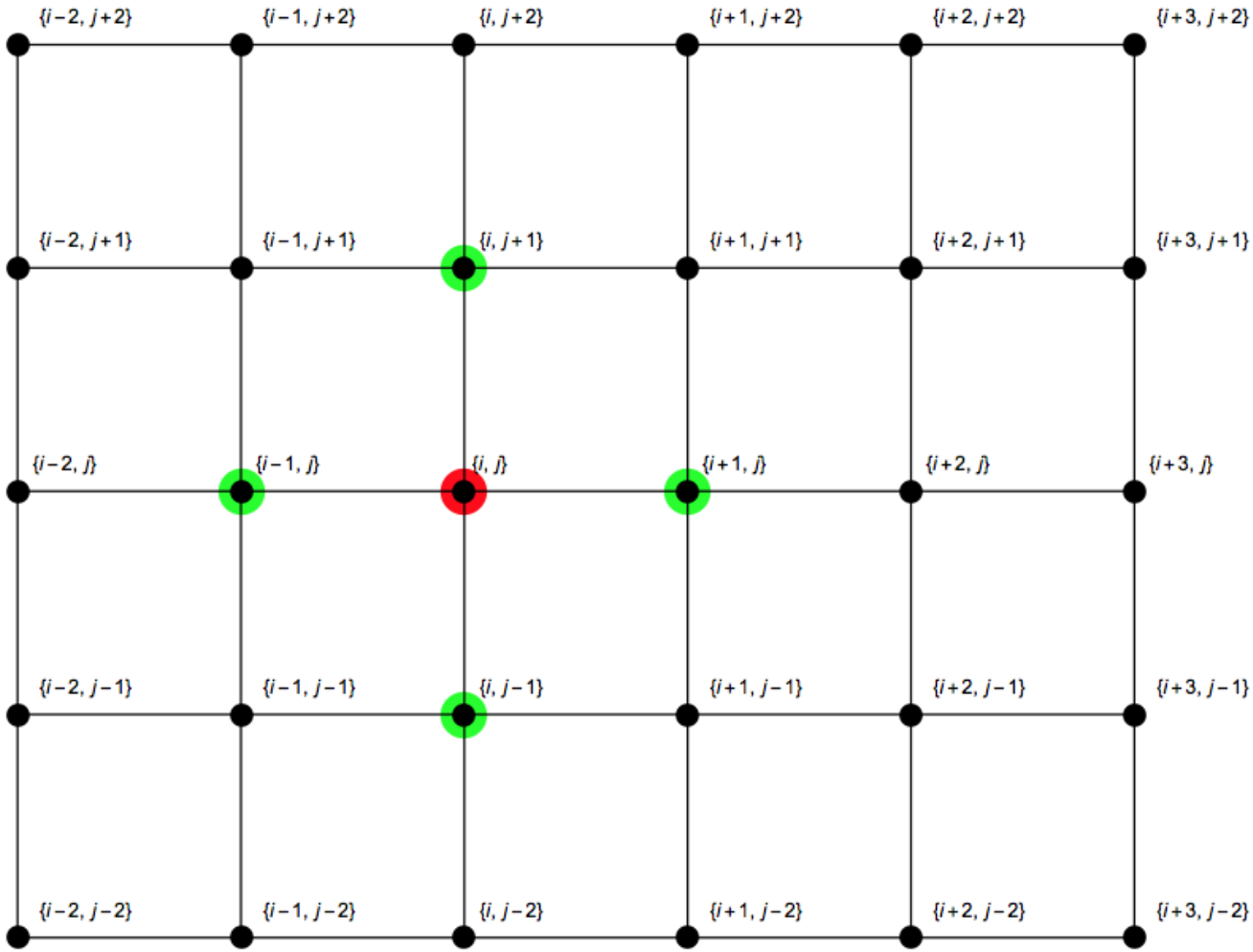
$$f(L,t) = f_{out}(t)$$

Objective: Minimize Compression Cost and Maximize Gas Delivered

$$\min_{\Delta p(\cdot)} \int_0^T \left(\alpha_P P(t) - \alpha_{gas} d_{deliver}(t)\right) dt$$

Space-Time Discretization

Key Idea: Convert PDE equations into set of nonlinear constraints



Space-Time Discretization

Continuous Form

$$\frac{\partial p(t,x)}{\partial t} + \frac{c^2}{A}\frac{\partial f(t,x)}{\partial x} = 0$$

$$\frac{\partial f(t,x)}{\partial t} + \frac{2c^2 f(t,x)}{Ap(t,x)}\frac{\partial f(t,x)}{\partial x} - \frac{c^2 f(t,x)^2}{Ap(t,x)^2}\frac{\partial p(t,x)}{\partial x} + \frac{A\partial p(t,x)}{\partial x} = -\frac{8c^2\lambda A}{\pi^2 D^5}\frac{f(t,x)\,|f(t,x)|}{p(t,x)}$$

Discretized Form (Backward Euler Scheme)

$$\frac{p_{t+1,k} - p_{t,k}}{\Delta t} = -\frac{c^2}{A}\frac{f_{t+1,k+1} - f_{t+1,k}}{\Delta x}, \quad t \in \mathcal{T}, k \in \mathcal{X}$$

$$\frac{f_{t+1,k} - f_{t,k}}{\Delta t} = -\frac{2c^2}{A}\frac{f_{t+1,k}}{p_{t+1,k}}\frac{f_{t+1,k+1} - f_{t+1,k}}{\Delta x} + \frac{c^2}{A}\frac{f_{t+1,k}^2}{p_{t+1,k}^2}\frac{p_{t+1,k+1} - p_{t,k}}{\Delta x} - A\frac{p_{t+1,k+1} - p_{t,k}}{\Delta x}$$
$$-\frac{8c^2\lambda A}{\pi^2 D^5}\frac{f_{t+1,k}\,|f_{t+1,k}|}{p_{t+1,k}}, \quad t \in \mathcal{T}, k \in \mathcal{X}$$

Where $\mathcal{T}$ and $\mathcal{X}$ are sets of discrete time and space points.

Modeling Tricks

- Nonlinear solvers benefit from scaling and reformulations (e.g., $y = x$ vs. $\frac{y}{x} = 1$).
- In the context of the gas compression problem, we can aid the solver by using a *lifting* procedure to the moment equation:

$$s_{1,t,k} p_{t,k} = \frac{8c^2 \lambda A}{\pi^2 D^5} f_{t,k} \left| f_{t,k} \right|$$

$$s_{2,t,k} p_{t,k} = \frac{2c^2}{A} f_{t,k}$$

$$s_{3,t,k} p_{t,k}^2 = A f_{t,x}^2$$

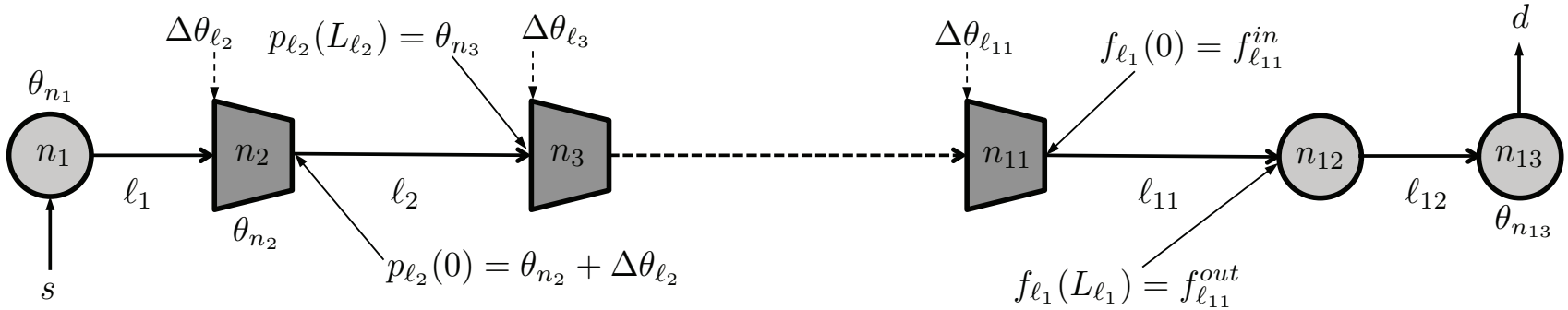
To obtain:

$$\frac{f_{t+1,k} - f_{t,k}}{\Delta t} = -s_{2,t+1,k} \frac{f_{t+1,k+1} - f_{t+1,k}}{\Delta x} + s_{3,t+1,k} \frac{p_{t+1,k+1} - p_{t,k}}{\Delta x} - A \frac{p_{t+1,k+1} - p_{t,k}}{\Delta x} - s_{1,t+1,k}$$

Gas Pipeline Modeling with JuMP

Julia notebook (`gas_pipeline.ipynb`)

Natural Gas Networks



We can easily model a network of pipelines by defining sets of nodes (junctions) and links (pipelines).

- $\mathcal{N}$: Set of nodes (junctions)
- $\mathcal{L}$: Set of links (pipelines)
- $\mathcal{S}$: Set of gas supply flows
- $\mathcal{D}$: Set of gas demand flows
- $\mathcal{L}_a \subseteq \mathcal{L}$: Set of active links (pipelines with compressors)
- $\mathcal{L}_p \subseteq \mathcal{L}$: Set of passive links (pipelines without compressors)

Gas Compression over Network

The equations are analogous to the single pipeline case but we now need to replicate pipelines and impose conservation at the junctions.

Mass and Momentum Balances

$$\frac{\partial p_\ell(t,x)}{\partial t} + \frac{c^2}{A_\ell}\frac{\partial f_\ell(t,x)}{\partial x} = 0, \quad \ell \in \mathcal{L}$$

$$\frac{\partial f_\ell(t,x)}{\partial t} + \frac{2c^2 f_\ell(t,x)}{A_\ell p_\ell(t,x)}\frac{\partial f_\ell(t,x)}{\partial x} - \frac{c^2 f_\ell(t,x)^2}{A_\ell p_\ell(t,x)^2}\frac{\partial p_\ell(t,x)}{\partial x} + A_\ell \frac{\partial p_\ell(t,x)}{\partial x} = -\frac{8c^2\lambda A_\ell}{\pi^2 D_\ell^5}\frac{f_\ell(t,x)}{p_\ell(t,x)}\left|f_\ell(t,x)\right|, \quad \ell \in \mathcal{L}$$

Compressor Power

$$P_\ell(t) = c_p \cdot T \cdot f_{in,\ell}(t) \cdot \left(\left(\frac{p_{in,\ell}(t)+\Delta p_\ell(t)}{p_{in,\ell}(t)}\right)^{\frac{\gamma-1}{\gamma}} - 1\right), \ell \in \mathcal{L}_a$$

Boundary Conditions

$$p_\ell(0,t) = p_{in,\ell}(t) + \Delta p_\ell(t), \ell \in \mathcal{L}_a$$

$$p_\ell(0,t) = p_{in,\ell}(t), \ell \in \mathcal{L}_p$$

$$p_\ell(L_\ell,t) = p_{out,\ell}(t), \ell \in \mathcal{L}$$

Node Conservation

$$\sum_{\ell \in \mathcal{L}_n^{rec}} f_{out,\ell}(t) - \sum_{\ell \in \mathcal{L}_n^{snd}} f_{in,\ell}(t) + \sum_{i \in \mathcal{S}_n} g_i(t) - \sum_{j \in \mathcal{D}_n} d_j^{deliver}(t) = 0, \ n \in \mathcal{N}$$

Objective: Minimize Compression Cost and Maximize Gas Delivered

$$\min_{\Delta p_\ell(\cdot)} \int_0^T \left(\sum_{\ell \in \mathcal{L}_a} \alpha_P P_\ell(t) - \sum_{j \in \mathcal{D}} \alpha_{gas} d_j^{deliver}(t)\right) dt$$

Gas Networks - Modeling with JuMP

Julia notebook (`pipeline_network.ipynb`)

Plasmo.jl - What is it?

Platform for **S**calable **M**odeling and **O**ptimization

A Graph-based modeling and optimization framework

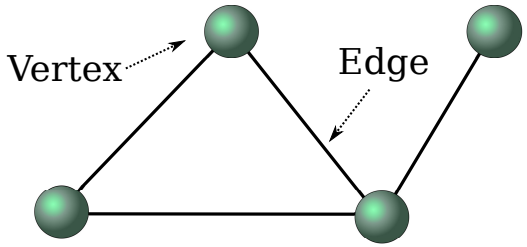
Key Features:

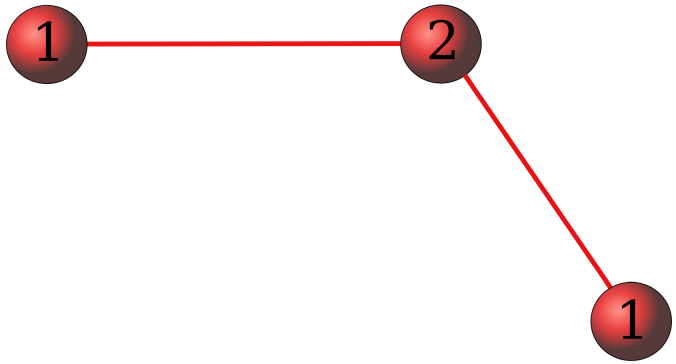
- Component models associated with nodes **and** edges
- Facilitates construction of hierarchical graphs (uses subgraphs)
- Modularization of component models
- Manipulate graph structure for solver interface
- Ease of modeling complex systems

Relevant Graph Concepts

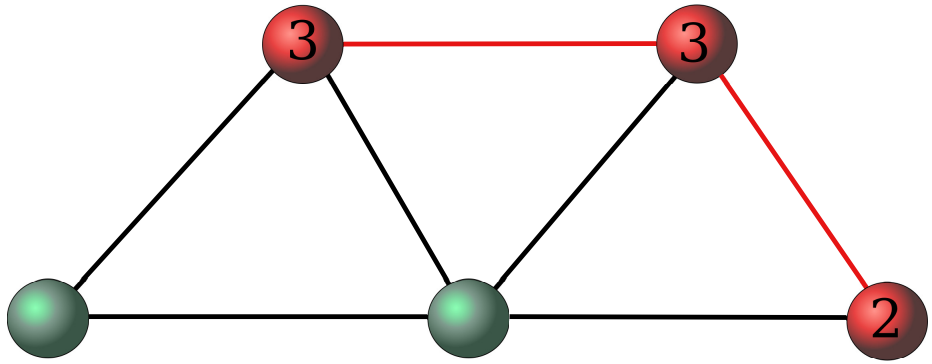
Graph Definition

A graph (G) is a finite set V(G) of vertices (nodes) and a finite family E(G) of pairs of elements of V(G) called edges





A

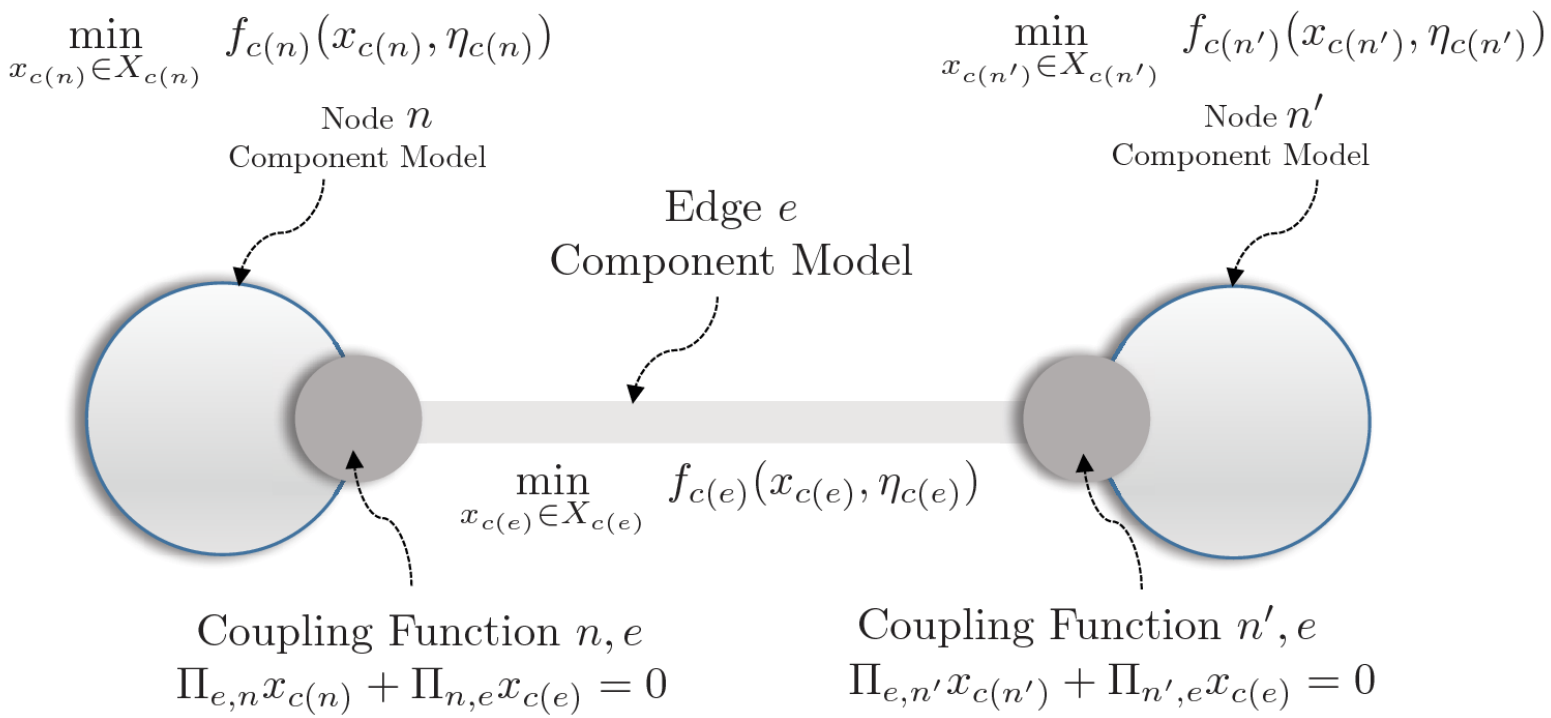


B

- A is a subgraph of B
- The degree of a node is specific to its graph

Graph Based Modeling

Plasmo associates model components with nodes and edges



Plasmo Syntax

```
using Plasmo
using Ipopt
#Create a graph model
m = GraphModel(solver = IpoptSolver())
graph = getgraph(m)
n1 = add_node!(m)
n2 = add_node!(m)
edge = add_edge!(m,n1,n2)
#Set component models
setmodel!(n1,node1_model())
setmodel!(n2,node2_model())
#link the two node models using the edge
@linkconstraint(m,graph,n1,n1[:x] == n2[:x])
solve(m,graph)
```

$$\min_{x,y} \; x$$
$$s.t.\; 0 \leq x \leq 5$$
$$0 \leq y \leq 5$$
$$x + y \geq 6$$

$$\max_{x,y} \; y$$
$$s.t.\; x \geq 0$$
$$0 \leq y \leq 5$$
$$e^x + y \leq 7$$

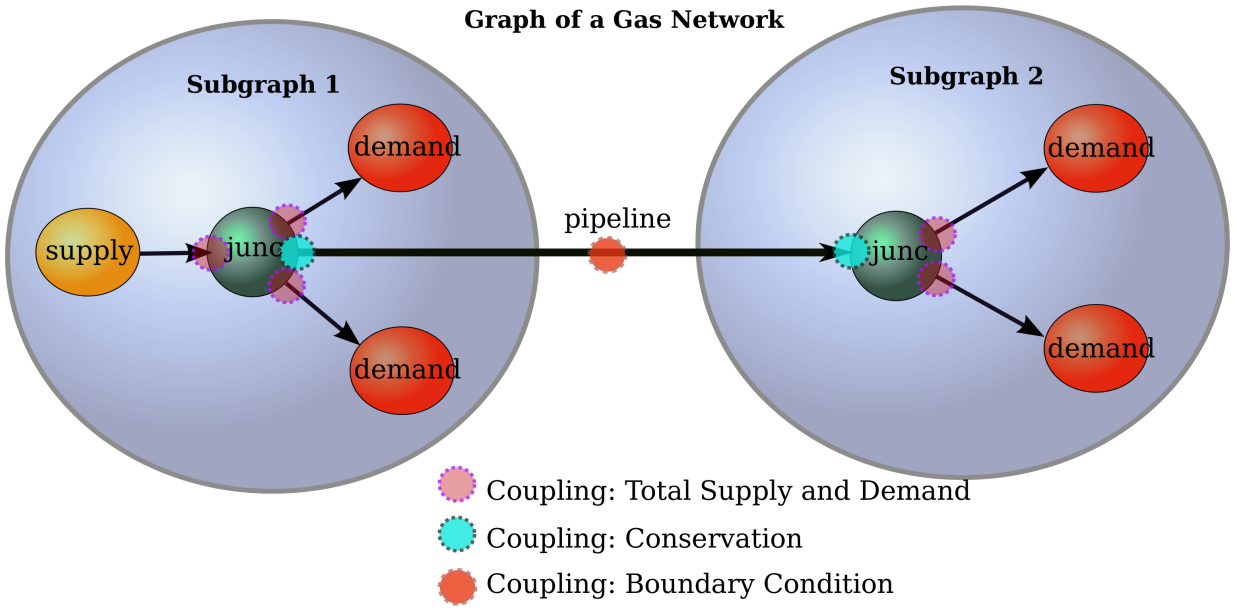
$$m1.x == m2.x$$

Plasmo Notebooks

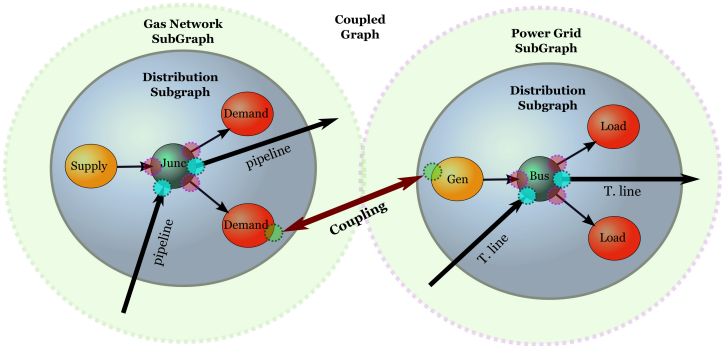
- A simple Plasmo example coupling two models
  - Julia notebook (simple_plasmo_model.ipynb)
- Modeling our pipeline example with Plasmo
  - Julia notebook (plasmo_pipeline_model.ipynb)
- Modeling the pipeline network with Plasmo
  - Julia notebook (plasmo_network_model.ipynb)

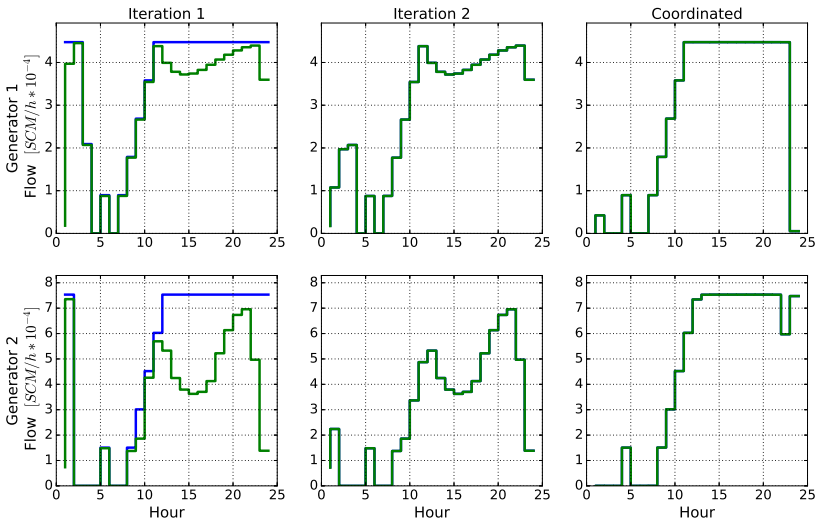
Graph Based Modeling - Even More Modular

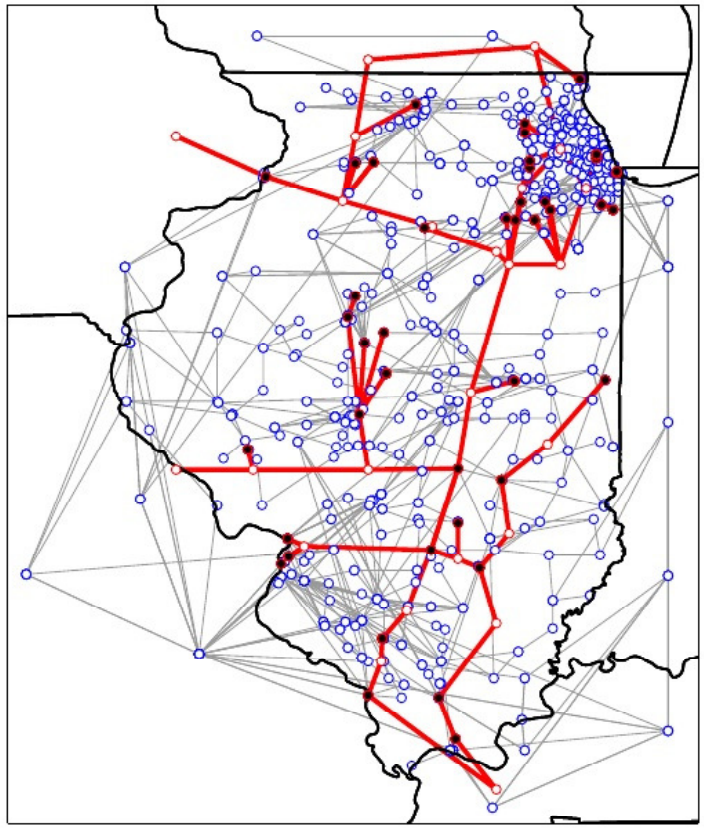
- The subgraph abstraction allows multiple couplings on the same node
- This can be used to build modular systems and couple them at higher levels!



Plasmo Applications - Coupled Infrastructure

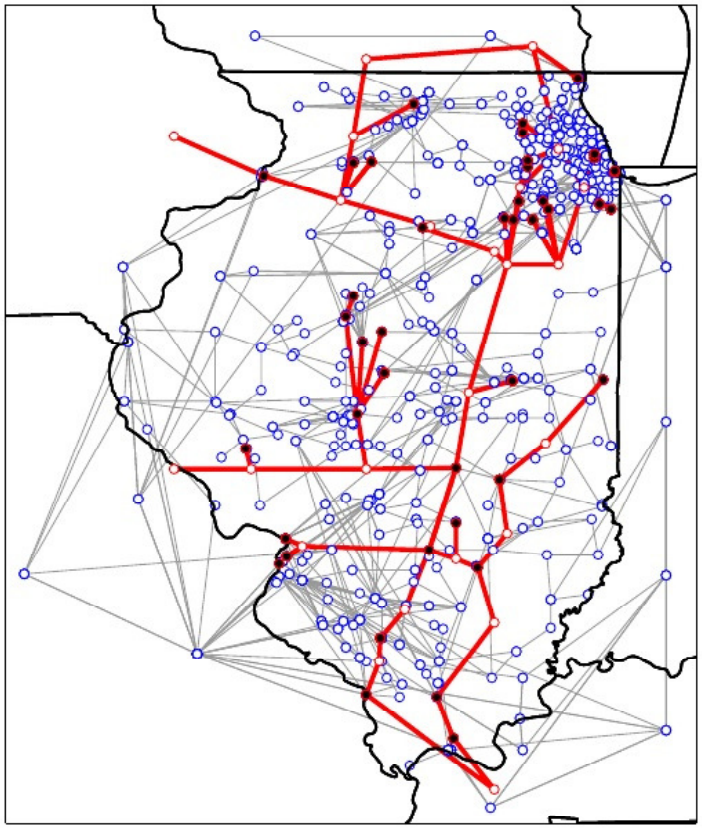






Plasmo Applications - Coupled Infrastructure

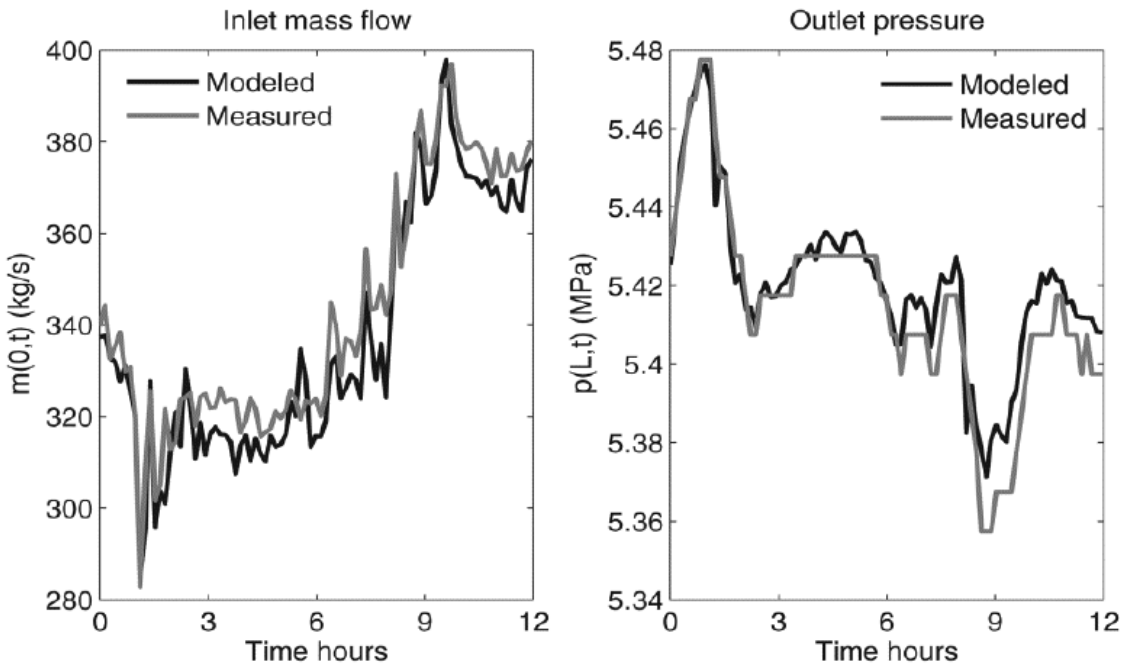
Coupling infrastructures can be done by introducing new graphs and embedding subgraphs (infrastructures).



```
m = GraphModel()
graph = getgraph(m)
add_subgraph!(graph,power_network)
add_subgraph!(graph,gas_network)
generator = getnode(power_network,:gen)
demand = getnode(gas_network,:demand)
link = add_edge!(graph,generator,demand)
@linkconstraint(link,getconnectedfrom(graph,link)[:Pgend] ==
getconnectedto(graph,link)[:fdemand])
@linkconstraint(link,getconnectedfrom(graph,link)[:Pgend] <=
getconnectedto(graph,link)[:fdeliver])
```

Plasmo Applications - Data Assimilation

Idea: Use flow and pressure measurements at pipeline junctions to estimate the state of the entire system



Note: We use the same system model, but pose a different objective function

Plasmo Applications - Data Assimilation

State evolution

$$x^+ = f(x, w)$$
$$y = h(x) + v$$

Merit function

$$V_T(x(0), \mathbf{w}) = \ell_x(x(0) - \overline{x_0}) + \sum_{i=0}^{T-1} \ell_i(w(i), v(i))$$

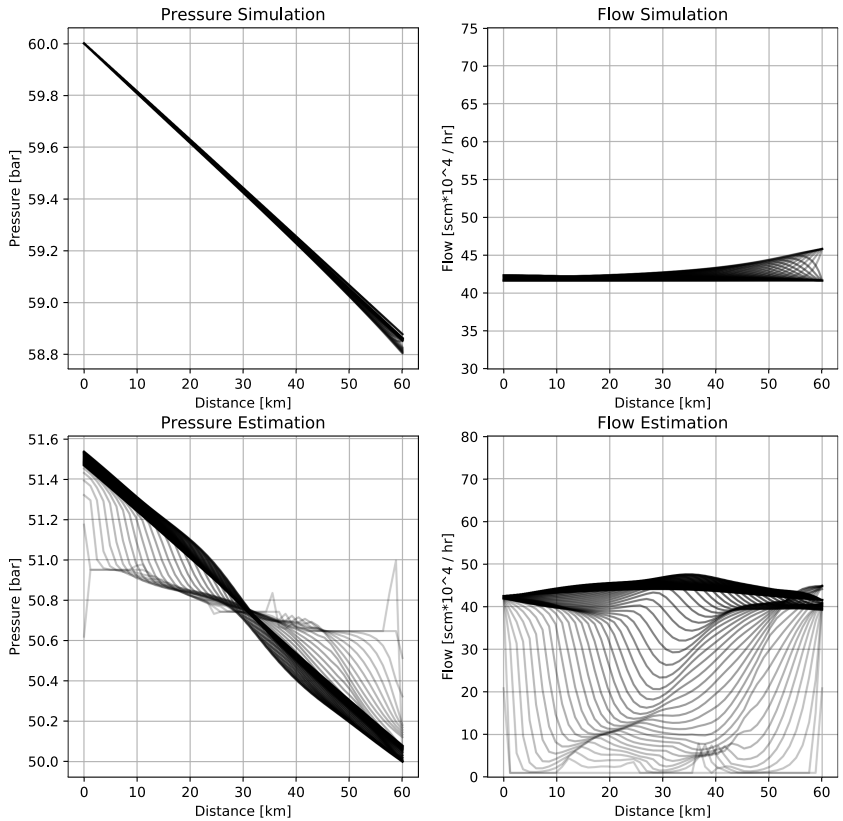
Full information estimator

$$\min_{x(0), \mathbf{w}} \; V_T(x(0), \mathbf{w})$$
$$s.t. \;\; x^+ = f(x, w)$$
$$y = h(x) + v$$

Data Assimilation - Bad Prior

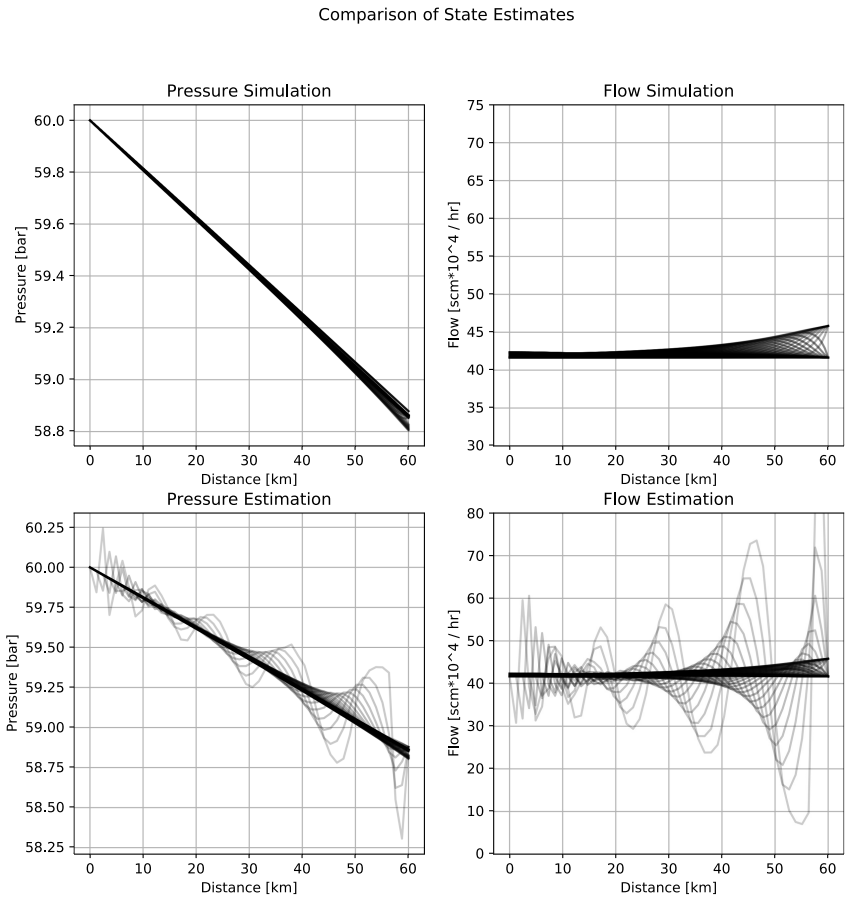
$$\min \sum_{\ell \in \mathcal{L}} \sum_{k=0}^{N_x} \Big(\big((p_{\ell,0,k} - \mathbf{0} + (f_{\ell,0,k} - \mathbf{0}) + \sum_{t=0}^{T-1} \Big(\sum_{n \in \mathcal{N}} (\theta_{n,t} - \overline{\theta_n})^2 + \sum_{\ell \in \mathcal{L}} \Big((f_{\ell,in,t} - \overline{f_{\ell,in,t}})^2 + (f_{\ell,out,t} - \overline{f_{\ell,out,t}})^2 \Big) \Big)$$

Comparison of State Estimates



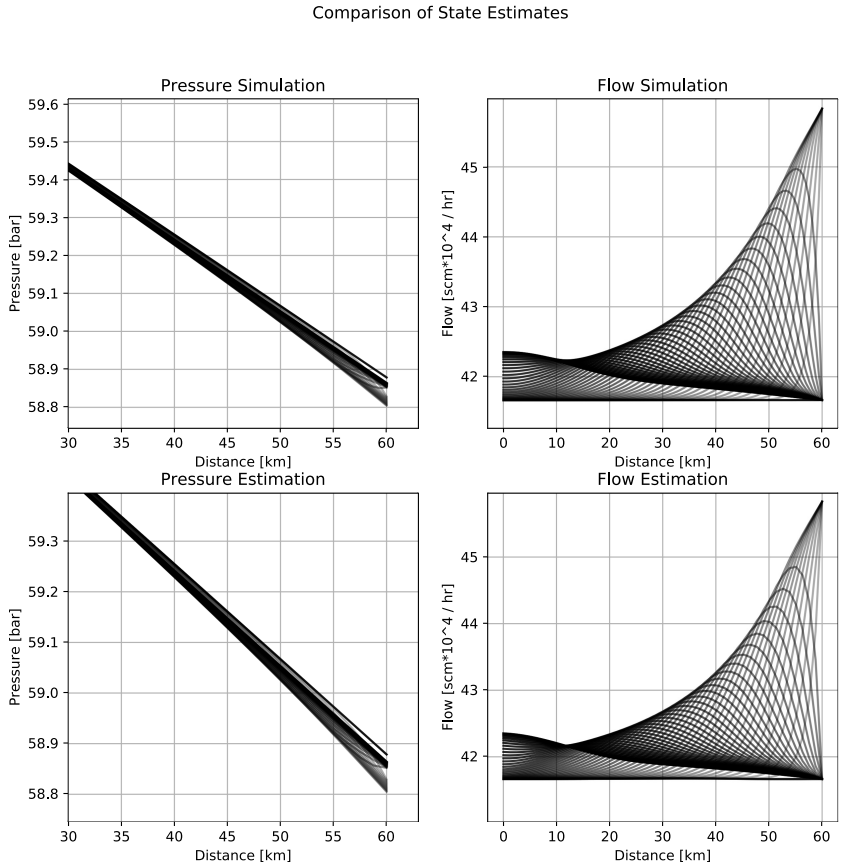
Data Assimilation - Only Measurements

$$\min \sum_{t=0}^{T-1} \left(\sum_{n \in \mathcal{N}} (\theta_{n,t} - \overline{\theta_n})^2 + \sum_{\ell \in \mathcal{L}} \left((f_{\ell,in,t} - \overline{f_{\ell,in,t}})^2 + (f_{\ell,out,t} - \overline{f_{\ell,out,t}})^2 \right) \right)$$

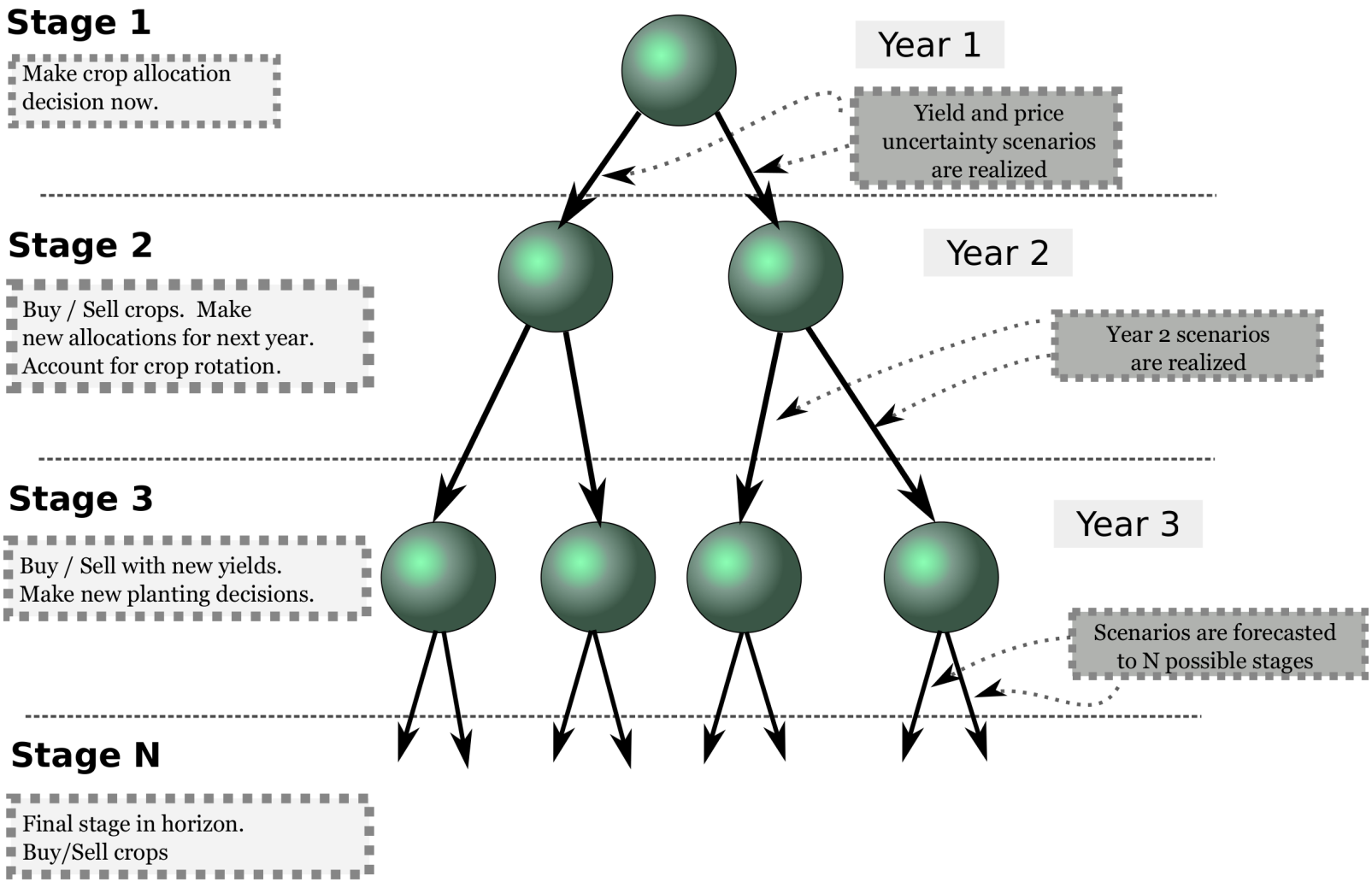


Data Assimilation - Steady State Prior

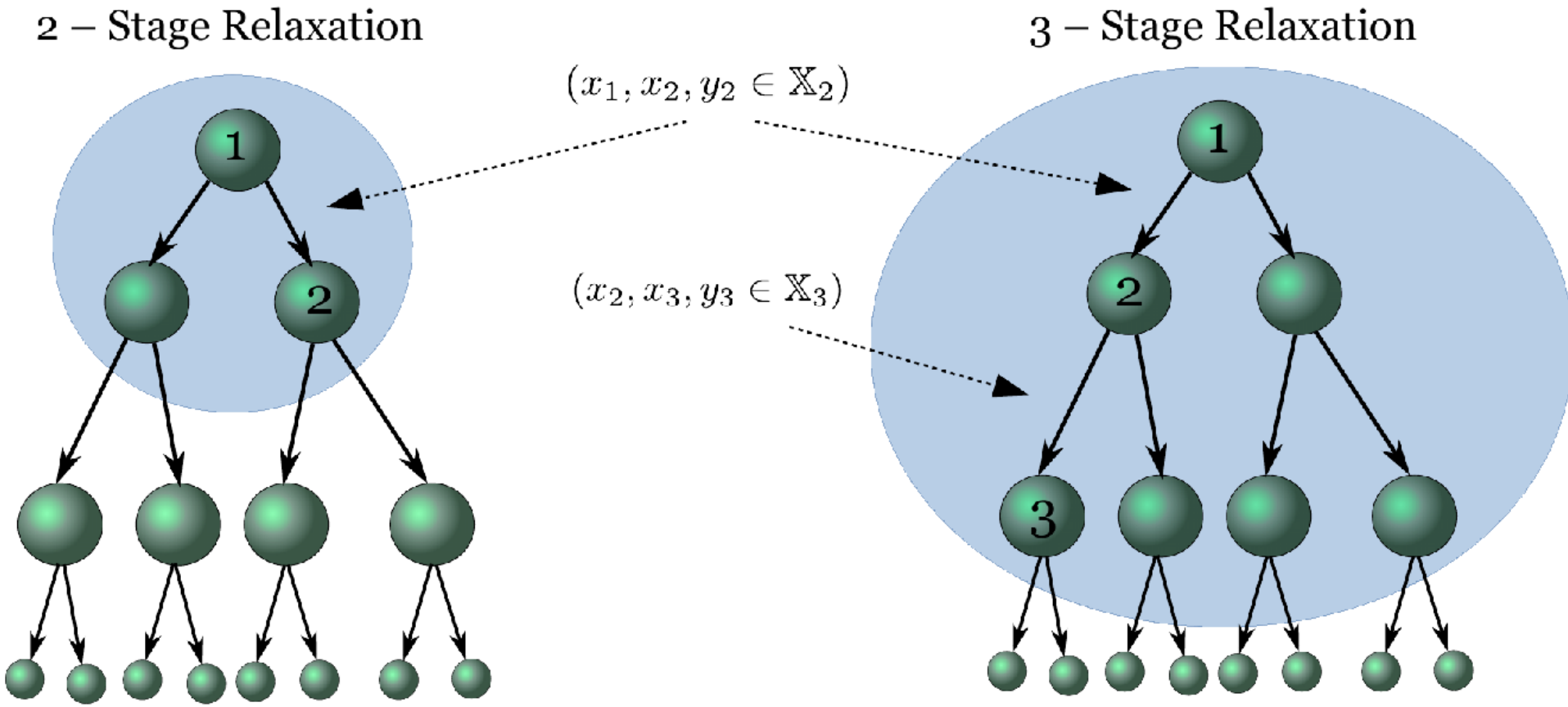
$$\min \sum_{\ell \in \mathcal{L}} \sum_{k=0}^{N_x} \left(\left((p_{\ell,0,k} - p_{ss,\ell,k})^2 + (f_{\ell,0,k} - f_{ss,\ell,k})^2 \right) + \right.$$
$$\left. \sum_{t=0}^{T-1} \left(\sum_{n \in \mathcal{N}} (\theta_{n,t} - \overline{\theta_n})^2 + \sum_{\ell \in \mathcal{L}} \left((f_{\ell,in,t} - \overline{f_{\ell,in,t}})^2 + (f_{\ell,out,t} - \overline{f_{\ell,out,t}})^2 \right) \right) \right.$$



Plasmo Applications - Stochastic Programming



Plasmo Applications - Stochastic Programming



Plasmo Applications - Stochastic Programming

Problem	Expected Cost (MM$)	Solution Time (seconds)
Perfect Information	-6.98	–
2 - Stage Relaxation	-4.29	0.00366
3 - Stage Relaxation	-3.99	0.0927
4 - Stage Relaxation	-3.99	1.68
5 - Stage Program	-2.93	11.7