Introducing Plasmo.jl

A Package for Graph-Based Modeling using JuMP

Jordan Jalving and Victor Zavala

Department of Chemical and Biological Engineering
University of Wisconsin-Madison

JuMP Developers Meetup
June 13th, 2017

Plasmo.jl - What is it?

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

A Graph-based modeling and optimization framework

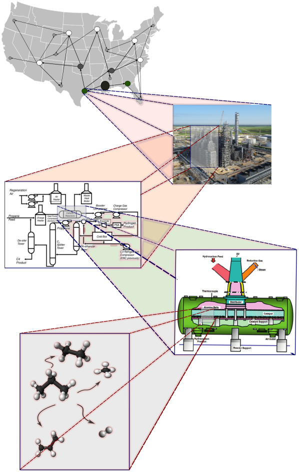
Key Features:

- 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

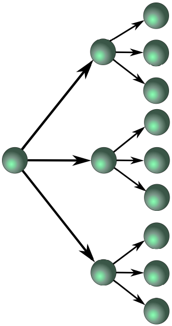
Overview

- Motivation - Complex systems
- Modeling Systems with Components (Graphs)
- Applications
- Design considerations
- Goals right now

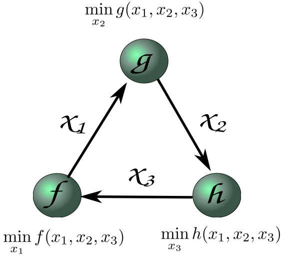
Group Research Theme:Complex Systems



Multi-scale systems



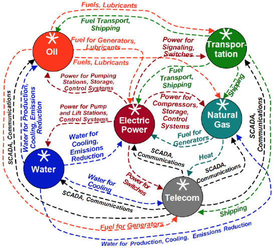
Multi-stage stochastic programs

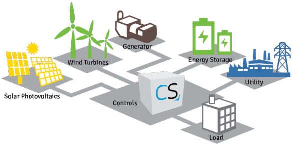


Asynchronous systems

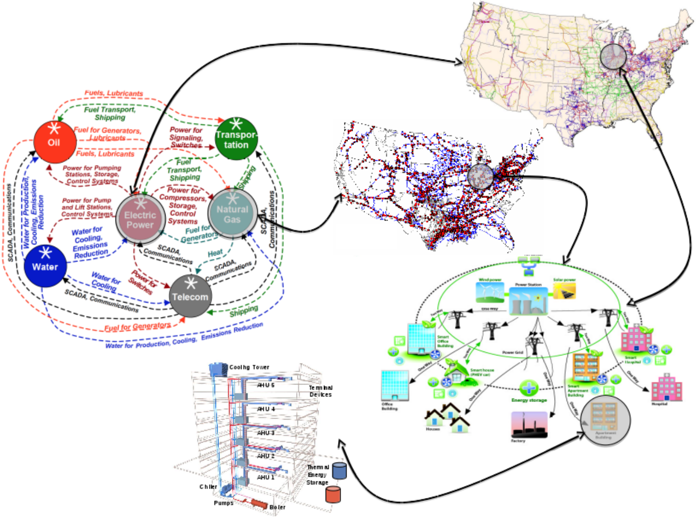
Types of Problems

- Nonlinear (nonconvex) optimization
- Stochastic programming
- Model predictive control
- Some Applications
  - Enery storage systems
  - Connected infrastructure
  - Microgrids

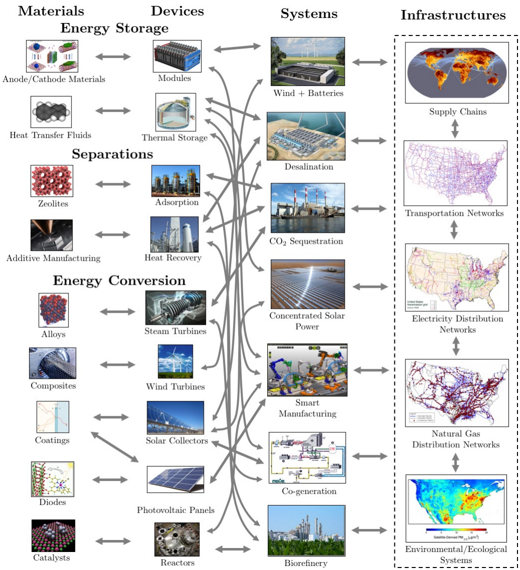




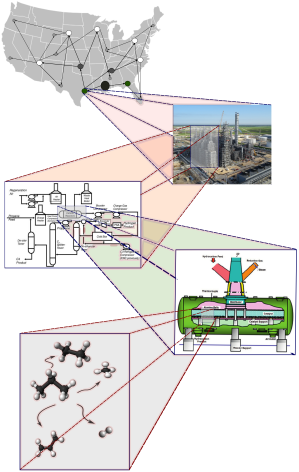
Hierarchical Networks



Technology Landscapes



Challenges with complex physical systems



- Millions of constraints and variables can make the computation intractable
  - Generally apply ad-hoc methods to perform some model reduction
- Millions of system connections makes model instantiation non-trivial
  - Multiple scenarios
  - Solution inspection
- Modeling asynchronicity in large communicating systems is non trivial
  - Decentralized control

Some existing modeling frameworks

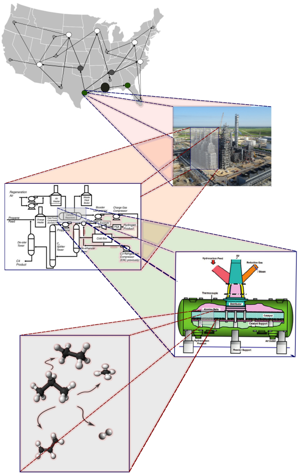
- Modelica
  - Components, hierarchies, architectures (highly abstracted)
  - Designed for simulation (Optimica extension does some optimization)
  - Write connectors for coupling (I always found this difficult)
- gProms
  - Equation oriented chemical flowsheeting software
  - Custom modeling language
  - Commercial

Revisiting our Goals

- Model encapsulation
- Modularity and reuse
- Navigate solutions to complex optimization problems
- Facilitate modeling of communicating systems

The Power of Abstraction - Graphs

Multiscale Physical Systems



Hierarchical Graphs

Supply Chain Network

(A **Graph** with Production **Nodes** and Transport **Edges**)

Polypropylene Plant

(A **Node** with a Process Unit **Subgraph**)

PDH Process Flowsheet

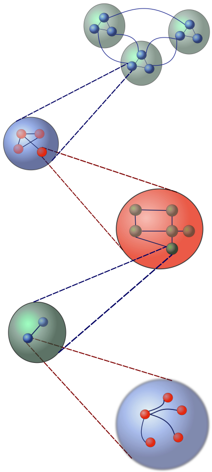
(A **Graph** with Unit Operation **Nodes**)

Reactor

(A **Node** with a Physical Model **Subgraph**)

Reaction Network

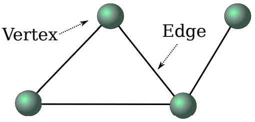
(A **Graph** with Chemical Species **Nodes** and Reaction **Edges**)

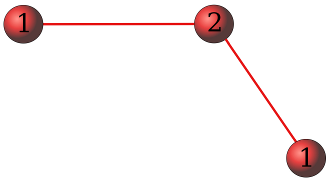


Relevant Graph Concepts

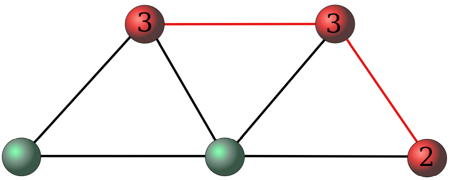
Graph Definition

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

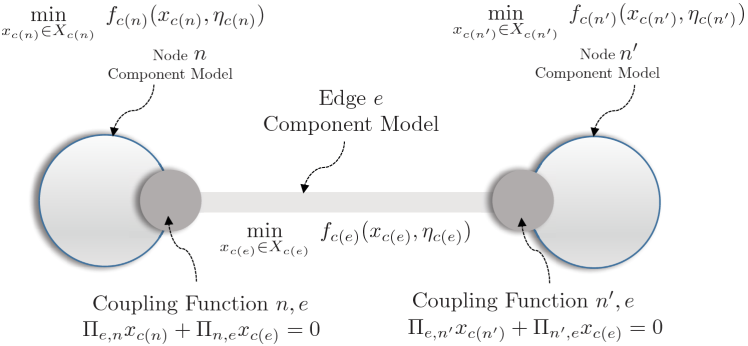


B

- 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

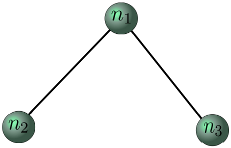


Plasmo.jl - Key Features

Key Features (some in progress):

- Associates models (JuMP Models) and linking information (constraints) with nodes and edges within a graph
- Exploits a subgraph abstraction to enable hierarchies of models (multiple graphs defined on a set of nodes)
- Uses LightGraphs.jl as the graph backend
- Accesses model information on nodes and edges
- Provides interfaces with structured solvers (PIPS, etc...)

Plasmo - Old syntax (still supported)



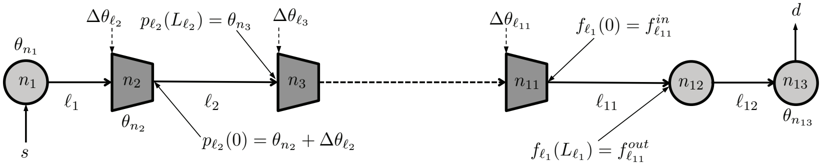

```
using Plasmo
graph = PlasmoGraph()    #Create a graph
n1 = add_node!(graph)
n2 = add_node!(graph)
n3 = add_node!(graph)
edge1 = add_edge!(graph,n1,n2)
edge2 = add_edge!(graph,n1,n3)
#Set component models
setmodel!(n1,simple_model())
setmodel!(n2,simple_model())
setmodel!(n3,simple_model())
#provide linking information
setcouplingfunction!(graph,edge1,couplenodes)
setcouplingfunction!(graph,edge2,couplenodes)
model = generate_model(graph)
setsolver(model,IpoptSolver())
solve(model)

function couplenodes(m::Model,graph,edge)
    @constraint(m,getconnectedfrom(graph,edge)[:x]
    == getconnectedto(graph,edge[:x]))
end
```

New Syntax - In Progress

```julia
using Plasmo
#Create a graph
m = GraphModel(solver = IpoptSolver())
n1 = add_node!(m)
n2 = add_node!(m)
n3 = add_node!(m)
edge1 = add_edge!(m,n1,n2)
edge2 = add_edge!(m,n1,n3)
#Set component models
setmodel!(n1,simple_model())
setmodel!(n2,simple_model())
setmodel!(n3,simple_model())
#link the two models
@linkconstraint(edge1,getconnectedfrom(m,edge1)[:x]
== getconnectedto(edge1)[:x])
@linkconstraint(edge2,getconnectedfrom(m,edge2)[:x]
== getconnectedto(edge2)[:x])
solve(m)  #solve with Ipopt
#solve_pips(m,n1,[n2,n3])  #solve with PIPS NLP
```

Gas Networks



- $\mathcal{N}$: Set of nodes in the gas network (junctions)
- $\mathcal{L}$: Set of links (pipelines)
- $\mathcal{S} \subseteq \mathcal{N}$: Set of gas supplies
- $\mathcal{D} \subseteq \mathcal{N}$: Set of gas demands
- $\mathcal{L}_a \subseteq \mathcal{L}$: Set of active links (pipelines with compressors)
- $\mathcal{L}_p \subseteq \mathcal{L}$: Set of passive links (pipelines without compressors)

Gas Networks

Mass and Momentum Balances on a Network

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

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

Compressor Power

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

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

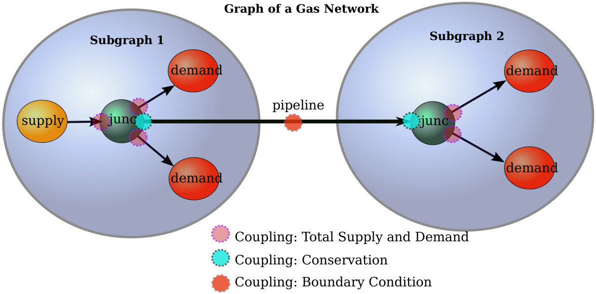
$$\sum_{\ell \in \mathcal{L}_n^{rec}} f_{out,\ell}(t) - \sum_{\ell \in \mathcal{L}_n^{snd}} f_{in,\ell}(t) + \sum_{i \in \mathcal{S}_n} g_i(t) - \sum_{j \in \mathcal{D}_n} d_j^{gas}(t) = 0, \; n \in \mathcal{N}$$

Supply and Demand

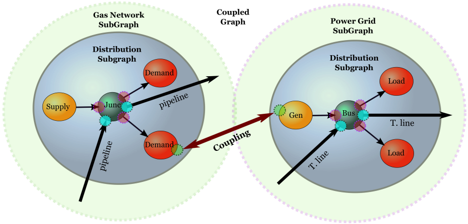
$f_{deliver,n}(t) \leq f_{demand,n}(t), n \in \mathcal{D}$

Graph Based Modeling - Gas Networks

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



Graph Based Modeling - Coupled Networks



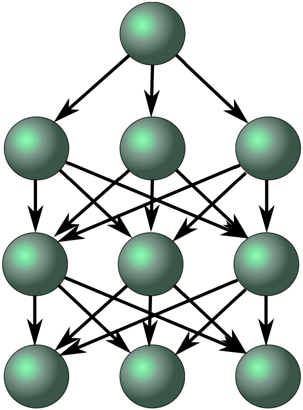

```
m = GraphModel()
graph = getgraph(m)
add_subgraph!(graph,power_network)
add_subgraph!(graph,gas_network)
generator = getnode(power_network,:gen)
demand = getnode(gas_network,:demand)
link = add_edge!(graph,generator,demand)
@linkconstraint(link,getconnectedfrom(graph,link)[:Pgend] <=
getconnectedto(graph,link)[:fdeliver])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

Graph Based Modeling - Coupled Networks

Key Findings:

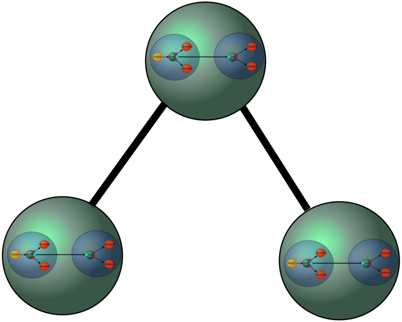
- Infrastructure models (graphs) can be developed independently and coupled within larger systems (graphs)
- Illinois Case Study: 7% more gas delivered to generators; 27% revenue increase versus uncoordinated case
- Uncoordinated case simulated by solving successive optimization problems

Multistage Stochastic Programming



- Our graph abstraction corresponds to the node-based abstraction in multistage stochastic programming
- Component models associated within nodes (scenarios)
- Link constraints propogate transition from stage to stage

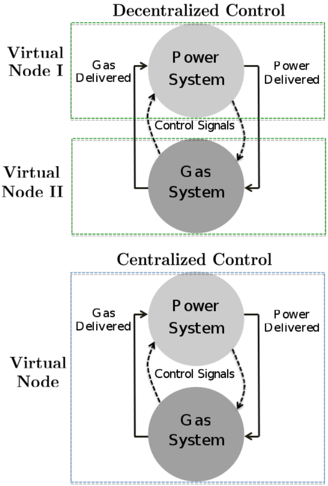
Embedding Graph Models



- Graph models can themselves be embedded as models in nodes or edges
- Simplifies construction of multiple layers in systems

Future Direction

- Generalize the model interface if possible (strictly uses JuMP)
- Find suitable abstraction for computational workflows
  - Decentralized control
  - Algorithmic strategies (e.g. scheduling and operations)
  - Graph partitioning and model reduction
  - Initialization strategies
- Simulation interfaces



Goals right now

- Finalize physical model abstraction
- Push first version to github
- Figure out a suitable graph communication abstraction