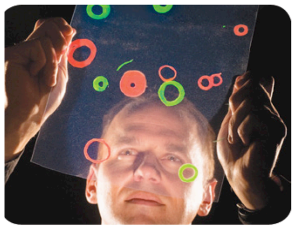
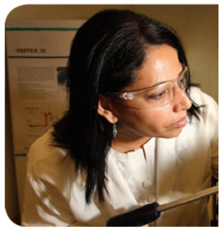



Sandia National Laboratories

Exceptional service in the national interest

11. Generalized Disjunctive Programming

PYOMO

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

Disjunctive programs

- Disjunctions: selectively enforce sets of constraints
  - Sequencing decisions: x ends before y or y ends before x
  - Switching decisions: a process unit is built or not
  - Alternative selection: selecting from a set of pricing policies

- Implementation
  - `Disjunct`:
    - Block of Pyomo components
      - (Var, Param, Constraint, etc.)
    - Boolean (binary) indicator variable determines if block is enforced
  - `Disjunction`:
    - Enforces logical XOR across a set of Disjunct indicator variables
  - (Logic constraints on indicator variables)

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

Example: Task sequencing

- Prevent tasks colliding on a single piece of equipment
  - Derived from Raman & Grossmann (1994)
  - Given:
    - Tasks I processed on a sequence of machines (with no waiting)
    - Task i starts processing at time t_i with duration τ_{im} on machine m
    - $J(i)$ is the set of machines used by task i
    - C_{ik} is the set of machines used by both tasks i and j

$$\begin{bmatrix} Y_{ik} \\ t_i + \sum_{\substack{m \in J(i) \\ m \leq j}} \tau_{im} \leq t_k + \sum_{\substack{m \in J(k) \\ m < j}} \tau_{km} \end{bmatrix} \vee \begin{bmatrix} Y_{ki} \\ t_k + \sum_{\substack{m \in J(k) \\ m \leq j}} \tau_{km} \leq t_i + \sum_{\substack{m \in J(i) \\ m < j}} \tau_{im} \end{bmatrix}$$

$$\forall j \in C_{ik}, \forall i, k \in I, i < k$$

Example: Task sequencing in Pyomo

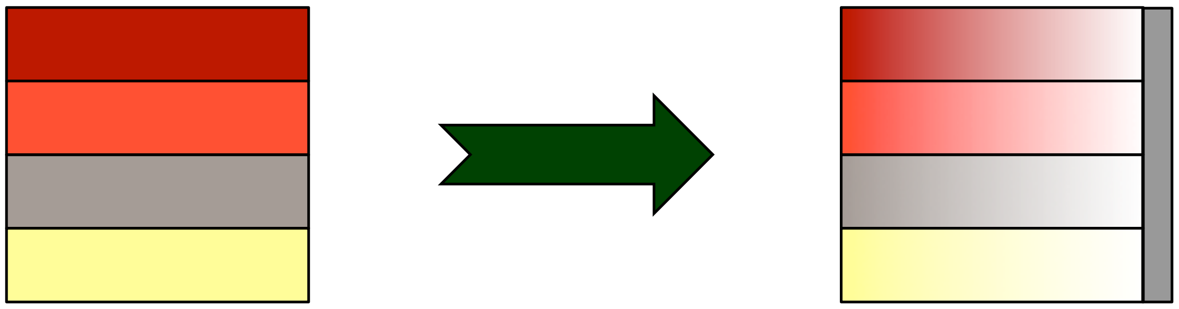
```
from pyomo.dae import *
def _NoCollision(disjunct, i, k, j, ik):
    model = disjunct.model()
    lhs = model.t[i] + sum(model.tau[i,m] for m in model.STAGES if m<j)
    rhs = model.t[k] + sum(model.tau[k,m] for m in model.STAGES if m<j)
    if ik:
        disjunct.c = Constraint( expr= lhs + model.tau[i,j] <= rhs )
    else:
        disjunct.c = Constraint( expr= rhs + model.tau[k,j] <= lhs )
model.NoCollision = Disjunct( model.L, [0,1], rule=_NoCollision )

def _setSequence(model, i, k, j):
    return [ model.NoCollision[i,k,j,ik] for ik in [0,1] ]
model.setSequence = Disjunction(model.L, rule=_setSequence)
```

$$\left[t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} + \overset{Y_{ik}}{\tau_{ij}} \le t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} \right] \vee \left[t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} + \overset{Y_{ki}}{\tau_{kj}} \le t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} \right]$$

$$\forall j \in C_{ik}, \forall i,k \in I, i<k$$

Solving disjunctive models

- Few solvers “understand” disjunctive models
  - *Transform* model into standard math program
  - Big-M relaxation:
    - Convert logic variables to binary
    - Split equality constraints in disjuncts into pairs of inequality constraints
    - Relax all constraints in the disjuncts with “appropriate” M values



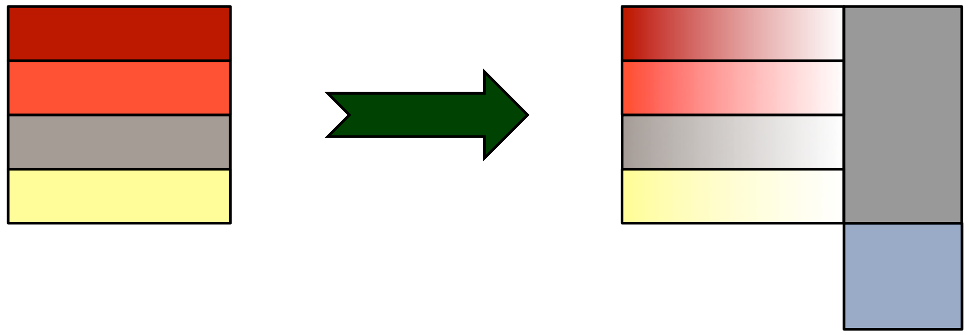
```
pyomo solve --solver cbc --transform=gdp.bigm jobshop.py jobshop.dat
```

Why is the transformation interesting?

- Model preserves explicit disjunctive structure
- Automated transformation reduces errors
- Automatically identifies appropriate M values (for bounded linear)

Why is the transformation interesting?

- Model preserves explicit disjunctive structure
- Automated transformation reduces errors
- Automatically identifies appropriate M values (for bounded linear)
- Big-M is not the only way to relax a disjunction!
  - Convex hull transformation (Balas, 1985; Lee and Grossmann, 2000)



```
pyomo solve --solver cbc --transform=gdp.chull jobshop.py jobshop.dat
```

  - Algorithmic approaches
    - e.g., Trespalacios and Grossmann (submitted 2013)
  - Prematurely choosing one relaxation makes trying others difficult