Advances in Model-Based Optimization with AMPL

Robert Fourer, Marcos Dominguez Velad

`{4er,marcos}@ampl.com`

AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

23rd Conference of the
International Federation of Operational Research Societies

Santiago, Chile — 10-14 July 2023

Software for Optimization Stream, Session TB-6

Advances in Model-Based Optimization with AMPL

The ideal of model-based optimization is to describe your problem the way you think about it, while computers do the work of getting and reporting solutions. Recent enhancements aim to bring the AMPL modeling language and system closer to this ideal, on two fronts.

First, modeling language extensions are enabling more natural expressions to be used directly in AMPL model formulations. Conversions to the forms required by large-scale solvers are handled automatically, with support from a new C++ AMPL-solver interface library that adapts to handle diverse solver requirements. The new extensions include general quadratic expressions, numerous logical operators and constraints, and common near-linear and nonlinear functions, all combinable with familiar algebraic expressions.

Second, modeling system extensions are letting AMPL fit more naturally into the increasingly popular Python application programming environment: installing as an “amplpy” Python package, importing and exporting data naturally from/to Python data structures and Pandas dataframes, and supporting Jupyter notebooks that mix AMPL modeling and Python programming. In contrast to Python-only modeling solutions, AMPL’s Python API offers straightforward, efficient model processing while leveraging Python’s vast ecosystem for data pre-processing, solution analysis, and visualization. Examples include free prototyping in Google Colab and fast web app development using Streamlit.

Outline

Modeling more naturally

- New *open source* solver interface
- Logical and other “not linear” expressions
- Support for many mixed-integer (MIP) solvers

Using AMPL everywhere

- Python API
- Jupyter notebooks
- NEOS with Kestrel
- Docker containers
- Streamlit visualizations

Formulating Models More Like You Think About Them

Describe an optimization problem

- ❖ In a form *you find natural or convenient*
- ❖ Using readily recognized expressions

Send it to a solver

- ❖ In a form *the solver will accept*
- ❖ Relying on the modeling software to translate

Get back a result

- ❖ In the form you originally used

New Solver Interface Library (MP)

Design

- C++ library for building efficient, configurable solver drivers
- Substitutes for AMPL's C interface library (ASL)
- *Extensive toolset for problem recognition and transformation*

Motivation . . .

Typical User Complaint

Thank you so much for replying.
Let me show my "if-then" constraint in a more clear way as follows:

```
set veh := {1..16 by 1};

param veh_ind {veh};
param theory_time {veh};
param UP := 400000;

var in_lane_veh {veh} integer >=1, <=2;
var in_in_time {veh} >=0, <=UP;
```

Note that "in_lane_veh {veh}" are integer variables which equal 1 or 2,
and "in_in_time {veh}" are continuous variables.

```
subject to IfConstr {i in 1..card(veh)-1, j in i+1..card(veh):
  veh_ind[i] = veh_ind[j] and theory_time[i] <= theory_time[j]}:

    in_lane_veh[i] = in_lane_veh[j]  ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

When I run my program, there appears the following statement:

```
CPLEX 20.1.0.0: logical constraint _slogcon[1] is not an indicator constraint.
```

Typical Reply

To reformulate this model in a way that your MIP solver would accept,
you could define some more binary variables,

```
var in_lane_same {veh,veh} binary;
```

with the idea that in_lane_same[i,j] should be 1 if and only if in_lane_veh[i] = in_lane_veh[j].
Then the desired relation could be written as two constraints:

```
in_lane_veh[i] = in_lane_veh[j] ==> in_lane_same[i,j] = 1
in_lane_same[i,j] = 1 ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

The second one is an indicator constraint, but you would just need
to replace the first one by equivalent linear constraints.

Given that in_lan_veh can only be either 1 or 2, those constraints could be

```
in_lane_same[i,j] >= 3 - in_lane_veh[i] - in_lane_veh[j]
in_lane_same[i,j] >= in_lane_veh[i] + in_lane_veh[j] - 3
```

New Solver Interface Library (MP)

Design

- ❖ C++ library for building efficient, configurable solver drivers
- ❖ Support for features of current C interface library
- ❖ *Extensive toolset for problem recognition and transformation*

Motivation . . .

- ❖ AMPL has logical and "not linear" expressions
  for *writing models the way you think of them*
- ❖ Current interfaces have very limited support for these
- ❖ New interfaces, built with MP,
  allow these expressions to be used and combined freely

. . . converting them to forms each solver requires

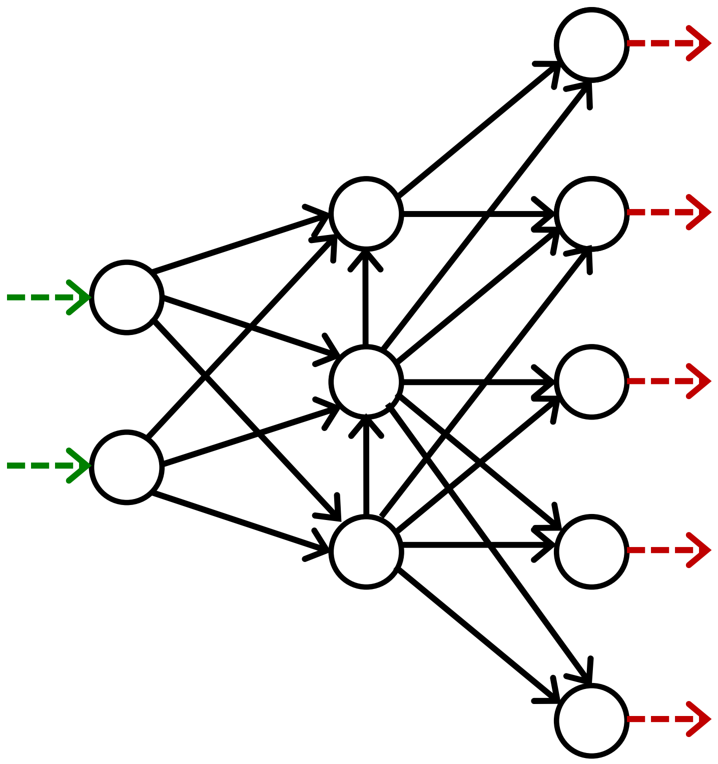
Example:
Multi-Product Network Flow

Motivation

- ❖ Ship products efficiently
  to meet demands

Context

- ❖ a transportation network
  - ∗ nodes ◯ representing cities
  - ∗ arcs ⟶ representing roads
- ❖ supplies - - -> at nodes
- ❖ demands - - -> at nodes
- ❖ capacities on arcs
- ❖ shipping costs on arcs



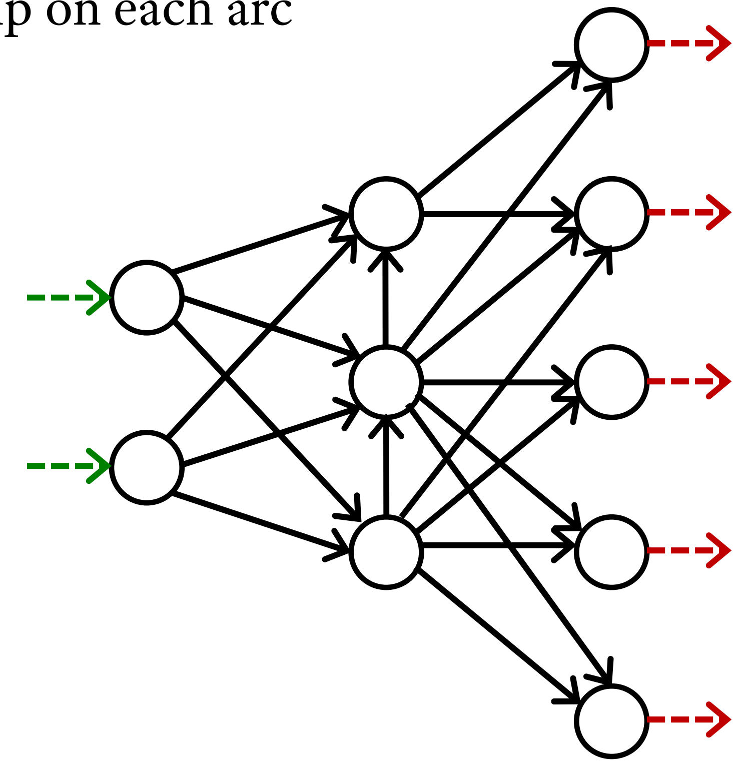
Example: Multi-Product Network Flow

Decide

- how much of each product to ship on each arc

So that

- shipping costs are kept low
- shipments on each arc respect capacity of the arc
- supplies, demands, and shipments are in balance at each node



Multi-Product Flow

Model in AMPL

Symbolic data, variables, objective

```
set PRODUCTS;
set NODES;
param net_inflow {PRODUCTS,NODES};

set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;

param var_cost {PRODUCTS,ARCS} >= 0;
var Flow {PRODUCTS,ARCS} >= 0;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j];

subject to Capacity {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

subject to Conservation {p in PRODUCTS, j in NODES}:
   sum {(i,j) in ARCS} Flow[p,i,j] + net_inflow[p,j] =
   sum {(j,i) in ARCS} Flow[p,j,i];
```

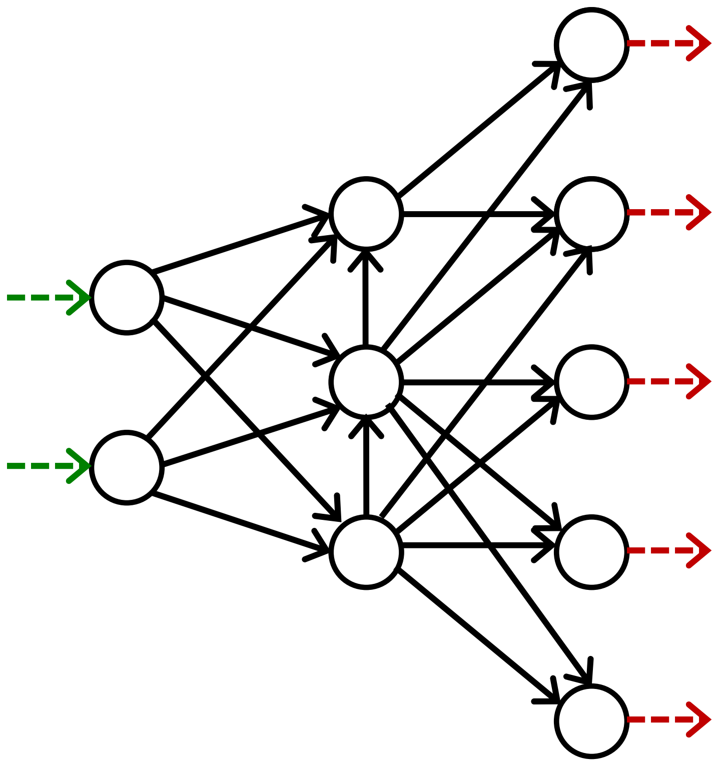
Example with conditions: Multi-Product Network Flow

Decide also

- ❖ whether to use each arc

So that

- ❖ variable costs plus fixed costs for shipping are kept low
- ❖ shipments are not too small
- ❖ not too many arcs are used



Formulating

Positive Shipments Incur Fixed Costs

Linearization

```
param fix_cost {ARCS} >= 0;
var Use {ARCS} binary;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

How you think about it

```
param fix_cost {ARCS} >= 0;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS}
      if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

Formulating

Shipments Can’t Be Too Small

Linearization

```
subject to Min_Shipment {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] >= min_ship * Use[i,j];

subject to Capacity {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j] * Use[i,j];
```

How you think about it

```
subject to Shipment_Limits {(i,j) in ARCS}:
  sum {p in PRODUCTS} Flow[p,i,j] = 0 or
  min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];
```

Formulating

Can’t Use Too Many Arcs

Linearization

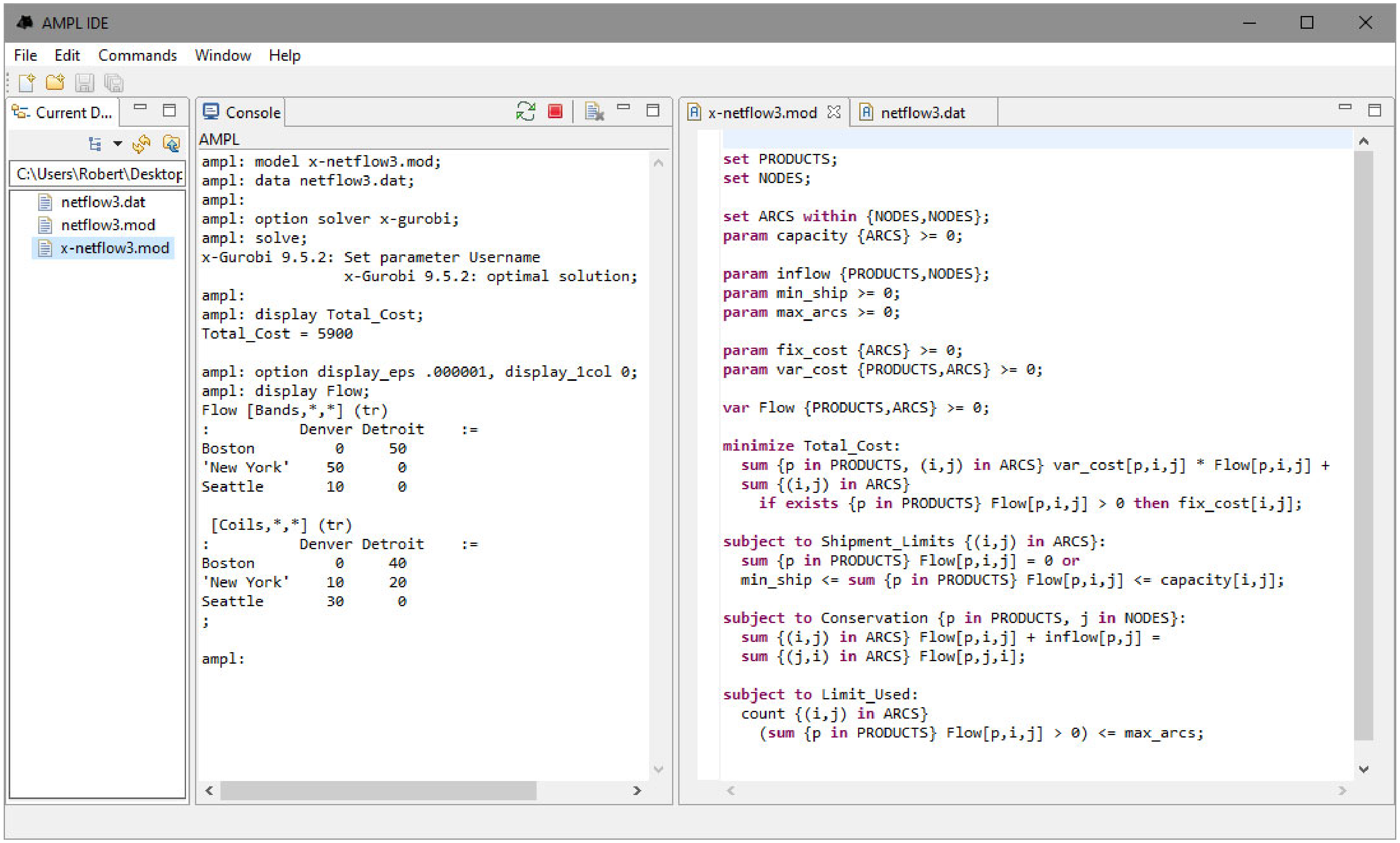
```
subject to Max_Used:
   sum {(i,j) in ARCS} Use[i,j] <= max_arcs;
```

How you think about it

```
subject to Limit_Used:
  atmost max_arcs {(i,j) in ARCS}
    (sum {p in PRODUCTS} Flow[p,i,j] > 0);
```

Formulating

Optimization by Same MIP Solver *(x-gurobi)*



```
ampl: model x-netflow3.mod;
ampl: data netflow3.dat;
ampl:
ampl: option solver x-gurobi;
ampl: solve;
x-Gurobi 9.5.2: Set parameter Username
                x-Gurobi 9.5.2: optimal solution;
ampl:
ampl: display Total_Cost;
Total_Cost = 5900

ampl: option display_eps .000001, display_1col 0;
ampl: display Flow;
Flow [Bands,*,*] (tr)
:          Denver Detroit    :=
Boston        0     50
'New York'   50      0
Seattle      10      0

 [Coils,*,*] (tr)
:          Denver Detroit    :=
Boston        0     40
'New York'   10     20
Seattle      30      0
;

ampl:
```

```
set PRODUCTS;
set NODES;

set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;

param inflow {PRODUCTS,NODES};
param min_ship >= 0;
param max_arcs >= 0;

param fix_cost {ARCS} >= 0;
param var_cost {PRODUCTS,ARCS} >= 0;

var Flow {PRODUCTS,ARCS} >= 0;

minimize Total_Cost:
  sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
  sum {(i,j) in ARCS}
    if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];

subject to Shipment_Limits {(i,j) in ARCS}:
  sum {p in PRODUCTS} Flow[p,i,j] = 0 or
  min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

subject to Conservation {p in PRODUCTS, j in NODES}:
  sum {(i,j) in ARCS} Flow[p,i,j] + inflow[p,j] =
  sum {(j,i) in ARCS} Flow[p,j,i];

subject to Limit_Used:
  count {(i,j) in ARCS}
    (sum {p in PRODUCTS} Flow[p,i,j] > 0) <= max_arcs;
```

Formulating

Linearization is Usually Not That Easy!

```
subject to IfConstr {i in 1..card(veh)-1, j in i+1..card(veh):
         veh_ind[i] = veh_ind[j] and theory_time[i] <= theory_time[j]}:
   in_lane_veh[i] = in_lane_veh[j]
      ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

```
minimize total_fuelcost:
   sum{(i,j) in A} sum{k in V} X[i,j,k] *
      ((if H[i,k] <= 300 then dMor[i,j] else
        if H[i,k] <= 660 then dAft[i,j] else
        if H[i,k] <= 901 then dEve[i,j]) * 5 +
       (if H[i,k] <= 300 then tMor[i,j] else
        if H[i,k] <= 660 then tAft[i,j] else
        if H[i,k] <= 901 then tEve[i,j]) * 0.0504);
```

```
subject to NoPersonIsolated
         {l in TYPES['loc'], r in TYPES['rank'], j in 1..numberGrps}:
   sum {i in LOCRANK[l,r]} Assign[i,j] = 0 or
   sum {i in LOCRANK[l,r]} Assign[i,j] +
     sum {a in ADJACENT[r]} sum {i in LOCRANK[l,a]} Assign[i,j] >= 2;
```

Formulating

Supported Extensions and Solvers

Operators and functions

- Conditional: `if-then-else`; `==>`, `<==`, `<==>`
- Logical: `or`, `and`, `not`; `exists`, `forall`
- Piecewise linear: `abs`; `min`, `max`; `<<breakpoints; slopes>>`
- Counting: `count`; `atmost`, `atleast`, `exactly`; `numberof`
- Comparison: `>`, `<`, `!=`; `alldiff`
- Complementarity: `complements`
- Nonlinear: `*`, `/`, `^`; `exp`, `log`; `sin`, `cos`, `tan`; `sinh`, `cosh`, `tanh`
- Set membership: `in`

Expressions and constraints

- High-order polynomials
- Second-order and exponential cones

Supported Extensions and Solvers

Operators and functions

Expressions and constraints

Solvers

- COPT, CPLEX, Gurobi, Xpress
- CBC, HiGHS, *SCIP soon*

Modeling guide

- *https://amplmp.readthedocs.io/en/latest/rst/model-guide.html*

See complete slides for . . .

- How it works with different MIP solvers
- Implementations issues
- Efficiency issues

Formulating

How It Works with MIP Solvers

Read problem instance from AMPL nl file

- Store initially as linear coefficients + expression trees
- Analyze trees to determine if linearizable

Generate linearizations

- Walk trees to build linearizations (flatten)
- Define auxiliary variables (usually zero-one)
- Generate equivalent constraints

Solve

- Send to solver through its API
- Convert optimal solution back to the original AMPL variables
- Write solution to AMPL

Formulating

Special Alternatives in *Gurobi*

Apply our linearization (`count`)

- ❖ Use Gurobi's linear API

Have Gurobi linearize (`or`, `abs`)

- ❖ Simplify and "flatten" the expression tree
- ❖ Use Gurobi's "general constraint" API
  - ∗ **`addGenConstrOr`** (resbinvar, [binvars])
    tells Gurobi: resbinvar = 1 iff at least one item in [binvars] = 1
  - ∗ **`addGenConstrAbs`** (resvar, argvar)
    tells Gurobi: resvar = |argvar|

Have Gurobi piecewise-linearize (`log`)

- ❖ Replace univariate nonlinear functions by p-l approximations
- ❖ Use Gurobi's "function constraint" API
  - ∗ **`addGenContstrLog`** (xvar, yvar)
    tells Gurobi: yvar = a piecewise-linear approximation of log(xvar)

Formulating

Implementation Issues

Is an expression repeated?

- Detect common subexpressions

```
subject to Shipment_Limits {(i,j) in ARCS}:

sum {p in PRODUCTS} Flow[p,i,j] = 0 or
min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];
```

Is there a simplified formulation?

- Yes for min-max — no for max-max

```
minimize Max_Cost:
  max {i in PEOPLE} sum {j in PROJECTS} cost[i,j] * Assign[i,j];
```

```
maximize Max_Value:
  sum {t in T} max {n in N} weight[t,n] * Value[n];
```

Formulating

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- ❖ Depends on constraint set: yes if "closed" — no if "open"
- ❖ *Usually* yes for >= or <= but no for > or < . . .

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Flow[i,j] = 0 ==> Use[i,j] = 0 else Use[i,j] = 1;
```

Formulating

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- Depends on constraint set: yes if "closed" — no if "open"
- *Usually* yes for >= or <= but no for > or < . . .

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] >= 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] > 0;
```

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- ❖ Depends on constraint set: yes if "closed" — no if "open"
- ❖ *Usually* yes for >= or <= but no for > or < . . .

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] >= 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] >= 1e-4;
```

Formulating

Solver Efficiency Issues

Bounds on subexpressions

- Define auxiliary variables that can be bounded

```
var x {1..2} <= 2, >= -2;

minimize Goldstein-Price:
  (1 + (x[1] + x[2] + 1)^2
    * (19 - 14*x[1] + 3*x[1]^2 - 14*x[2] + 6*x[1]*x[2] + 3*x[2]^2))
* (30 + (2*x[1] - 3*x[2])^2
    * (18 - 32*x[1] + 12*x[1]^2 + 48*x[2] - 36*x[1]*x[2] + 27*x[2]^2));
```

```
var t1 >= 0, <= 25;   subj to t1def: t1 = (x[1] + x[2] + 1)^2;
var t2 >= 0, <= 100;  subj to t2def: t2 = (2*x[1] - 3*x[2])^2;

minimize Goldstein-Price:
  (1 + t1
    * (19 - 14*x[1] + 3*x[1]^2 - 14*x[2] + 6*x[1]*x[2] + 3*x[2]^2))
* (30 + t2
    * (18 - 32*x[1] + 12*x[1]^2 + 48*x[2] - 36*x[1]*x[2] + 27*x[2]^2));
```

Formulating

Solver Efficiency Issues *(cont'd)*

Simplification of logic

- Replace an iterated **`exists`** with a **`sum`**

```
minimize TotalCost: ...
  sum {(i,j) in ARCS}
    if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

```
minimize TotalCost: ...
  sum {(i,j) in ARCS}
    if sum {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

Formulating

Solver Efficiency Issues *(cont'd)*

Creation of common subexpressions

❖ Substitute a stronger bound from a constraint

```
subject to Shipment_Limits {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] = 0 or
   min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

minimize TotalCost: ...
   sum {(i,j) in ARCS}
      if sum {p in PRODUCTS} Flow[p,i,j] > 0
         then fix_cost[i,j];
```

```
minimize TotalCost: ...
   sum {(i,j) in ARCS}
      if sum {p in PRODUCTS} Flow[p,i,j] >= min_ship
         then fix_cost[i,j];
```

. . . consider automating all these improvements