Advances in Model-Based Optimization with AMPL

Robert Fourer, Gleb Belov, Filipe Brandão

`[4er,gleb,fdabrandao]@ampl.com`

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

ICCOPT / MOPTA 2022:
International Conference on Continuous Optimization
Modeling and Optimization: Theory and Applications
Bethlehem, PA — 25-28 July 2022

Software Cluster, Wednesday, 2:20-3:40 pm

Advances in Model-Based Optimization with AMPL

The ideal of model-based optimization is to describe your problem the way you think about it, and then let the computer do the work of getting a solution. Recent enhancements aim to bring the AMPL modeling language and system closer to this ideal. Using a variety of modeling language extensions, common formulations are described more naturally, with the AMPL translator, the AMPL-solver interface, or the solver itself doing most of the needed transformations.

Extensions described in this presentation include quadratic expressions, logical operators and constraints, simple near-linear and nonlinear functions, and combinations of these together with linear terms. All are supported by a new C++ AMPL-solver interface library that can be adapted to handle the multiple detection and transformation strategies required by large-scale solvers.

New Developments in AMPL

Availability

- ❖ Community Edition
  - ∗ unlimited free use with free solvers
- ❖ New licensing for cloud machines and docker containers
- ❖ New implementation of the NEOS Server client (Kestrel)

Modeling language

- ❖ Snapshot utility
- ❖ New plug-in framework for
  user-defined functions, table handlers, other utilities

Data

- ❖ Extended and faster ODBC support for database software
- ❖ Direct support for .csv and .xlsx (spreadsheet) files
  - ∗ Support for two-dimensional spreadsheet tables

New Developments in AMPL

Examples

- Free AMPL Model Colaboratory
  - AMPL in Jupyter notebooks using Google Colab, Kaggle, etc.
- Portfolio optimization and deployment in the amplpy API

Solvers

- Callbacks from AMPL APIs
- *New interface library . . .*

New Solver Interface Library (MP)

Design

- C++ library for building efficient, configurable solver drivers
- Complementary to AMPL's C interface library (ASL)
- *Extensive toolset for problem transformations*

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)

Interface design

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

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

. . . initial emphasis on MIP solvers

Outline

Example

- Multi-product network flow *with complications*
- Model-based optimization
- Linearization for MIP solvers: in math and in AMPL

Formulating models more like you think about them

- *Example:* Natural vs. linearized formulations
- Supported operators, functions, expressions
- Implementation issues
- Efficiency issues

New C++ interface

- General use with COPT, HiGHS
- Special alternatives for Gurobi

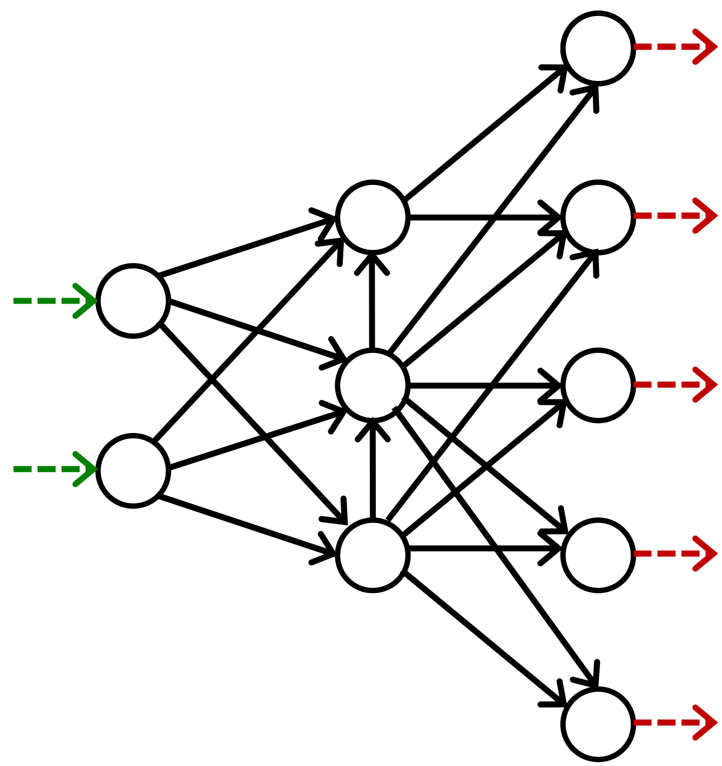
Example:
Multi-Product Network Flow

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



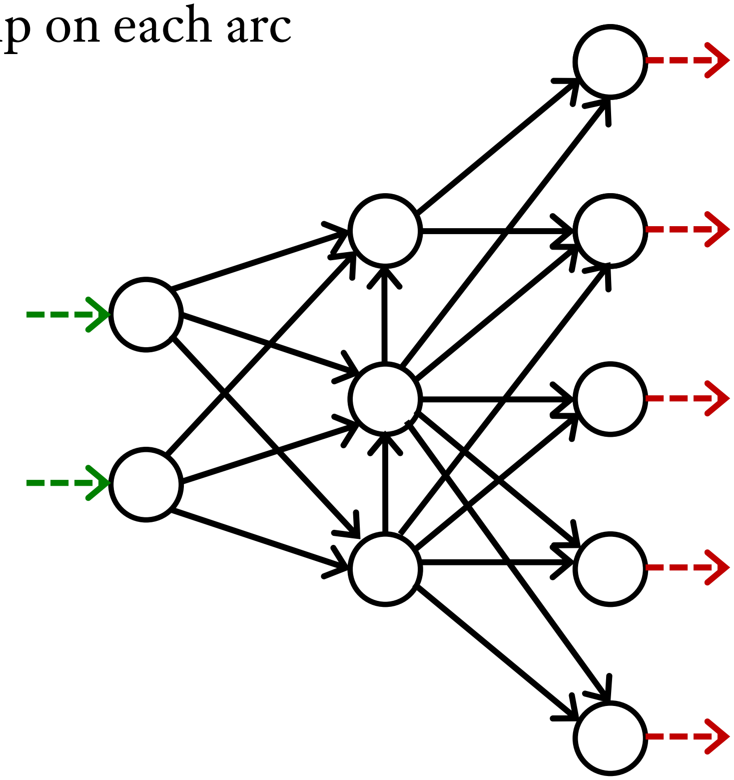
Example:
Multi-Product Network Flow

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



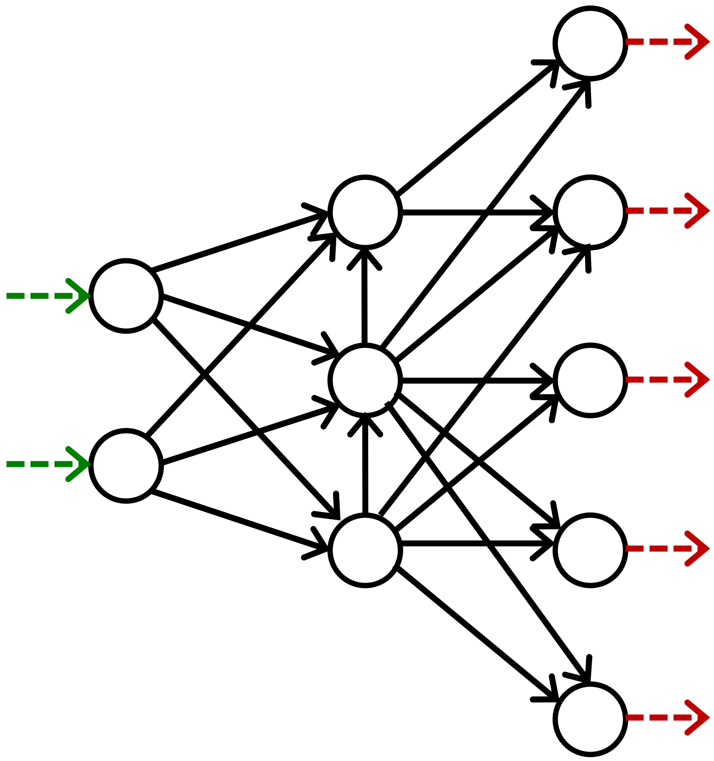
Example *with complications:*
Multi-Product Network Flow

Decide also

- whether to use each arc

So that

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



Model-Based Optimization

Formulate a minimum shipping cost model

- *decision variables:* What arcs are used and how much is shipped
- *objective:* Total fixed and variable costs
- *constraints:* Equations that the variables must satisfy to meet the requirements of the problem

Apply model-based optimization software

- *modeling language:* Write a formulation that a computer system can read
- *data:* Read costs, capacities, supplies, demands, and limits that define a specific case to be solved
- *solver:* Send to an off-the-shelf optimization engine that accepts a broad class of problems

Formulation *(data)*

Given

P set of products

N set of network nodes

$A \subseteq N \times N$ set of arcs connecting nodes

and

u_{ij} capacity of arc from i to j, for each $(i, j) \in A$

s_{pj} supply/demand of product p at node j, for each $p \in P$, $j \in N$
> 0 implies supply, < 0 implies demand

c_{pij} cost per unit to ship product p on arc (i, j),
for each $p \in P$, $(i, j) \in A$

d_{ij} fixed cost for using the arc from i to j, for each $(i, j) \in A$

m smallest total shipments on any arc that is used

n largest number of arcs that may be used

Linearized Formulation *(variables, objective)*

Determine

X_{pij} amount of commodity p to be shipped on arc (i, j), for each $p \in P$, $(i, j) \in A$

Y_{ij} 1 if any amount is shipped from node i to node j, 0 otherwise, for each $(i, j) \in A$

to minimize

$$\sum_{p \in P} \sum_{(i,j) \in A} c_{pij} X_{pij} + \sum_{(i,j) \in A} d_{ij} Y_{ij}$$

total cost of shipments

Multi-Product Flow

Linearized Formulation *(constraints)*

Subject to

$\sum_{p \in P} X_{pij} \leq u_{ij} Y_{ij}$, for all $(i, j) \in A$

when the arc from node i to node j is used for shipping,
total shipments must not exceed capacity, and Y_{ij} must be 1

$\sum_{p \in P} X_{pij} \geq m Y_{ij}$, for all $(i, j) \in A$

when the arc from node i to node j is used for shipping,
total shipments from i to j must be at least m

$\sum_{(i,j) \in A} X_{pij} + s_{pj} = \sum_{(j,i) \in A} X_{pji}$, for all $p \in P$, $j \in N$

shipments in plus supply/demand must equal shipments out

$\sum_{(i,j) \in A} Y_{ij} \leq n$

At most n arcs can be used

Multi-Product Flow

Linearized Model in AMPL

Symbolic data, variables, objective

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

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];
```

Multi-Product Flow

Linearized Model in AMPL

Constraints

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

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

$$\sum_{p \in P} X_{pij} \leq u_{ij} Y_{ij}, \text{ for all } (i,j) \in A$$

Data Instance in AMPL Text Format

Data: Limits

```
set PRODUCTS := Bands Coils ;
set NODES := Detroit Denver Boston 'New York' Seattle ;

param: ARCS: capacity:

        Boston 'New York' Seattle :=
 Detroit   100       80       120
 Denver    120      120       120   ;

param inflow:

         Detroit Denver Boston 'New York' Seattle :=
  Bands      50     60     -50     -50       -10
  Coils      60     40     -40     -30       -30;

param min_ship := 15 ;

param max_arcs := 4 ;
```

Multi-Product Flow

Data Instance in AMPL Text Format

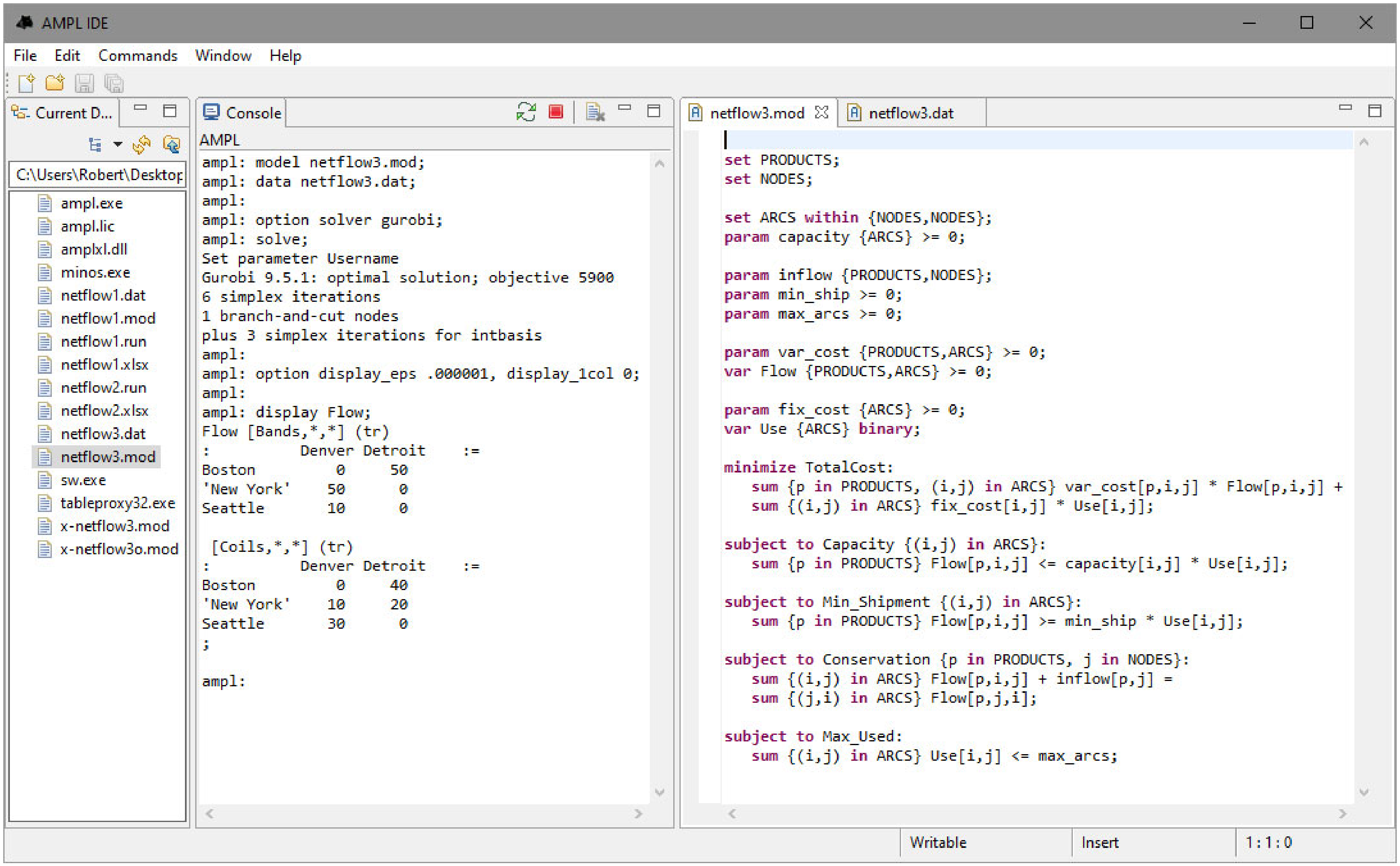
Data: Costs

```
param var_cost:
 [Bands,*,*]  Boston 'New York' Seattle :=
     Detroit     10        20        60
     Denver      40        40        30
 [Coils,*,*]  Boston 'New York' Seattle :=
     Detroit     20        20        80
     Denver      60        70        30 ;

param fix_cost default 75 ;
```

Multi-Product Flow

Optimization: MIP Solver *(gurobi)*



Formulating (MIP) **Models More Like You Think About Them**

Describe an optimization problem

- In a form *you find natural or convenient*
- Using existing AMPL expressions, functions, and operators

Send the problem to a solver

- In a form *that solver will accept*
- Relying on the AMPL-solver interface to translate

Get back a result

- In the form you originally used

Formulating

Positive Shipments Incur Fixed Costs

Linearized formulation

```
sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

Natural formulation

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

Shipments Can't Be Too Small

Linearized formulation

```
sum {p in PRODUCTS} Flow[p,i,j] >= min_ship * Use[i,j];
sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j] * Use[i,j];
```

Natural formulation

```
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

Linearized formulation

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

Natural formulation

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

Formulating

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

AMPL IDE

Current Directory: C:\Users\Robert\Desktop

- ampl.exe
- ampl.lic
- amplxl.dll
- minos.exe
- netflow1.dat
- netflow1.mod
- netflow1.run
- netflow1.xlsx
- netflow2.run
- netflow2.xlsx
- netflow3.dat
- netflow3.mod
- sw.exe
- tableproxy32.exe
- x-netflow3.mod
- x-netflow3o.mod

Console

```
AMPL
ampl: model x-netflow3.mod;
ampl: data netflow3.dat;
ampl:
ampl: option solver x-gurobi;
ampl: solve;
x-Gurobi 9.5.1: Set parameter Username
                x-Gurobi 9.5.1: optimal solution;
ampl:
ampl: display Total_Cost;
Total_Cost = 5900

ampl: option display_eps .000001, display_1col 0;
ampl:
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: |
```

x-netflow3.mod

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

Extensions for MIP Solvers

Conditional operators

- **`if`** *constraint* **`then`** *var-expr1* [**`else`** *var-expr2*]
- *constraint1* ==> *constraint2* [**`else`** *constraint3*]
  constraint1 <== *constraint2*
  constraint1 <==> *constraint2*

```
minimize TotalCost:
   sum {j in JOBS, k in MACHINES}
      if MachineForJob[j] = k then cost[j,k];
```

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] >= 1 ==>
      minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Extensions for MIP Solvers

Logical operators

- *constraint1* **or** *constraint2*
  constraint1 **and** *constraint2*
  not *constraint2*
- **exists** {*indexing*} *constraint-expr*
  forall {*indexing*} *constraint-expr*

```
subject to NoMachineConflicts
      {m1 in 1..nMach, m2 in m1+1..nMach, j in 1..nJobs}:
   Start[m1,j] + duration[m1,j] <= Start[m2,j] or
   Start[m2,j] + duration[m2,j] <= Start[m1,j];
```

```
subj to HostNever {j in BOATS}:
   isH[j] = 1 ==> forall {t in TIMES} H[j,t] = j;
```

Extensions for MIP Solvers

Piecewise-linear functions and operators

- << *breakpoint-list* ; *slope-list* >> *variable*
  << *breakpoint-list* ; *slope-list* >> (*variable* , *zero-point*)
- **abs**(*var-expr*)
  min(*var-expr-list*)   **min** {*indexing*} *var-expr*
  max(*var-expr-list*)   **max** {*indexing*} *var-expr*

```
minimize Total_Cost:
   sum {i in ORIG, j in DEST}
      <<{p in 1..npiece[i,j]-1} limit[i,j,p];
        {p in 1..npiece[i,j]} rate[i,j,p]>> Trans[i,j];
```

```
maximize WeightSum:
   sum {t in TRAJ} max {n in NODE} weight[t,n] * Use[n];
```

Extensions for MIP Solvers

Counting operators

- **count** {*indexing*} (*constraint-expr*)
- **atmost** *k* {*indexing*} (*constraint-expr*)
  atleast *k* {*indexing*} (*constraint-expr*)
  exactly *k* {*indexing*} (*constraint-expr*)
- **numberof** *k* **in** (*var-expr-list*)

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

```
subj to CapacityOfMachine {k in MACHINES}:
   numberof k in ({j in JOBS} MachineForJob[j]) <= cap[k];
```

Extensions for MIP Solvers

Comparison operators

- *var-expr1* **!=** *var-expr2*
  var-expr1 **>** *var-expr2*
  var-expr1 **<** *var-expr2*
- **alldiff(***var-expr-list***)**
  alldiff {*indexing***}** *var-expr*

```
subj to Different_Colors {(c1,c2) in Neighbors}:
   Color[c1] != Color[c2];
```

```
subject to OnePersonPerPosition:
   alldiff {i in 1..nPeople} Pos[i];
```

Extensions for MIP Solvers

Complementarity operators

- *single-inequality1* **complements** *single-inequality2*
- *double-inequality* **complements** *var-expr*
  var-expr **complements** *double-inequality*

```
subject to Pri_Compl {i in PROD}:
   max(500.0, Price[i]) >= 0 complements
      sum {j in ACT} io[i,j] * Level[j] >= demand[i];
```

```
subject to Lev_Compl {j in ACT}:
   level_min[j] <= Level[j] <= level_max[j] complements
      cost[j] - sum {i in PROD} Price[i] * io[i,j];
```

Extensions for MIP Solvers

Nonlinear expressions and operators

- *var-expr1* ***** *var-expr2*
  var-expr1 **/** *var-expr2*
  var-expr **^** *k*
- **exp**(*var-expr*) **log**(*var-expr*)
  sin(*var-expr*) **cos**(*var-expr*) **tan**(*var-expr*)

```
subj to Eq {i in J} :
   x[i+neq] / (b[i+neq] * sum {j in J} x[j+neq] / b[j+neq]) =
   c[i] * x[i] / (40 * b[i] * sum {j in J} x[j] / b[j]);
```

```
minimize Chichinadze:
     x[1]^2 - 12*x[1] + 11 + 10*cos(pi*x[1]/2)
   + 8*sin(pi*5*x[1]) - exp(-(x[2]-.5)^2/2)/sqrt(5);
```

Formulating

Extensions for MIP Solvers

Discrete variable domains

- **var** *varname* **{***indexing***}** **in** *set-expr***;**

```
var Buy {f in FOODS} in {0,10,30,45,55};
```

```
var Ship {(i,j) in ARCS}
   in {0} union interval[min_ship,capacity[i,j]];
```

```
var Work {j in SCHEDS} integer
   in {0} union interval[least, max {i in SHIFT_LIST[j]} req[i]];
```

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 an easy reformulation?

- Yes for min-max, no for max-min

```
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?

- Yes if constraint set is "closed"
- No if constraint set is "open"

```
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?

- Yes if constraint set is "closed"
- No if constraint set is "open"

```
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;
```

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));
```

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

General use with COPT, HiGHS

Read objectives & constraints from AMPL

- 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

Special Alternatives in *x-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)

Learn More

https://github.com/ampl/

- all AMPL open-source projects

https://github.com/ampl/mp

- *MP solver interface*

https://dev.ampl.com

- new AMPL development projects

https://colab.ampl.com/

- AMPL Colaboratory links