Teaching, Learning & Applying Optimization

New Developments in the AMPL Modeling System

Filipe Brandão, Robert Fourer
{fdabrandao,4er}@ampl.com
AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Annual Meeting
Phoenix, Arizona — 14 October 2023
Exhibitor Workshop
Teaching, Learning, and Applying Optimization: New Developments in the AMPL Modeling System

Optimization is the most widely adopted technology of Operations Research and Analytics, yet it must steadily evolve to remain relevant. After an introductory example, this presentation takes you on a tour through new developments in the AMPL modeling language and system that have been changing the ways that people learn and apply large-scale optimization:

- A new approach to connecting the modeling language to solvers, which lets you write many common logical conditions and “not quite linear” functions in a much more natural way, avoiding complicated and error-prone reformulations.

- A Python-first alternative to learning AMPL and model-building, supported by new teaching materials that leverage the power of Jupyter notebooks and Google Colab to bring modern computing to the study of optimization.

- Enhancements to the AMPL Python interface (amplpy), including faster data input and closer solver integration, which expand the possibilities for model-based application development.

We conclude with deployment examples, showing how Python scripts can be turned quickly into OR and Prescriptive Analytics applications using amplpy, Pandas, and the Streamlit app framework. Deployments are supported on traditional servers and in a variety of modern virtual environments including containers, clusters, and clouds.
Mathematical optimization

Definitions
Definitions from Oxford Languages · Learn more

**optimization**

noun
the action of making the best or most effective use of a situation or resource.
"companies interested in the optimization of the business"

Translate optimization to French

---

People also ask

What is optimization used for?

What do you mean optimize?
Mathematical optimization

From Wikipedia, the free encyclopedia

*Mathematical programming* redirects here. For the peer-reviewed journal, see Mathematical Programming.

*Optimization* and *Optimum* redirect here. For other uses, see Optimization (disambiguation) and Optimum (disambiguation).

In mathematics, computer science and operations research, mathematical optimization (alternatively spelled optimisation or mathematical programming) is the selection of a best element (with regard to some criterion) from some set of available alternatives.¹

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations constitutes a large area of applied mathematics. More generally, optimization includes finding “best available” values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains.
New Developments in AMPL

**Part 1: Writing models more like you think about them**

- Background: algebraic model-based optimization
- Motivation
  - Typical user complaints
  - Previous advances
- MP: a new AMPL-solver interface library
  - Example: Multi-product flow with logical conditions
  - Extensions supported in AMPL
  - Implementation issues
- Using MP-based solvers with AMPL
  - Direct interface to spreadsheet data
  - AMPL model colaboratory on Google Colab
New Developments in AMPL

Part 2: Teaching, Learning & Applying AMPL with Python

❖ A Python-first alternative
  ❖ Interfacing with Python using amplpy
  ❖ Jupyter notebooks & Google Colab

❖ Enhancements to the AMPL Python interface
  ❖ Installing AMPL and solvers as Python packages
  ❖ Importing and exporting data naturally from/to Python data structures such as Pandas dataframes
  ❖ Turning Python scripts into prescriptive analytics applications in minutes with Pandas, amplpy, and Streamlit
It is time for us to outgrow our love affair with linear programming. As a starting point, I would like to offer my own definition of “optimization”:

**Optimization is the science of making the best possible decisions,** either at a point in time or over time, **to optimize a metric (or combination of metrics) while respecting constraints** and (for sequential problems) the dynamics of a problem.

Mathematical Optimization

In concept,

- Given an objective in terms of some *decision variables*
- Choose values of the variables to make a given objective as large or as small as possible
- Subject to constraints on the values of the variables

In practice,

- A paradigm for a very broad variety of *decision problems*
- A valuable approach to making decisions
Optimization in OR & Analytics

*Given a recurring need to make many interrelated decisions*
  - Purchases, production and shipment amounts, assignments, . . .

*Consistently make highly desirable choices*

*By applying ideas from mathematical optimization*
  - Ways of describing problems (*models*)
  - Ways of solving problems (*algorithms*)
Model-Based Optimization

Steps

- model: Formulate a general description of a class of optimization problems
- data: Get values that define a scenario to be solved; combine them with the model to generate a problem instance
- solver: Apply algorithmic software to compute good decisions for the problem instance
- results: Analyze or deploy the solution

Independence

- model is independent of data
- model & data are independent of solver
**Algebraic Model-Based Optimization**

**Mathematical model formulation**

- *sets & parameters*: Description of the data required
- *decision variables*: Solution values to be determined
- *objective*: Function of the variables that one would like to minimize or maximize
- *constraints*: Conditions that the variables must satisfy to meet the requirements of the problem

**Model-based optimization software**

- *modeling language*: System for expressing model formulations in a way that a computer system can process
- *solver*: Ready-to-run optimization engine that finds solutions for broad classes of model types
Writing Models
More Like You Think About Them

Motivation
- Typical user complaints
- Precursors
- Example: Multi-product flow with logical conditions

Realization
- MP: a new AMPL-solver interface library
- Supports more natural & direct ways of expressing models
- Facilitates connections to a range of solvers
**Motivation**

**Typical MIP 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]}:

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

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

Typical Response

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

\[
\text{var in_lane_same \{veh,veh\} binary;}
\]

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

\[
\text{in_lane_veh}[i] = \text{in_lane_veh}[j] \implies \text{in_lane_same}[i,j] = 1
\]
\[
\text{in_lane_same}[i,j] = 1 \implies \text{in_in_time}[j] \geq \text{in_in_time}[i] + \frac{1\_veh}{V};
\]

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

Given that \(\text{in_lane_veh}\) can only be either 1 or 2, those constraints could be

\[
\text{in_lane_same}[i,j] \geq 3 - \text{in_lane_veh}[i] - \text{in_lane_veh}[j]
\]
\[
\text{in}_\text{lane_same}[i,j] \geq \text{in_lane_veh}[i] + \text{in_lane_veh}[j] - 3
\]
**Typical Nonlinear User Complaint**

So I tried out gurobi with the two commands I mentioned in my previous email, and I receive the message

**Gurobi 9.0.2: Gurobi can't handle nonquadratic nonlinear constraints.**

I went over the constraints, and it seems to me
the only constraint that is nonquadratic nonlinear is

subject to A2 {t in 2..card(POS), i in PATIENTS}:
  sum {a in DONORS, b in PATIENTS, c in PATIENTS: ceil(a/2) = c}
    x[b,t] * x[c,t-1] * y[a,b] = 2 * x[i,t];

where x and y are binary variables.

Is this now sufficient for gurobi to solve if I only linearize one of the term on the LHS of this constraint (e.g. x[b, t]), while keeping the other two terms the same?
**Motivation**

**Typical Response**

*You are right, A2 has a cubic term \( x[b,t] \times x[c,t-1] \times y[a,b] \) that you will have to transform before you can get Gurobi to accept it.*

*You can transform to quadratic* by picking two of the three variables and replacing their product by a new variable. For example, if you define a new binary variable \( z[b,c,t] \) to replace \( x[b,t] \times x[c,t-1] \), you can write

```plaintext
var z {t in 2..card(POS), b in PATIENTS, c in PATIENTS} binary;
subject to zDefn {t in 2..card(POS), b in PATIENTS, c in PATIENTS}:
    z[b,c,t] = x[b,t] * x[c,t-1];
```

Then write your constraint A2 as \( z[b,c,t] \times y[a,b] = 2 \times x[i,t] \). There are two other possibilities, corresponding to the two other ways you can pick two of the three variables.

*You can also linearize the cubic term* directly. In that case, you would define a new binary variable \( z[a,b,c,t] \) to replace \( x[b,t] \times x[c,t-1] \times y[a,b] \), and you would add the following four constraints:

```
z[a,b,c,t] >= x[b,t] + x[c,t-1] + y[a,b] - 2
z[a,b,c,t] <= x[b,t]
z[a,b,c,t] <= x[c,t-1]
z[a,b,c,t] <= y[a,b]
```
Motivation

Precursors

MiniZinc

- Modeling language for Constraint Programming
- Linearization of diverse logical conditions to support MIP solvers

gurobipy

- Python-based modeling language & interface to Gurobi solver
- Linear, quadratic + “general” constraints
  - min/max, abs, and/or, norm, if-then, piecewise-linear
  - exp, log, power, sin, cos . . . handled by p-l approximation
- Transformation to linear-quadratic MIPs
  - https://www.gurobi.com/documentation/current/refman/constraints.html
Motivation

Typical gurobipy User Complaint

```
production_change_cost = gp.quicksum(3 * gp.max_(0,(x[i] - x[i-1] for i in periods)) \  
+ 0.8 * gp.max_(0,(x[i-1] - x[i] for i in periods)))
```

Hi

I'm trying to solve a production problem. When the x change, it will cost a different additional cost. I need to compare the \([x[i] - x[i-1]]\) with 0. How can I solve this.
Motivation

Typical gurobipy Response

General constraints are meant to be used to define single constraints. It is not possible to use these constructs in other expressions, i.e., it is not possible to use gp.max_ in a more complex constraint other than $y = \text{gp.max}_-$. 
Motivation

Typical gurobi Response

General constraints are meant to be used to define single constraints. It is not possible to use these constructs in other expressions, i.e., it is not possible to use gp.max_ in a more complex constraint other than \( y = \text{gp.max}_- \).

Moreover, as described in the documentation of the addGenConstrMax method, \( \text{gp.max}_- \) only accepts single variables as inputs. Thus, it is not possible to pass expressions \( x[i] - x[i-1] \). To achieve what you want, you have to introduce additional auxiliary variables \( \text{aux}[i] = x[i] - x[i-1] \) and additional equality constraints \( z1 = \text{gp.max}_- \) and \( z2 = \text{gp.max}_- \).

```python
aux1 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar1")
aux2 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar2")

# are you sure that i-1 does not lead to a wrong key access?
m.addConstrs((aux1[i] = x[i]-x[i-1] for i in periods), name = "auxconstr1")
m.addConstrs((aux2[i] = x[i-1]-x[i] for i in periods), name = "auxconstr2")
z1 = m.addVar(lb = -GRB.INFINITY, name="z1")
z2 = m.addVar(lb = -GRB.INFINITY, name="z2")
m.addConstr(z1 = gp.max_(0,aux1),name="maxconstr1")
m.addConstr(z2 = gp.max_(0,aux2),name="maxconstr2")

[...]
production_change_cost = gp.quicksum(3 * z1 + 0.8 * z2)
```
New in AMPL

**MP Solver Interface Library**

*Design*

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

MP Solver Interface Library

General context

- AMPL has logical and “not linear” expressions
- Previous ASL interface had very limited support for these
- New interfaces, built with MP, allow these expressions to be used freely

Gurobi context

- AMPL should support Gurobi’s general constraints
- Existing gurobipy offers only limited range of expressions
- New interfaces, built with MP, convert much more general expressions to work with Gurobi
Motivation

- Ship products efficiently to meet demands

Context

- A transportation network
  - Nodes representing cities
  - Arcs representing roads
- Supplies to nodes
- Demands at nodes
- Capacities on arcs
- Shipping costs on arcs

Example: Multi-Product Network Flow
**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**

**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 \( p \in P, (i, j) \in A \)
Multi-Product Flow

Formulation (variables, objective, constraints)

Determine

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

to minimize

\[ \Sigma_{p \in P} \Sigma_{(i,j) \in A} c_{pij} X_{pij} \]

total cost of shipments

Subject to

\[ \Sigma_{p \in P} X_{pij} \leq u_{ij}, \text{ for all } (i, j) \in A \]

total shipments must not exceed capacity

\[ \Sigma_{(i,j) \in A} X_{pij} + s_{pj} = \Sigma_{(j,i) \in A} X_{pji}, \text{ for all } p \in P, j \in N \]

shipments in plus supply/demand must equal shipments out
Multi-Product Flow

Model in AMPL

Symbolic data, variables, objective

```AMPL
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];
```

\[
\sum_{(i,j) \in A} X_{p ij} + s_{pj} = \sum_{(j,i) \in A} X_{p ji}, \quad \text{for all } p \in P, j \in N
\]
Multi-Product Flow

**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

```ampl
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

```ampl
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} fix_cost[i,j];
    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

Linearization is Often Not So 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 LOCRAANK[l,r]} Assign[i,j] = 0 or
    sum {i in LOCRAANK[l,r]} Assign[i,j] +
    sum {a in ADJACENT[r]} sum {i in LOCRAANK[l,a]} Assign[i,j] >= 2;
Formulating Extensions

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];
Formulating Extensions

Logical operators

- \( \text{constraint1 or constraint2} \)
- \( \text{constraint1 and constraint2} \)
- \( \text{not constraint2} \)
- \( \text{exists \{indexing\} constraint-expr} \)
- \( \text{forall \{indexing\} constraint-expr} \)

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 ==\(\text{forall \{t in TIMES\} H[j,t] = j;}}
Formulating

Extensions

Piecewise-linear functions and operators

- \(<\langle \text{breakpoint-list}; \text{slope-list} \rangle \rangle \text{ variable}
- \(<\langle \text{breakpoint-list}; \text{slope-list} \rangle \rangle \text{ (variable, zero-point)}
- \text{abs}(\text{var-expr})
- \text{min}(\text{var-expr-list}) \text{ min}\{\text{indexing}\} \text{ var-expr}
- \text{max}(\text{var-expr-list}) \text{ max}\{\text{indexing}\} \text{ var-expr}

maximize WeightSum:
\[
\sum \{t \text{ in TRAJ} \} \text{ max}\{n \text{ in NODE} \} \text{ weight}[t,n] \ast \text{ Use}[n];
\]

minimize Total_Cost:
\[
\sum \{i \text{ in ORIG, j in DEST}\}
\langle\langle \{p \text{ in } 1..\text{npiece}[i,j]-1\} \text{ limit}[i,j,p];
\{p \text{ in } 1..\text{npiece}[i,j]\} \text{ rate}[i,j,p]\rangle \text{ Trans}[i,j];
\]
Formulating Extensions

Piecewise-linear functions and operators

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

```plaintext
x = mod.addVars(periods)
production_change_cost = \
    gp.quicksum(3.0 * gp.max_(0, (x[i] - x[i-1] for i in periods)) \n        + 0.8 * gp.max_(0, (x[i-1] - x[i] for i in periods)))
```

```plaintext
var x {0..T} >= 0;
var production_change_cost =
    3.0 * max(0, {i in 1..T} x[i] - x[i-1]) +
    0.8 * max(0, {i in 1..T} x[i-1] - x[i]);
```
Formulating

Extensions

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)`

```plaintext
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];
```
Formulating

Extensions

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];
**Formulating**

**Extensions**

**Complementarity operators**

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

```plaintext
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];
```
Formulating

Extensions

Nonlinear expressions and operators

- \( \text{var-expr1} \times \text{var-expr2} \)
- \( \text{var-expr1} / \text{var-expr2} \)
- \( \text{var-expr} ^ {k} \)
- \( \exp(\text{var-expr}) \log(\text{var-expr}) \)
- \( \sin(\text{var-expr}) \cos(\text{var-expr}) \tan(\text{var-expr}) \)

\[
\text{subj to } \sum_{i \in J} \left( x[i+\text{neq}] / (b[i+\text{neq}] \times \sum_{j \in J} x[j+\text{neq}] / b[j+\text{neq}]) \right) = \\
c[i] \times x[i] / (40 \times b[i] \times \sum_{j \in J} x[j] / b[j]);
\]

\[
\text{minimize Chichinadze:}
\begin{align*}
& x[1]^2 - 12\times x[1] + 11 + 10\times \cos(\pi \times x[1]/2) \\
& + 8\times \sin(\pi \times 5 \times x[1]) - \exp(-(x[2]-.5)^2/2)/\sqrt{5};
\end{align*}
\]
Formulating Extensions

Discrete variable domains

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

\begin{verbatim}
var Buy \{f in FOODS\} in \{0,10,30,45,55\};
\end{verbatim}

\begin{verbatim}
var Ship \{(i,j) in ARCS\}
in \{0\} union interval[min_ship,capacity[i,j]];
\end{verbatim}

\begin{verbatim}
var Work \{j in SCHEDS\}
in \{0\} union integer[least,max \{i in SHIFT_LIST[j]\} req[i]];
\end{verbatim}
**MP Interface**

**General use with MIP solvers**

**Read objectives & constraints from AMPL**
- Store initially as linear coefficients + expression graphs
- 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 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 an easy reformulation?

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

```
minimize Worst_Rank:
  max {i in PEOPLE} sum {j in PROJECTS} rank[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”

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

Implementation Issues (cont’d)

Does an exact linearization exist?

- Yes if constraint set is “closed”
- No if constraint set is “open”

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

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

```plaintext
subj to Use_DIFFinition {(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

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

minimize Goldstein-Price:
  (1 + (x[1] + x[2] + 1)^2
 * (30 + (2*x[1] - 3*x[2])^2
```

```plaintext
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
 * (30 + t2
```
**Formulating**

**Solver Efficiency Issues (cont’d)**

**Simplification of logic**

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

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

```plaintext
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
**Formulating**

**Solver Tolerance Issues**

*Solver tolerances are applied after automatic conversion*

- Anomalous results are possible in rare circumstances

```
var x {1..2} >=0, <=100;

maximize Total:
        then x[1] + x[2] else 0;

subj to con: x[1] = x[2];
```

```
ampl: solve;
Gurobi 10.0.2: optimal solution; objective 9.999998

ampl: display x;
1  4.9999999
2  4.9999999

ampl: display Total;
Total = 0
```
Formulating

Solver Tolerance Issues

Warning added
  ✗ (but needs work)

```ampl
var x {1..2} >=0, <=100;

maximize Total:
  then x[1] + x[2] else 0;

subj to con: x[1] = x[2];
```

```
ampl: solve;
Gurobi 10.0.2: optimal solution; objective 9.9999998

---------- WARNING ----------
WARNING: "Solution Check (Idealistic)"
[ sol:chk:feastol=1e-06, :feastolrel=1e-06, :inttol=1e-05,
  :round='', :prec='\'
]
Objective value violations:
  - 1 objective value(s) violated,
    up to 1E+01 (abs)
Idealistic check is an indicator only, see documentation.
```
AMPL Environments

Native
- Interactive command line
- Model, data, and script (“run”) files

IDEs
- AMPL IDE, VScode

APIs
- C++, C#, Java, MATLAB, Python, R

Python
- Jupyter notebooks
- AMPL model colaboratory . . .
Direct Spreadsheet Interface

“1D” spreadsheet ranges
Spreadsheet interface

Data Handling

Script (input)

```plaintext
model x-netflow3.mod;

table Products IN "amplxl" "netflow2.xlsx" "Items":
    PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow2.xlsx":
    NODES <- [NODES];

table Capacity IN "amplxl" "netflow2.xlsx":
    ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow2.xlsx":
    [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow2.xlsx":
    [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```
Spreadsheet interface

Data Handling

Script (input)

```AMPL
model x-netflow3.mod;

table Products IN "amplxl" "netflow2.xlsx" "Items":
  PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow2.xlsx":
  NODES <- [NODES];

table Capacity IN "amplxl" "netflow2.xlsx" "2D":
  ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow2.xlsx" "2D":
  [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow2.xlsx" "2D":
  [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```
Direct Spreadsheet Interface

“2D” spreadsheet ranges

<table>
<thead>
<tr>
<th>ITEMS</th>
<th>capacity</th>
<th>TO</th>
<th>cost</th>
<th>ITEMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bands</td>
<td>FROM</td>
<td>Boston</td>
<td>New York</td>
<td>Seattle</td>
</tr>
<tr>
<td></td>
<td>Detroit</td>
<td>100</td>
<td>80</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Denver</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coils</td>
<td>FROM</td>
<td>Detroit</td>
<td>Boston</td>
<td></td>
</tr>
<tr>
<td></td>
<td>New York</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detroit</td>
<td>inflow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Denver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boston</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New York</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seattle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Excel spreadsheet showing data for items and nodes with capacities and costs.
Spreadsheet interface

Data Handling

**Script (output)**

```AMPL
option solver gurobi;
solve;

table Results OUT "amplxl" "netflow1.xlsx" "2D":
    [ITEMS, FROM, TO], Flow;

table Summary OUT "amplxl" "netflow1.xlsx":
    {(i,j) in ARCS} -> [FROM, TO],
    sum {p in PRODUCTS} Flow[p, i, j] ~ TotFlow,
    sum {p in PRODUCTS} Flow[p, i, j] / capacity[i, j] ~ "%Used";

write table Results;
write table Summary;
```
### Spreadsheet interface

#### Data Results

**“2D” spreadsheet range**

![Spreadsheet interface](image)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Detroit</td>
<td>Boston</td>
<td>50</td>
<td>30</td>
<td>Detroit</td>
<td>Boston</td>
<td>80</td>
<td>80.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Detroit</td>
<td>New York</td>
<td>0</td>
<td>30</td>
<td>Detroit</td>
<td>New York</td>
<td>30</td>
<td>37.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Detroit</td>
<td>Seattle</td>
<td>0</td>
<td>0</td>
<td>Detroit</td>
<td>Seattle</td>
<td>0</td>
<td>0.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Denver</td>
<td>Boston</td>
<td>0</td>
<td>10</td>
<td>Denver</td>
<td>Boston</td>
<td>10</td>
<td>8.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Denver</td>
<td>New York</td>
<td>50</td>
<td>0</td>
<td>Denver</td>
<td>New York</td>
<td>50</td>
<td>41.7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Denver</td>
<td>Seattle</td>
<td>10</td>
<td>30</td>
<td>Denver</td>
<td>Seattle</td>
<td>40</td>
<td>33.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**shipments** | **ITEMS** | **FROM** | **TO** | **TotFlow** | **%Used**

**Data and Results**

![Spreadsheet interface](image)
AMPL Model Colaboratory in Google Colab

Multi-Product Flow

Set up AMPL and solvers with just a few lines

```python
%pip install -q amplpy pandas numpy
from amplpy import AMPL, ampl_notebook

ampl = ampl_notebook(
    modules=['highs', 'gurobi'],  # modules to install
    license_uid='3450be66-b04a-11eb-9e10-c75c7742e3ae',  # license to use
)  # Instantiate AMPL object and register magic
```

https://colab.research.google.com/drive/1RteMlfHd2N9hdV4q7luEf5X9E1gxeYR0?usp=sharing
**Colaboratory**

**AMPL Model in Notebook Cell**

```
[ ] %%%ampl_eval
set PRODUCTS;
set NODES;
set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;

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);
```
Colaboratory

Python Data for the Model

In a large-scale application, this would be read or derived from data sources external to the notebook.

```python
import pandas as pd

PRODUCTS = ['Bands', 'Coils']
NODES = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']

capacity = pd.DataFrame(
    [
        [100, 80, 120],
        [120, 120, 120],
    ],
    columns=['Boston', 'New York', 'Seattle'],
    index=['Detroit', 'Denver'],
).stack().to_frame(name='capacity')

inflow = pd.DataFrame(
    ...
)
Colaboratory

Passing the Data to AMPL

```plaintext
# product and node sets
AMPL.set("PRODUCTS") = PRODUCTS
AMPL.set("NODES") = NODES

# arc set and capacity
AMPL.set_data(capacity, "ARCS")

# inflow
AMPL.param("inflow") = inflow

# min_ship, max_arcs
AMPL.param("min_ship") = min_ship
AMPL.param("max_arcs") = max_arcs

# var_cost, fix_cost
AMPL.param("var_cost") = var_cost
AMPL.eval("data; param fix_cost default 75;")
```

0s completed at 1:28 PM