Model-Based Optimization with AMPL: New Connections to Analytics Tools and Environments

Robert Fourer
4er@ampl.com
AMPL Optimization Inc.
www.ampl.com — +1 773-336-2675

INFORMS Annual Meeting
Seattle — 20-23 October 2019
Session TD58a, Technology Tutorials
Outline

Why AMPL?

- Mathematical optimization:
  Model-based vs. method-based approaches
- Model-based optimization:
  Modeling language vs. programming language approaches
- Modeling languages for optimization:
  Declarative vs. executable approaches

New in AMPL

- Direct spreadsheet interface
- Solver callbacks
- Jupyter notebooks
- Beyond the desktop . . .
Mathematical Optimization
Approaches to Optimization

Method-based approach
- Program a method (algorithm) for computing solutions

Model-based approach
- Formulate a description (model) of the desired solutions

Which should you prefer?
- For simple problems, any approach can be easy
- But real optimization problems must be revised . . .
  - to get the formulation right
  - to address new client requirements
  - to address new circumstances
**Example:**
Multi-Product Optimal 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
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

Consider the two approaches . . .
Method-Based Approach

Program a method to build a shipping plan

- a method says how to compute a solution

Order-driven

- Develop rules for how each order should be met
  - Given some demand and given available capacity, determine where to ship it from and which route to use
  - Fill orders one by one, according to the rules
    - Decrement capacity as each one is filled

Route-driven

- Repeat until all demands are met
  - Choose a shipping route and a product
  - Add as much flow as possible of that product along that route without exceeding supply, demand, or capacity

Program refinements to the method to get better results...
Multi-Product Flow

Method-Based Refinements

Develop rules for choosing good routes
- Generate batches of routes
- Apply routes in some systematic order

Improve the initial solution
- *Local optimization*: swaps and other simple improvements
- *Local-search metaheuristics*:
  - simulated annealing, tabu search, GRASP
- *Population-based metaheuristics*:
  - evolutionary methods, particle swarm optimization
Model-Based Approach

**Formulate a minimum shipping cost model**

- **model** says what conditions a solution should satisfy
- Identify amounts shipped as the decisions of the model (*variables*)
- Specify feasible shipment amounts by writing equations that the variables must satisfy (*constraints*)
- Write total shipping cost as a summation over the variables (*objective*)
- Collect costs, capacities, supplies, demands (*data*)

**Send to a solver that computes optimal solutions**

- Handles broad problem classes efficiently
  - Ex: Linear constraints and objective, continuous or integer variables
- Recognizes and exploits special cases
- Available ready to run, without programming
**Multi-Product Flow**

**Model-Based Formulation**

**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

Model-Based Formulation (cont’d)

Determine

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

_to minimize_

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

total cost of shipping

_subject to_

\[
\sum_{p \in P} X_{pij} \leq u_{ij}, \text{ for all } (i, j) \in A
\]
on each arc, total shipped must not exceed capacity

\[
\sum_{(i, j) \in A} X_{pij} + s_{pj} = \sum_{(j, i) \in A} X_{pji}, \text{ for all } p \in P, j \in N
\]
at each node, shipments in plus
supply/demand must equal shipments out
Example revised:
Complications in Multi-Product Flow

Additional restrictions imposed by the user
- Cost has fixed and variable parts
  - Each arc incurs a cost if it is used for shipping
- Shipments cannot be too small
- Not too many arcs can be used

Additional data for the problem
- \(d_{ij}\) fixed cost for using the arc from \(i\) to \(j\), for each \((i, j) \in A\)
- \(m\) smallest total that may be shipped on any arc used
- \(n\) largest number of arcs that may be used
Complications

Method-Based (cont’d)

What has to be done?
- Revise or re-think the solution approach
- Update or re-implement the algorithm

What are the challenges?
- In this example,
  - Shipments have become more interdependent
  - Good routes are harder to identify
  - Improvements are harder to find
- In general,
  - Even small revisions to a problem can necessitate major changes to the method and its implementation
  - Each problem revision requires more method development

... and revisions are frequent!
Model-Based (cont’d)

What has to be done?
- Update the objective expression
- Formulate additional constraint equations
- Send back to the solver

What are the challenges?
- In this example,
  - New variables and expressions to represent fixed costs
  - New constraints to impose shipment and arc-use limits
- In general,
  - The formulation tends to get more complicated
  - A new solver type or solver options may be needed

...but it’s easier to revise formulations than methods
...and a few solver types handle most cases
Model-Based Formulation (revised)

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
Complications

Model-Based Formulation (revised)

Subject to

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

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

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

shipments in plus supply/demand must equal shipments out

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

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

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

At most \( n \) arcs can be used
Method-Based Remains Popular for . . .

Applications of heuristic methods

- Simple heuristics
  - Greedy algorithms, local improvement methods
- Metaheuristics
  - Evolutionary methods, simulated annealing, tabu search, GRASP, . . .

Situations hard to formulate mathematically

- Difficult combinatorial constraints
- Black-box objectives and constraints

Extremely large, intensive applications

- Routing huge fleets of delivery trucks
- Finding shortest routes in mapping apps
- Training neural networks on gigantic datasets

. . . and it appeals to programmers
Model-Based Has Been Adopted in . . .

Diverse industries

- Manufacturing, distribution, supply-chain management
- Air and rail operations, trucking, delivery services
- Medicine, medical services
- Refining, electric power flow, gas pipelines, hydropower
- Finance, e-commerce, . . .
Model-Based Has Been Adopted in . . .

Diverse industries

Diverse fields

- Operations research & management science
- Business analytics
- Engineering & science
- Economics & finance
Model-Based Has Been Adopted by . . .

*Diverse industries*

*Diverse fields*

*Diverse kinds of users*

- Anyone who took an “optimization” class
- Anyone else with a technical background
- Newcomers to optimization

*These have in common . . .*

- Analysts inclined toward modeling; focus is
  - more on *what* should be solved
  - less on *how* it should be solved
- Good algebraic formulations for off-the-shelf solvers
- Emphasis on fast prototyping and long-term maintenance
**Approaches to Model-Based Optimization**

*Two forms of an optimization problem*
- Modeler’s form
  - Mathematical description, easy for people to work with
- Solver’s form
  - Explicit data structure, easy for solvers to compute with

**Programming language approach**
- Write a *program* to generate the solver’s form

**Modeling language approach**
- Write a *model formulation* in a language that a computer can read and translate
Programming Language Approach

Write a program to generate the solver’s form
- Read data and compute objective & constraint coefficients
- Send the solver the data structures it needs
- Receive solution data structure for viewing or processing

Some attractions
- Ease of embedding into larger systems
- Access to advanced solver features

Serious disadvantages
- Difficult environment for modeling
  * program does not resemble the modeler’s form
  * model is not separate from data
- Very slow modeling cycle
  * hard to check the program for correctness
  * hard to distinguish modeling from programming errors
Modeling Language Approach

Use a computer language to describe the modeler’s form

- Write your model
- Prepare data for the model
- Let the computer translate to & from the solver’s form

Manageable drawbacks

- Need to learn a new language
- Incur overhead in translation
- Create valuable formulations that must be protected?

Great advantages

- Faster modeling cycles
- More reliable modeling
- More maintainable applications
Algebraic Modeling Languages

Most popular today

- Computer language based on *algebraic* formulations
- Both familiar and general

**Determine**

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

**to minimize**

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

total cost of shipping

**subject to**

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

on each arc, total shipped must not exceed capacity

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

at each node, shipments in plus
supply/demand must equal shipments out
Approaches to Modeling Languages for Optimization

Executable approach
- Simulate an algebraic modeling language inside a general-purpose programming language
- Redefine operators like + and <= to return constraint objects rather than simple values

Declarative approach
- Design a language specifically for optimization modeling
- Extend with basic programming concepts: loops, tests, assignments
- Access from popular programming languages via APIs
Example:
Multi-Product Optimal Network Flow

Executable approach: gurobipy
- Based on the Python programming language
- Generates problems for the Gurobi solver

Declarative approach: AMPL
- Based on algebraic notation (like our sample formulation)
- Designed specifically for optimization
- Generates problems for Gurobi and other solvers
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

Statements: Data

gurobipy

- Assign values to Python lists and dictionaries

products = ['Pencils', 'Pens']

nodes = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']

arcs, capacity = multidict({
    ('Detroit', 'Boston'): 100,
    ('Detroit', 'New York'): 80,
    ('Detroit', 'Seattle'): 120,
    ('Denver', 'Boston'): 120,
    ('Denver', 'New York'): 120,
    ('Denver', 'Seattle'): 120
})

- Provide data later in a separate file

AMPL

- Define symbolic model sets and parameters

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

set PRODUCTS := Pencils Pens ;
set NODES := Detroit Denver
    Boston 'New York' Seattle ;

param: ARCS: capacity:
    Boston 'New York' Seattle :=
    Detroit 100 80 120
    Denver 120 120 120 ;
## Multi-Product Flow

### Statements: Data (cont’d)

**gurobipy**

<table>
<thead>
<tr>
<th>inflow =</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>('Pencils', 'Detroit'):</td>
<td>50,</td>
<td></td>
</tr>
<tr>
<td>('Pencils', 'Denver'):</td>
<td>60,</td>
<td></td>
</tr>
<tr>
<td>('Pencils', 'Boston'):</td>
<td>-50,</td>
<td></td>
</tr>
<tr>
<td>('Pencils', 'New York'):</td>
<td>-50,</td>
<td></td>
</tr>
<tr>
<td>('Pencils', 'Seattle'):</td>
<td>-10,</td>
<td></td>
</tr>
<tr>
<td>('Pens', 'Detroit'):</td>
<td>60,</td>
<td></td>
</tr>
<tr>
<td>('Pens', 'Denver'):</td>
<td>40,</td>
<td></td>
</tr>
<tr>
<td>('Pens', 'Boston'):</td>
<td>-40,</td>
<td></td>
</tr>
<tr>
<td>('Pens', 'New York'):</td>
<td>-30,</td>
<td></td>
</tr>
<tr>
<td>('Pens', 'Seattle'):</td>
<td>-30</td>
<td></td>
</tr>
</tbody>
</table>

**AMPL**

```
param inflow {PRODUCTS,NODES};

param inflow (tr):
    Pencils  Pens :=
    Detroit  50   60
    Denver   60   40
    Boston   -50  -40
    'New York' -50  -30
    Seattle -10  -30 ;
```
Statements: Data (cont’d)

gurobipy

```python
cost = {
    ('Pencils', 'Detroit', 'Boston'): 10,
    ('Pencils', 'Detroit', 'New York'): 20,
    ('Pencils', 'Detroit', 'Seattle'): 60,
    ('Pencils', 'Denver', 'Boston'): 40,
    ('Pencils', 'Denver', 'New York'): 40,
    ('Pencils', 'Denver', 'Seattle'): 30,
    ('Pens', 'Detroit', 'Boston'): 20,
    ('Pens', 'Detroit', 'New York'): 20,
    ('Pens', 'Detroit', 'Seattle'): 80,
    ('Pens', 'Denver', 'Boston'): 60,
    ('Pens', 'Denver', 'New York'): 70,
    ('Pens', 'Denver', 'Seattle'): 30
}
```
Multi-Product Flow

Statements: Data (cont’d)

AMPL

```
param cost {PRODUCTS,ARCS} >= 0;

param cost {
[Pencils,*,*] (tr) Detroit Denver :=
    Boston  10  40
    'New York'  20  40
    Seattle  60  30

[Pens,*,*] (tr) Detroit Denver :=
    Boston  20  60
    'New York'  20  70
    Seattle  80  30
};
```
**Multi-Product Flow**

**Formulation: Model**

**Determine**

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

**to minimize**

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

total cost of shipping

**subject to**

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

total shipped on each arc must not exceed capacity

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

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

Statements: Model

---

**gurobipy**

```python
m = Model('netflow')
flow = m.addVars(products, arcs, obj=cost, name="flow")
m.addConstrs(
    (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")
m.addConstrs(
    (flow.sum(p,'*',j) + inflow[p,j] == flow.sum(p,j,'*')
    for p in products for j in nodes), "node")

for i,j in arcs:
    m.addConstr(sum(flow[p,i,j] for p in products) <= capacity[i,j],
            "cap[%s,%s]" % (i,j))

m.addConstrs(
    (quicksum(flow[p,i,j] for i,j in arcs.select('*',j)) + inflow[p,j] ==
    quicksum(flow[p,j,k] for j,k in arcs.select(j,'*'))
    for p in products for j in nodes), "node")
```

---

alternatives

---
(Note on Summations)

gurobipy quicksum

```python
m.addConstrs(
    (quicksum(flow[p,i,j] for i,j in arcs.select('*','j')) + inflow[p,j] ==
     quicksum(flow[p,j,k] for j,k in arcs.select('j','*'))
    for p in commodities for j in nodes), "node")
```

**quicksum** (data)

A version of the Python `sum` function that is much more efficient for building large Gurobi expressions (LinExpr or QuadExpr objects). The function takes a list of terms as its argument.

Note that while quicksum is much faster than `sum`, it isn’t the fastest approach for building a large expression. Use `addTerms` or the `LinExpr()` constructor if you want the quickest possible expression construction.
Multi-Product Flow

Statements: Model (cont’d)

AMPL

\[
\begin{align*}
\text{var } & \text{ Flow } \{\text{PRODUCTS}, \text{ARCS}\} \geq 0; \\
\text{minimize } & \text{ TotalCost:} \\
& \text{sum } \{p \text{ in } \text{PRODUCTS}, (i,j) \text{ in } \text{ARCS}\} \text{ cost}[p,i,j] \times \text{Flow}[p,i,j]; \\
\text{subject to } & \text{ Capacity } \{(i,j) \text{ in } \text{ARCS}\}: \\
& \text{sum } \{p \text{ in } \text{PRODUCTS}\} \text{ Flow}[p,i,j] \leq \text{capacity}[i,j]; \\
\text{subject to } & \text{ Conservation } \{p \text{ in } \text{PRODUCTS}, j \text{ in NODES}\}: \\
& \text{sum } \{(i,j) \text{ in } \text{ARCS}\} \text{ Flow}[p,i,j] + \text{inflow}[p,j] = \\
& \text{sum } \{(j,i) \text{ in } \text{ARCS}\} \text{ Flow}[p,j,i]; \\
\end{align*}
\]

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

Solution

gurobipy

```python
m.optimize()
if m.status == GRB.Status.OPTIMAL:
    solution = m.getAttr('x', flow)
    for p in products:
        print('
Optimal flows for %s:' % p)
        for i,j in arcs:
            if solution[p,i,j] > 0:
                print('%s -> %s: %g' % (i, j, solution[p,i,j]))
```

Solved in 0 iterations and 0.00 seconds
Optimal objective 5.500000000e+03

Optimal flows for Pencils:
Detroit -> Boston: 50
Denver -> New York: 50
Denver -> Seattle: 10

Optimal flows for Pens: ...
Multi-Product Flow

Solution (cont’d)

AMPL

```AMPL
AMPL

ampl: model netflow.mod;
ampl: data netflow.dat;

option solver gurobi;
ampl: solve;

Gurobi 8.1.0: optimal solution; objective 5500
2 simplex iterations

ampl: display Flow;

Flow [Pencils,*,*]
: Boston 'New York' Seattle :=
Denver  0  50  10
Detroit 50  0   0

[Pens,*,*]
: Boston 'New York' Seattle :=
Denver 10  0  30
Detroit 30 30  0

;
```
Multi-Product Flow

Solution (cont’d)

AMPL

```AMPL
ampl: model netflow.mod;
ampl: data netflow.dat;

option solver cplex;
ampl: solve;

CPLEX 12.9.0.0: optimal solution; objective 5500
0 dual simplex iterations (0 in phase I)
ampl: display Flow;

Flow [Pencils,*,*]
  : Boston 'New York' Seattle :=
Denver 0 50 10
Detroit 50 0 0

[Pens,*,*]
  : Boston 'New York' Seattle :=
Denver 10 0 30
Detroit 30 30 0
```


Multi-Product Flow

Solution (cont’d)

AMPL

```
AMPL: model netflow.mod;
AMPL: data netflow.dat;

option solver xpress;
AMPL: solve;

XPRESS 8.6.0(32.01.08): Optimal solution found
Objective 5500, 1 simplex iteration

AMPL: display Flow;

Flow [Pencils,*,*]
:       Boston 'New York' Seattle :=
Denver      0       50       10
Detroit    50        0        0

[Pens,*,*]
:       Boston 'New York' Seattle :=
Denver    10        0       30
Detroit   30       30        0
```

Integration with Applications

gurobipy

- Everything can be developed in Python
  - Extensive data, visualization, deployment tools available
- Limited modeling features also in C++, C#, Java

AMPL

- Modeling language extended with loops, tests, assignments
- Application programming interfaces (APIs) for calling AMPL from C++, C#, Java, MATLAB, Python, R
  - Efficient methods for data interchange
Integration with Solvers

gurobipy

- Works closely with the Gurobi solver: callbacks during optimization, fast re-solves after problem changes
- Supports Gurobi’s extended expressions: min/max, and/or, if-then-else, univariate nonlinear

AMPL

- Supports all popular solvers
- Extends to general nonlinear and logic expressions
  - Connects to nonlinear function libraries and user-defined functions
  - Automatically computes nonlinear function derivatives
  - Connects to global optimization and constraint programming solvers
Multi-Product Flow

Complications

Easily accommodated

- Add variables to the model
- Add a term to the objective
- Update one constraint and add two
- Send to the same solver
New in AMPL

*Direct spreadsheet interface*

*Solver callbacks*

*Jupyter notebooks*

*Beyond the desktop . . .
Direct spreadsheet interface

Read & write any .xlsx file

- Independent of the spreadsheet software used
- Works on all popular platforms (Windows, Linux, macOS)
- Bypasses database drivers such as ODBC

Use existing AMPL data-interface statements

- `table` for making associations between AMPL model parameters and spreadsheet data
- `read table` and `write table` for importing and exporting data
Example: Multi-Product Flow

Model

set PRODUCTS;
set NODES;
set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;
param inflow {PRODUCTS,NODES};
param cost {PRODUCTS,ARCS} >= 0;
var Flow {PRODUCTS,ARCS} >= 0;

minimize TotalCost:
    sum {p in PRODUCTS, (i,j) in ARCS} 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] + inflow[p,j] =
    sum {(j,i) in ARCS} Flow[p,j,i];
Example: Multi-Product Flow

Data in text file

```plaintext
set PRODUCTS := Pencils Pens ;
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 :=
    Pencils   50    60  -50   -50  -10
    Pens      60    40  -40   -30  -30;

param cost:

    [Pencils,*,*] Boston 'New York' Seattle :=
    Detroit    10    20    60
    Denver     40    40    30

    [Pens,*,*]  Boston 'New York' Seattle :=
    Detroit    20    20    80
    Denver     60    70    30 ;
```
**Direct spreadsheet interface**

**Example: Multi-Product Flow**

**Data in spreadsheet file**

<table>
<thead>
<tr>
<th></th>
<th>Items</th>
<th>FROM</th>
<th>TO</th>
<th>capacity</th>
<th>ITEMS</th>
<th>FROM</th>
<th>TO</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Pencils</td>
<td>Detroit</td>
<td>Boston</td>
<td>100</td>
<td>Pencils</td>
<td>Detroit</td>
<td>Boston</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Pens</td>
<td>Detroit</td>
<td>New York</td>
<td>80</td>
<td>Pencils</td>
<td>Detroit</td>
<td>New York</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Detroit</td>
<td>Detroit</td>
<td>Seattle</td>
<td>120</td>
<td>Pencils</td>
<td>Detroit</td>
<td>Seattle</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>New York</td>
<td>Denver</td>
<td>Boston</td>
<td>120</td>
<td>Pencils</td>
<td>Denver</td>
<td>Boston</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>Seattle</td>
<td>Denver</td>
<td>New York</td>
<td>120</td>
<td>Pencils</td>
<td>Denver</td>
<td>New York</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>Denver</td>
<td>Denver</td>
<td>Seattle</td>
<td>120</td>
<td>Pencils</td>
<td>Denver</td>
<td>Seattle</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>Boston</td>
<td>Pens</td>
<td>Detroit</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Pencils</td>
<td>New York</td>
<td>Seattle</td>
<td>inflow</td>
<td>Pens</td>
<td>Detroit</td>
<td>New York</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Pencils</td>
<td>Detroit</td>
<td>Denver</td>
<td>50</td>
<td>Pens</td>
<td>Detroit</td>
<td>Seattle</td>
<td>80</td>
</tr>
<tr>
<td>12</td>
<td>Pencils</td>
<td>Denver</td>
<td>Boston</td>
<td>60</td>
<td>Pens</td>
<td>Denver</td>
<td>Boston</td>
<td>60</td>
</tr>
<tr>
<td>13</td>
<td>Pencils</td>
<td>Boston</td>
<td>Denver</td>
<td>-50</td>
<td>Pens</td>
<td>Denver</td>
<td>New York</td>
<td>70</td>
</tr>
<tr>
<td>14</td>
<td>Pencils</td>
<td>New York</td>
<td>Denver</td>
<td>-50</td>
<td>Pens</td>
<td>Denver</td>
<td>New York</td>
<td>70</td>
</tr>
<tr>
<td>15</td>
<td>Pens</td>
<td>Seattle</td>
<td>Pens</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Pens</td>
<td>Detroit</td>
<td>Pens</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Pens</td>
<td>Denver</td>
<td>Pens</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Pens</td>
<td>Boston</td>
<td>Pens</td>
<td>-40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Pens</td>
<td>New York</td>
<td>Pens</td>
<td>-30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Pens</td>
<td>Seattle</td>
<td>Pens</td>
<td>-30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Multi-Product Flow

Script file (input)

```plaintext
model netflow1.mod;

table Products IN "amplxl" "netflow1.xlsx" "Items":
    PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow1.xlsx":
    NODES <- [NODES];

table Capacity IN "amplxl" "netflow1.xlsx":
    ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow1.xlsx":
    [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow1.xlsx":
    [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```
Example: Multi-Product Flow

Script file (output)

```plaintext
option solver gurobi;
solve;

table Results OUT "amplxl" "netflow1.xlsx":
    [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;
```
**Direct spreadsheet interface**

**Example: Multi-Product Flow**

**Results in spreadsheet file**

![Spreadsheet Image]

<table>
<thead>
<tr>
<th>ITEMS</th>
<th>FROM</th>
<th>TO</th>
<th>Flow</th>
<th>FROM</th>
<th>TO</th>
<th>TotFlow</th>
<th>%Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pencils</td>
<td>Detroit</td>
<td>Boston</td>
<td>50</td>
<td>Detroit</td>
<td>Boston</td>
<td>80</td>
<td>0.8</td>
</tr>
<tr>
<td>Pencils</td>
<td>Detroit</td>
<td>New York</td>
<td>0</td>
<td>Detroit</td>
<td>New York</td>
<td>30</td>
<td>0.375</td>
</tr>
<tr>
<td>Pencils</td>
<td>Detroit</td>
<td>Seattle</td>
<td>0</td>
<td>Detroit</td>
<td>Seattle</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pencils</td>
<td>Denver</td>
<td>Boston</td>
<td>0</td>
<td>Denver</td>
<td>Boston</td>
<td>10</td>
<td>0.08333</td>
</tr>
<tr>
<td>Pencils</td>
<td>Denver</td>
<td>New York</td>
<td>50</td>
<td>Denver</td>
<td>New York</td>
<td>50</td>
<td>0.41667</td>
</tr>
<tr>
<td>Pens</td>
<td>Detroit</td>
<td>Seattle</td>
<td>10</td>
<td>Denver</td>
<td>Seattle</td>
<td>40</td>
<td>0.33333</td>
</tr>
<tr>
<td>Pens</td>
<td>Detroit</td>
<td>New York</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pens</td>
<td>Detroit</td>
<td>Seattle</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pens</td>
<td>Denver</td>
<td>Boston</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pens</td>
<td>Denver</td>
<td>New York</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pens</td>
<td>Denver</td>
<td>Seattle</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Direct spreadsheet interface

And There’s More . . .

All existing features supported

- Indexed collections of tables
- Dynamic file, range & header names in tables
- \texttt{readtable}, \texttt{writetable} in loops and conditionals

To come: Data not limited to relational tables

- Support for two-dimensional spreadsheet tables
- Extensions for handling higher-dimensional data
Solver Callbacks

Current example

- AMPL Python API (amplpy, from us)
- Gurobi Python API (gurobipy, from Gurobi Optimization)

Coming soon

- AMPL Python API (amplpy, from us)
- AMPL Gurobi connector (amplpy_gurobi, from us)

... connectors for other solvers, too
AMPL Python API

**Principles**

- APIs for “all” popular languages
  - C++, C#, Java, MATLAB, Python, R
- Common overall design
- Common implementation core in C++
- Customizations for each language and its data structures

**Python support: amplpy**

- Versions: 2.7, 3.3 and up
- Data structures: Lists, dictionaries, dataframes
- Libraries: Pandas, Bokeh
- Easy installation: `pip install amplpy`

**Example**

- Roll cutting by pattern generation . . .
Iterative scheme: Solve a series of problems

- Solve continuous relaxation using subset of “easy” patterns
- Add “most promising” pattern to the subset
  - Minimize reduced cost given dual values
  - Equivalent to a one-constraint (“knapsack”) problem
- Iterate as long as there are promising patterns
  - Stop when minimum reduced cost is zero
- Form integer program using all patterns found
  - Apply a solver for a “reasonable” amount of time
  - Return the best (possibly optimal) solution found

...using a callback to implement a user-specified stopping rule
Roll Cutting Implementation

Logic
- Iterative scheme in Python
- Modeling and solving in AMPL, via API calls
- Solution reporting in Python

AMPL objects
- **Master** is the cutting model with current pattern subset
- **Sub** is the one-constraint knapsack problem
Python Callbacks from Gurobi

Example: User-Specified Stopping Rule

Data
- Times $t_1 < t_2 < t_3$ etc.
- Optimality gap tolerances $g_1 < g_2 < g_3$ etc.

Execution
- When elapsed time reaches $t_i$ . . .
- Increase the gap tolerance to $g_i$
Jupyter Notebooks

Support for all parts of an AMPL API application

- Python code cells
- Python data cells
- AMPL model cells
Beyond the Desktop

*Alternative computing environments*

- Cloud computing services
- High-performance compute clusters
- Containers

*Alternatives for access to AMPL*

- Streamlined / flexible licensing
- Free AMPL for courses with no licensing worries
- AMPL web server (*coming soon*)
Streamlined / Flexible Licensing

Flexible licensing for alternative computing environments

- For academic research and business applications
- Contact us to discuss your needs

Streamlined installation for traditional licensing setups

- Get a token, issue a command

---

You can download and setup a license for AMPL with a single command as follows:

- Linux/macOS:
  ```bash
eval "$\{\text{curl -sSL get.ampl.online/nix.sh\}"
```

- Windows:
Streamlined / Flexible Licensing

Flexible licensing for alternative computing environments
- For academic research and business applications
- Contact us to discuss your needs

Streamlined installation for traditional licensing setups
- Get a token, issue a command
AMPL for Courses

Streamlined for quick setup

- Short online application form for each course offering
- AMPL & solvers in one compressed file for each platform
  - No problem size limitations
- Freely install on any computer supporting the course
- Freely distribute to students for their own computers
  - Times out after your specified course end date

Includes top-quality solvers

- CPLEX, Gurobi, Xpress, Knitro, BARON, MINOS, ILOG CP, SNOPT, CONOPT, LOQO, LGO, (soon) LINDO Global

Used this year in 685 courses, 312 universities, 53 countries

- Details and application form at ampl.com/courses.html
**AMPL Cloud Services** *(coming soon)*

**Development environment**

- AMPL modeling environment in a web browser
  - Selection of solvers
  - Tour of examples
  - Up to 1000 concurrent users
- User accounts with file storage
  - For trial and purchase

**Deployment alternatives**

- Support for solver cloud platforms
  - Gurobi Instant Cloud available now
- AMPL cloud platforms under development
  
  
  . . . contact us for details

... contact us for details