New Connections to the AMPL Modeling Language: Spreadsheets and Callbacks

Robert Fourer

4er@ampl.com

AMPL Optimization Inc.
www.ampl.com — +1 773-336-6273

31st European Conference on Operational Research

Athens, Greece (online) — **11-14 July 2021**
Software for Optimization, Session TD-53: Modelling Tools I

New Connections to the AMPL Modeling Language: Spreadsheets and Callbacks

Optimization applications are often concerned as much with making connections as with building models. This presentation describes two connections recently implemented in the AMPL modeling language and system. A direct spreadsheet connection reads and writes xlsx-format files, defining correspondences between common spreadsheet layouts and AMPL's algebraic data definitions. Support is included for "two-dimensional" spreadsheet tables in which one index labels the columns and one or more indices label the rows. A solver callback connection enables AMPL's APIs to communicate with algorithms as they are running, uniting the ease of modeling in AMPL with the flexibility of programming to customize algorithmic behavior. This facility can be used to write specialized routines that report progress, change settings, and generate constraints that cut off fractional solutions.

Outline

Direct interface to spreadsheet files

- Implementation as a new AMPL table handler
- Example: Multicommodity network flow

Callbacks from solvers

- Implementation in AMPL APIs for Python, C++, C#
  - Python in Jupyter notebooks with AMPL cells
- Example: Custom solver stopping criterion
  - Optimal pattern selection for roll cutting
- Example: Generation of subtour elimination cuts
  - Minimum tour (TSP) of a network

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

Now testing . . .

- *Direct .csv file handler*

Direct spreadsheet interface

Network Flow *(symbolic 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 $p \in P$, $(i, j) \in A$

Network Flow *(symbolic 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

Direct spreadsheet interface

Network Flow in AMPL

Symbolic data and 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];
```

Direct spreadsheet interface

Data Instance

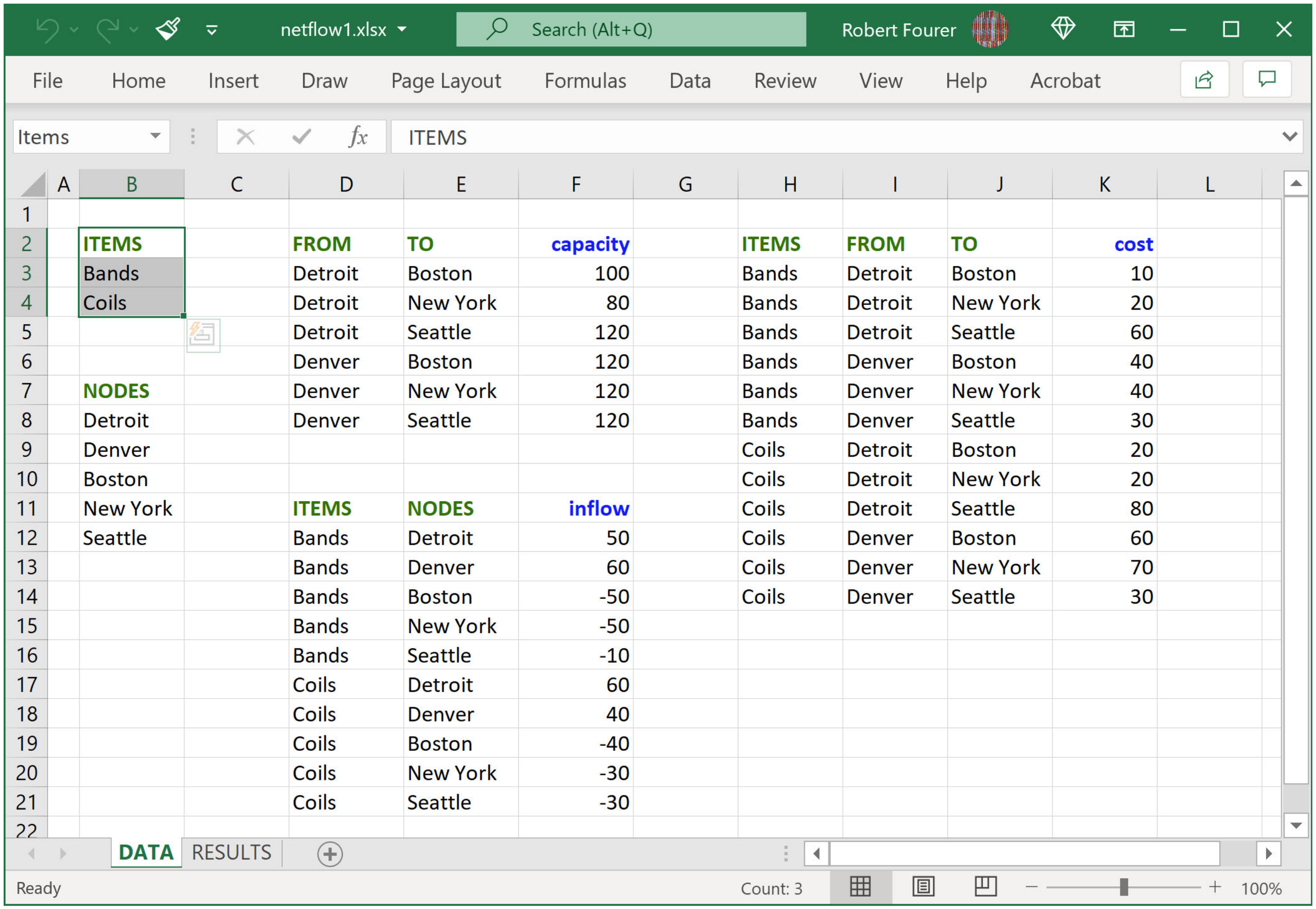
in AMPL text format

```
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 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 ;
```

Direct spreadsheet interface

Data Instance

in spreadsheet ranges



	A	B	C	D	E	F	G	H	I	J	K
1											
2		ITEMS		FROM	TO	capacity		ITEMS	FROM	TO	cost
3		Bands		Detroit	Boston	100		Bands	Detroit	Boston	10
4		Coils		Detroit	New York	80		Bands	Detroit	New York	20
5				Detroit	Seattle	120		Bands	Detroit	Seattle	60
6				Denver	Boston	120		Bands	Denver	Boston	40
7		NODES		Denver	New York	120		Bands	Denver	New York	40
8		Detroit		Denver	Seattle	120		Bands	Denver	Seattle	30
9		Denver						Coils	Detroit	Boston	20
10		Boston						Coils	Detroit	New York	20
11		New York		ITEMS	NODES	inflow		Coils	Detroit	Seattle	80
12		Seattle		Bands	Detroit	50		Coils	Denver	Boston	60
13				Bands	Denver	60		Coils	Denver	New York	70
14				Bands	Boston	-50		Coils	Denver	Seattle	30
15				Bands	New York	-50					
16				Bands	Seattle	-10					
17				Coils	Detroit	60					
18				Coils	Denver	40					
19				Coils	Boston	-40					
20				Coils	New York	-30					
21				Coils	Seattle	-30					
22											

Direct spreadsheet interface

Data Handling

Script file (input)

```
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;
```

Direct spreadsheet interface

Data Handling

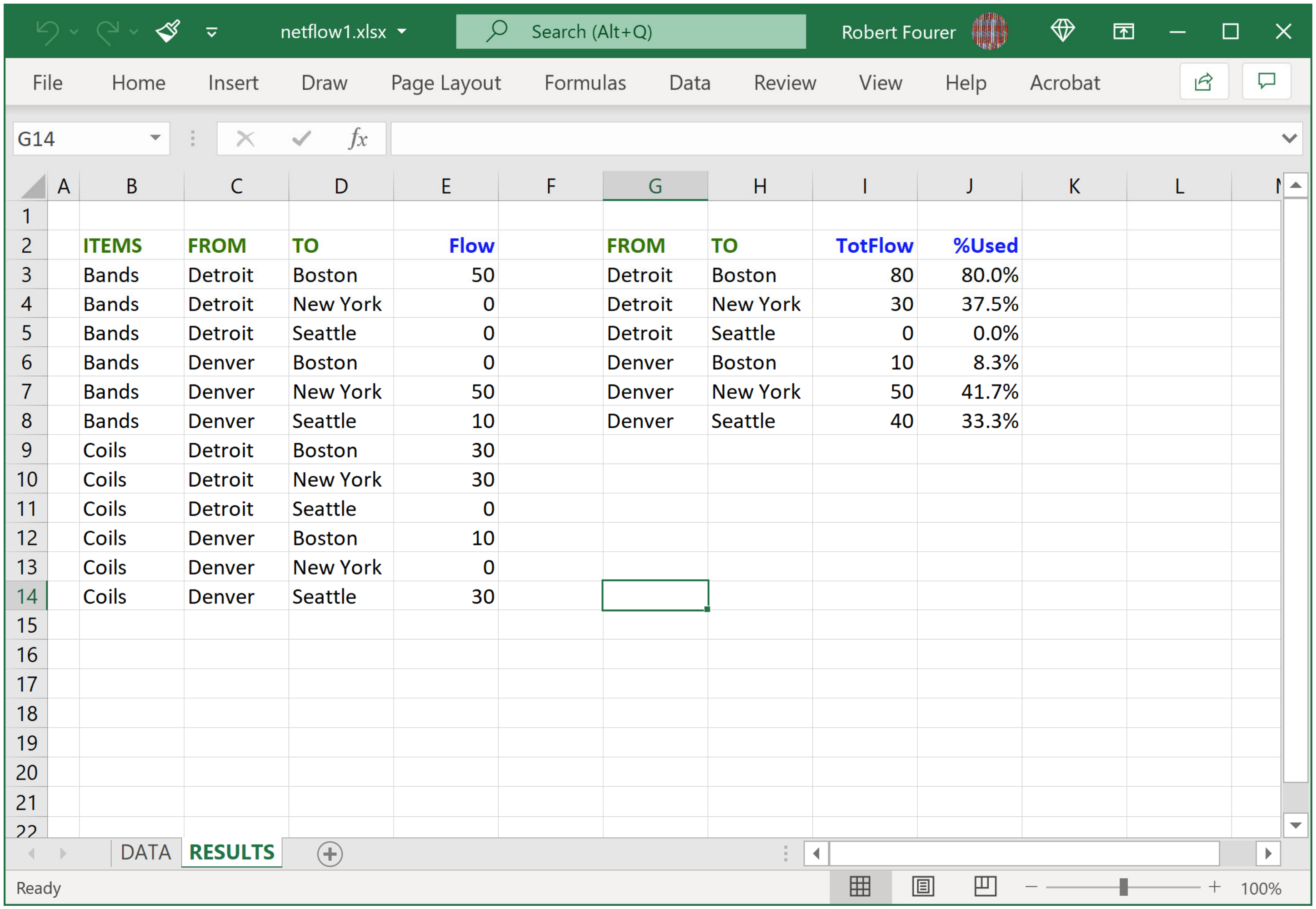
Script file (output)

```
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

Data Results

in spreadsheet ranges



ITEMS	FROM	TO	Flow		FROM	TO	TotFlow	%Used
Bands	Detroit	Boston	50		Detroit	Boston	80	80.0%
Bands	Detroit	New York	0		Detroit	New York	30	37.5%
Bands	Detroit	Seattle	0		Detroit	Seattle	0	0.0%
Bands	Denver	Boston	0		Denver	Boston	10	8.3%
Bands	Denver	New York	50		Denver	New York	50	41.7%
Bands	Denver	Seattle	10		Denver	Seattle	40	33.3%
Coils	Detroit	Boston	30					
Coils	Detroit	New York	30					
Coils	Detroit	Seattle	0					
Coils	Denver	Boston	10					
Coils	Denver	New York	0					
Coils	Denver	Seattle	30					

And There's More . . .

All existing features supported

- Indexed collections of tables
- Dynamic file, range & header names in tables
- **`read table`**, **`write table`** in loops and conditionals

New spreadsheet-specific features

- Recognize both sheet and range names
- Properly interpret empty data cells
- Process "two-dimensional" spreadsheet tables

Direct spreadsheet interface

Data Instance *(revisited)*

"1D" spreadsheet ranges

netflow1.xlsx

Capacity | FROM

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		ITEMS		FROM	TO	capacity		ITEMS	FROM	TO	cost	
3		Bands		Detroit	Boston	100		Bands	Detroit	Boston	10	
4		Coils		Detroit	New York	80		Bands	Detroit	New York	20	
5				Detroit	Seattle	120		Bands	Detroit	Seattle	60	
6				Denver	Boston	120		Bands	Denver	Boston	40	
7		NODES		Denver	New York	120		Bands	Denver	New York	40	
8		Detroit		Denver	Seattle	120		Bands	Denver	Seattle	30	
9		Denver						Coils	Detroit	Boston	20	
10		Boston						Coils	Detroit	New York	20	
11		New York		ITEMS	NODES	inflow		Coils	Detroit	Seattle	80	
12		Seattle		Bands	Detroit	50		Coils	Denver	Boston	60	
13				Bands	Denver	60		Coils	Denver	New York	70	
14				Bands	Boston	-50		Coils	Denver	Seattle	30	
15				Bands	New York	-50						
16				Bands	Seattle	-10						
17				Coils	Detroit	60						
18				Coils	Denver	40						
19				Coils	Boston	-40						
20				Coils	New York	-30						
21				Coils	Seattle	-30						
22												

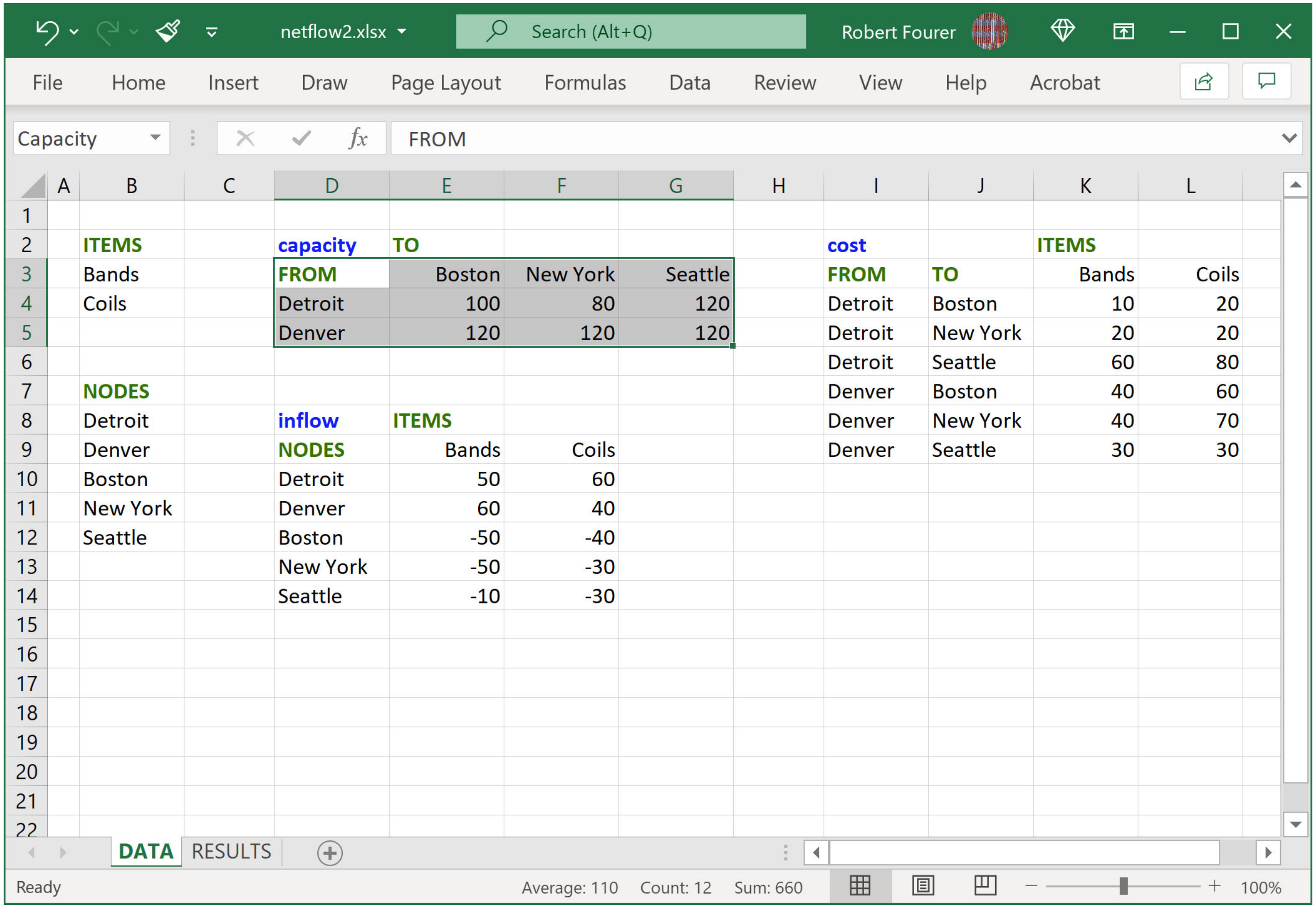
DATA | RESULTS

Ready — Average: 110 Count: 21 Sum: 660 — 100%

Direct spreadsheet interface

Data Instance *(revisited)*

"2D" spreadsheet ranges



Direct spreadsheet interface

Data Handling *(revisited)*

Script file (input)

```
model netflow1.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

Data Handling

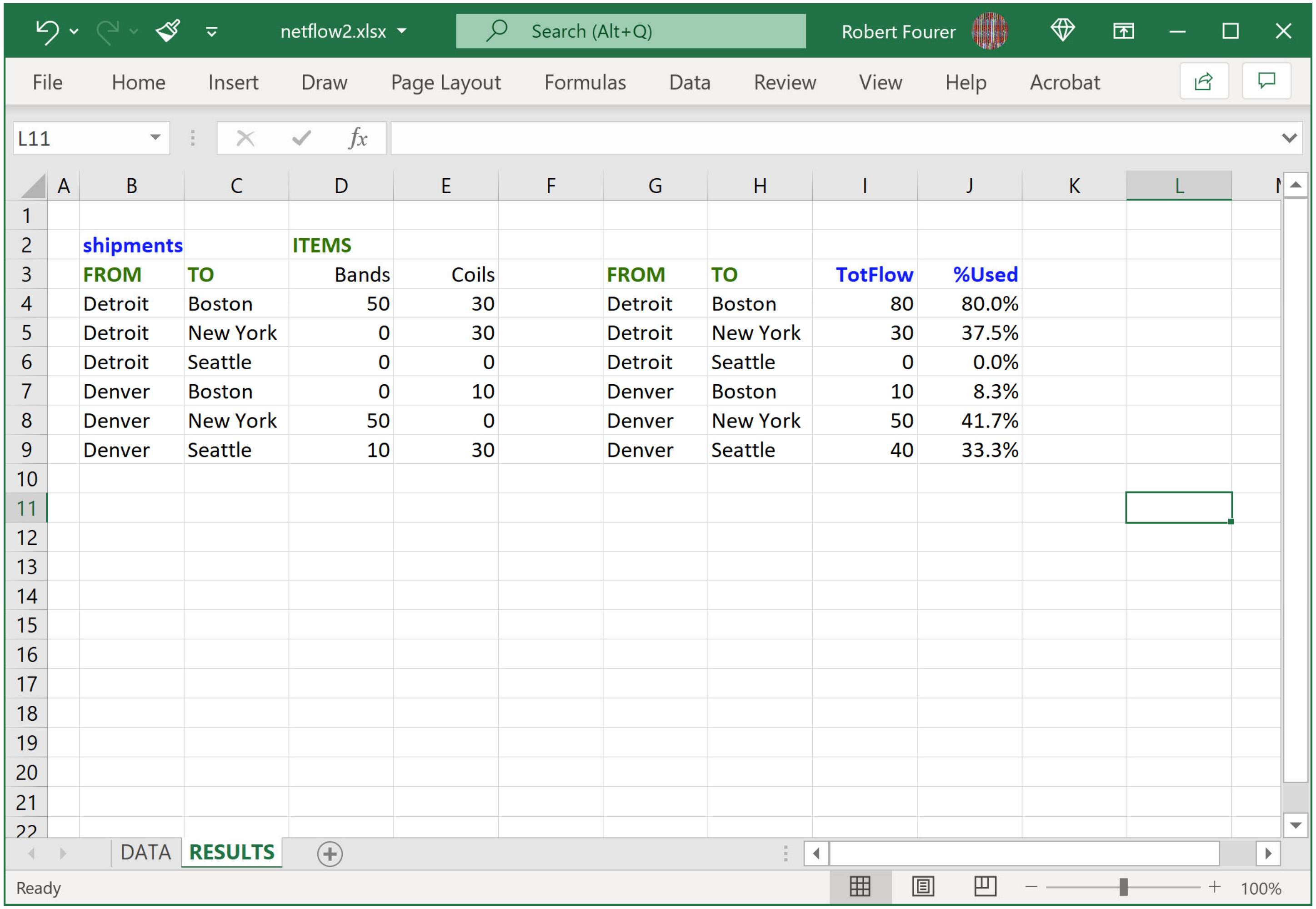
Script file (output)

```
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;
```

Direct spreadsheet interface

Data Results

"2D" spreadsheet ranges



	A	B	C	D	E	F	G	H	I	J
1										
2		shipments		ITEMS						
3		FROM	TO	Bands	Coils		FROM	TO	TotFlow	%Used
4		Detroit	Boston	50	30		Detroit	Boston	80	80.0%
5		Detroit	New York	0	30		Detroit	New York	30	37.5%
6		Detroit	Seattle	0	0		Detroit	Seattle	0	0.0%
7		Denver	Boston	0	10		Denver	Boston	10	8.3%
8		Denver	New York	50	0		Denver	New York	50	41.7%
9		Denver	Seattle	10	30		Denver	Seattle	40	33.3%

Callbacks from Solvers

Generic implementation in AMPL APIs

- In multiple languages: Python, C++, C#
- For multiple solvers: CPLEX, Gurobi, Xpress
  - based on our AMPL-enabled distribution
- With multiple *generic* callback types: MIPNODE, MIPSOL, . . .

Sample uses

- Implementing a solver stopping criterion
  - optimal pattern selection for roll cutting
- Adding user-generated constraints
  - minimum tour (TSP) of a network

Examples in Python

- Jupyter notebooks with AMPL cells

Pattern Selection for Roll Cutting

Given

- Raw rolls of a large (fixed) width
- Demands for various (smaller) ordered widths
- Selected cutting patterns that may be used

Determine

- Number of times to cut each pattern

So that

- Demands are met (or slightly exceeded)
- *Number of raw rolls cut* is minimized

Solution Strategy

Generate many "good" cutting patterns

- Example: solve knapsack subproblems

Solve integer program using all patterns generated

- Apply a solver for a "reasonable" amount of time
- Return the best (possibly optimal) solution found

. . . using a callback to implement an adaptive stopping rule

Stopping Rule

Data

- Times $t_1 < \mathrm{t}_2 < \mathrm{t}_3$ 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

Implementation Highlights

Stopping rule data in Python dictionary

```
stopdict = { 'time'   : (  15,    30,   60 ),
             'gaptol' : ( .0002, .002, .02 )
           }
```

Callback class (constructor)

```
class MyCallback(ampls.GenericCallback):

    def __init__(self, stoprule):
        super(MyCallback, self).__init__()
        self._stoprule = stoprule
        self._current = 0
        self._continueOpt = True

    .......
```

Stopping Rule

Implementation Highlights

Callback class (process callback, update gap tolerance)

```
def run(self):
    where = self.getAMPLWhere()
    if where == ampls.Where.MIPNODE:
        runtime = self.getValue(ampls.Value.RUNTIME).dbl
        if runtime >= self._stoprule['time'][self._current]:
            self._continueOpt = True
            return -1
    return 0

def setCurrentGap(self):
    gaptolpct = 100*self._stoprule['gaptol'][self._current]
    stoptime = self._stoprule['time'][self._current]
    print("Increasing gap tolerance to "
          f"{gaptolpct:.2f}% after {stoptime:.1f} seconds")
    ampls_model.setAMPLsParameter(ampls.SolverParams.DBL_MIPGap,
                                  self._stoprule['gaptol'][self._current])
    self._current += 1
```

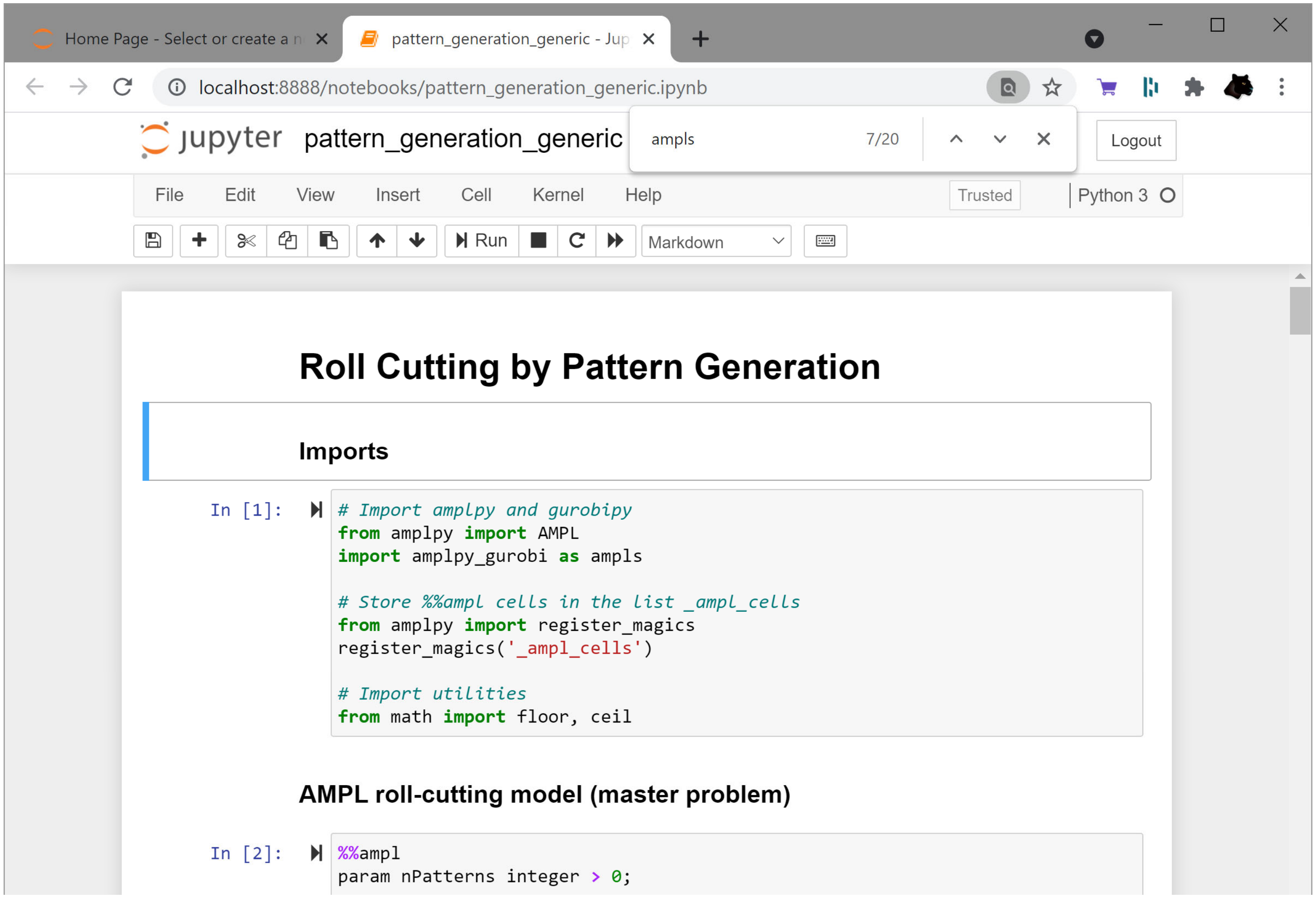
Stopping Rule

Implementation Highlights

Solve using callbacks

```
# Export
Master.option['relax_integrality'] = 0
ampls_model = Master.exportModel(solver, ["return_mipgap=5"])

# Initialize with stopping rule
callback = MyCallback(stopdict)
ampls_model.setCallback(callback)

# Invoke solver
while callback._continueOpt:
    callback._continueOpt = False
    ampls_model.optimize()
    if callback._continueOpt:
        callback.setCurrentGap()

# Import solution from solver
Master.importSolution(ampls_model)
```

Roll-Cutting Notebook



https://ampl.com/dl/Notebooks/patgen_callback.zip

Minimum Tour (TSP)

Given

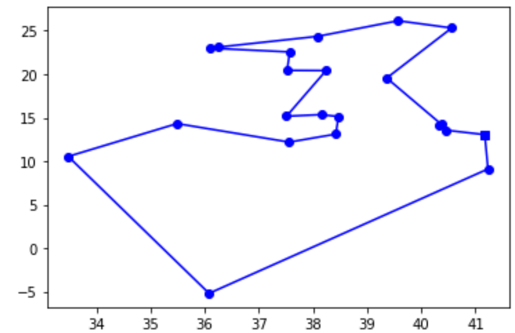
- A set of locations ("nodes")
- Distances between pairs of locations ("arcs")

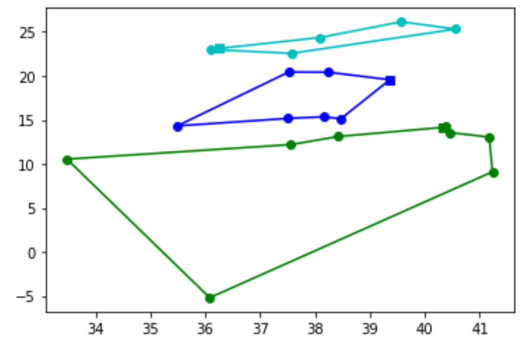
Choose

- A subset of arcs having minimum total distance

So that

- Exactly two arcs meet each node *(adjacency)*
- At least two arcs connect every subset of nodes to the other nodes *(subtour elimination)*





Solution Strategy

Start with only adjacency constraints

For each feasible solution found,

- Check for subtours
- Add an elimination constraint for each subtour found
- Continue with the optimization

. . . using a callback to add the elimination constraints

Implementation Highlights

Set a few execution parameters

```
solver = "cplex"
tspFile = "TSP/ch150.tsp"

PLOTSUBTOURS = True
```

Read TSPLIB file into a dictionary

```
def getDictFromTspFile(tspFile):

  p = tsp.load(tspFile)

  nnodes = len(list(p.get_nodes()))
  formatString = f"{{:0{ceil(log10(nnodes+1))}d}}"

  nodes = {formatString.format(value) : p.node_coords[index+1]
      for index, value in enumerate(p.get_nodes())}

  return nodes
```

Minimum Tour

Implementation Highlights

Create AMPL object, load model and data

```
# Create AMPL object and set solver

ampl = AMPL()
ampl.option["solver"] = solver

# Load model in AMPL

tspAMPLModel = _ampl_cells[0]
ampl.eval(tspAMPLModel)

# Read TSPLIB file and pass data to AMPL

nodes = getDictFromTspFile(tspFile)

df = DataFrame(index=[('NODES')], columns=['hpos', 'vpos'])
df.setValues(nodes)
ampl.setData(df, "NODES")
```

Minimum Tour

Implementation Highlights

Define callback

```
class MyCallback(ampls.GenericCallback):
    def __init__(self):
      # Constructor, simply sets the iteration number to 0
      super().__init__()
      self.iteration = 0

    def run(self):
      try:
         # For each solution
         if self.getAMPLWhere() == ampls.Where.MIPSOL:
            self.iteration += 1
            print(f"\nIteration {self.iteration}: Finding subtours")
            sol = self.getSolutionVector()
            arcs = [xvars[i] for i,value in enumerate(sol) if value > 0]
            subTours = findSubTours(set(arcs), set(vertices))
            if len(subTours) == 1:
                print("No subtours detected. Not adding any cut")
                return 0
```

Implementation Highlights

Define callback (cont'd)

```
class MyCallback(ampls.GenericCallback):
...
    def run(self):
    ...
             for subTour in subTours:
                st1 = set(subTour)
                nst1 = set(vertices) - st1
                externalArcs = [(i,j) if i < j else (j,i)
                                for i in st1 for j in nst1]
                varsExternalArcs = [xinverse[(i,j)]
                                    for (i,j) in externalArcs]
                coeffs = [1 for i in range(len(varsExternalArcs))]
                varsExternalArcs = sorted(varsExternalArcs)
                self.addLazyIndices(varsExternalArcs , coeffs,
                                    ampls.CutDirection.GE, 2)
                if len(subTours) == 2:
                   return 0
             print("Continue solving")
          return 0
```

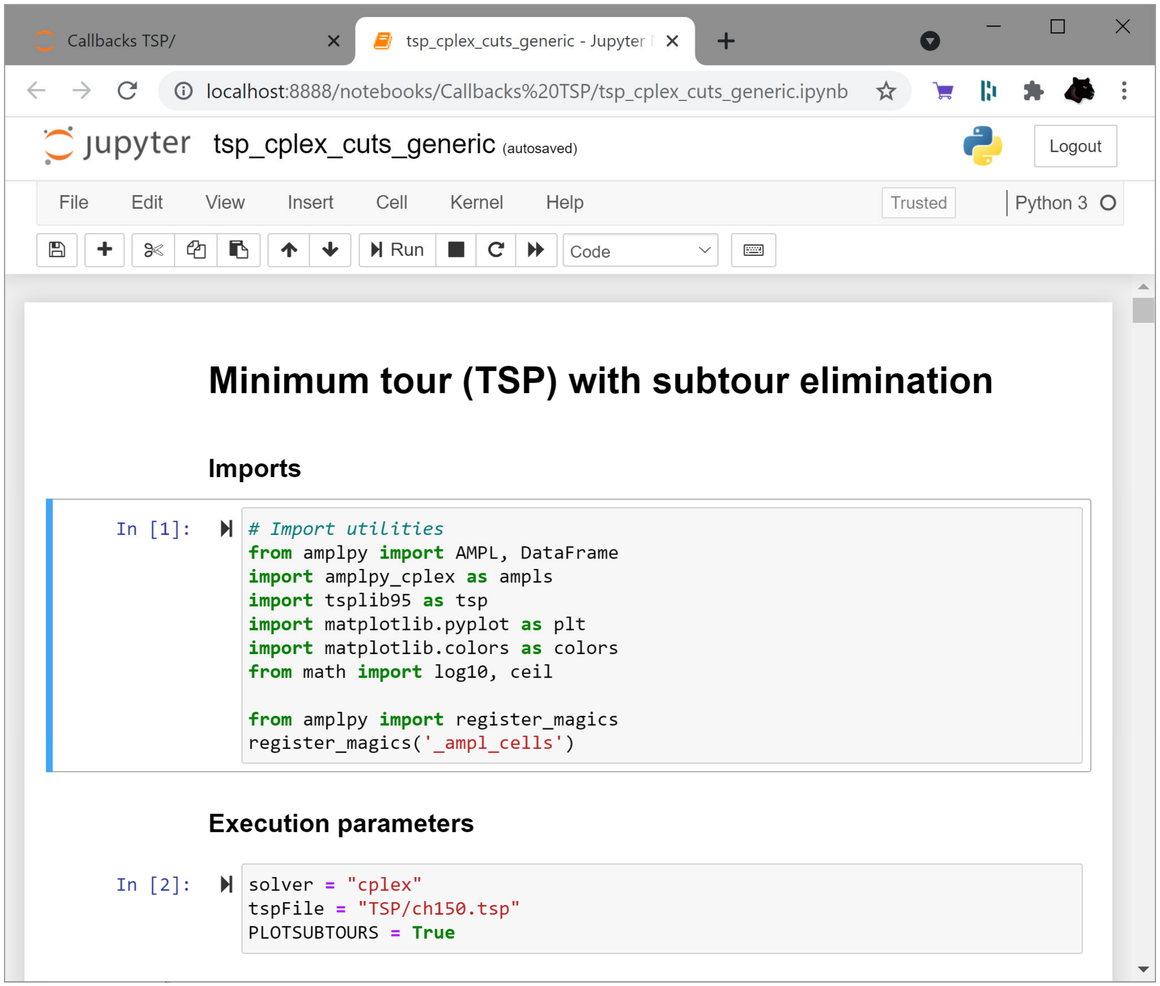
Minimum Tour

Implementation Highlights

Run subtour elimination

```
# Export the model using ampls
model = ampl.exportModel(solver)
model.enableLazyConstraints()

# Get the global maps between solver vars and AMPL entities
varMap = model.getVarMapFiltered("X")
inverse = model.getVarMapInverse()
xvars = ...
xinverse = ...
vertices = ...

# Assign the callback
callback = MyCallback()
model.setCallback(callback)

# Start the optimization
model.optimize()

# Import the solution back to AMPL
ampl.importSolution(model)
```

Minimum Tour Notebook



https://ampl.com/dl/Notebooks/mintour_callback.zip

Availability

Direct spreadsheet interface

* *implementation by Nicolau Santos*

- Included in all AMPL distributions
- Details at *ampl.com/resources/new-features/spreadsheets/*

Solver callbacks

* *implementation by Christian Valente, Filipe Brandão*

- Available for beta testing
- Write to *support@ampl.com* for details