AMPL Solver Interfaces With Callbacks

Featuring: Python callbacks for Gurobi and CPLEX!

Filipe Brandão
fdabrandao@ampl.com

IFORS 2021 Virtual – August 23-27, 2021

What is AMPL?

- AMPL: **A** **M**athematical **P**rogramming **L**anguage
- Algebraic modeling language built specially for optimization
- Designed to support many solvers
- Natural, easy-to-learn modeling principles
- Efficient processing that scales well with problem size

Model: diet.mod

```
# Choose prepared foods to meet certain nutritional requirements:
set NUTR;
set FOOD;

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];

param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];

param amt {NUTR,FOOD} >= 0;

var Buy {j in FOOD} >= f_min[j], <= f_max[j];

minimize Total_Cost:  sum {j in FOOD} cost[j] * Buy[j];

subject to Diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

How to interact with the model and the data?

Using the Python API (amplpy):

```
>>> from amplpy import AMPL
>>> ampl = AMPL()                          # > ampl
>>> ampl.read('diet.mod')                  # ampl: model diet.mod;
>>> ampl.read('diet.dat')                  # ampl: data diet.dat;
>>> ampl.option['solver'] = 'gurobi' # ampl: option solver gurobi;
>>> ampl.solve()                           # ampl: solve;
Gurobi 7.5.0: optimal solution; objective 88.2
1 simplex iterations
>>> ampl.getVariable('Buy').getValues().toPandas()
        Buy.val
BEEF   0.000000
CHK    0.000000
FISH   0.000000
HAM    0.000000
MCH   46.666667
MTL    0.000000
SPG    0.000000
TUR    0.000000
```

Setting data from Python

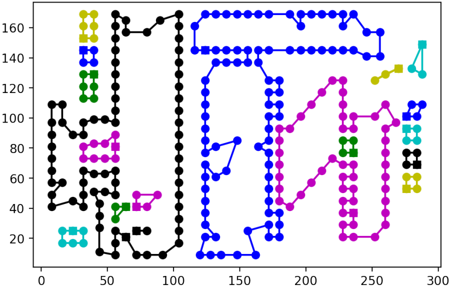
In Python:

```
>>> ampl.set['FOOD'] = [
    'BEEF', 'CHK', 'FISH', 'HAM', 'MCH', 'MTL', 'SPG', 'TUR']
>>> ampl.param['cost'] = [
    3.59, 2.59, 2.29, 2.89, 1.89, 1.99, 1.99, 2.49]
>>> ampl.param['f_min'] = [2, 2, 2, 2, 2, 2, 2, 2]
>>> ampl.param['f_max'] = [10, 10, 10, 10, 10, 10, 10, 10]
>>> ampl.eval('display cost, f_min, f_max;')
:      cost f_min f_max    :=
BEEF   3.59    2     10
CHK    2.59    2     10
FISH   2.29    2     10
HAM    2.89    2     10
MCH    1.89    2     10
MTL    1.99    2     10
SPG    1.99    2     10
TUR    2.49    2     10
;
```

Travelling Salesman Problem (TSP)

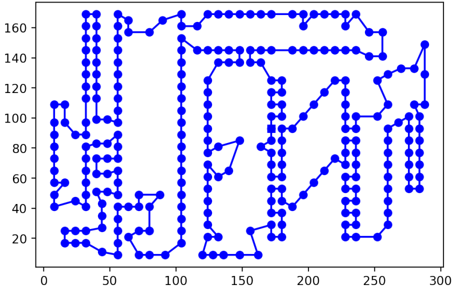
"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Invalid solution with sub-tours:



TSP instance: a280.tsp from TSPLIB.

Optimal solution without sub-tours:



Travelling Salesman Problem (TSP)

```
from amplpy import AMPL
ampl = AMPL()
ampl.eval('''
param n;
set V := 1..n;
set A := {(i,j) in V cross V : i != j};
param c{A} >= 0 default Infinity;
var x{A}, binary;
minimize Tour_Length: sum{(i,j) in A} c[i,j] * x[i,j];
s.t. enter{j in V}: sum{i in V: i != j} x[i, j] == 1;
s.t. leave{i in V}: sum{j in V: j != i} x[i, j] == 1;

# subtour elimination Miller, Tucker and Zemlin (MTZ) (1960)
var u{V} >= 0;
subject to MTZ{(i,j) in A: i != 1}: u[i]-u[j] + (n-1)*x[i,j] <= n-2;
''')
n, dist = load_tsp_instance('instance.txt')
ampl.param['n'] = n
ampl.param['c'] = dist
```

Travelling Salesman Problem (TSP)

What happens if you try to solve a280.tsp with just 280 nodes without callbacks using just MTZ constraints:

```
Root relaxation: objective 2.431206e+03, 579 iterations, 0.28 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 2431.20637    0  447          - 2431.20637      -     -    2s
     0     0 2515.52499    0  525          - 2515.52499      -     -   11s
     0     0 2515.52499    0  526          - 2515.52499      -     -   36s
     0     0 2531.19773    0  496          - 2531.19773      -     -   38s
     0     0 2534.89984    0  510          - 2534.89984      -     -   38s
     0     0 2534.89984    0  510          - 2534.89984      -     -   38s
     0     0 2541.20314    0  496          - 2541.20314      -     -   40s
     0     0 2541.20314    0  496          - 2541.20314      -     -   89s
     0     0 2541.20314    0  496          - 2541.20314      -     -   91s
     0     0 2541.22366    0  501          - 2541.22366      -     -  109s
     0     0 2541.22366    0  501          - 2541.22366      -     -  109s
     0     0 2541.22366    0  385          - 2541.22366      -     -  112s
     0     0 2541.22366    0  385          - 2541.22366      -     -  112s
     0     0 2541.22366    0  378          - 2541.22366      -     -  114s
     0     0 2541.22366    0  378          - 2541.22366      -     -  114s
     0     0 2541.22366    0  376          - 2541.22366      -     -  117s
     0     0 2541.22366    0  375          - 2541.22366      -     -  258s
     0     0 2541.22366    0  447          - 2541.22366      -     -  260s
```

Defining a simple generic callback in Python

This is a very simple callback that just provides additional information to the user.

```python
import amplpy_gurobi as ampls
# Define my generic callback function
class MyCallback(ampls.GenericCallback):
    def __init__(self):
        self.nMIPnodes = 0
    def run(self):
        t = self.getAMPLWhere()
        if t == ampls.Where.MSG:
            print('>' + self.getMessage())
        elif t == ampls.Where.MIPNODE:
            self.nMIPnodes += 1
            print("New MIP node, count {}".format(self.nMIPnodes))
        elif t == ampls.Where.MIPSOL:
            print("MIP Solution = {}".format(self.getObj()))
        return 0
```

Using the callback with Gurobi

Export model object, optimize, and import solution:

```
m = ampl.exportModel('gurobi')  # export model object
cb = MyCallback()               # instantiate callback
m.setCallback(cb)               # set the callback to use
m.optimize()                    # run the optimization process
if m.getStatus() == ampls.Status.OPTIMAL:
    ampl.importSolution(m)      # load the solution into ampl
    ampl.display('Tour_Length') # display objective value form ampl
```

Output of m.optimize():

```
...
MIP Solution = 460.07905777356814
>*  206   208            44    460.0790578  419.71967  8.77%  19.5    1s
MIP Solution = 440.442642507044
>H  230   206                  440.4426425  419.75510  4.70%  18.6    1s
MIP Solution = 439.9556742664898
>*  354   250            42    439.9556743  419.75510  4.59%  18.3    1s
MIP Solution = 439.1123891294378
>*  490   284            55    439.1123891  420.06206  4.34%  18.1    1s
MIP Solution = 436.18563128541143
>H 1046   627                  436.1856313  423.97345  2.80%  18.0    3s
>  1074   648  426.46288   12  128  436.18563  426.46288  2.23%  19.9    5s
MIP Solution = 435.47746822093836
>H 1082   619                  435.4774682  426.47191  2.07%  19.7    5s
MIP Solution = 432.502179738009
>H 1089   593                  432.5021797  426.47191  1.39%  22.3    6s
MIP Solution = 431.10323765840644
>H 1191   621                  431.1032377  426.75789  1.01%  24.6    8s
MIP Solution = 428.871756392034
>* 1567   655           120    428.8717564  426.75789  0.49%  23.4    8s
...
```

Symmetric Traveling Salesman Problem

```
set NODES ordered;
param hpos {NODES};
param vpos {NODES};

set PAIRS := {i in NODES, j in NODES: ord(i) < ord(j)};

param distance {(i,j) in PAIRS}
   := sqrt((hpos[j]-hpos[i])**2 + (vpos[j]-vpos[i])**2);

var X {PAIRS} binary;

minimize Tour_Length: sum {(i,j) in PAIRS} distance[i,j] * X[i,j];

subject to Visit_All {i in NODES}:
   sum {(i, j) in PAIRS} X[i,j] + sum {(j, i) in PAIRS} X[j,i] = 2;
```

Generate sub-tour elimination cuts with a callback

```python
import amplpy_cplex as ampls # use cplex instead of gurobi
class TSPCuts(ampls.GenericCallback):
    def run(self):
        if self.getAMPLWhere() == ampls.Where.MIPSOL:
            self.mipsol()
        return 0
    def mipsol(self):
        sol = self.getSolutionVector()
        uf = UnionFind()
        for i, (u, v) in xvars.items():
            if sol[i] > 1e-5:
                uf.link(u, v)
        groups = uf.groups()
        if len(groups) == 1:
            print('Valid solution!')
            return
        for g in groups:
            print('> sub-tour: ', g)
            vnames = [tuple2var('X', i, j) for i in g for j in g if i < j]
            coeffs = [1 for i in range(len(vnames))]
            self.addLazy(vnames, coeffs, ampls.CutDirection.LE, len(grp)-1)
```

Using the callback with CPLEX

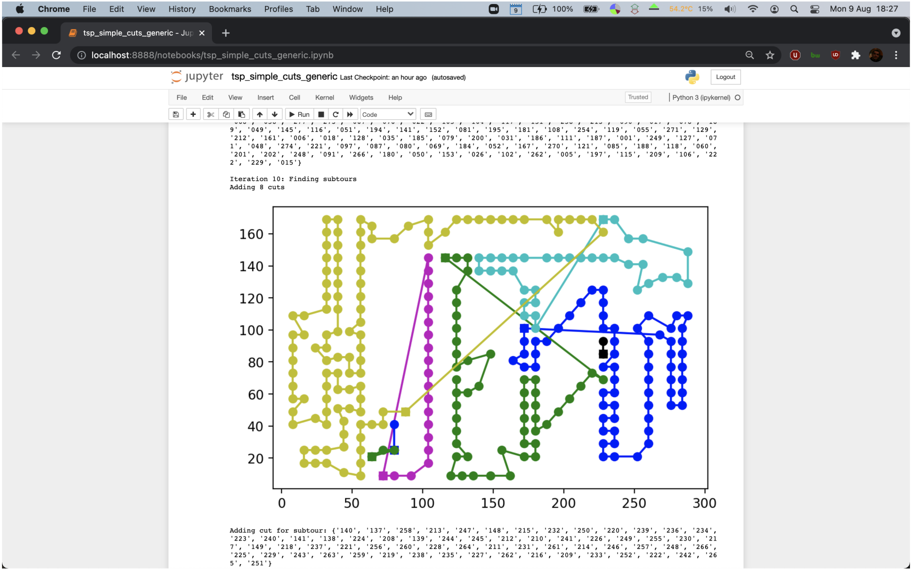
Export model object, optimize, and import solution:

```
m = ampl.exportModel('cplex').   # export model object
m.enableLazyConstraints().       # enable lazy constraints
cb = TSPCuts()                   # instantiate callback
m.setCallback(cb)                # set the callback to use
m.optimize()                     # run the optimization process
if m.getStatus() == ampls.Status.OPTIMAL:
    ampl.importSolution(m)       # load the solution into ampl
    ampl.display('Tour_Length') # display objective value form ampl
```

Output of m.optimize():

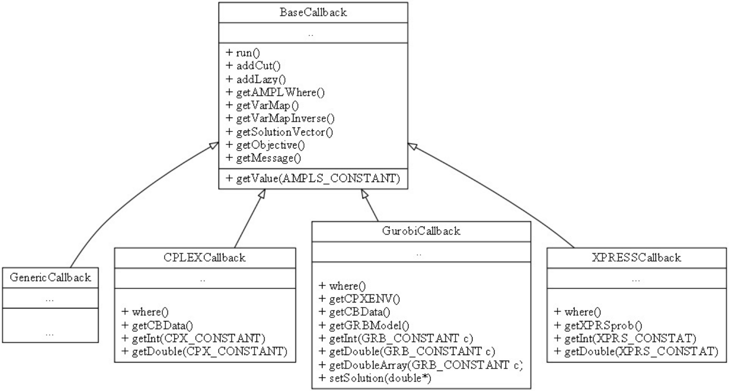
```
...
MIPSOL #16
Valid solution!
MIPSOL #17
> sub-tour:  ['1', '6', '3', '29', '4', '47', '5', '35', '7', '37', '8', '20', '9', '10', '46', '11', '12', '19', '13', '
> sub-tour:  ['2', '32']
> sub-tour:  ['21', '26', '38']
> sub-tour:  ['22', '44']
> sub-tour:  ['30', '43']
MIPSOL #18
> sub-tour:  ['1', '34', '2', '32', '3', '6', '4', '28', '5', '9', '7', '37', '8', '14', '47', '10', '11', '29', '12', '4
> sub-tour:  ['25', '42']
MIPSOL #19
> sub-tour:  ['1', '34', '3', '29', '4', '47', '5', '35', '6', '7', '37', '8', '20', '9', '10', '46', '11', '13', '31', '
> sub-tour:  ['2', '26', '21']
> sub-tour:  ['12', '43', '22', '38', '30', '44']
> sub-tour:  ['23', '32']
MIPSOL #20
Valid solution!
Solved for 2550 variables, objective 428.87175639203394
```

Jupyter Notebok



Conclusions 1/2

- Even though this presentation was focused on callbacks in Python, we provide solver extensions with callbacks for other languages and solvers.
- We provide a generic interface that allows implementing callbacks that work seamlessly with multiple solvers, and solver specific interfaces so that users can take full advantage of everything a specific solver provides.



Conclusions 2/2

- The solver libraries for Gurobi and CPLEX can be installed as easily as:

```
python -m pip install amplpy_gurobi
python -m pip install amplpy_cplex
```

- Last but not least, all this interfaces are **open-source** so that users can contribute to them and extend them to support additional languages and solver features.
- GitHub project: `https://github.com/ampl/ampls-api`
- Documentation for solver libraries: `https://ampls.readthedocs.io`
- Documentation for amplpy: `https://amplpy.readthedocs.io`