AMPL Solver Interfaces With Callbacks

Featuring: Python callbacks for Gurobi and CPLEX!

Filipe Brandão
fdabrandao@ampl.com

Virtual 2020 INFORMS Annual Meeting – November 11, 2020
What is AMPL?

- AMPL: **A Mathematical Programming Language**
- 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):

```python
>>> 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:

```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
;```
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 total: 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;
''

Filipe Brandão (AMPL)
AMPL Solver Interfaces With Callbacks
November 11, 2020 6 / 11
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:

```python
m = ampl.exportGurobiModel()  # 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('total')     # display objective value form ampl
```

Output of `m.optimize()`:

```
...  
MIP Solution = 460.0790577356814  
>= 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 = 435.4774682209386  
>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 = 431.1032765840644  
>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
...  
```
Generate subtour 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 = [ampl_var('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:

```python
m = ampl.exportCplexModel()  # export model object
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('total')  # display objective value form ampl
```

Output of `m.optimize()`:

```
...  
MIPSOL #16
Valid solution!
MIPSOL #17
> 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', '43']
> sub-tour: ['25', '42']
MIPSOL #19
> 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
CPLEX 12.10.0.0: optimal integer solution; objective 428.87175639203394
1245 MIP simplex iterations
217 branch-and-bound nodes
```

Filipe Brandão (AMPL)
Conclusions

- 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.
- 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.