New Python Integration Features of the AMPL Modeling Language

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

30th European Conference on Operational Research
Dublin, Ireland — 23-26 June 2019
Session TA-49: Software for optimization model deployment
New Python Integration Features of the AMPL Modeling Language

Optimization modeling languages are fundamentally declarative in design, but are frequently put to use within broader contexts that require a variety of programming options. Thus while programming is not employed to describe models, it facilitates the integration of models into broader algorithmic schemes and business applications. This presentation focuses on integration of the widely used AMPL modeling language with Python and Jupyter, the most popular environment for programming in data science. A single running example illustrates multiple topics, which include integrating model-based optimization into applications using AMPL's Python API, embedding Python in AMPL models and scripts, implementing complex AMPL constraint generators in Python, and setting up solver callbacks using Python programs.
Examples

AMPL Python API
- Example: Roll cutting by pattern enumeration

Python data embedded in an AMPL model
- Example: Roll cutting data

Python code embedded in an AMPL model
- Example: Generating advanced lot-sizing constraints

Python callbacks from Gurobi
- Example: User-specified stopping rule

AMPL in Jupyter notebooks
- Example: Roll cutting by pattern generation
- Example: Lot sizing using advanced formulations
**AMPL Python API**

*Example: Roll Cutting by Pattern Enumeration*
- Fill orders for rolls of various widths

**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
**AMPL Model**

**Mathematical Formulation**

*Given*

\[ w \quad \text{width of “raw” rolls} \]
\[ W \quad \text{set of (smaller) ordered widths} \]
\[ n \quad \text{number of cutting patterns considered} \]

*and*

\[ a_{ij} \quad \text{occurrences of width } i \text{ in pattern } j, \]
\[ \text{for each } i \in W \text{ and } j = 1, \ldots, n \]
\[ b_i \quad \text{orders for width } i, \text{ for each } i \in W \]
\[ o \quad \text{limit on overruns} \]
**AMPL Model**

**Mathematical Formulation (cont’d)**

**Determine**

\[ X_j \text{ number of rolls to cut using pattern } j, \]
\[ \text{for each } j = 1, \ldots, n \]

**to minimize**

\[ \sum_{j=1}^{n} X_j \]
\[ \text{total number of rolls cut} \]

**subject to**

\[ b_i \leq \sum_{j=1}^{n} a_{ij}X_j \leq b_i + o, \text{ for all } i \in W \]
\[ \text{number of rolls of width } i \text{ cut} \]
\[ \text{must be at least the number ordered,} \]
\[ \text{and must be within the overrun limit} \]
**AMPL Model**

**AMPL Formulation**

**Symbolic model**

```AMPL
param rawWidth;
set WIDTHS;

param nPatterns integer > 0;
set PATTERNS = 1..nPatterns;

param rolls {WIDTHS,PATTERNS} >= 0, default 0;
param order {WIDTHS} >= 0;
param overrun;

var Cut {PATTERNS} integer >= 0;

minimize TotalCut: sum {p in PATTERNS} Cut[p];

subject to OrderLimits {w in WIDTHS}:
    order[w] <= sum {p in PATTERNS} rolls[w,p] * Cut[p] <= order[w] + overrun;
```

\[ b_i \leq \sum_{j=1}^{n} a_{ij} X_j \leq b_i + o \]
**AMPL Model**

**AMPL Formulation (cont’d)**

**Explicit data (independent of model)**

```
param rawWidth := 64.5 ;
param: WIDTHS: order :=
    6.77   10
    7.56   40
    17.46  33
    18.76  10 ;
param nPatterns := 9 ;
param rolls:  1  2  3  4  5  6  7  8  9 :=
    6.77   0  1  1  0  3  2  0  1  4
    7.56   1  0  2  1  1  4  6  5  2
   17.46  0  1  0  2  1  0  1  1  1
   18.76  3  2  2  1  1  1  0  0  0 ;
param overrun := 6 ;
```
AMPL Model

Command Language (cont’d)

Solver choice independent of model and data

```ampl
ampl: model cut.mod;
ampl: data cut.dat;
ampl: option solver gurobi;
ampl: solve;
Gurobi 8.1.0: optimal solution; objective 20
3 simplex iterations
1 branch-and-cut nodes
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
4 13 7 4 9 3
```
AMPL Model

AMPL Command Language

Model + data = problem instance to be solved

```
ampl: model cut.mod;
ampl: data cut.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.9.0.0: optimal integer solution; objective 20
3 MIP simplex iterations
0 branch-and-bound nodes
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
4 13 7 4 9 3
```
**AMPL Model**

**AMPL Command Language**

*Model + data = problem instance to be solved*

```
AMPL: model cut.mod;
AMPL: data cut.dat;
AMPL: option solver xpress;
AMPL: solve;

XPRESS 8.5(32.01.08): Global search complete
Best integer solution found 20
3 integer solutions have been found
1 branch and bound node

AMPL: option omit_zero_rows 1;
AMPL: option display_1col 0;
AMPL: display Cut;
4 13 7 4 9 3
```
**AMPL Model**

**Command Language (cont’d)**

**Results available for browsing**

```plaintext
ampl: display \{p in PATTERNS\} sum \{w in WIDTHS\} w * rolls[w,p];
1 63.84  3 59.41  5 64.09  7 62.82  9 59.66  # material used
2 61.75  4 61.24  6 62.54  8 62.0   # in each pattern

ampl: display sum \{p in PATTERNS\}
ampl? Cut[p] * (rawWidth - sum \{w in WIDTHS\} w * rolls[w,p]);
62.32  # total waste
        # in solution

ampl: display OrderLimits.lslack;
  6.77  0  # overruns
  7.56  0  # of each pattern
  17.46 0
  18.76 5
```
**AMPL APIs**

*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*
Roll Cutting by Pattern Enumeration

**Principles**
- Generate a long list of candidate patterns
  - for this example, all nondominated patterns
- Solve the cutting problem using this entire candidate list

**Implementation**
- Pattern enumeration in Python
- Modeling and solving in AMPL, via API calls
- Solution reporting in Python

**Key to examples**
- AMPL entities
- AMPL API Python objects
- AMPL API Python methods
- Python functions etc.
### AMPL Model File

**Same pattern-cutting model**

```AMPL
param nPatterns integer > 0;
set PATTERNS = 1..nPatterns;   # patterns
set WIDTHS;                     # finished widths
param order {WIDTHS} >= 0;      # rolls of width j ordered
param overrun;                  # permitted overrun on any width
param rawWidth;                 # width of raw rolls to be cut
param rolls {WIDTHS,PATTERNS} >= 0, default 0;  
                                 # rolls of width i in pattern j

var Cut {PATTERNS} integer >= 0;  # raw rolls to cut in each pattern
minimize TotalRawRolls: sum {p in PATTERNS} Cut[p];

subject to FinishedRollLimits {w in WIDTHS}:
    order[w] <= sum {p in PATTERNS} rolls[w,p] * Cut[p] <= order[w] + overrun;
```
AMPL Python API

Some Python Data

A float, an integer, and a dictionary

```
roll_width = 64.5
overrun = 6
Orders = {
    6.77: 10,
    7.56: 40,
    17.46: 33,
    18.76: 10
}
```

... can also work with lists and Pandas dataframes
**AMPL Python API**

**Pattern Enumeration**

*Load & generate data, set up AMPL model*

```python
def cuttingEnum(dataset):
    from amplpy import AMPL

    # Read orders, roll_width, overrun
    exec(open(dataset+'.py').read(), globals())

    # Enumerate patterns
    widths = list(sorted(orders.keys(), reverse=True))
    patmat = patternEnum(roll_width, widths)

    # Set up model
    ampl = AMPL()
    ampl.option['ampl_include'] = 'models'
    ampl.read('cut.mod')```
**Pattern Enumeration**

**Send data to AMPL**

```python
# Send scalar values
ampl.param['nPatters'] = len(patmat)
ampl.param['overrun'] = overrun
ampl.param['rawWidth'] = roll_width

# Send order vector
ampl.set['WIDTHS'] = widths
ampl.param['order'] = orders

# Send pattern matrix
ampl.param['rolls'] = {
    (widths[i], 1+p): patmat[p][i]
    for i in range(len(widths))
    for p in range(len(patmat))
}
```
**AMPL Python API**

**Pattern Enumeration**

* Solve and get results *

```python
# Solve
ampl.option['solver'] = 'gurobi'
ampl.solve()

# Retrieve solution
CuttingPlan = ampl.var['Cut'].getValues()
cutvec = list(CuttingPlan.getColumn('Cut.val'))
```
# Prepare solution data

```python
summary = {
    'Data': dataset,
    'Obj': int(ampl.obj['TotalRawRolls'].value()),
    'Waste': ampl.getValue(
        'sum {p in PATTERNS} Cut[p] * 
        (rawWidth - sum {w in WIDTHS} w*rolls[w,p])'
    )
}
```

```python
solution = [(patmat[p], cutvec[p])
            for p in range(len(patmat))
            if cutvec[p] > 0]
```

# Create plot of solution

```python
cuttingPlot(roll_width, widths, summary, solution)
```
**Pattern Enumeration**

*Enumeration routine*

```python
def patternEnum(roll_width, widths, prefix=[]):
    from math import floor
    max_rep = int(floor(roll_width/widths[0]))
    if len(widths) == 1:
        patmat = [prefix+[max_rep]]
    else:
        patmat = []
        for n in reversed(range(max_rep+1)):
            patmat += patternEnum(roll_width-n*widths[0], widths[1:], prefix+[n])
    return patmat
```
**AMPL Python API**

**Pattern Enumeration**

**Plotting routine**

```python
def cuttingPlot(roll_width, widths, summ, solution):
    import numpy as np
    import matplotlib.pyplot as plt
    ind = np.arange(len(solution))
    acc = [0]*len(solution)
    colorlist = ['red','lightblue','orange','lightgreen',
                 'brown','fuchsia','silver','goldenrod']
```
Pattern Enumeration

Plotting routine (cont’d)

```python
for p, (patt, rep) in enumerate(solution):
    for i in range(len(widths)):
        for j in range(patt[i]):
            vec = [0]*len(solution)
            vec[p] = widths[i]
            plt.barh(ind, vec, 0.6, acc,
                color=colorlist[i%len(colorlist)], edgecolor='black')
            acc[p] += widths[i]

plt.title(summ['Data'] + ": " +
    str(summ['Obj']) + " rolls" + ", " +
    str(round(100*summ['Waste']/(roll_width*summ['Obj']),2)) + "% waste"
)
plt.xlim(0, roll_width)
plt.xticks(np.arange(0, roll_width, 10))
plt.yticks(ind, tuple("x {x}".format(rep) for patt, rep in solution))
plt.show()
```
AMPL Python API

Pattern Enumeration

In [1]: from pattern Enumeration import *

In [2]: cuttingEnum('Sorrentino')

43 patterns enumerated

Gurobi 8.1.0: optimal solution; objective 18
7 simplex iterations
1 branch-and-cut nodes

Figure 1

Sorrentino: 18 rolls, 2.34% waste
Python Data Embedded in an AMPL Model

Example: *Roll Cutting Data*

**Data transfer method approach (amplpy)**
- Read the model into AMPL
- Use Python API methods to send data to AMPL

**Embedded data approach (PyMPL)**
- Specify Python data correspondences in the model
- Read the model into AMPL
Python Integration

Getting Data

Imported and generated data in Python

```python
roll_width = 64.5
overrun = 6

orders = {
    6.77: 10,
    7.56: 40,
    17.46: 33,
    18.76: 10
}

patmat = patternEnum(roll_width, list(sorted(orders.keys(), reverse=True)))
```
Python Integration

Sending Data using the Python API

Specify symbolic sets and parameters in AMPL

```ampl
param nPatterns integer > 0;
set PATTERNS = 1..nPatterns;
set WIDTHS;

param order {WIDTHS} >= 0;
param overrun;

param rawWidth;
param rolls {WIDTHS,PATTERNS} >= 0, default 0;
```
Python Integration

Sending Data using the Python API (cont’d)

Call `ampl` methods to read model, send data

```python
ampl = AMPL()
ampl.read('cut.mod')
........
ampl.param['nPatterns'] = len(patmat)
ampl.param['overrun'] = overrun
ampl.param['rawWidth'] = roll_width

ampl.set['WIDTHS'] = widths
ampl.param['order'] = orders

ampl.param['rolls'] = {
    (widths[i], 1+p): patmat[p][i]
    for i in range(len(widths))
    for p in range(len(patmat))
}
```
Sending Data using PyMPL

Specify Python data correspondences in the model

```plaintext
$PARAM[nPatterns]{ len(patmat) };  
set PATTERNS = 1..nPatterns;

$SET[WIDTHS]{ widths };  
$PARAM[order~WIDTHS] { orders };  
$PARAM[overrun] { overrun };  
$PARAM[rawWidth] { roll_width };  
$PARAM[rolls~WIDTHS,~PATTERNS] {  
  {  
    (widths[i], 1+p): patmat[p][i]  
    for i in range(len(widths))  
    for p in range(len(patmat))  
  }  
};

ampl = AMPL(langext=PyMPL())  
ampl.read('cutpy.mod')
```
Python Code Embedded in an AMPL Model

Example: *Generating advanced lot-sizing constraints*

- Create a tighter formulation that solves faster

New constructs for embedding Python in AMPL

- `python-expression`
- `$EXEC{ python-statements }$`
Executing Python inside AMPL

Fix AMPL variables according to Python variable

```plaintext
$PARAM[NT]\{8\};

var x \{1..NT\}, >= 0;    # production lot size
var y \{1..NT\}, binary; # production set-up
var s \{0..NT\}, >= 0;    # inventory level

var r \{1..NT\}, ${">= 0" if BACKLOG else ">= 0, <= 0"}$;

    # use these variables iff BACKLOG > 0
```
## Executing Python inside AMPL

**Invoke Python generators for special lot-sizing constraints**

```python
$EXEC{
def mrange(a, b):
    return range(a, b+1)

s = ['s[{}]'.format(t) for t in mrange(0, NT)]
y = ['y[{}]'.format(t) for t in mrange(1, NT)]
d = [demand[t] for t in mrange(1, NT)]

if BACKLOG is False:
    WW_U_AMPL(s, y, d, NT, prefix='w')
else:
    r = ['r[{}]'.format(t) for t in mrange(1, NT)]
    WW_U_B_AMPL(s, r, y, d, NT, prefix='w')
};
```

```python
ampl = AMPL(langext=PyMPL())
ampl.read('lotsize.mod')
ampl.solve()
```
Python Integration

Executing Python inside AMPL

Optional listing of generated constraints

```plaintext
var ws {wi in 0..8} = s[wi];
var wr {wi in 1..8} = r[wi];
var wy {wi in 1..8} = y[wi];
param wD {1..8, 1..8};
data;
param wD :=
;
model;
```
Python Integration

Executing Python inside AMPL

Optional listing of generated constraints (cont’d)

```plaintext
var wa {1..8};
var wb {1..8};
subject to wXY {wt in 1..8}: wa[wt] + wb[wt] + wy[wt] >= 1;
subject to wXA {wk in 1..8, wt in wk..min(8, wk+8-1): wD[wt,wt]>0}:
    ws[wk-1] >=
    sum {wi in wk..wt} wD[wi,wi] * wa[wi]
    - sum {wi in wk..wt-1} wD[wi+1,wt] * wy[wi];
subject to wXB {wk in 1..8, wt in max(1, wk-8+1)..wk: wD[wt,wt]>0}:
    wr[wk] >=
    sum {wi in wt..wk} wD[wi,wi] * wb[wi]
    - sum {wi in wt+1..wk} wD[wt,wi-1] * wy[wi];
```
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 $g_i$
Python Integration

Callbacks

Stopping rule data in Python dictionary

```python
stopdict = { 'time' : (  15,  30,  60 ),
             'gaptol' : ( 0.0002, 0.002, 0.02 )
           }
```

Main routine for cutting by pattern generation

```python
def cuttingGen(cutdata, stopdata = ""): from amplpy import AMPL
                   .......
           # begin pattern generation phase
           # finish when continuous relaxation of cutting problem has been solved
```
Python Integration

Callbacks

Set up callback and solve final integer program

```python
# Instead of Master.solve(), export to a gurobipy object
grb_model = Master.exportGurobiModel()

# Assign AMPL stopping data to gurobipy objects
if len(stopdata) == 0:
    grb_model._stoprule = {'time': (1e+10,), 'gaptol': (1,)}
else:
    exec(open(stopdata+'.py').read(), globals())
    stopdict['time'] += (1e+10,)
    stopdict['gaptol'] += (1,)
    grb_model._stoprule = stopdict

grb_model._current = 0

# Solve and import results
grb_model.optimize(callback)
Master.importGurobiSolution(grb_model)
```
Python Integration

Callbacks

Callback function

```python
def callback(m, where):
    """Gurobi callback function."""
    if where == gpy.GRB.Callback.MIP:
        runtime = m.cbGet(gpy.GRB.Callback.RUNTIME)
        if runtime >= m._stoprule['time'][m._current]:
            print("Reducing gap tolerance to %f at %d seconds" %
                  (m._stoprule['gaptol'][m._current], m._stoprule['time'][m._current]))
            m.Params.MIPGap = m._stoprule['gaptol'][m._current]
        m._current += 1
```
AMPL in Jupyter Notebooks

Example: Roll Cutting by Pattern Generation

Example: Lot Sizing Using Advanced Formulations
AMPL in Jupyter Notebooks

Contact the speaker (4er@ampl.com) for the notebook files

AMPLPY: Pattern Generation

Documentation: http://amplpy.readthedocs.io

GitHub Repository: https://github.com/ampl/amplpy

PyPI Repository: https://pypi.python.org/pypi/amplpy

Imports

In [1]:

from __future__ import print_function
from amplpy import AMPL
import os, sys
from math import floor, ceil
Status

**AMPL API 2.0**
- Released

*Embedded Python data & code*

*Python callbacks from Gurobi*

**AMPL in Jupyter notebooks**
- All available for beta testing
- Contact support@ampl.com for more information