Integrating Optimization Modeling with General-Purpose Programming for Efficient and Reliable Application Deployment

Robert Fourer, Filipe Brandão
AMPL Optimization
{4er,fdabrandao}@ampl.com

Christian Valente
OptiRisk Systems
christian@optirisk-systems.com

INFORMS Annual Meeting
Houston — 22-25 October 2017 — Session TB74
INFORMS Computing Society
4 Ways to Use an Optimization Modeling Language

Command language
- Browse results & debug model interactively
- Make changes and re-run

Scripting language
- Write programs using modeling language constructs

Programming interface (API)
- Embed a modeling language and system within a general-purpose programming language

Application-building toolkit
- Use a application-building system designed specifically for the modeling language
Features

- Algebraic modeling language
- Built specially for optimization
- Designed to support many solvers

APIs

- C++, C#, Java, MATLAB, Python
- R coming soon

Application-building toolkit

- QuanDec / built on Java API
Example

Roll cutting in Python API
- Solution by pattern enumeration
- Solution by pattern generation
- Tradeoff between waste and overruns

QuanDec
- Overview
Roll Cutting Problem

Motivation

- Fill orders for rolls of various widths
  - by cutting raw rolls of one (large) fixed width
  - using a variety of cutting patterns

Optimization model

- Decision variables
  - number of raw rolls to cut according to each pattern
- Objective
  - minimize number of raw rolls used
- Constraints
  - meet demands for each ordered width
Given

- \( w \) width of “raw” rolls
- \( W \) set of (smaller) ordered widths
- \( n \) number of cutting patterns considered

and

- \( a_{ij} \) occurrences of width \( i \) in pattern \( j \), for each \( i \in W \) and \( j = 1, \ldots, n \)
- \( b_i \) orders for width \( i \), for each \( i \in W \)
- \( o \) limit on overruns
Roll cutting

Mathematical Formulation (cont’d)

Determine

\( X_j \) number of rolls to cut using pattern \( j \),
for each \( j = 1, \ldots, n \)

to minimize

\[ \sum_{j=1}^{n} X_j \]

total number of rolls cut

subject to

\[ b_i \leq \sum_{j=1}^{n} a_{ij} X_j \leq b_i + o, \text{ for all } i \in W \]

number of rolls of width \( i \) cut
must be at least the number ordered,
and must be within the overrun limit
Roll cutting

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 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;
```

\[ b_i \leq \sum_{j=1}^{n} a_{ij} X_j \leq b_i + o \]
Roll Cutting

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 ;
```
Roll Cutting

AMPL Command Language

Model + data = problem instance to be solved

```ampl
ampl: model cut.mod;
ampl: data cut.dat;
ampl: option solver cplex;
ampl: solve;
```

CPLEX 12.7.1.0: optimal integer solution; objective 20
3 MIP simplex iterations
0 branch-and-bound nodes

```ampl
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
```

4 13 8 5 9 2
Roll Cutting

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 7.5.0: optimal solution; objective 20
8 simplex iterations
1 branch-and-cut nodes
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
4 13 5 2 7 4 9 1
```
Roll Cutting

Command Language (cont’d)

Results available for browsing

```ampl
ampl: display {j in 1..nPatterns, i in WIDTHS: Cut[j] > 0} rolls[i,j];
: 4 5 7 9 := # patterns used
   6.77 0 3 0 4
   7.56 1 1 6 2
   17.46 2 1 1 1
   18.76 1 1 0 0

ampl: display {j in 1..nPatterns} sum {i in WIDTHS} i * rolls[i,j];
1 63.84 3 59.41 5 64.09 7 62.82 9 59.66 # pattern
2 61.75 4 61.24 6 62.54 8 62.0 # total widths

ampl: display FinishedRollLimits.lslack;
   6.77 0 # overruns
   7.56 1
   17.46 0
   18.76 5
```
Cutting via Pattern Enumeration

Build the pattern list, then solve

- Read general model
- Read data
  * demands, raw width
  * orders, overrun limit
- Compute data: all “good” patterns
  * extract widths from demand list
  * enumerate all non-dominated patterns
- Solve problem instance
Pattern Enumeration

Example Using the AMPL API

Hybrid approach
- Control & pattern enumeration in Python
- Model & modeling expressions in AMPL

Key to program examples
- AMPL entities
- AMPL API Python objects
- AMPL API Python methods
- Python functions etc.
### AMPL API

#### 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;
```
Some Python Data

A float, an integer, and a dictionary

```python
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 API

**Pattern Enumeration in Python**

*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')```
**AMPL API**

**Pattern Enumeration in Python**

**Send data to AMPL**

```python
# Send scalar values
ampl.param['nPatterns'] = 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 API**

**Pattern Enumeration in Python**

**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'))
```
**AMPL API**

**Pattern Enumeration in Python**

**Display solution**

```python
# Prepare solution data

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

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

# Create plot of solution

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

**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 API**

**Pattern Enumeration in Python**

*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 in Python**

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 {}".format(rep) for patt, rep in solution))
plt.show()
```
AML API

Pattern Enumeration in Python

```
sw: running ipython
Python 3.4.2 (v3.4.2:ab2c823a9432, Oct 6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]: from patternEnumeration import *
In [2]: cuttingEnum('Sorrentino')
Gurobi 7.5.0: optimal solution; objective 18
9 simplex iterations
1 branch-and-cut nodes
```
Cutting via Pattern Generation

*Generate the pattern list by a series of solves*

- 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
- Solve IP using all patterns found
Pattern Generation

Example Using the API

Two AMPL objects

- **Master** is the cutting model with current pattern subset
- **Sub** is the one-constraint knapsack problem

Key to program examples

- AMPL entities
- AMPL API Python objects
- AMPL API Python methods
- Python functions etc.
Pattern Generation in Python

Get data, set up master problem

```python
function cuttingGen(dataset):
    from amplpy import AMPL

    # Read orders, roll_width, overrun; extract widths
    exec(open(dataset+'.py').read(), globals())
    widths = list(sorted(orders.keys(), reverse=True))

    # Set up cutting (master problem) model
    Master = AMPL()
    Master.option['ampl_include'] = 'models'
    Master.read('cut.mod')

    # Define a param for sending new patterns
    Master.eval('param newPat {WIDTHS} integer >= 0;')

    # Set solve options
    Master.option['solver'] = 'gurobi'
    Master.option['relax_integrality'] = 1
```
Pattern Generation in Python

Send data to master problem

```python
# Send scalar values
Master.param['nPatterns'] = len(widths)
Master.param['overrun'] = overrun
Master.param['rawWidth'] = roll_width

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

# Generate and send initial pattern matrix
Master.param['rolls'] = {
    (widths[i], 1+i): int(floor(roll_width/widths[i]))
    for i in range(len(widths))
}
```
**AMPL API**

Pattern Generation in Python

Set up subproblem

```python
# Define knapsack subproblem
Sub = AMPL()
Sub.option['solver'] = 'gurobi'
Sub.eval(''
    set SIZES;
    param cap >= 0;
    param val {SIZES};
    var Qty {SIZES} integer >= 0;
    maximize TotVal: sum {s in SIZES} val[s] * Qty[s];
    subject to Cap: sum {s in SIZES} s * Qty[s] <= cap;
'')

# Send subproblem data
Sub.set['SIZES'] = widths
Sub.param['cap'] = roll_width
```
**AMPL API**

**Pattern Generation in Python**

*Generate patterns and re-solve cutting problems*

```python
# Alternate between master and sub solves
while True:
    Master.solve()
    Sub.param['val'].setValues(Master.con['OrderLimits'].getValues())
    Sub.solve()
    if Sub.obj['TotVal'].value() <= 1.00001:
        break
    Master.param['newPat'].setValues(Sub.var['Qty'].getValues())
    Master.eval('let nPatterns := nPatterns + 1;')
    Master.eval('let {w in WIDTHS} rolls[w, nPatterns] := newPat[w];')

# Compute integer solution
Master.option['relax_integrality'] = 0
Master.solve()
```

Fourer, Brandão, Valente, Integrating Optimization Modeling with Programming
INFORMS Annual Meeting, Houston, 22-25 October 2017
Pattern Generation in Python

Display solution

```python
# Retrieve solution
cutting_plan = Master.var['Cut'].getValues()
cutvec = list(cutting_plan.getColumn('Cut.val'))

# Prepare summary data
summary = {
    'Data': dataset,
    'Obj': int(Master.obj['TotalRawRolls'].value()),
    'Waste': Master.getValue(
        'sum {p in PATTERNS} Cut[p] * \n        (rawWidth - sum {w in WIDTHS} w*rolls[w,p])'
    )
}

# Retrieve patterns and solution
npatterns = int(Master.param['nPaterns'].value())
rolls = Master.param['rolls'].getValues().toDict()
cutvec = Master.var['Cut'].getValues().toDict()
```
# Pattern Generation in Python

## Display solution

```python
# Prepare solution data
solution = [
    ([int(rolls[widths[i], p+1][0])
      for i in range(len(widths))], int(cutvec[p+1][0]))
    for p in range(npatterns)
    if cutvec[p+1][0] > 0
]

# Create plot of solution

cuttingPlot(roll_width, widths, summary, solution)
```
AMPL API

Pattern Generation in Python

```
In [1]: from pattern_generation import *

In [2]: cuttingGen('Sorrentino')
Gurobi 7.5.0: optimal solution; objective 20.44444444
2 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 18.799
3 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 18.375
3 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 17.958
5 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 17.941
5 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 17.941
5 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 17.941
5 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 17.941
5 simplex iterations
1 branch-and-cut nodes
Gurobi 7.5.0: optimal solution; objective 19
3 simplex iterations
1 branch-and-cut nodes
```

Figure 1

Sorrentino: 19 rolls, 4.41% waste
QuanDec

Server side
- AMPL model and data
- Standard AMPL-solver installations

Client side
- Interactive tool for collaboration & decision-making
- Runs on any recent web browser
- Java-based implementation
  - AMPL API for Java
  - Eclipse Remote Application Platform

...developed / supported by Cassotis Consulting
Getting started

**step 1:** install QuanDec on a server

**step 2:** copy & paste your model files (.mod and .dat) into QuanDec’s workspace

**step 3:** create AMPL tables and link them to QuanDec explorer
Workbench
Features

- Server application
- Centralized data
- Several models on a single server

- Web-based
- Multi-users
- Concurrent access
- Secure access

- Scenario-based
- Sharing between users
- Sharing rights (edit / comment / view)

And much more…
Scenario-based environment

Sharing system

Permission: Edit – Comment - View
3 levels:
- Report
- Input parameters
- Variables

Chart and tables

Colored values for easier analysis

Constraint (min/max) on any variable
Collaborative work

Notification system

Comments between users
Scenarios with changes history

Traceability and undo system
**Scenario comparison**

All variables can be compared

Display of relative difference

Custom reports

---

**Figure Description**

The image shows a software interface for scenario comparison, highlighting the ability to compare all variables, display relative differences, and generate custom reports. The interface includes a table comparing economic and production data across different scenarios, such as BUDGET 2016, My Scenario, and FORECAST 2017.
Data cleaning

Regression tool

Any variable can be added to a regression

Manual coefficients if no data available
Regression tool

1. Collect data
2. Import data into QuanDec
3. Clean the data
4. Add potential predictors
5. Select the best regression
6. The regression is included in the optimization model as a constraint

The equation is:

\[ X = 2.1 + 1.5 B - 0.3 C + 4.9 D \]
Sensitivity analysis

For both parameters AND variables

All variables can be compared

Display of relative difference
Predefined analyses

Script parameters
QuanDec Availability

*Ready now for commercial applications*
- Free trials available
- Pricing keyed to number of models & users

*First year’s support included*
- Tailored setup support from Cassotis Consulting
- Customizations possible

... contact sales@ampl.com for details