Adding Optimization to Your Applications

Quickly and Reliably

1. A Guide to Model-Based Optimization
2. From Prototyping to Integration with AMPL

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

INFORMS Business Analytics Conference
Technology Workshop, 14 April 2019
Outline

1. Model-based optimization
   - Comparison of method-based and model-based approaches
   - Modeling languages for optimization
   - Algebraic modeling languages: AMPL
   - Off-the-shelf solvers for common model types

2. From prototyping to integration
   - Building models: AMPL’s interactive environment
   - Developing optimization-based procedures: AMPL scripts
   - Integrating into decision-making systems: AMPL APIs
     - Integrating with Python applications: pyMPL
     - Building a decision-making tool for deployment: QuanDec
Example: Roll Cutting

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
Roll Cutting

Mathematical Formulation

Given

- $W$ set of ordered widths
- $n$ number of 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 $i \in W$
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

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

number of rolls cut of width \( i \)
must be at least the number ordered
Roll Cutting

AMPL Formulation

Symbolic model

```AMPL
set WIDTHS;
param orders {WIDTHS} > 0;
param nPAT integer >= 0;
param nbr {WIDTHS,1..nPAT} integer >= 0;
var Cut {1..nPAT} integer >= 0;
minimize Number:
  sum {j in 1..nPAT} Cut[j];
subj to Fulfill {i in WIDTHS}:
  sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];
```

\[ \sum_{j=1}^{n} a_{ij} X_j \geq b_i, \text{ for all } i \in W \]
Roll Cutting

AMPL Formulation (cont’d)

Explicit data (independent of model)

```
param: WIDTHS: orders :=
     6.77   10
     7.56   40
    17.46   33
    18.76   10 ;
param nPAT := 9 ;
param nbr:  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  0  0  0  0 ;
```
Command Environment

Model + data = problem instance to be solved

```ampl
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
```
Command Language (cont’d)

Solver choice independent of model and data

```
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
```
Command Language (cont’d)

Results available for browsing

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

ampl: display {j in 1..nPAT} sum {i in WIDTHS} i * nbr[i,j];
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 62.03 # in each pattern

ampl: display Fulfill.slack;
  6.77 2 # overruns
  7.56 3 # of each width
  17.46 0
  18.76 3
```
Symbolic model

```plaintext
param roll_width > 0;
set WIDTHS;
param orders {WIDTHS} > 0;

param nPAT integer >= 0;
param nbr {WIDTHS,1..nPAT} integer >= 0;

var Cut {1..nPAT} integer >= 0;

minimize Number:
  sum {j in 1..nPAT} Cut[j];

minimize Waste:
  sum {j in 1..nPAT}
    Cut[j] * (roll_width - sum {i in WIDTHS} i * nbr[i,j]);

subj to Fulfill {i in WIDTHS}:
  sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];
```
Roll Cutting

Revision 1 (cont’d)

Explicit data

```
param roll_width := 64.5;

param: WIDTHS: orders :=
    6.77   10
    7.56   40
    17.46  33
    18.76  10 ;

param nPAT := 9 ;

param nbr:  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  0  0  0 ;
```
Revision 1 (cont’d)

Solutions

ampl: model cutRev1.mod;
ampl: data cutRev1.dat;
ampl: objective Number; solve;
Gurobi 8.1.0: optimal solution; objective 20
3 simplex iterations
1 branch-and-cut nodes
ampl: display Number, Waste;
Number = 20
Waste = 63.62

ampl: objective Waste; solve;
Gurobi 8.1.0: optimal solution; objective 15.62
2 simplex iterations
1 branch-and-cut nodes
ampl: display Number, Waste;
Number = 35
Waste = 15.62
Roll Cutting

Revision 2

Symbolic model

```plaintext
param roll_width > 0;
param over_lim integer >= 0;
set WIDTHS;
param orders {WIDTHS} > 0;
param nPAT integer >= 0;
param nbr {WIDTHS,1..nPAT} integer >= 0;

var Cut {1..nPAT} integer >= 0;

...

subj to Fulfill {i in WIDTHS}:
  orders[i] <= sum {j in 1..nPAT} nbr[i,j] * Cut[j]
  <= orders[i] + over_lim;
```
Roll Cutting

Revision 2 (cont’d)

Explicit data

```plaintext
param roll_width := 64.5;
param over_lim := 8;

param: WIDTHS: orders :=
  6.77  10
  7.56  40
  17.46  33
  18.76  10;

param nPAT := 9;

param nbr: 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 0 0 0;
```
Revision 2 (cont’d)

Solutions

ampl: model cutRev2.mod;
ampl: data cutRev2.dat;
ampl: objective Number; solve;
Gurobi 8.1.0: optimal solution; objective 20
7 simplex iterations
1 branch-and-cut nodes
ampl: display Number, Waste;
Number = 20
Waste = 62.04

ampl: objective Waste; solve;
Gurobi 8.1.0: optimal solution; objective 46.72
4 simplex iterations
ampl: display Number, Waste;
Number = 22
Waste = 46.72
Scripting

Bring the programmer to the modeling language

Extend modeling language syntax . . .
- Algebraic expressions
- Set indexing expressions
- Interactive commands

. . . with programming concepts
- Loops of various kinds
- If-then and If-then-else conditionals
- Assignments

Examples
- Tradeoffs between number cut and waste
- Cutting via pattern enumeration
- Cutting via pattern generation
Tradeoffs Between Objectives

Minimize rolls cut

- Set large overrun limit

Minimize waste

- Reduce overrun limit 1 roll at a time
- If there is a change in number of rolls cut
  * record total waste (increasing)
  * record total rolls cut (decreasing)
- Stop when no further progress possible
  * problem becomes infeasible
  * total rolls cut falls to the minimum
- Report table of results
Scripting

Parametric Analysis (cont’d)

Script (setup and initial solve)

```plaintext
model cutRev2.mod;
data cutRev2.dat;

set OVER default {} ordered by reversed Integers;

param minNumber;
param minNumWaste;
param minWaste {OVER};
param minWasteNum {OVER};

param prev_number default Infinity;

option solver gurobi;
option solver_msg 0;

objective Number;
solve >Nul;

let minNumber := Number;
let minNumWaste := Waste;

objective Waste;
```
Scripting

Parametric Analysis (cont’d)

Script (looping and reporting)

```plaintext
for {k in over_lim .. 0 by -1} {
    let over_lim := k;
    solve >Nul;
    if solve_result = 'infeasible' then break;
    if Number < prev_number then {
        let OVER := OVER union {k};
        let minWaste[k] := Waste;
        let minWasteNum[k] := Number;
        let prev_number := Number;
    }
    if Number = minNumber then break;
}
printf 'Min%3d rolls with waste%6.2f\n\n', minNumber, minNumWaste;
printf ' Over Waste  Number\n';
printf {k in OVER}: '%4d%8.2f%6d\n', k, minWaste[k], minWasteNum[k];
```
Scripting

Parametric Analysis (cont’d)

Script run

```
ampl: include cutWASTE.run

Min 20 rolls with waste 62.04

<table>
<thead>
<tr>
<th>Over</th>
<th>Waste</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>40.57</td>
<td>24</td>
</tr>
<tr>
<td>19</td>
<td>43.01</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>45.45</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>47.89</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>54.76</td>
<td>20</td>
</tr>
</tbody>
</table>

ampl:
```
Scripting

Cutting via Pattern Enumeration

Build the pattern list, then solve

- Read general model
- Read data: demands, raw width
- Compute data: all usable patterns
- Solve problem instance
Pattern Enumeration

Model

```plaintext
param roll_width > 0;
set WIDTHS ordered by reversed Reals;
param orders {WIDTHS} > 0;
param maxPAT integer >= 0;
param nPAT integer >= 0, <= maxPAT;
param nbr {WIDTHS,1..maxPAT} integer >= 0;

var Cut {1..nPAT} integer >= 0;

minimize Number:
    sum {j in 1..nPAT} Cut[j];

subj to Fulfill {i in WIDTHS}:
    sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];
```
Scripting

Pattern Enumeration

Data

```plaintext
param roll_width := 64.50;
param: WIDTHS: orders :=
    6.77  10
    7.56  40
    17.46  33
    18.76  10;
```
Scripting

Pattern Enumeration

Script (initialize)

```plaintext
model cutPAT.mod;
param dsetname symbolic;
printf "Enter dataset name:
";
read dsetname <-;
data (dsetname & ".dat");

model;
param curr_sum >= -1e-10;
param curr_width > 0;
param pattern {WIDTHS} integer >= 0;
let maxPAT := 100000000;
let nPAT := 0;
let curr_sum := 0;
let curr_width := first(WIDTHS);
let {w in WIDTHS} pattern[w] := 0;
```
Scripting

Pattern Enumeration

Script (loop)

```plaintext
repeat {
    if curr_sum + curr_width <= roll_width then {
        let pattern[curr_width] := floor((roll_width-curr_sum)/curr_width);
        let curr_sum := curr_sum + pattern[curr_width] * curr_width;
    }
    if curr_width != last(WIDTHS) then
        let curr_width := next(curr_width,WIDTHS);
    else {
        let nPAT := nPAT + 1;
        let {w in WIDTHS} nbr[w,nPAT] := pattern[w];
        let curr_sum := curr_sum - pattern[last(WIDTHS)] * last(WIDTHS);
        let pattern[last(WIDTHS)] := 0;
        let curr_width := min {w in WIDTHS: pattern[w] > 0} w;
        if curr_width < Infinity then {
            let curr_sum := curr_sum - curr_width;
            let pattern[curr_width] := pattern[curr_width] - 1;
            let curr_width := next(curr_width,WIDTHS);
        } else break;
    }
}
```
Pattern Enumeration

Script (solve, report)

```plaintext
printf "\nAT LEAST %d ROLLS REQUIRED\n\n",
    ceil((sum {i in WIDTHS} i * orders[i]) / roll_width);

option solver gurobi;
solve;

printf \n\n%5i patterns, %3i rolls\n\n",
    nPAT, sum {j in 1..nPAT} Cut[j];
printf \n\n Cut \n",
    {j in 1..nPAT: Cut[j] > 0}: "%3i", Cut[j];
printf \n\n;
for {i in WIDTHS} {
    printf \n\n%7.2f , i;
    printf \n\n{j in 1..nPAT: Cut[j] > 0}: "%3i", nbr[i,j];
    printf \n\n;
}  
```
Pattern Enumeration

Results

ampl: include cutPatEnum.run

AT LEAST 18 ROLLS REQUIRED

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

43 patterns, 18 rolls

<table>
<thead>
<tr>
<th>Cut</th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.76</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17.46</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7.56</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>6.77</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Scripting

Pattern Enumeration

Data 2

param roll_width := 349;
param: WIDTHS: orders :=
   28.75   7
   33.75   23
   34.75   23
   37.75   31
   38.75   10
   39.75   39
   40.75   58
   41.75   47
   42.25   19
   44.75   13
   45.75   26;
Pattern Enumeration

Results 2

<table>
<thead>
<tr>
<th>Cut</th>
<th>7</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>5</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.75</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44.75</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>42.25</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41.75</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40.75</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>39.75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>38.75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>37.75</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>34.75</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>33.75</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>28.75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
**Scripting**

**Pattern Enumeration**

**Data 3**

```plaintext
param roll_width := 172 ;
param: WIDTHS: orders :=
    25.000   5
    24.750   73
    18.000   14
    17.500   4
    15.500   23
    15.375   5
    13.875   29
    12.500   87
    12.250   9
    12.000   31
    10.250   6
    10.125   14
    10.000   43
    8.750    15
    8.500    21
    7.750    5 ;
```
Scripting

Pattern Enumeration

Results 3 (using 1% of generated patterns)

ampl: include cutPatEnum.run

AT LEAST 33 ROLLS REQUIRED

Gurobi 8.1.0: optimal solution; objective 33
321 simplex iterations
1 branch-and-cut nodes

273380 patterns, 33 rolls

<table>
<thead>
<tr>
<th>Cut</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>6</th>
<th>4</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.00</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24.75</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>18.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>17.50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8.75</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8.50</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7.75</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Generating the pattern list by a series of solves

- Solve an easy cutting problem using a subset of patterns
  - Allow *fractional* amounts cut
  - Get dual values (shadow prices) on order requirements
- Find a “most promising” pattern to add to the subset
  - Minimize pattern’s 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 no longer negative
- Solve full cutting problem using all patterns found
  - Require *integer* amounts cut
Pattern Generation

Cutting model

```plaintext
set WIDTHS ordered by reversed Reals;
param orders {WIDTHS} > 0;

param nPAT integer >= 0;
param nbr {WIDTHS,1..nPAT} integer >= 0;

var Cut {1..nPAT} integer >= 0;

minimize Number:
    sum {j in 1..nPAT} Cut[j];

subj to Fill {i in WIDTHS}:
    sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];
```
Pattern Generation

Pattern-generating model

```plaintext
param roll_width > 0;
param price {WIDTHS} default 0.0;

var Use {WIDTHS} integer >= 0;

minimize Reduced_Cost:
    1 - sum {i in WIDTHS} price[i] * Use[i];

subj to Width_Limit:
    sum {i in WIDTHS} i * Use[i] <= roll_width;
```
Scripting

Pattern Generation

Script (define models)

```markdown
model cutPatGen.mod;

problem Cutting_Opt: Cut, Number, Fill;
    option relax_integrality 1;
    option presolve 0;

problem Pattern_Gen: Use, Reduced_Cost, Width_Limit;
    option relax_integrality 0;
    option presolve 1;
```
Scripting

Pattern Generation

Script (data, initial patterns)

```plaintext
param dsetname symbolic;
print "Enter dataset name:";
read dsetname <-;
data (dsetname & ".dat");

let nPAT := 0;
for {i in WIDTHS} {
    let nPAT := nPAT + 1;
    let nbr[i,nPAT] := floor (roll_width/i);
    let {i2 in WIDTHS: i2 <> i} nbr[i2,nPAT] := 0;
};

printf "AT LEAST %d ROLLS REQUIRED",
    ceil((sum {i in WIDTHS} i * orders[i]) / roll_width);
```
Scripting

Pattern Generation

Script (generation loop)

```plaintext
repeat {
    solve Cutting_Opt;
    let {i in WIDTHS} price[i] := Fill[i].dual;
    solve Pattern_Gen;
    printf "\n%7.2f%11.2e ", Number, Reduced_Cost;
    if Reduced_Cost < -0.00001 then {
        let nPAT := nPAT + 1;
        let {i in WIDTHS} nbr[i,nPAT] := Use[i];
    } else break;
    for {i in WIDTHS} printf "%3i", Use[i];
};
```
Pattern Generation

Script (final integer solution)

```plaintext
option Cutting_Opt.relax_integrality 0;
option Cutting_Opt.presolve 10;
solve Cutting_Opt;

if Cutting_Opt.result = "infeasible" then
  printf "\n*** No feasible integer solution ***\n\n";
else {
  printf "Best integer: %3i rolls\n\n", sum {j in 1..nPAT} Cut[j];
  for {j in 1..nPAT: Cut[j] > 0} {
    printf "%3i of:\n", Cut[j];
    printf {i in WIDTHS: nbr[i,j] > 0}: "%3i x %6.3f", nbr[i,j], i;
    printf "\n";
  }
  printf "\nWASTE = %5.2f\n\n",
    100 * (1 - (sum {i in WIDTHS} i * orders[i]) / (roll_width * Number));
}
```
### Pattern Generation

#### Results (relaxation)

```plaintext
ampl: include cutPatGen.run
Enter dataset name:
ampl? Sorrentino

<table>
<thead>
<tr>
<th>Value</th>
<th>Coefficient</th>
<th>Type</th>
<th>Category</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.44</td>
<td>-1.53e-01</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>18.78</td>
<td>-1.11e-01</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>18.37</td>
<td>-1.25e-01</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17.96</td>
<td>-4.17e-02</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>17.94</td>
<td>-1.00e-06</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimal relaxation: **17.9412 rolls**

- 10.0000 of:  2 x 17.460  3 x  7.560  1 x  6.770
- 4.1961 of:  3 x 17.460  1 x  7.560
- 3.3333 of:  3 x 18.760  1 x  7.560
- 0.4118 of:  1 x 17.460  6 x  7.560

WASTE = 23.33 (2.02%)
```
Pattern Generation

Results (integer)

<table>
<thead>
<tr>
<th>Cut</th>
<th>10</th>
<th>5</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.76</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>17.46</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7.56</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>6.77</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

WASTE = 28.42 (2.20%)

Best integer: 19 rolls

<table>
<thead>
<tr>
<th>Cut</th>
<th>10</th>
<th>4</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.76</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>17.46</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7.56</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>6.77</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

WASTE = 23.86 (1.95%)
Scripting in practice...

Large and complex scripts

- Multiple files
- Hundreds of statements
- Millions of statements executed

Coordination with enterprise systems

- Your system
  - writes data files
  - invokes `ampl optapp.run`
- AMPL’s script
  - reads the data files
  - processes data, generates problems, invokes solvers
  - writes result files
- Your system
  - reads the result files
Scripting

Limitations

Scripts can be slow
- Interpreted, not compiled
- Very general set & data structures

Script programming constructs are limited
- Based on a declarative language
- Not object-oriented

Scripts are stand-alone
- Close AMPL environment before returning to system

What are the alternatives?
- Bring the modeling language to the programmer (AMPL APIs)
- Enhance integration of modeling and programming (pyMPL)
- Build a deployment tool (QuanDec)
**APIs** (application programming interfaces)

*Bring the modeling language to the programmer*

- Data and result management in a general-purpose programming language
- Modeling and solving through calls to AMPL

*Add-ons to all AMPL distributions*

- Java, MATLAB, C++, C#
  - Download from http://ampl.com/products/api/
- **Python** 2.7, 3.3, 3.4, 3.5, 3.6
  - pip install amplpy
- **R now available!**
  - install.packages("Rcpp", type="source")
  - install.packages(
    "https://ampl.com(dl/API/rAMPL.tar.gz", repos=NULL)
Cutting Revisited

Hybrid approach

- Control & pattern enumeration from a programming language
- Model definition & modeling commands in AMPL

Key to examples: Python and R

- AMPL entities
- AMPL API Python/R objects
- AMPL API Python/R methods
- Python/R functions etc.
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')
```
Pattern Enumeration in R

Load & generate data, set up AMPL model

```r

`cuttingEnum <- function(dataset) {
  library(rAMPL)

  # Read orders, roll_width, overrun
  source(paste(dataset, "\R", sep=""))

  # Enumerate patterns
  patmat <- patternEnum(roll_width, orders$width)
  cat(sprintf("\n%d patterns enumerated\n\n", ncol(patmat)))

  # Set up model
  ampl <- new(AMPL)
  ampl$setOption("ampl_include", "models")
  ampl$read("cut.mod")
}

```

AMPL API
### 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 R**

**Send data to AMPL**

```r
# Send scalar values
ampl$getParameter("nPatterns")$set(ncol(patmat))
ampl$getParameter("overrun")$set(overrun)
ampl$getParameter("rawWidth")$set(roll_width)

# Send order vector
ampl$getSet("WIDTHS")$setValues(orders$width)
ampl$getParameter("order")$setValues(orders$demand)

# Send pattern matrix
df <- as.data.frame(as.table(patmat))
df[,1] <- orders$width[df[,1]]
df[,2] <- as.numeric(df[,2])
ampl$getParameter("rolls")$setValues(df)
```
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 R

Solve and get results

```r
# Solve
ampl$setOption("solver", "gurobi")
ampl$solve()

# Retrieve solution
CuttingPlan <- ampl$getVar("Cut")$getValues()
solution <- CuttingPlan[CuttingPlan[,-1] != 0,]
```
**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] * \
        (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)
```
**AMPL API**

**Pattern Enumeration in R**

**Display solution**

```r
# Prepare solution data
data <- dataset
obj <- ampl$getObjective("TotalRawRolls")$value()
waste <- ampl$getValue(
  "sum {p in PATTERNS} Cut[p] * (rawWidth - sum {w in WIDTHS} w*rolls[w,p])"
)
summary <- list(data=dataset, obj=obj, waste=waste)

# Create plot of solution
cuttingPlot(roll_width, orders$width, patmat, 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
```
**Enumeration routine**

```r
patternEnum <- function(roll_width, widths, prefix=c()) {
  cur_width <- widths[length(prefix)+1]
  max_rep <- floor(roll_width/cur_width)
  if (length(prefix)+1 == length(widths)) {
    return (c(prefix, max_rep))
  } else {
    patterns <- matrix(nrow=length(widths), ncol=0)
    for (n in 0:max_rep) {
      patterns <- cbind(
        patterns,
        patternEnum(roll_width-n*cur_width, widths, c(prefix, n))
      )
    }
    return (patterns)
  }
}
```
Pattern Enumeration in Python

Plotting routine

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']
**AMPL API**

**Pattern Enumeration in R**

**Plotting routine**

```r
cuttingPlot <- function(roll_width, widths, patmat, summary, solution) {
  pal <- rainbow(length(widths))
  par(mar=c(1,1,1,1))
  par(mfrow=c(1,nrow(solution)))
  for(i in 1:nrow(solution)) {
    pattern <- patmat[, solution[i, 1]]
    data <- c()
    color <- c()
  }
}
```
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()
```
**AMPL API**

**Pattern Enumeration in R**

**Plotting routine (cont’d)**

```r
for(j in 1:length(pattern)) {
    if(pattern[j] >= 1) {
        for(k in 1:pattern[j]) {
            data <- rbind(data, widths[j])
            color <- c(color, pal[j])
        }
    }
}

label <- sprintf("x %d", solution[i, -1])

barplot(data, main=label, col=color, border="white", space=0.04, axes=FALSE, ylim=c(0, roll_width))

print(summary)

}```
**AMPL API**

**Pattern Enumeration in Python**

```python
In [1]: from patternEnumeration 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](image)

Sorrentino: 18 rolls, 2.34% waste
**AMPL API**

**Pattern Enumeration in R**

```r
> source("PatternEnumeration.R")
> cuttingEnum("Sorrentino")

95 patterns enumerated

$data
[1] "Sorrentino"

$obj
[1] 18

$waste
[1] 27.12

> |
```
Much to do in Python, R, MATLAB, etc.
- Prepare order
- Generate & sample patterns
- Feed results to visualization and implementation

Key role for modeling in AMPL
- Prototype and refine a model
- Evolve and maintain the model reliably
- Manage the interface to your choice of solvers
PyMPL (Python integration with AMPL)

Enhance integration of modeling and programming

Roll Cutting enhanced

- Sending Python data to an AMPL model
  - via AMPL API for Python
  - via Python references in the AMPL model
- Programming a custom stopping criterion in Python
  - via callbacks from the Gurobi solver
- Maintaining a view of the integrated application
  - via Jupyter notebooks

Lot Sizing using advanced formulations

- Generating specialized constraints
  - via Python embedded in AMPL scripts
Python Integration

Sending Python Data to an AMPL model

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

Symbolic sets and parameters in 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;
```
Sending Data using the Python API (cont’d)

*Call ampl methods to read model, send data*

```python
ampl = AMPL()

.......  # omissions

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))
}
```
Python Integration

Sending Data using PyMPL

Specify Python data correspondences in the model

```python
ampl = AMPL(langext=PyMPL())
........
$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))
    }
};
```
Python Integration

Callbacks

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': (.0002, .002, .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
```
Python Integration

Callbacks

Run inside Jupyter notebook

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
Python Integration

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
```

`lotsize.mod`
Python Integration

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)*

```ampl
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[wk,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];
```
QuanDec

Building a decision-making tool for deployment

Implemented in the Java API for AMPL

- Developed and supported by Cassotis Consulting
**QuanDec**

**Architecture**

**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
QuanDec

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
QuanDec

Workbench

Explorer

- section 1
- section 2
  - category 2.1
  - category 2.2
  - category 2.3

Viewer

- Charts
- Water
- Barley
- Hops
- Yeast

Report tables

- Export
- Edit bounds
- Comment
- Analyze sensitivity

Input tables

- Import
- Edit values
- Edit set: new/remove/duplicate
- Comment

Journal | Bounds | Regressions | Comments

Console

>_ _
**QuanDec**

**Scenarios**

- Scenario comparison
- All variables can be compared
- Display of relative difference
- Custom reports