Building Optimization-Enabled Applications Using AMPL API

Robert Fourer
4er@ampl.com
AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Annual Meeting
Nashville — 13-16 November 2016 — Session TA19
Optimization Modeling and Beyond with a Focus on Practice
Building Optimization-Enabled Applications Using AMPL API

We describe how to combine the power of the AMPL modeling system and a general-purpose programming language to build rich optimization-enabled client applications. Having an optimization model expressed in a high-level declarative form with model and data separation facilitates its evolution and maintenance, and makes switching between different solvers and data sources easy. At the same time it is possible to use a familiar development environment and have access to a wide variety of programming libraries for data management and interface development.
Features

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

Design goals

- Powerful, general expressions
- Natural, easy-to-learn modeling principles
- Efficient processing that scales well with problem size

3 ways to use . . .
3 Ways to Use AMPL

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

Scripting language
- Bring the programmer to the modeling language

Programming interface (API)
- Bring the modeling language to the programmer
Example

Roll cutting model
  ❖ Solution via command language
  ❖ Tradeoff analysis via scripting

Roll cutting by pattern enumeration
  ❖ via scripting
  ❖ via API

Roll cutting by pattern generation
  ❖ via scripting
  ❖ via API
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
**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 of width \( i \) cut

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];
```

$$
\sum_{j=1}^{n} a_{ij} X_j \geq b_i
$$
Roll Cutting

AMPL Formulation (cont’d)

Explicit data (independent of model)

\[
\begin{align*}
\text{param: WIDTHS: } & \text{orders :=} \\
& \begin{array}{ll}
6.77 & 10 \\
7.56 & 40 \\
17.46 & 33 \\
18.76 & 10 \\
\end{array} \\
\text{param nPAT := 9 ;} \\
\text{param nbr: } & 1 2 3 4 5 6 7 8 9 := \\
& \begin{array}{llllllllll}
6.77 & 0 & 1 & 1 & 0 & 3 & 2 & 0 & 1 & 4 \\
7.56 & 1 & 0 & 2 & 1 & 1 & 4 & 6 & 5 & 2 \\
17.46 & 0 & 1 & 0 & 2 & 1 & 0 & 1 & 1 & 1 \\
18.76 & 3 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\end{align*}
\]
**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 12.6.3.0: optimal integer solution; objective 20
3 MIP simplex iterations
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
4 13 7 4 9 3
```
Solver choice independent of model and data

AML: model cut.mod;
AML: data cut.dat;
AML: option solver gurobi;
AML: solve;
Gurobi 7.0.0: optimal solution; objective 20
3 simplex iterations
AML: option omit_zero_rows 1;
AML: option display_1col 0;
AML: 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  # pattern
  2 61.75 4 61.24 6 62.54 8 62.0  # total widths

ampl: display Fulfill.slack;
  6.77 2  # overruns
  7.56 3
  17.46 0
  18.76 3
Roll Cutting

Revision 1: Waste vs. # of Rolls

Symbolic model

```AMPL
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
Roll Cutting

Revision 1 (cont’d)

Explicit data

```ampl
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  1  0  0  0;
```
Revision 1 (cont’d)

Solutions

ampl: model cutRev1.mod;
ampl: data cutRev1.dat;

ampl: objective Number; solve;
Gurobi 7.0.0: optimal solution; objective 20
3 simplex iterations
ampl: display Number, Waste;
  Number = 20
  Waste = 63.62

ampl: objective Waste; solve;
Gurobi 7.0.0: optimal solution; objective 15.62
2 simplex iterations
ampl: display Number, Waste;
  Number = 35
  Waste = 15.62
Roll Cutting

Revision 2: Overrun Limit

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

```ampl
param roll_width := 64.5;
param over_lim := 6 ;

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  1  0  0  0 ;
```
Revision 2 (cont’d)

Solutions

```
ampl: model cutRev2.mod;
ampl: data cutRev2.dat;

ampl: objective Number; solve;
Gurobi 7.0.0: optimal solution; objective 20
7 simplex iterations
ampl: display Number, Waste;
Number = 20
Waste = 54.76

ampl: objective Waste; solve;
Gurobi 7.0.0: optimal solution; objective 49.16
4 simplex iterations
ampl: display Number, Waste;
Number = 21
Waste = 49.16
```
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 objectives
- Cutting via pattern enumeration
- Cutting via pattern generation
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
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];
```
Parametric Analysis (cont’d)

Script run

```ampl
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>10</td>
<td>46.72</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
Scripting

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

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)

model cutPAT.mod;
data Sorrentino.dat;

param curr_sum >= 0;
param curr_width > 0;
param pattern {WIDTHS} integer >= 0;

let maxPAT := 1000000;
let nPAT := 0;
let curr_sum := 0;
let curr_width := first(WIDTHS);
let {w in WIDTHS} pattern[w] := 0;
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;
    }
}
```
Scripting

Pattern Enumeration

Script (solve, report)

```ampl
option solver gurobi;
solve;
printf "\n%5i patterns, %3i rolls", nPAT, sum {j in 1..nPAT} Cut[j];
printf "\n\n Cut  ";
printf {j in 1..nPAT: Cut[j] > 0}: "%3i", Cut[j];
printf "\n\n";
for {i in WIDTHS} {
    printf "%7.2f ", i;
    printf {j in 1..nPAT: Cut[j] > 0}: "%3i", nbr[i,j];
    printf "\n";
}
printf "\nWASTE = %5.2f%%\n\n",
    100 * (1 - (sum {i in WIDTHS} i * orders[i]) / (roll_width * Number));
```
Scripting

Pattern Enumeration

Results

```
ampl: include cutPatEnum.run
Gurobi 7.0.0: optimal solution; objective 18
5 simplex iterations

43 patterns, 18 rolls

Cut    2  2  3 11
18.76  3  2  0  0
17.46  0  1  3  2
 7.56  1  1  1  3
 6.77  0  0  1  1
WASTE =  2.34%
```
Cutting via Pattern Generation

Generate the pattern list by a series of solves

- Solve LP relaxation using subset of patterns
- Add “most promising” pattern to the subset
  - Minimize reduced cost given dual values
  - Equivalent to a 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

Cutting model

set WIDTHS ordered by reversed Reals;
param orders {WIDTHS} > 0;
param nPAT integer >= 0, <= maxPAT;
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];
Pattern Generation

Knapsack model

```AMPL
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;
```
Pattern Generation

Script (problems, initial patterns)

```ampl
model cutPatGen.mod;
data Sorrentino.dat;

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;

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;
};
```
Scripting

Pattern Generation

Script (generation loop)

```AMPL
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];
};
```
Scripting

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));
}
```
Scripting

Pattern Generation

Results (relaxation)

```
AMPL: include cutpatgen.run

20.44  -1.53e-01    1  3  2  0
18.78  -1.11e-01    0  1  3  0
18.37  -1.25e-01    0  1  0  3
17.96  -4.17e-02    0  6  0  1
17.94  -1.00e-06

Optimal relaxation: 17.9412 rolls

10.0000 of:  1 x 6.770  3 x  7.560  2 x 17.460
4.3333 of:  1 x 7.560  3 x 17.460
3.1961 of:  1 x 7.560  3 x 18.760
0.4118 of:  6 x 7.560  1 x 18.760

WASTE = 2.02%
```
Pattern Generation

Results (integer)

Rounded up to integer: 20 rolls

Cut
6.77  1  0  0  0
7.56  3  1  1  6
17.46  2  3  0  0
18.76  0  0  3  1

WASTE = 12.10%

Best integer: 19 rolls

Cut
6.77  1  0  0  0
7.56  3  1  1  6
17.46  2  3  0  0
18.76  0  0  3  1

WASTE = 7.48%
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
Cutting Revisited

Hybrid approach

- Control & pattern creation from a programming language
  - Pattern enumeration: finding all patterns
  - Pattern generation: solving knapsack problems
- Model & modeling commands in AMPL

Choice of programming languages

- Java, MATLAB, C++

Key to program examples

- AMPL entities
- objects
- methods for working with AMPL
- functions
**AMPL API**

**AMPL Model File**

Basic pattern-cutting model

```plaintext
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 rolls {WIDTHS,PATTERNS} >= 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;
```

function cuttingEnum(dataFile)

% Get data from .mat file: roll_width, overrun, widths, orders
load(dataFile);

% Generate pattern matrix
[widthsDec, ind] = sort(widths, 'descend');
patmat = patternEnum(roll_width, widthsDec);
patmat(:, ind) = patmat;

% Initialize and load cutting-stock model from file
ampl = AMPL();
ampl.read('cut.mod');
**AMPL API**

**Pattern Enumeration in MATLAB**

**Send data to AMPL**

```matlab
% Send scalar values
ampl.getParameter('overrun').setValues(overrun);
ampl.getParameter('nPaterns').setValues(length(patmat));

% Send order vector
WidthOrder = DataFrame(1, 'WIDTHS', 'order');
WidthOrder.setColumn('WIDTHS', num2cell(widths));
WidthOrder.setColumn('order', orders);
ampl.setData(WidthOrder, 'WIDTHS');

% Send pattern matrix
AllPatterns = DataFrame(2, 'WIDTHS', 'PATTERNS', 'rolls');
AllPatterns.setMatrix(patmat', num2cell(widths), num2cell(1:length(patmat)));
ampl.setData(AllPatterns)
```
**AMPL API**

Pattern Enumeration in MATLAB

**Solve and report**

```matlab
% Solve
ampl.setOption('solver', 'gurobi');
ampl.solve

% Retrieve solution
CuttingPlan = ampl.getVariable('Cut').getValues();
cutvec = CuttingPlan.getColumnAsDoubles('val');

% Display solution
cuttingPlot (roll_width, widths, patmat(cutvec>0,:), cutvec(cutvec>0))
```
**AMPL API**

**Pattern Enumeration in MATLAB**

*Enumeration routine*

```
function patmat = patternEnum(rollwidth, widths)
if length(widths) == 1
    patmat = floor(rollwidth/widths(1));
else
    patmat = [];
    for n = floor(rollwidth/widths(1)):-1:0
        patnew = patternEnum (rollwidth-n*widths(1), widths(2:end));
        patmat = [patmat; n*ones(size(patnew,1),1) patnew];
    end
end
```

function cuttingPlot (roll_width, widths, patmat, cutvec)

plotmat = zeros(length(cutvec), sum(max(patmat)));
colors = jet(length(widths));

plotpos = 0;
for j = 1:length(widths)
    for i = 1:length(cutvec)
        plotmat(i, plotpos+1:plotpos+patmat(i,j)) = widths(j);
    end
    for i = 1:max(patmat(:,j))
        colormat(plotpos+i,:) = colors(j,:);
    end
    plotpos = plotpos + max(patmat(:,j));
end

colormap(colormat); shading faceted
h = barh(plotmat,'stacked');
set (h, 'edgecolor','black'); set(gca,'YTickLabel',num2cell(cutvec))
xlim([0,roll_width]); ylim([0,numel(get(gca,'YTick'))+1])
Pattern Enumeration in MATLAB
**AMPL API**

**Pattern Enumeration in Java**

*Generate patterns, set up AMPL model*

```java
public static void main(String[] args) throws IOException {
    import static com.ampl.examples.CuttingStock.Sorrentino;
    int[] sortedWidths = widths.clone();
    sortDescending(sortedWidths);
    ArrayList<Integer> patterns = new ArrayList<>();
    patternEnum (roll_width, sortedWidths, 0, patterns);

    // Initialize and load cutting-stock model from file
    AMPL ampl = new AMPL();
    try {
        ampl.read("cut.mod");
    }
```
Pattern Enumeration in Java

Send data to AMPL

```java
AMPL API

ampl.getParameter("overrun").setValues(overrun);
int numPatterns = patterns.size() / widths.length;
ampl.getParameter("nPatterns").setValues(numPatterns);

DataFrame widthOrder = new DataFrame(1, "WIDTHS", "order");
widthOrder.setColumn("WIDTHS", widths);
widthOrder.setColumn("order", orders);
ampl.setData(widthOrder, true);

DataFrame allPatterns = new DataFrame(2, "WIDTHS", "PATTERNS", "rolls");
for (int i = 0; i < widths.length; i++) {
    for (int j = 0; j < numPatterns; j++) {
        allPatterns.addRow(
            sortedWidths[i], j + 1, patterns.get(j * widths.length + i));
    }
}
ampl.setData(allPatterns, false);
```
**AMPL API**

**Pattern Enumeration in Java**

*Solve and report solution*

```java
ampl.setOption("solver", "gurobi");
ampl.solve();
printSolution (ampl.getVariable("Cut"), ampl.getParameter("rolls"));
} finally {
    ampl.close();
}
```
In practice . . .

Integrate within a larger scheme

Retain benefits of algebraic modeling
  ❖ work with natural representation of optimization models
  ❖ efficient prototyping, reliable maintenance

Use the best tools for each part of the project
  ❖ program data manipulation in your choice of language
  ❖ work with optimization models in AMPL
Pattern Generation in MATLAB

Set up AMPL, get data

```matlab
function cuttingGen(dataFile)

% Initialize
ampl = AMPL();

% Load cutting-stock model from file
ampl.read('cut.mod');
Cut = ampl.getVariable('Cut');
Limits = ampl.getConstraint('FinishedRollLimits');

% Get data from .mat file: roll_width, overrun, widths, orders
load(dataFile);
```
% Send scalar values
ampl.getParameter('overrun').setValues(overrun);
ampl.getParameter('nPattens').setValues(length(widths));

% Send order vector
WidthOrder = DataFrame(1, 'WIDTHS', 'order');
WidthOrder.setColumn('WIDTHS', num2cell(widths));
WidthOrder.setColumn('order', orders);
ampl.setData(WidthOrder, 'WIDTHS');

% Generate and send initial pattern matrix
maxpat = floor(roll_width./widths);
patmat = diag(maxpat);
InitPatterns = DataFrame(2, 'WIDTHS', 'PATTENS', 'rolls');
InitPatterns.setMatrix(patmat, num2cell(widths), num2cell(1:length(widths)));
ampl.setData(InitPatterns);
Pattern Generation in MATLAB

Set up for generation loop

```matlab
% Set solve options
ampl.setOption('solver','gurobi');
ampl.setOption('relax_integrality','1');

% Set up DataFrame for sending AMPL new patterns
ampl.eval('param newpat {WIDTHS} integer >= 0;');
NewPattern = DataFrame(1, 'WIDTHS', 'newpat');
NewPattern.setColumn('WIDTHS', num2cell(widths));

% Compute multiplier for integer weights
[n,d] = rat(widths);
intmult = lcms(d);
```
**AMPL API**

**Pattern Generation in MATLAB**

**Loop 1: Retrieve duals & look for new pattern**

```matlab
while 1
    ampl.solve
    DualPrices = Limits.getValues;
    dualvec = DualPrices.getColumnAsDoubles('dual');

    wgt = []; val = [];
    for w = 1:length(widths)
        if dualvec(w) > 0
            wgt = [wgt widths(w)*ones(1,maxpat(w))];
            val = [val dualvec(w)*ones(1,maxpat(w))];
        end
    end

    % Solve knapsack problem for potential new pattern
    [kmax,z] = kp01 (round(intmult*wgt), val, intmult*roll_width);
    if kmax < 1.000001
        break;
    end
end
```
**AMPL API**

**Pattern Generation in MATLAB**

**Loop 2: Send new pattern to AMPL**

```matlab
widthlist = wgt(z);
for w = 1:length(widths)
    newpat(w) = length(find(widthlist==widths(w)));
end
patmat = [patmat; newpat];
NewPattern.setColumn('newpat', newpat);
ampl.setData(NewPattern);
ampl.eval('let nPatterns := nPatterns + 1;');
ampl.eval('let {w in WIDTHS} rolls[w,nPatterns] := newpat[w];');
end

% Compute and display integer solution
ampl.setOption('relax_integrality','0');
ampl.solve;
CuttingPlan = Cut.getValues();
cutvec = CuttingPlan.getColumnAsDoubles('val');
cuttingPlot (roll_width, widths, patmat(cutvec>0,:), cutvec(cutvec>0))
```
**AMPL API**

**Pattern Generation in MATLAB**
In practice . . .

Implement hybrid iterative schemes
  ❖ build powerful software for hard problems

Alternate between optimization & other analytics
  ❖ invoke specialized optimizers for subproblems
AMPL API

Data Transfer: Alternatives

Process

- Define symbolic sets & parameters in AMPL model
- Create corresponding objects in program
- Transfer data using API methods
  * Program to AMPL
  * AMPL to program

Methods for transfer between . . .

- Scalar values
- Collections of values
  * AMPL indexed expressions
  * Java arrays, MATLAB matrices
- Relational tables
  * AMPL “table” structures
  * API DataFrame objects in Java, MATLAB
Availability

AMPL API version 1.2 released

- Java, MATLAB, C++ (beta)
- Add-ons to all AMPL distributions
- Download from www.ampl.com/products/api/

More languages to follow

- Python
- R
- .NET: C#, Visual Basic