Developing and Deploying Optimization Applications with AMPL

Speaker Introduction

- **Greg Glockner**

  - Vice President of Engineering at Gurobi Optimization
  - Manages Gurobi worldwide support
  - Global experience in optimization training and consulting
  - Winner of the INFORMS Transportation Science dissertation prize

Speaker Introduction

- **Bob Fourer**

  - Co-founder of AMPL Optimization
  - Professor of Industrial Engineering & Management Sciences, Northwestern University
  - Expert in optimization modeling systems & methods
  - Co-recipient of the INFORMS Impact Prize for development of algebraic modeling languages
  - Created the AMPL modeling language and system with David Gay and Brian Kernighan of Bell Laboratories

Introduction

- **A wide variety of systems to express optimization models**
  - Gurobi provides APIs for popular programming languages
    - C, C++, C#, Java, Python, MATLAB, R
  - Others provide a variety of higher-level interfaces
    - Spreadsheet optimizers
    - Optimization modeling extensions to programming languages
    - Algebraic modeling systems for optimization
- **This webinar highlights AMPL**
  - A widely used algebraic modeling system
  - Designed for fast development and reliable deployment

Developing and Deploying Optimization Applications with AMPL

Robert Fourer

4er@ampl.com

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

Gurobi Webinar

19 May 2016

The Optimization Modeling Cycle

Steps

- Communicate with problem owner
- Build model
- Prepare data
- Generate optimization problem
- Submit problem to Gurobi solver
- Report & analyze results
- ***Repeat!***

Goals for optimization software

- Do this quickly and reliably
- Get results before client loses interest
- ***Deploy for application***

Optimization Modeling Languages

Two forms of an optimization problem

- Modeler's form
  - High-level description, easy for people to work with
- Algorithm's form
  - Explicit data structure, easy for solvers to compute with

Idea of a modeling language

- ***A computer-readable modeler's form***
  - You write optimization problems in a modeling language
  - Computers translate to algorithm's form for solution

Reasons to consider a modeling language

- Faster modeling cycles
- More reliable modeling
- More maintainable applications

Algebraic Modeling Languages

Formulation concept

- Define data in terms of sets & parameters
  - Analogous to database keys & records
- Define decision variables
- Minimize or maximize a function of decision variables
- Subject to equations or inequalities
  that constrain the values of the variables

Advantages

- Familiar
- Powerful
- Proven

Features

- Algebraic modeling language
- Built specially for optimization

Design goals

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

A Note on Performance

- **Frequent customer question:**
  - "What's the best language for building my optimization model?"
- **Underlying questions:**
  - Which is easiest to use?
  - Which fits best with my application?
  - *Which gives the best performance?*

A Simple Example

- **Let's build a simple network-flow model**
  - 100K nodes and 1M edges
  - Resulting model is a continuous linear program
- **Consider time required to**
  - Generate the LP and send it to Gurobi
  - Solve the LP in Gurobi
  - Average from 5 trials on Xeon E3-1240 (3.40 GHz)

In C

```
for (j = 0; j < nedges; j++) {
    ind[0] = s[j];
    ind[1] = t[j];
    val[0] = -1;
    val[1] = 1;

    error = GRBaddvar(model, 2, ind, val, 0, 0,(double)c[j], GRB_CONTINUOUS, NULL);
    if (error) goto QUIT;
}
```

- **Runtime to build model: 0.30 sec**

In Python

```
for i in range(N):
  incoming = sum(x[e] for e in edges.select('*', i))
  outgoing = sum(x[e] for e in edges.select(i, '*'))
  m.addConstr(incoming == outgoing)
```

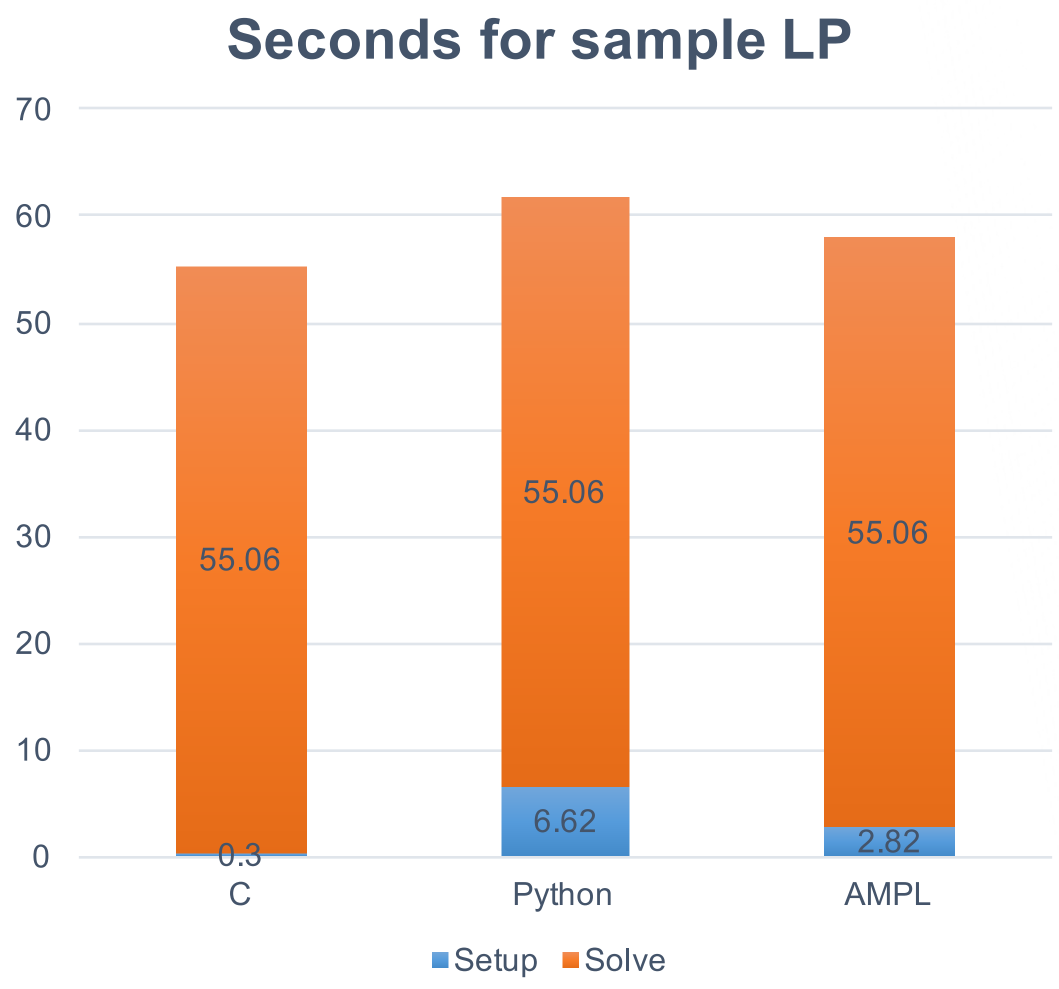
- **Runtime to build model: 6.62 sec**

In AMPL

```
subject to Balance {i in NODES diff {0,n-1}}:
   sum {(j,i) in ARCS} Flow[j,i] = sum {(i,j) in ARCS} Flow[i,j];
```

- **Runtime to build model: 2.81 sec**

Should Performance Guide Language Choice?



- **Language features should be top priority**
  - Ease of development, deployment and maintenance
  - How well does language fit the needs of the application
- **Performance should be much lower in priority**
  - Time and memory for model setup should be much less than for solving

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

Series of Examples

Roll cutting model

- Solution via command language
- Tradeoff analysis via scripting

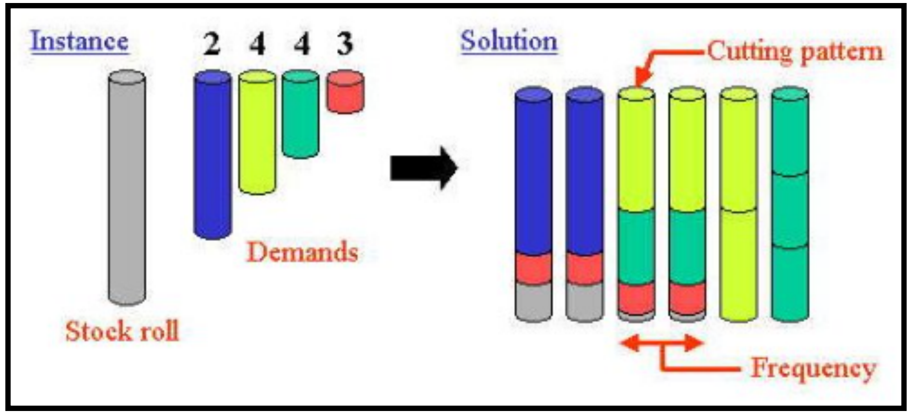
Roll cutting by pattern enumeration

- 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 stock rolls of 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

Algebraic 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 $i \in W$

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

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

number of rolls of width i cut
must be at least the number ordered

AMPL Formulation

Symbolic model

```
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];

subject to Fulfill {i in WIDTHS}:

   sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];
```

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

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  1  0  0  0 ;
```

Command Language

Model + data = problem instance to be solved

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

ampl: option solver gurobi;

ampl: solve;

Gurobi 6.5.0: optimal solution; objective 20
3 simplex iterations

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         # sum of widths
2 61.75   4 61.24   6 62.54   8 62.0                    # in each pattern

ampl: display Fulfill.slack;

 6.77  2                                                # overruns
 7.56  3
17.46  0
18.76  3
```

Revision 1: An Alternate Objective

Symbolic model

```
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];
```

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  1  0  0  0 ;
```

Revision 1 *(cont'd)*

Solutions

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

ampl: objective Number; solve;

Gurobi 6.5.0: optimal solution; objective 20
3 simplex iterations

ampl: display Number, Waste;
Number = 20
Waste = 63.62

ampl: objective Waste; solve;

Gurobi 6.5.0: optimal solution; objective 15.62
2 simplex iterations

ampl: display Number, Waste;
Number = 35
Waste = 15.62
```

Revision 2: Limit on Overruns

Symbolic model

```
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;
```

Revision 2 *(cont'd)*

Explicit data

```
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 6.0.4: optimal solution; objective 20
8 simplex iterations

ampl: display Number, Waste;
Number = 20
Waste = 63.62

ampl: objective Waste; solve;

Gurobi 6.0.4: optimal solution; objective 49.16
2 simplex iterations

ampl: display Number, Waste;
Number = 21
Waste = 49.16
```

Further revisions

Overruns

- Limit to percentage of amount ordered
- Limit total extra rolls

Pattern restrictions

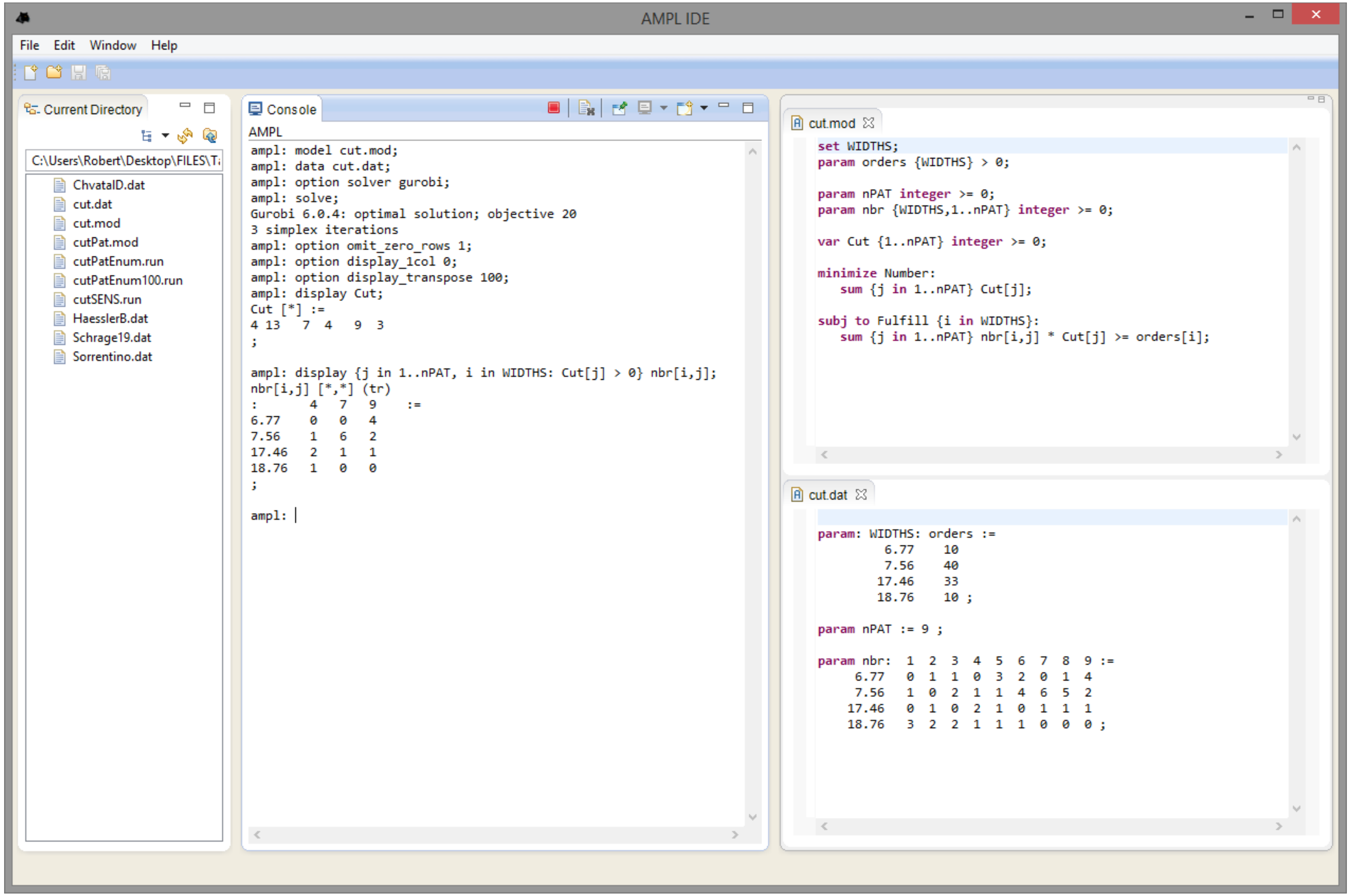
- Cut at least a specified number of each pattern used
- Limit the number of patterns used

Costs

- Account for setups
- Account for complications of cutting

Anything else you can imagine . . .

IDE for Command Language



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

Extended analysis

- Tradeoffs between objectives

Data generation and result processing

- Cutting *via* pattern enumeration

Optimization schemes

- Cutting *via* pattern generation

Tradeoffs Between Objectives

Minimize rolls cut

- Record total rolls cut (low), total waste (high)

Minimize waste

- Set large overrun limit
- Record total rolls cut (high), total waste (low)

Explore tradeoffs

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

Parametric Analysis *(cont'd)*

Script (setup and initial solve)

```
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;
```

Parametric Analysis *(cont'd)*

Script (looping and reporting)

```
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: include cutWASTE.run

Min 20 rolls with waste 63.62

 Over  Waste  Number
  10   46.72    22
   7   47.89    21
   5   54.76    20

ampl:
```

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

```
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];
```

Pattern Enumeration

Data

```
param roll_width := 64.50 ;

param: WIDTHS: orders :=

         6.77    10
         7.56    40
        17.46    33
        18.76    10 ;
```

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)

```
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)

```
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";
   }
```

Pattern Enumeration

Results

```
ampl: include cutPatEnum.run

Gurobi 6.5.0: optimal solution; objective 18
7 simplex iterations

43 patterns, 18 rolls

  Cut     3  1  4 10

  18.76   3  1  0  0
  17.46   0  1  3  2
   7.56   1  3  1  3
   6.77   0  0  0  1
```

Pattern Enumeration

Bigger data

```
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

Far more patterns, still fast results

```
ampl: include cutPatEnum.run

Gurobi 6.5.0: optimal solution; objective 34
291 simplex iterations
60 branch-and-cut nodes

54508 patterns,  34 rolls

  Cut     8  1  1  2  2  3  1  1  1  1  2  1  4  6

  45.75   3  1  1  0  0  0  0  0  0  0  0  0  0  0
  44.75   1  3  2  0  0  0  0  0  0  0  0  0  0  0
  42.25   0  4  5  4  1  0  0  0  0  0  0  0  0  0
  41.75   4  0  0  0  0  3  2  2  1  1  0  0  0  0
  40.75   0  0  0  3  0  3  2  0  4  3  6  4  3  1
  39.75   0  0  0  0  0  0  0  4  0  0  0  2  4  3
  38.75   0  0  0  0  0  0  3  1  1  3  0  2  0  0
  37.75   0  0  0  0  0  0  0  0  1  0  0  0  0  5
  34.75   0  0  0  0  8  0  0  0  0  1  3  0  0  0
  33.75   0  0  0  0  0  3  2  2  2  1  0  0  2  0
  28.75   0  0  0  2  1  0  0  0  0  0  0  1  0  0
```

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 one-constraint “knapsack” optimization 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

```
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)

```
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;
   };
```

Pattern Generation

Script (generation loop)

```
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)

```
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:", 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)

```
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     10   5   4   1

   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     10   5   3   1

   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%
```

Integration and Deployment

Your system

- writes data files
- Invokes AMPL with script name

AMPL's script

- reads the data files
- processes data, generates problems, invokes solvers
- writes result files

Your system

- reads the result files

. . . multi-file implementations with hundreds of statements, millions of statements executed

Limitations

Script statements can be slow

- Interpreted, not compiled
- Must support very general set & data structures

Script programming constructs are limited

- Based on a declarative language
- Not object-oriented

Scripts are stand-alone

- Must close AMPL environment before returning to system

So . . .

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 an active AMPL process

Cutting Revisited

Hybrid approach

- Supervision in a general programming language
  - Data preparation
  - Pattern enumeration or generation
  - Result reporting
- Model & solving in AMPL

Example: Pattern enumeration using MATLAB

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

AMPL Model File

Basic pattern-cutting model

```
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;
```

Pattern Enumeration in MATLAB

Load & generate data, set up AMPL model

```
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');
```

Pattern Enumeration in MATLAB

Send data to AMPL

```
% Send scalar values

ampl.getParameter('overrun').setValues(overrun);
ampl.getParameter('nPatterns').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)
```

Pattern Enumeration in MATLAB

Solve and report

```
% 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))
```

Pattern Enumeration in MATLAB

Enumeration routine

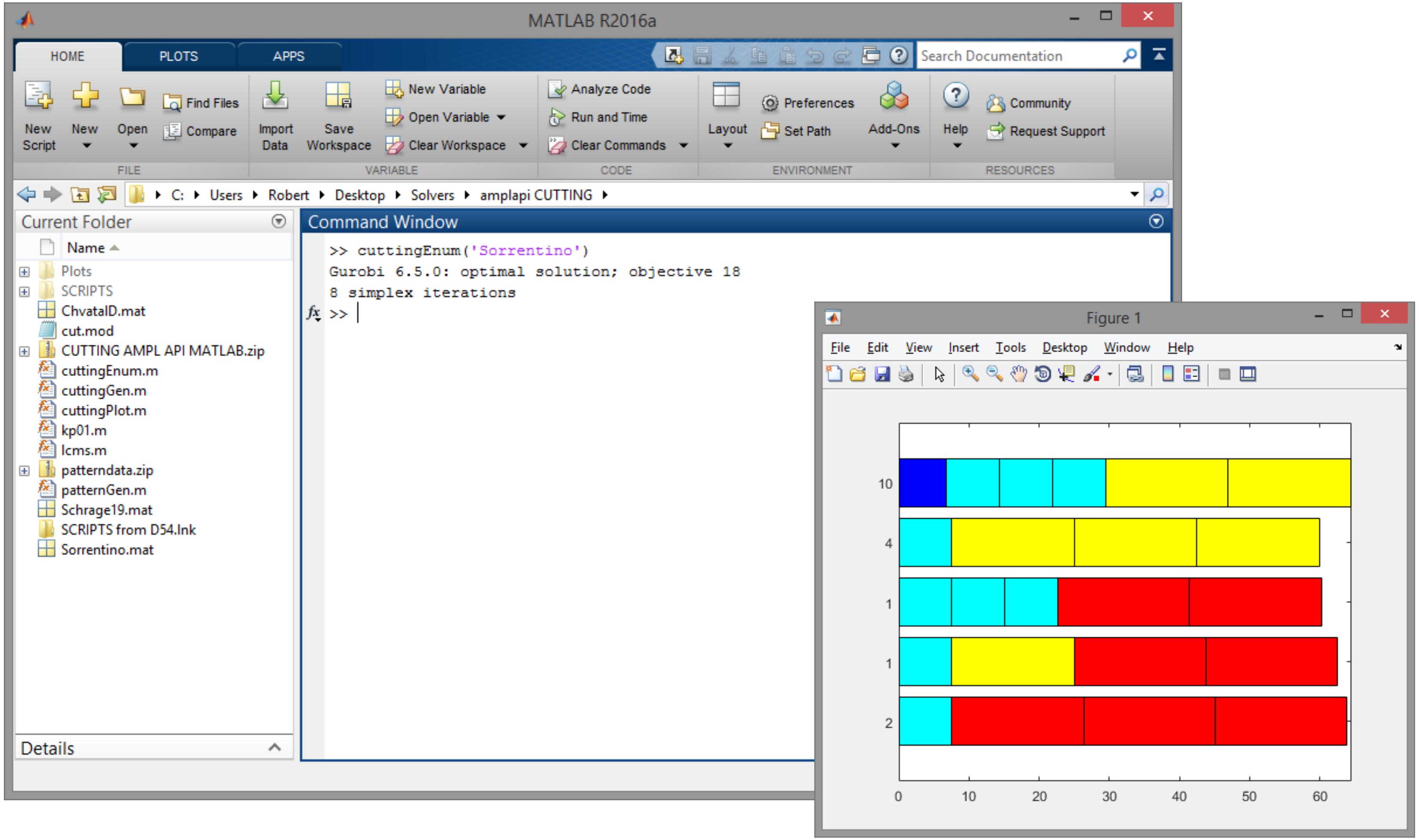
```
function patmat = patternEnum(rollwidth,widths)
if length(widths) == 1
    patmat = floor(rollwidth/widths(1));
else
   patmat = [];
   for n = floor(rollwidth/widths(1)):-1:0
       patnew = patternEnum (rollwidth-n*widths(1), widths(2:end));
       patmat = [patmat; n*ones(size(patnew,1),1) patnew];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

Pattern Enumeration in MATLAB

Plotting routine

```
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



Other Examples

Pattern enumeration in other languages

- Java, C++, MATLAB currently available
- Python, R, .NET (VB & C#) under development

Pattern generation, looping over two solves

- Knapsack solver to find a new pattern
  - freely available routine using dynamic programming approach
- Linear solver to find new solution & dual values
  - Gurobi solver applied to an AMPL model

AMPL Summary

Prototyping and development

- ❖ Interactive modeling interface
  - ∗ Plain command window
  - ∗ Multi-window IDE
- ❖ Scripting facility
  - ∗ Execute AMPL commands
  - ∗ Invoke user-defined AMPL functions
  - ∗ Run external programs

Deployment

- ❖ Interaction with scripts through files
- ❖ Control of AMPL through API in popular programming languages

Trying Gurobi and AMPL for yourself

- *If you already have Gurobi but not AMPL...*
  - Contact sales@gurobi.com and request a free 30-day evaluation license of AMPL
- *If you already have AMPL but not Gurobi...*
  - Commercial: Contact sales@ampl.com and request a free 30-day evaluation license of Gurobi

    - or -
  - Academic: Follow the instructions on: http://www.gurobi.com/academia/for-universities
- *If you are new to both AMPL and Gurobi...*
  - Contact either sales@gurobi.com or sales@ampl.com and you will be helped