Building AMPL Models into Your Applications

Robert Fourer, Victor Zverovich

`[4er,viz]@ampl.com`

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

INFORMS Conference on
Business Analytics and Operations Research

Huntington Beach, CA — 12-14 April 2015
Track 10, Monday 3:40, *Software Tutorial*

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

1

Building AMPL Models into Applications

AMPL is well known for making optimization models easy to develop and to debug, but you don't have to give up these advantages when embedding your model into a larger system and deploying it to users. We present and contrast two facilities for building models into your applications: AMPL scripting, which provides powerful programming constructs within the modeling language; and the new AMPL API, which permits access to AMPL objects and methods from popular general-purpose programming languages.

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

2

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

3

The Optimization Modeling Cycle

Steps

- ❖ Communicate with problem owner
- ❖ Build model
- ❖ Prepare data
- ❖ Generate optimization problem
- ❖ Submit problem to solver
  - ∗ CPLEX, Gurobi, KNITRO, CONOPT, MINOS, . . .
- ❖ Report & analyze results
- ❖ ***Repeat!***

Goals for optimization software

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

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

4

Optimization Modeling Languages

Two forms of an optimization problem

- ❖ Modeler's form
  - ∗ Mathematical 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

Advantages of a modeling language

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

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

5

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

❖ Implemented

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

6

The AMPL Modeling Language

Features

- ❖ Algebraic modeling language
- ❖ Variety of data sources
- ❖ Connections to all solver features

Advantages

- ❖ Powerful, general expressions
- ❖ Natural, easy-to-learn design
- ❖ Efficient processing scales well with problem size

3 ways to use

- ❖ Command language
- ❖ Scripting language
- ❖ Programming interface (API)

Outline

Example: Multicommodity transportation

- ❖ Solution via command language
- ❖ Sensitivity analysis via scripting

Example: Roll cutting

- ❖ Pattern enumeration
    - ∗ via scripting
    - ∗ via MATLAB API
    - ∗ via Java API
- ❖ Pattern generation
    - ∗ via scripting
    - ∗ via MATLAB API

Availability . . .

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

8

Command Language

Multicommodity transportation . . .

- ❖ Products available at factories
- ❖ Products needed at stores
- ❖ Plan shipments at lowest cost

. . . with practical restrictions

- ❖ Cost has fixed and variable parts
- ❖ Shipments cannot be too small
- ❖ Factories cannot serve too many stores

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

9

Multicommodity Transportation

Given

    O Set of origins (factories)

    D Set of destinations (stores)

    P Set of products

and

    a_{ip} Amount available, for each $i \in O$ and $p \in P$

    b_{jp} Amount required, for each $j \in D$ and $p \in P$

    l_{ij} Limit on total shipments, for each $i \in O$ and $j \in D$

    c_{ijp} Shipping cost per unit, for each $i \in O, j \in D, p \in P$

    d_{ij} Fixed cost for shipping any amount from $i \in O$ to $j \in D$

    s Minimum total size of any shipment

    n Maximum number of destinations served by any origin

Robert Fourer, Victor Zverovich,Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

10

Multicommodity Transportation

Mathematical Formulation

Determine

X_{ijp} Amount of each $p \in P$ to be shipped from $i \in O$ to $j \in D$

Y_{ij} 1 if any product is shipped from $i \in O$ to $j \in D$
    0 otherwise

to minimize

$$\sum_{i \in O} \sum_{j \in D} \sum_{p \in P} c_{ijp} X_{ijp} + \sum_{i \in O} \sum_{j \in D} d_{ij} Y_{ij}$$

Total variable cost plus total fixed cost

Mathematical Formulation

Subject to

$\sum_{j \in D} X_{ijp} \leq a_{ip}$ for all $i \in O, p \in P$

Total shipments of product p out of origin i
must not exceed availability

$\sum_{i \in O} X_{ijp} = b_{jp}$ for all $j \in D, p \in P$

Total shipments of product p into destination j
must satisfy requirements

Multicommodity Transportation

Mathematical Formulation

Subject to

$$\sum_{p \in P} X_{ijp} \leq l_{ij} Y_{ij} \quad \text{for all } i \in O, j \in D$$

When there are shipments from origin i to destination j, the total may not exceed the limit, and Y_{ij} must be 1

$$\sum_{p \in P} X_{ijp} \geq s Y_{ij} \quad \text{for all } i \in O, j \in D$$

When there are shipments from origin i to destination j, the total amount of shipments must be at least s

$$\sum_{j \in D} Y_{ij} \leq n \quad \text{for all } i \in O$$

Number of destinations served by origin i
must be as most n

AMPL Formulation

Symbolic data

```
set ORIG;    # origins
set DEST;    # destinations
set PROD;    # products

param supply {ORIG,PROD} >= 0;  # availabilities at origins
param demand {DEST,PROD} >= 0;  # requirements at destinations
param limit {ORIG,DEST} >= 0;   # capacities of links

param vcost {ORIG,DEST,PROD} >= 0; # variable shipment cost
param fcost {ORIG,DEST} > 0;       # fixed usage cost

param minload >= 0;               # minimum shipment size
param maxserve integer > 0;       # maximum destinations served
```

Multicommodity Transportation

AMPL Formulation

Symbolic model: variables and objective

```
var Trans {ORIG,DEST,PROD} >= 0;    # actual units to be shipped

var Use {ORIG, DEST} binary;        # 1 if link used, 0 otherwise


minimize Total_Cost:

   sum {i in ORIG, j in DEST, p in PROD} vcost[i,j,p] * Trans[i,j,p]

 + sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];
```

$$\sum_{i \in O} \sum_{j \in D} \sum_{p \in P} c_{ijp} X_{ijp} + \sum_{i \in O} \sum_{j \in D} d_{ij} Y_{ij}$$

AMPL Formulation

Symbolic model: constraint

```
subject to Supply {i in ORIG, p in PROD}:

    sum {j in DEST} Trans[i,j,p] <= supply[i,p];
```

$$\sum_{j \in D} X_{ijp} \leq a_{ip}, \text{ for all } i \in O, p \in P$$

AMPL Formulation

Symbolic model: constraints

```
subject to Supply {i in ORIG, p in PROD}:

    sum {j in DEST} Trans[i,j,p] <= supply[i,p];


subject to Demand {j in DEST, p in PROD}:

    sum {i in ORIG} Trans[i,j,p] = demand[j,p];


subject to Multi {i in ORIG, j in DEST}:

    sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];


subject to Min_Ship {i in ORIG, j in DEST}:

    sum {p in PROD} Trans[i,j,p] >= minload * Use[i,j];


subject to Max_Serve {i in ORIG}:

    sum {j in DEST} Use[i,j] <= maxserve;
```

AMPL Formulation

Explicit data independent of symbolic model

```
set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;

param supply (tr):   GARY    CLEV    PITT :=
             bands    400     700     800
             coils    800    1600    1800
             plate    200     300     300 ;

param demand (tr):
            FRA    DET    LAN    WIN    STL    FRE    LAF :=
    bands   300    300    100     75    650    225    250
    coils   500    750    400    250    950    850    500
    plate   100    100      0     50    200    100    250 ;

param limit default 625 ;

param minload := 375 ;
param maxserve := 5 ;
```

AMPL Formulation

Explicit data (continued)

```
param vcost :=

 [*,*,bands]:   FRA   DET   LAN   WIN   STL   FRE   LAF :=
        GARY    30    10     8    10    11    71     6
        CLEV    22     7    10     7    21    82    13
        PITT    19    11    12    10    25    83    15

 [*,*,coils]:   FRA   DET   LAN   WIN   STL   FRE   LAF :=
        GARY    39    14    11    14    16    82     8
        CLEV    27     9    12     9    26    95    17
        PITT    24    14    17    13    28    99    20

 [*,*,plate]:   FRA   DET   LAN   WIN   STL   FRE   LAF :=
        GARY    41    15    12    16    17    86     8
        CLEV    29     9    13     9    28    99    18
        PITT    26    14    17    13    31   104    20 ;

param fcost:     FRA   DET   LAN   WIN   STL   FRE   LAF :=
        GARY    3000  1200  1200  1200  2500  3500  2500
        CLEV    2000  1000  1500  1200  2500  3000  2200
        PITT    2000  1200  1500  1500  2500  3500  2200 ;
```

AMPL Solution

Model + data = problem instance to be solved

```
ampl: model multmip3.mod;
ampl: data multmip3.dat;

ampl: option solver gurobi;

ampl: solve;

Gurobi 5.6.0: optimal solution; objective 235625
293 simplex iterations
28 branch-and-cut nodes

ampl: display Use;

Use [*,*]

:     DET FRA FRE LAF LAN STL WIN   :=
CLEV   1   1   1   0   1   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   1   0   1   0
;
```

AMPL Solution

Solver choice independent of model and data

```
ampl: model multmip3.mod;
ampl: data multmip3.dat;

ampl: option solver cplex;

ampl: solve;

CPLEX 12.6.0.0: optimal integer solution; objective 235625
136 MIP simplex iterations
0 branch-and-bound nodes

ampl: display Use;

Use [*,*]

:      DET FRA FRE LAF LAN STL WIN   :=
CLEV    1   1   1   0   1   1   0
GARY    0   0   0   1   0   1   1
PITT    1   1   1   1   0   1   0
;
```

AMPL Solution

Examine results

```
ampl: display {i in ORIG, j in DEST}
ampl?    sum {p in PROD} Trans[i,j,p] / limit[i,j];

:       DET    FRA    FRE    LAF    LAN    STL    WIN      :=
CLEV    1      0.6    0.88   0      0.8    0.88   0
GARY    0      0      0      0.64   0      1      0.6
PITT    0.84   0.84   1      0.96   0      1      0
;

ampl: display Max_Serve.body;

CLEV  5
GARY  3
PITT  5
;

ampl: display TotalCost,
ampl?    sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];

TotalCost = 235625
sum {i in ORIG, j in DEST} fcost[i,j]*Use[i,j] = 27600
```

Multicommodity Transportation
AMPL IDE

Scripting

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

Scripting

Parametric Analyses

Try different limits on destinations served

❖ Reduce parameter `maxserve` and re-solve
  * until there is no feasible solution

❖ Display results
  * parameter value
  * numbers of destinations actually served

Try different supplies of plate at Gary

❖ Increase parameter `supply['GARY','plate']` and re-solve
  * until dual is zero (constraint is slack)

❖ Record results
  * distinct dual values
  * corresponding objective values

. . . display results at the end

Parametric Analysis *on limits*

Script

```
model multmipG.mod;
data multmipG.dat;

option solver gurobi;

for {m in 7..1 by -1} {
    let maxserve := m;
    solve;
    if solve_result = 'infeasible' then break;
    display maxserve, Max_Serve.body;
}
```

```
subject to Max_Serve {i in ORIG}:
    sum {j in DEST} Use[i,j] <= maxserve;
```

Parametric Analysis *on limits*

Run

```
ampl: include multmipServ.run;

Gurobi 5.6.0: optimal solution; objective 233150
maxserve = 7
CLEV 5    GARY 3    PITT 6

Gurobi 5.6.0: optimal solution; objective 233150
maxserve = 6
CLEV 5    GARY 3    PITT 6

Gurobi 5.6.0: optimal solution; objective 235625
maxserve = 5
CLEV 5    GARY 3    PITT 5

Gurobi 5.6.0: infeasible
```

Parametric Analysis *on supplies*

Script

```
set SUPPLY default {};
param sup_obj {SUPPLY};
param sup_dual {SUPPLY};

let supply['GARY','plate'] := 200;
param supply_step = 10;
param previous_dual default -Infinity;

repeat while previous_dual < 0 {

  solve;

  if Supply['GARY','plate'].dual > previous_dual then {
    let SUPPLY := SUPPLY union {supply['GARY','plate']};
    let sup_obj[supply['GARY','plate']] := Total_Cost;
    let sup_dual[supply['GARY','plate']] := Supply['GARY','plate'].dual;
    let previous_dual := Supply['GARY','plate'].dual;
    }

  let supply['GARY','plate'] := supply['GARY','plate'] + supply_step;
  }
```

Parametric Analysis *on supplies*

Run

```
ampl: include multmipSupply.run;

ampl: display sup_obj, sup_dual;

:     sup_obj    sup_dual     :=
200    223504     -13
380    221171     -11.52
460    220260     -10.52
510    219754      -8.52
560    219413       0
;
```

Cutting *via* Pattern Enumeration

Roll cutting

❖ Meet orders for small widths by cutting large rolls

∗ using a variety of cutting patterns

❖ Decision variables: numbers of each pattern to cut

❖ Objective: minimize large rolls used (or material wasted)

❖ Constraints: meet demands for each ordered width

Enumerate cutting patterns

❖ Read general model

❖ Read data: demands, raw width

❖ Compute data: all usable patterns

❖ Solve problem instance

Robert Fourer, Victor Zverovich, Building AMPL Models Into Applications
INFORMS Analytics Huntington Beach — 12-14 Apr 2015 — Track 10 Software Tutorials

31

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;

model;
param curr_sum >= 0;
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;
```

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";
   }

printf "\nWASTE = %5.2f%%\n\n",
   100 * (1 - (sum {i in WIDTHS} i * orders[i]) / (roll_width * Number));
```

Pattern Enumeration

Results

```
ampl: include cutPatEnum.run

Gurobi 5.6.0: optimal solution; objective 18
7 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  0  1

WASTE =   2.34%
```

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

```
ampl: include cutPatEnum.run

Gurobi 4.6.1: optimal solution; objective 34
291 simplex iterations

54508 patterns,  34 rolls

  Cut      8  1  1  1  3  1  1  1  1  2  7  2  3  1  1

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

WASTE =   0.69%
```

Pattern Enumeration

Data 3

```
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 ;
```

Pattern Enumeration

Results 3 (using a subset of patterns)

```
ampl: include cutPatEnum.run

Gurobi 4.6.1: optimal solution; objective 33
722 simplex iterations
40 branch-and-cut nodes

273380 patterns,  33 rolls

  Cut       1  1  1  1  4  4  4  1  1  2  5  2  1  1  1  3

  25.00     2  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
  24.75     1  2  1  0  5  4  3  2  2  2  2  1  1  0  0  0
  18.00     0  0  0  0  1  0  0  1  0  0  0  1  1  5  1  0
  17.50     0  3  0  0  0  0  0  0  0  0  0  0  0  0  1  0

  .......
  10.12     0  2  0  0  0  1  2  0  0  0  0  0  0  0  0  0
  10.00     0  0  0  0  0  2  0  1  3  0  6  0  0  2  0  0
   8.75     0  0  1  0  0  0  0  0  0  2  0  2  0  0  0  2
   8.50     0  0  2  0  0  2  0  0  0  0  0  4  3  0  0  0
   7.75     0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  0

WASTE =   0.62%
```

Cutting *via* Pattern Generation

Same roll cutting application

Generate cutting patterns

- ❖ 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

```
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%
```

General Observations

Scripts in practice

- ❖ Large and complicated
  - ✱ Multiple files
  - ✱ Hundreds of statements
  - ✱ Millions of statements executed
- ❖ Run within broader applications

Prospective improvements

- ❖ Faster loops
- ❖ True script functions
  - ✱ Arguments and return values
  - ✱ Local sets & parameters
  - ✱ Callback functions

But . . .

Limitations

Performance

- ❖ Interpreted language
- ❖ Complex set & data structures

Expressiveness

- ❖ Based on a declarative language
- ❖ Not object-oriented

So . . .

AMPL API

Application Programming Interface

- ❖ General-purpose languages: C++, Java, .NET, Python
- ❖ Analytics languages: MATLAB, R

Facilitates use of AMPL for

- ❖ Complex algorithmic schemes
- ❖ Embedding in other applications
- ❖ Deployment of models

Development details

- ❖ Partnership with OptiRisk Systems
  - ✳ Christian Valente, principal developer
- ❖ Long-term development & maintenance by AMPL
  - ✳ Victor Zverovich, project coordinator

Cutting Revisited

Hybrid approach

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

Two programming languages

- ❖ Java
- ❖ MATLAB

Key to examples

- ❖ AMPL entities
- ❖ Java/MATLAB objects
- ❖ Java/MATLAB methods for working with AMPL

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

```matlab
% 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))
```

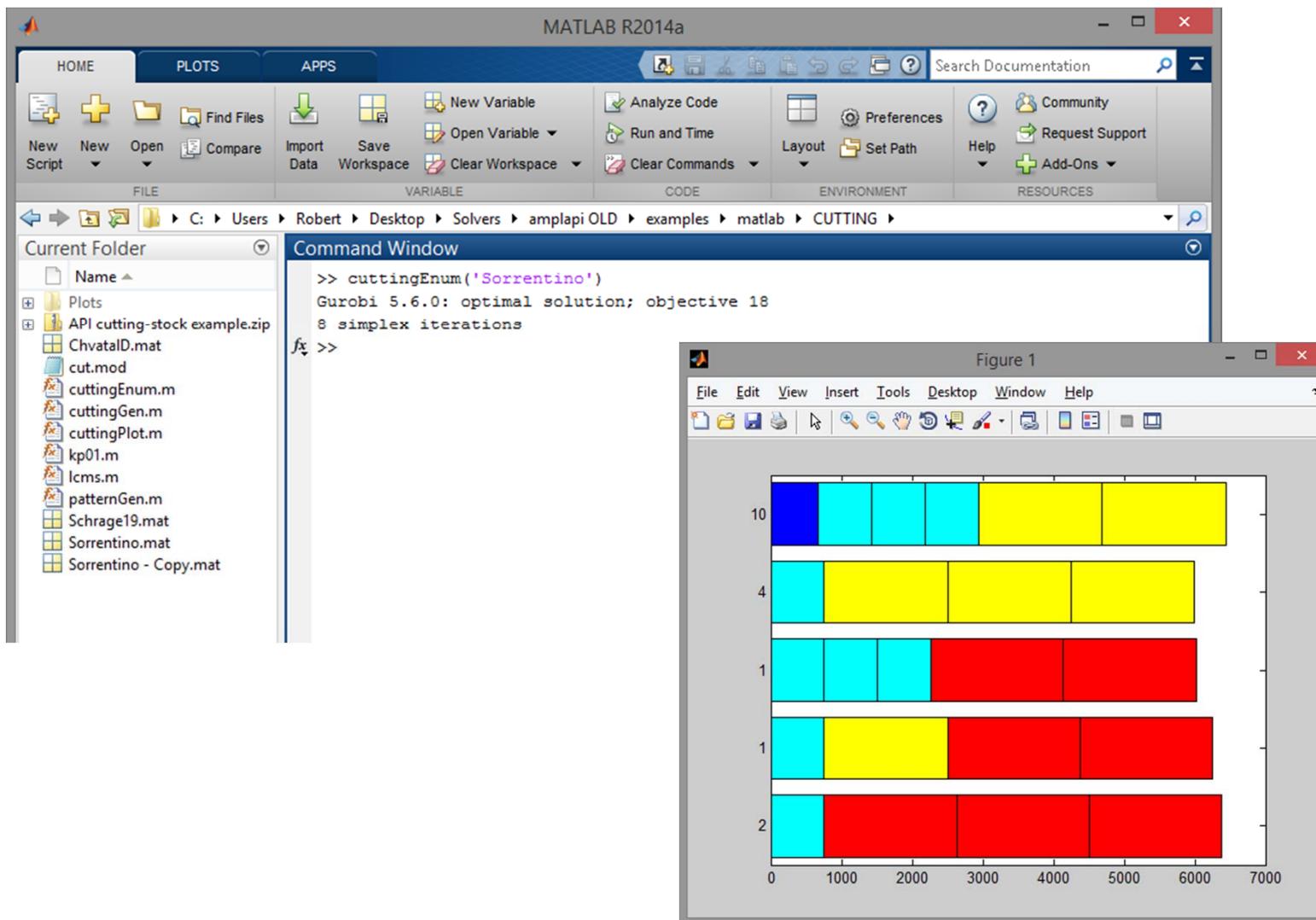# Pattern Enumeration in MATLAB

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 = patternGen (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))
```

AMPL API
Pattern Enumeration in MATLAB

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

```
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);
```

Pattern Enumeration in Java

Solve and report solution

```
    ampl.setOption("solver", "gurobi");

    ampl.solve();

    printSolution (ampl.getVariable("Cut"), ampl.getParameter("rolls"));
  } finally {

    ampl.close();

  }
}
```

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);
```

Pattern Generation in MATLAB

Send data to AMPL

```
% Send scalar values

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

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
```

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))
```

Pattern Generation in MATLAB
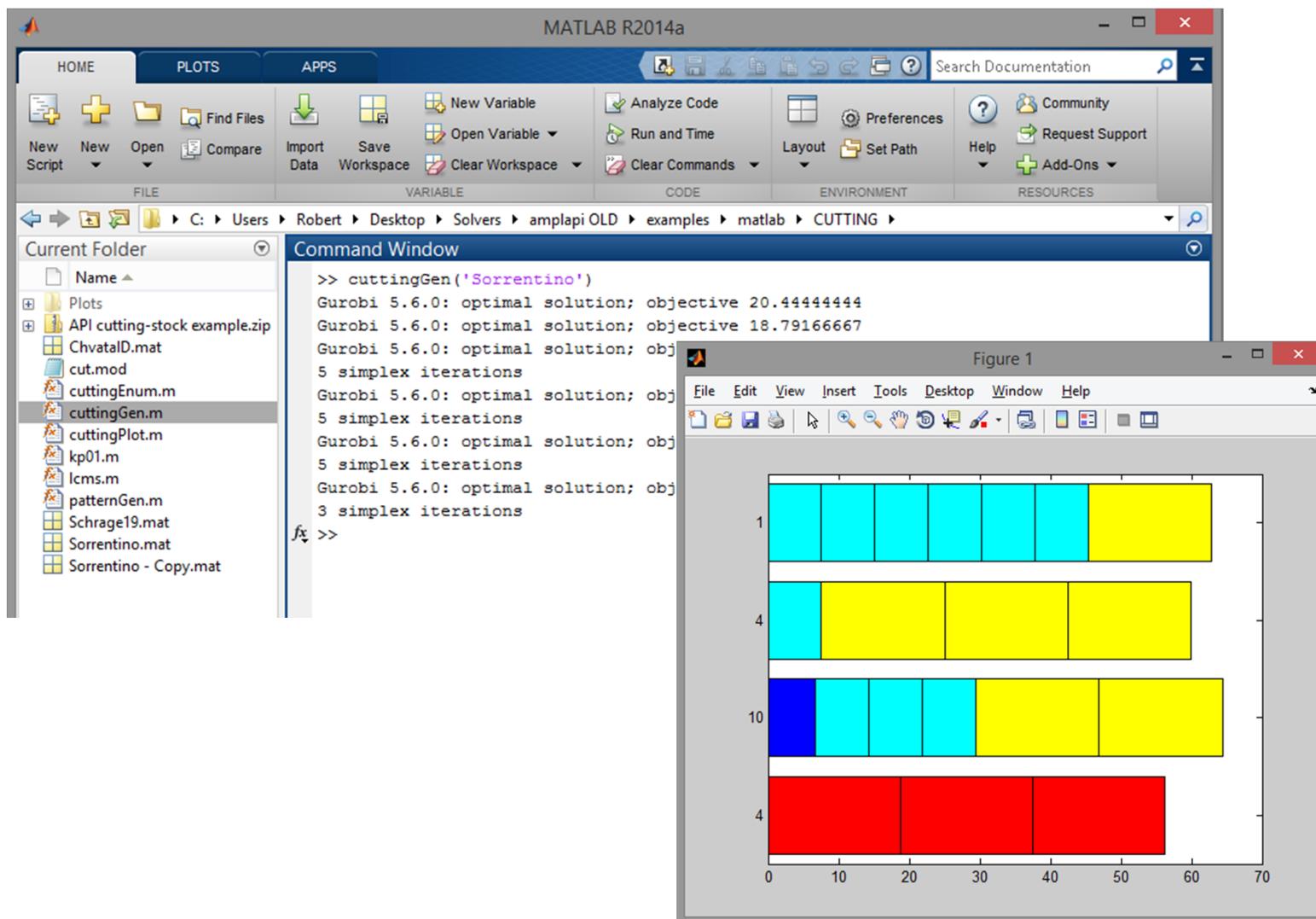
Data Transfer: Alternatives

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

Deployment: **Alternatives**

Scripting: Give (temporary) control to AMPL

- ❖ Write needed files
- ❖ Invoke AMPL to run some scripts
- ❖ Read the files that AMPL leaves on exit

API: Interact with AMPL

- ❖ Execute AMPL statements individually
- ❖ Read model, data, script files when convenient
- ❖ Exchange data tables directly with AMPL
  - ∗ populate sets & parameters
  - ∗ invoke any available solver
  - ∗ extract values of variables & result expressions

. . . all embedded within your program's logic

Availability

Java API version 1.0 released
MATLAB API version 1.0 released

❖ Add-ons to all AMPL distributions
❖ Download from www.ampl.com/products/api/

C++ API in final development

❖ Release planned for this spring

More languages to follow

❖ R
❖ Python
❖ .NET: C#, Visual Basic

www.ampl.com

AMPL Readings

❖ R. Fourer, "Modeling Languages versus Matrix Generators for Linear Programming." *ACM Transactions on Mathematical Software* **9** (1983) 143–183.

❖ R. Fourer, D.M. Gay, B.W. Kernighan, "A Modeling Language for Mathematical Programming." *Management Science* **36** (1990) 519–554.

❖ Robert Fourer, "Database Structures for Mathematical Programming Models." *Decision Support Systems* **20** (1997) 317–344.

❖ R. Fourer, D.M. Gay, B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Belmont, CA (first edition 1993, second edition 2003).

❖ Robert Fourer, On the Evolution of Optimization Modeling Systems. M. Groetschel (ed.), *Optimization Stories*. Documenta Mathematica (2012) 377-388.