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

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 $i \in W$

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 cut of width i
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];

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

$\sum_{j=1}^{n} a_{ij} X_j \geq b_i$, 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 Environment

Model + data = problem instance to be solved

```
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
```

Revision 1

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 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

```
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 := 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  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)

```
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 62.04

 Over  Waste   Number
  25   40.57     24
  19   43.01     23
  13   45.45     22
   7   47.89     21
   5   54.76     20

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

```
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;

param dsetname symbolic;
printf "\nEnter dataset name:\n";
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;
```

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;
      }
   }
```

Scripting

Pattern Enumeration

Script (solve, report)

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

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

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

  Cut      3  1  4  9  1

  18.76    3  1  0  0  0
  17.46    0  2  3  2  1
   7.56    1  1  1  3  5
   6.77    0  0  0  1  1
```

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 ;
```

Scripting

Pattern Enumeration

Results 2

```
ampl: include cutPatEnum.run

AT LEAST 34 ROLLS REQUIRED

Gurobi 8.1.0: optimal solution; objective 34
158 simplex iterations
33 branch-and-cut nodes

54508 patterns, 34 rolls

  Cut      7  2  1  2  3  1  1  1  5  1  2  1  1  3  2  1
  45.75    3  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
  44.75    1  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  42.25    0  4  0  0  2  2  2  1  0  0  0  0  0  0  0  0
  41.75    4  0  2  0  1  1  0  0  2  1  1  1  0  0  0  0
  40.75    0  0  1  2  0  0  1  0  2  5  4  3  6  4  3  2
  39.75    0  0  0  1  2  0  3  0  1  0  1  2  0  3  4  2
  38.75    0  0  0  2  0  2  0  0  0  0  0  1  0  0  0  3
  37.75    0  0  2  2  2  2  2  0  2  0  0  1  0  1  0  1
  34.75    0  0  2  0  0  2  0  3  1  2  3  0  3  0  0  0
  33.75    0  0  1  0  2  0  0  6  1  1  0  0  0  0  2  1
  28.75    0  0  0  1  0  0  1  0  0  0  0  1  0  1  0  0
```

Scripting

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 ;
```

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

  Cut     1  1  1  2  2  2  1  6  4  1  3  5  1  2  1

  25.00   1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
  24.75   3  2  0  0  5  4  3  3  2  2  2  2  1  1  0
  18.00   0  0  0  0  1  0  1  0  2  0  0  0  1  1  0
  17.50   0  0  0  0  0  0  0  0  0  4  0  0  0  0  0
  .......
  10.12   0  0  0  0  0  1  0  2  0  0  0  0  0  0  0
  10.00   0  1  0  0  0  2  0  0  0  0  3  6  0  0  0
   8.75   0  4  0  0  0  0  1  0  1  0  0  0  0  2  2
   8.50   1  0  1  0  0  2  4  0  0  0  0  0  3  4  0
   7.75   2  0  0  0  1  0  1  0  0  0  0  0  0  0  0
```

Cutting *via* Pattern Generation

Generate 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

Scripting

Pattern Generation

Cutting model

```
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

```
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 (define models)

```
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;
```

Pattern Generation

Script (data, initial patterns)

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

Scripting

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];
};
```

Scripting

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
Enter dataset name:
ampl? Sorrentino

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

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%)
```

Scripting

Pattern Generation

Results (integer)

```
Rounded up to integer: 20 rolls

  Cut     10  5  4  1

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

WASTE = 28.42 (2.20%)

Best integer: 19 rolls

  Cut     10  4  4  1

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

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

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

```
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

```
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")
```

Pattern Enumeration in Python

Send data to AMPL

```
# 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))
}
```

Pattern Enumeration in R

Send data to AMPL

```
# 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

```
# Solve

ampl.option['solver'] = 'gurobi'
ampl.solve()

# Retrieve solution

CuttingPlan = ampl.var['Cut'].getValues()
cutvec = list(CuttingPlan.getColumn('Cut.val'))
```

Pattern Enumeration in R

Solve and get results

```
# Solve

ampl$setOption("solver", "gurobi")
ampl$solve()

# Retrieve solution

CuttingPlan <- ampl$getVariable("Cut")$getValues()
solution <- CuttingPlan[CuttingPlan[,-1] != 0,]
```

Pattern Enumeration in Python

Display solution

```
# 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

```
  # 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

```
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
```

Pattern Enumeration in R

Enumeration routine

```
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']
```

Pattern Enumeration in R

Plotting routine

```
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)

```
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()
```

Pattern Enumeration in R

Plotting routine (cont'd)

```
    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)
  }
```

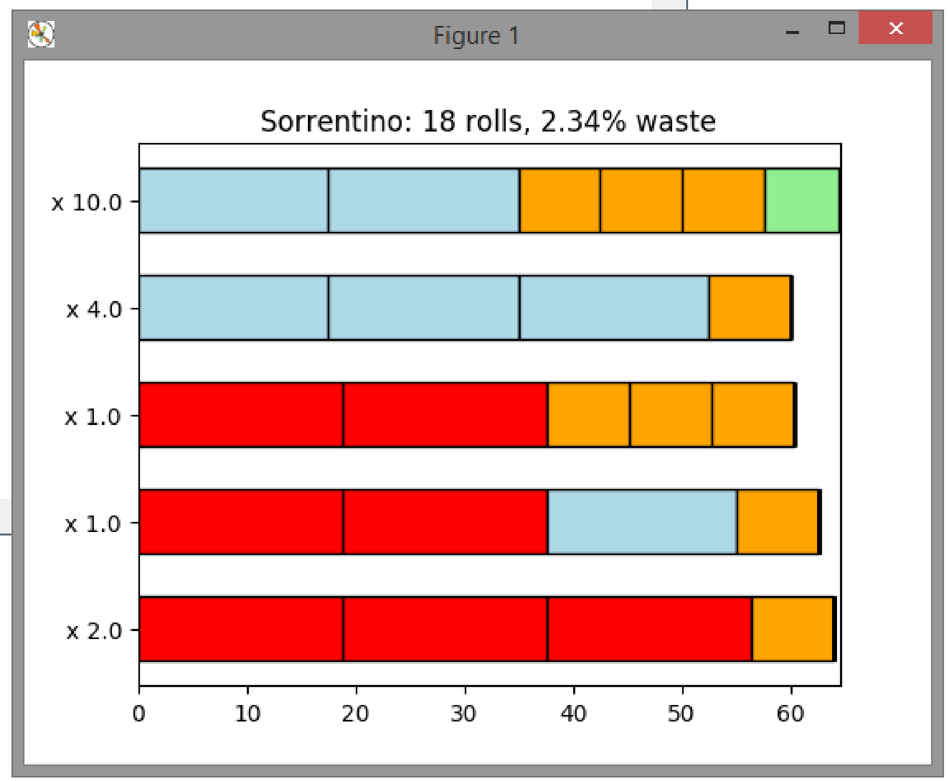
AMPL API

Pattern Enumeration in Python

Robert: running ipython

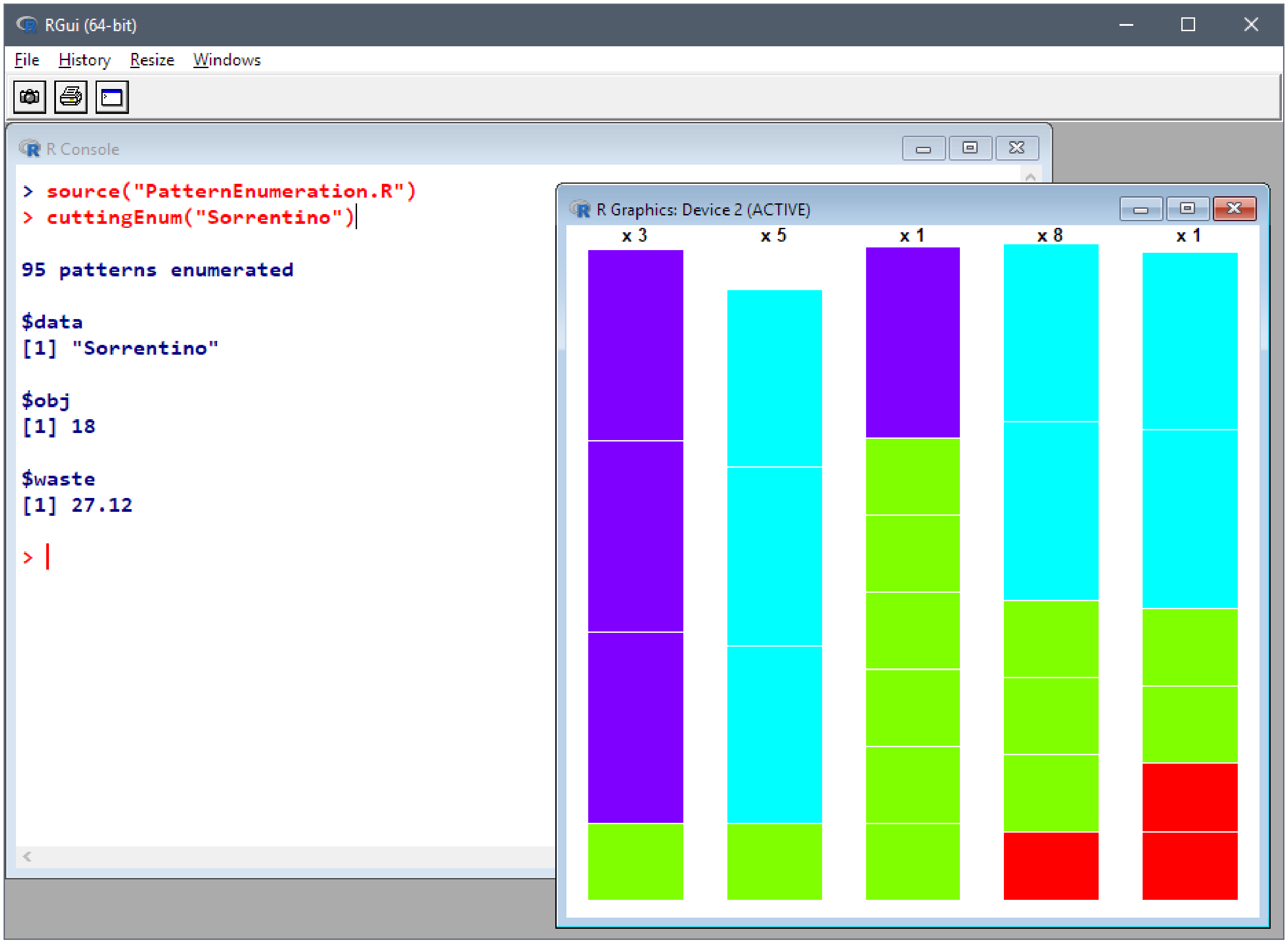
File Edit Help

```
sw: ipython
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from pattern_enumeration import *

In [2]: cuttingEnum('Sorrentino')

43 patterns enumerated

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



AMPL API

Pattern Enumeration in R



APIs in practice . . .

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

Sending Python Data to an AMPL model

Imported and generated data in 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)))
```

Sending Data using the Python API

Symbolic sets and parameters in AMPL

cut.mod

```
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

```
ampl = AMPL()

.......

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))
}
```

Sending Data using PyMPL

Specify Python data correspondences in the model

cutpy.mod

```
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))
    }
};
```

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

Callbacks

Stopping rule data in Python dictionary

stopping.py

```
stopdict = { 'time'   : (  15,    30,   60 ),
             'gaptol' : ( .0002, .002, .02 )
           }
```

Main routine for cutting by pattern generation

pattern_generation.py

```
def cuttingGen(cutdata, stopdata = ""):

  from amplpy import AMPL

  .......

  # begin pattern generation phase
  # finish when continuous relaxation of cutting problem has been solved
```

Callbacks

Set up callback and solve final integer program

```
# 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

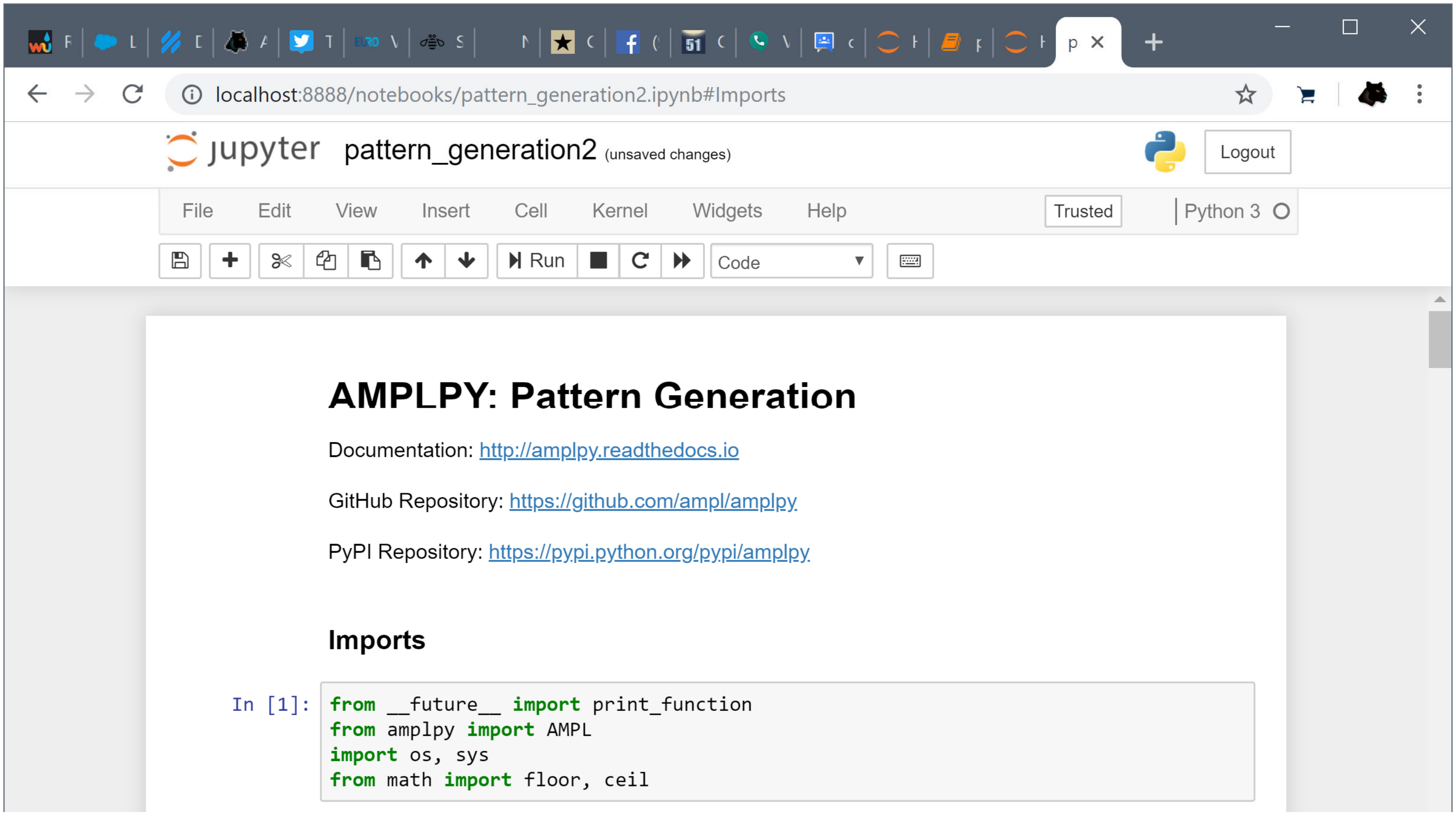
Callback function

```
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



Executing Python inside AMPL

Fix AMPL variables according to Python variable

lotsize.mod

```
$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
```

Executing Python inside AMPL

Invoke Python generators for special lot-sizing constraints

lotsize.mod

```
$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')
};
```

```
ampl = AMPL(langext=PyMPL())
ampl.read('lotsize.mod')
ampl.solve()
```

Executing Python inside AMPL

Optional listing of generated constraints

```
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 :=
[1,1]400 [1,2]800 [1,3]1600 [1,4]2400 [1,5]3600 [1,6]4800 [1,7]6000 [1,8]7200
[2,1]0   [2,2]400 [2,3]1200 [2,4]2000 [2,5]3200 [2,6]4400 [2,7]5600 [2,8]6800
[3,1]0   [3,2]0   [3,3]800  [3,4]1600 [3,5]2800 [3,6]4000 [3,7]5200 [3,8]6400
[4,1]0   [4,2]0   [4,3]0    [4,4]800  [4,5]2000 [4,6]3200 [4,7]4400 [4,8]5600
[5,1]0   [5,2]0   [5,3]0    [5,4]0    [5,5]1200 [5,6]2400 [5,7]3600 [5,8]4800
[6,1]0   [6,2]0   [6,3]0    [6,4]0    [6,5]0    [6,6]1200 [6,7]2400 [6,8]3600
[7,1]0   [7,2]0   [7,3]0    [7,4]0    [7,5]0    [7,6]0    [7,7]1200 [7,8]2400
[8,1]0   [8,2]0   [8,3]0    [8,4]0    [8,5]0    [8,6]0    [8,7]0    [8,8]1200
;

model;
```

Executing Python inside AMPL

Optional listing of generated constraints (cont'd)

```
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[wt,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

Architecture

Server side

- ❖ AMPL model and data
- ❖ Standard AMPL-solver installations

Client side

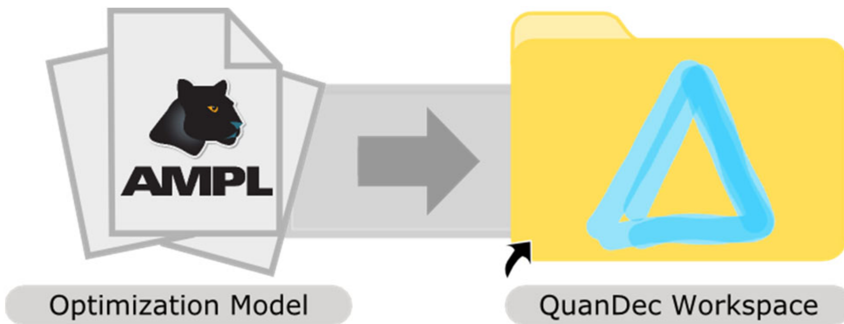
- ❖ Interactive tool for collaboration & decision-making
- ❖ Runs on any recent web browser
- ❖ Java-based implementation
  - ∗ AMPL API for Java
  - ∗ Eclipse Remote Application Platform

Getting Started

step 1: install QuanDec on a server

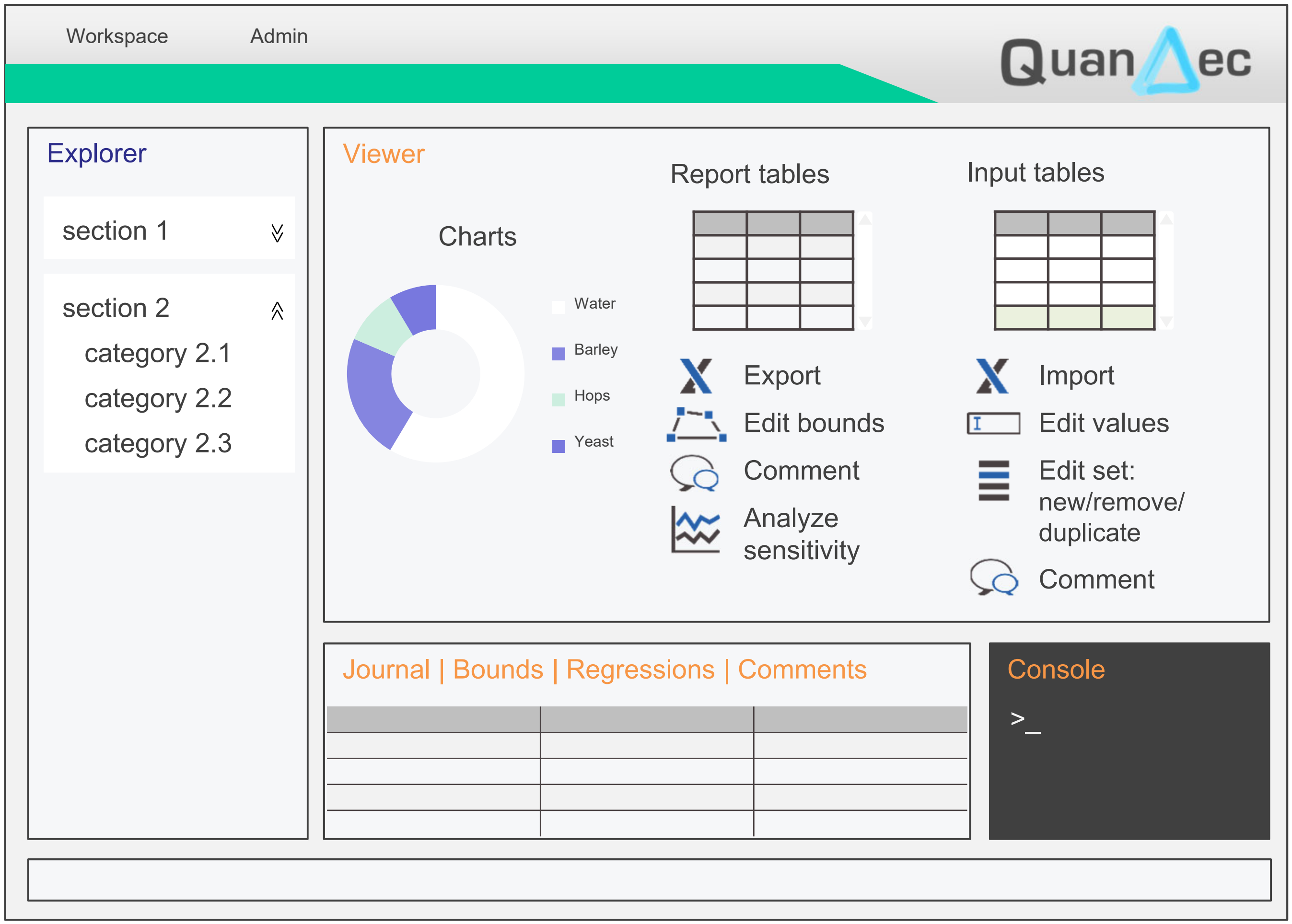
step 2: copy & paste your model files (.mod and .dat) into QuanDec's workspace

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



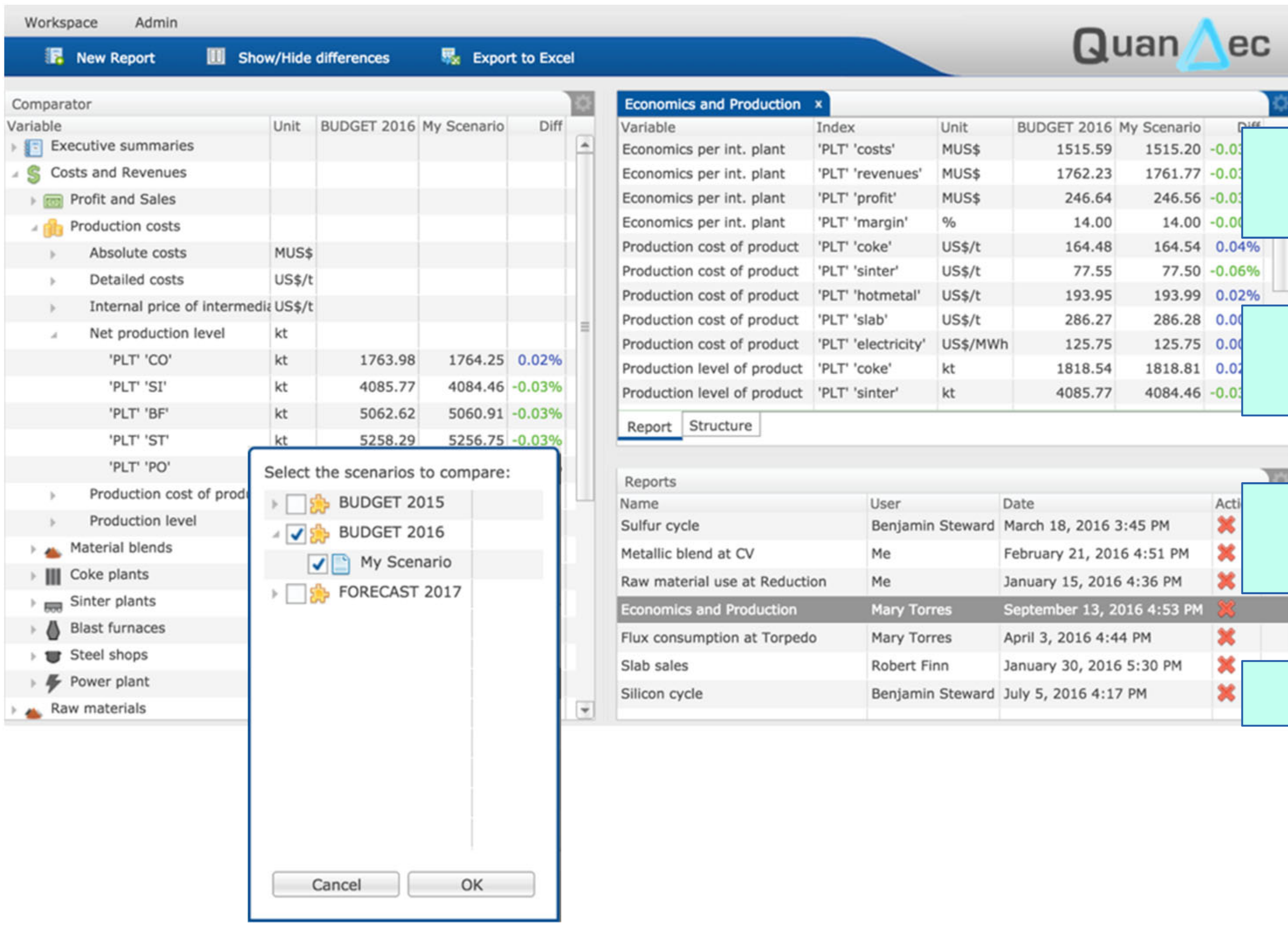
QuanDec

Workbench



QuanDec

Scenarios



Scenario comparison

All variables can be compared

Display of relative difference

Custom reports