Developing Optimization Applications Quickly and Effectively with Algebraic Modeling in AMPL

Robert Fourer

4er@ampl.com

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

INFORMS Annual Meeting

Houston— 22-25 October 2017
Session MB69, *Technology Tutorials*

Approaches to Optimization

Application-based

- Use a software package designed for your problems

Method-based

- Implement an optimization algorithm for your problems

Model-based

- Develop a general description of your problems
- Send problem instances to an off-the-shelf solver
- *Compared to application-based:*
  better tailored to your needs
- *Compared to method-based:*
  much easier to develop and maintain

The Optimization Modeling Cycle

Steps

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

Goals for optimization modeling 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
  - 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

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

Categorizations of Algebraic Modeling Languages

By language design

- Extended from a general programming language
- Built specially for optimization

By solver support

- Specialized for one particular solver
- Designed to support many solvers

AMPL

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

4 ways to use . . .

4 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

Deployment tool (QuanDec)

- Embed models into an interactive decision-making tool

Example

Roll cutting model

- Solution via command language
- Tradeoff analysis via scripting

Roll cutting by pattern enumeration

- via scripting
- via API

Roll cutting by pattern generation

- via scripting
- via API

. . . featuring new AMPL API for Python

In practice . . .

A general tool for applying optimization

- Based on a broadly applicable paradigm
- Readily accommodates unanticipated requirements

Ideally positioned for new projects

- More control
  - compared to application-specific software
- Faster, more flexible prototyping
  - compared to development in a programming language

Scalable for integration and deployment

Roll Cutting Problem

Motivation

- Fill orders for rolls of various widths
  - by cutting raw rolls of one (large) fixed width
  - using a variety of cutting patterns

Optimization model

- Decision variables
  - number of raw rolls to cut according to each pattern
- Objective
  - minimize number of raw rolls used
- Constraints
  - meet demands for each ordered width

Roll cutting

Mathematical Formulation

Given

W set of ordered widths

n number of patterns considered

and

a_{ij} occurrences of width i in pattern j, for each $i \in W$ and $j = 1,\ldots,n$

b_i orders for width i, for each $i \in W$

Roll cutting

Mathematical Formulation *(cont'd)*

Determine

X_j number of rolls to cut using pattern j, for each $j = 1, \ldots, n$

to minimize

$\sum_{j=1}^{n} X_j$

total number of rolls cut

subject to

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

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

Roll Cutting

AMPL Formulation

Symbolic model

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

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

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

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

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

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

Roll Cutting

AMPL Formulation *(cont'd)*

Explicit data (independent of model)

```
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 ;
```

In practice . . .

Model: decision variables, objective, constraints

- Applicable for many problem types
  - Planning, scheduling, routing, packing, assignment
  - Network flow, portfolio selection, feedstock blending
- Successful in many business areas
  - Production, logistics, sequencing, assignment, design
  - Energy, manufacture, process, finance, commerce

Model + data = Optimization problem for solver

- Model defined & documented independently of data
- Varied data sources supported
  - Text files, spreadsheets, databases, API calls

Command Language

Model + data = problem instance to be solved

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

ampl: option solver cplex;

ampl: solve;

CPLEX 12.7.1.0: 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 7.5.0: 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        # pattern
2 61.75   4 61.24   6 62.54   8 62.0                   # total widths

ampl: display Fulfill.slack;

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

Roll Cutting

Revision 1

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 7.5.0: optimal solution; objective 20
3 simplex iterations

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

ampl: objective Waste; solve;

Gurobi 7.5.0: optimal solution; objective 15.62
2 simplex iterations

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 := 6 ;

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

param nPAT := 9 ;

param nbr:  1  2  3  4  5  6  7  8  9 :=
     6.77   0  1  1  0  3  2  0  1  4
     7.56   1  0  2  1  1  4  6  5  2
    17.46   0  1  0  2  1  0  1  1  1
    18.76   3  2  2  1  1  1  0  0  0 ;
```

Revision 2 *(cont'd)*

Solutions

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

ampl: objective Number; solve;

Gurobi 7.5.0: optimal solution; objective 20
8 simplex iterations
1 branch-and-cut nodes

ampl: display Number, Waste;
Number = 20
Waste = 54.76

ampl: objective Waste; solve;

Gurobi 7.5.0: optimal solution; objective 49.16
4 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

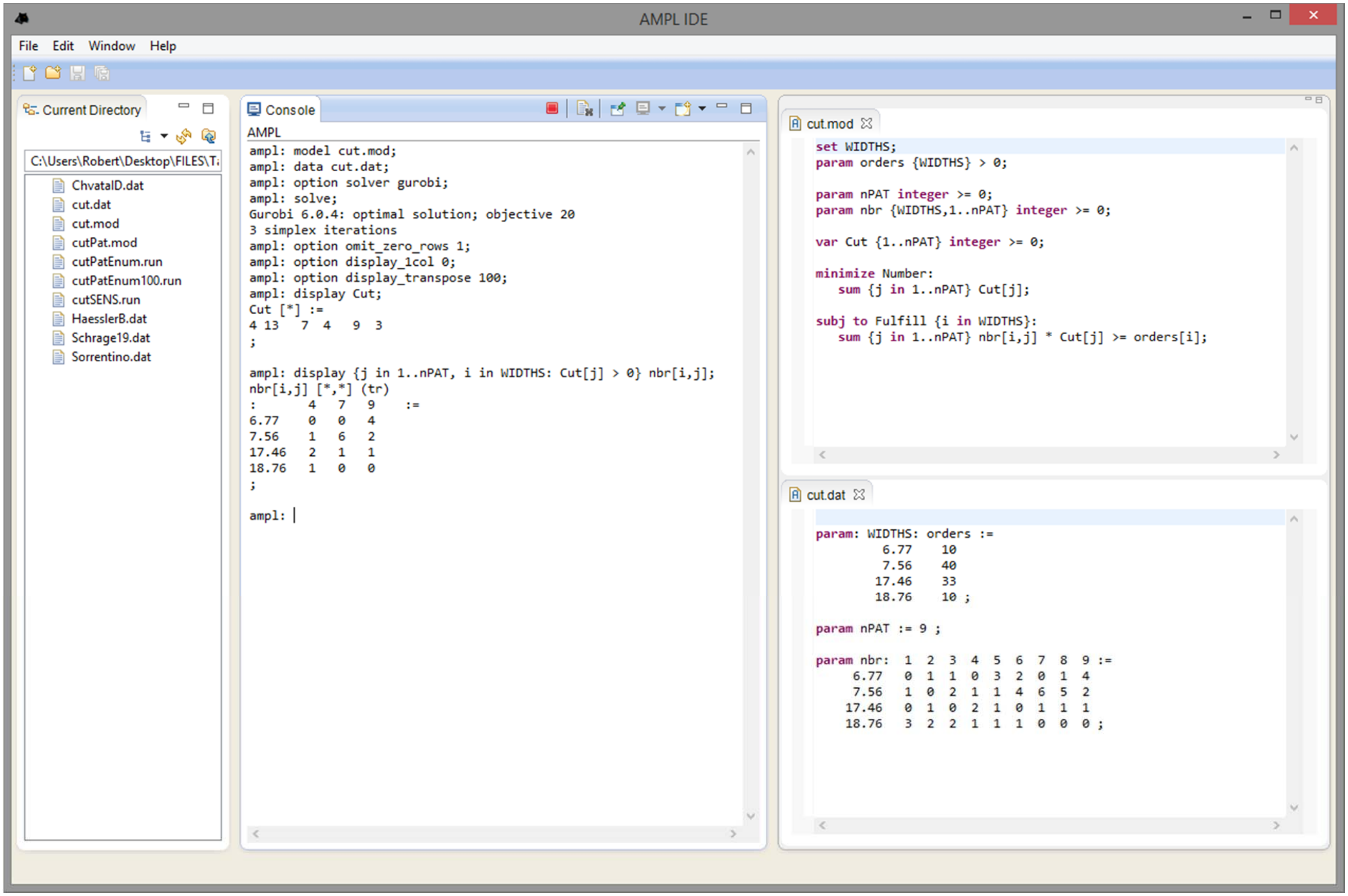
- Cut at least a specified number of each pattern used
- Limit the number of patterns used

Costs

- Account for setups
- Account for complications of cutting

Anything else you can imagine . . .

IDE for Command Language



In practice . . .

Work interactively

- Make changes
- Solve
- Browse results
- *Review and repeat*

Choose the best solver for your problem

- Linear/quadratic mixed-integer
  - CPLEX, Gurobi, Xpress
- Nonlinear continuous
  - CONOPT, Ipopt, LGO, LOQO, MINOS, SNOPT
- Nonlinear mixed-integer
  - BARON, Bonmin, Couenne, Knitro

Scripting

Bring the programmer to the modeling language

Extend modeling language syntax . . .

- Algebraic expressions
- Set indexing expressions
- Interactive commands

. . . with programming concepts

- Loops of various kinds
- If-then and If-then-else conditionals
- Assignments

Examples

- Tradeoffs between objectives
- Cutting *via* pattern enumeration
- Cutting *via* pattern generation

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

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";
   }

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 7.5.0: optimal solution; objective 18
9 simplex iterations
1 branch-and-cut node

43 patterns, 18 rolls

  Cut      3  1  3 11

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

WASTE =  2.34%
```

Scripting

Pattern Enumeration

Data 2

```
param roll_width := 349 ;

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

Pattern Enumeration

Results 2

```
ampl: include cutPatEnum.run

Gurobi 7.5.0: optimal solution; objective 34
130 simplex iterations

54508 patterns,  34 rolls

  Cut     2  5  3  3  1  1  6  2  1  7  1  2

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

WASTE =  0.69%
```

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 a subset of patterns)

```
ampl: include cutPatEnum.run

Gurobi 7.5.0: optimal solution; objective 33
362 simplex iterations
1 branch-and-cut nodes

273380 patterns,  33 rolls

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

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

WASTE =  0.62%
```

Cutting *via* Pattern Generation

Generate the pattern list by a series of solves

- Solve LP relaxation using subset of patterns
- Add "most promising" pattern to the subset
  - Minimize reduced cost given dual values
  - Equivalent to a knapsack problem
- Iterate as long as there are promising patterns
  - Stop when minimum reduced cost is zero
- Solve IP using all patterns found

Pattern Generation

Cutting model

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

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

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

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

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

Pattern Generation

Knapsack model

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

var Use {WIDTHS} integer >= 0;

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

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

Scripting

Pattern Generation

Script (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%
```

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

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 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

Cutting Revisited

Hybrid approach

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

Key to Python program examples

- ❖ AMPL entities
- ❖ AMPL API Python objects
- ❖ AMPL API Python methods
- ❖ Python functions etc.

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 rawWidth;                    # width of raw rolls to be cut
param rolls {WIDTHS,PATTERNS} >= 0, default 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;
```

Some Python Data

A float, an integer, and a dictionary

```
roll_width = 64.5

overrun = 6

orders = {
     6.77: 10,
     7.56: 40,
    17.46: 33,
    18.76: 10
}
```

. . . can also work with lists and Pandas dataframes

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 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 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 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)
```

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 Python

Plotting routine

```
def cuttingPlot(roll_width, widths, summary, 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 Python

Plotting routine (cont'd)

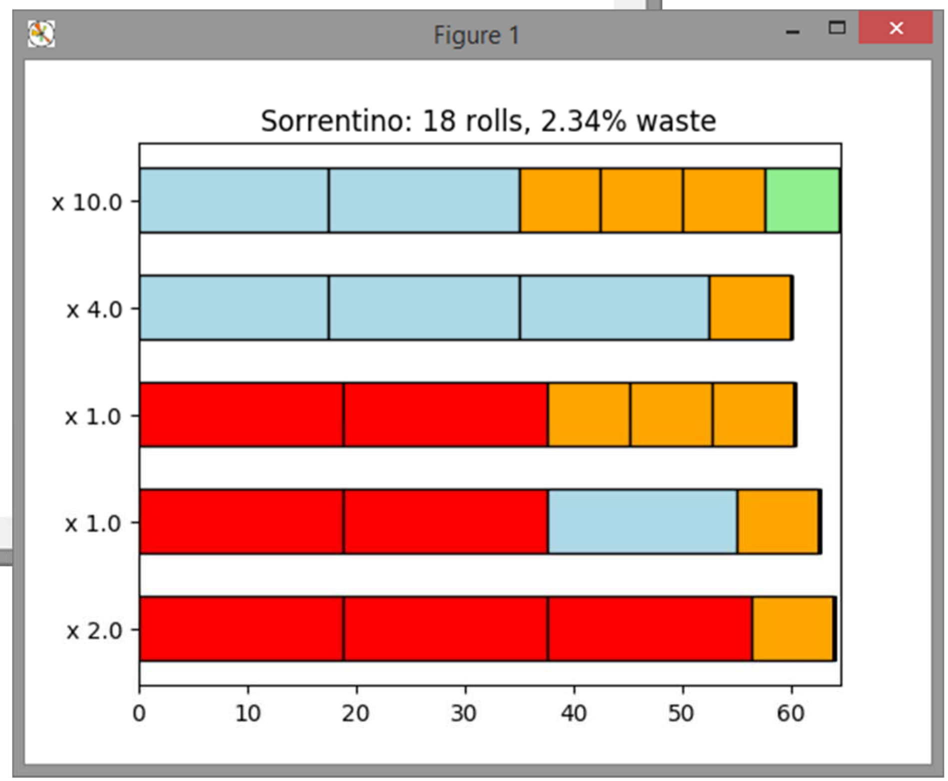
```
for p, (patt, rep) in enumerate(solution):
   for i in range(len(widths)):
      for j in range(patt[i]):
         vec = [0]*len(solution)
         vec[p] = widths[i]
         plt.barh(ind, vec, 0.6, acc,
                  color=colorlist[i%len(colorlist)], edgecolor='black')
         acc[p] += widths[i]

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 Python

2017: running ipython

File Edit Help

```
sw: ipython
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from pattern_enumeration import *

In [2]: cuttingEnum('Sorrentino')

43 patterns generated

Gurobi 7.5.0: optimal solution; objective 18
9 simplex iterations
1 branch-and-cut nodes
```



In practice . . .

Integrate within a larger scheme

Retain benefits of algebraic modeling

- work with natural representation of optimization models
- efficient prototyping, reliable maintenance

Use the best tools for each part of the project

- program data manipulation in your choice of language
- work with optimization models in AMPL

Pattern Generation in Python

Get data, set up master problem

```
function cuttingGen(dataset)
  from amplpy import AMPL

  # Read orders, roll_width, overrun; extract widths

  exec(open(dataset+'.py').read(), globals())
  widths = list(sorted(orders.keys(), reverse=True))

  # Set up cutting (master problem) model

  Master = AMPL()
  Master.option['ampl_include'] = 'models'
  Master.read('cut.mod')

  # Define a param for sending new patterns

  Master.eval('param newPat {WIDTHS} integer >= 0;')

  # Set solve options

  Master.option['solver'] = 'gurobi'
  Master.option['relax_integrality'] =  1
```

AMPL API

Pattern Generation in Python

Send data to master problem

```
# Send scalar values

Master.param['nPatterns'] = len(widths)
Master.param['overrun'] = overrun
Master.param['rawWidth'] = roll_width

# Send order vector

Master.set['WIDTHS'] = widths
Master.param['order'] = orders

# Generate and send initial pattern matrix

Master.param['rolls'] = {
   (widths[i], 1+i): int(floor(roll_width/widths[i]))
      for i in range(len(widths))
  }
```

Pattern Generation in Python

Set up subproblem

```
# Define knapsack subproblem

Sub = AMPL()
Sub.option['solver'] = 'gurobi'

Sub.eval('''
   set SIZES;
   param cap >= 0;
   param val {SIZES};
   var Qty {SIZES} integer >= 0;
   maximize TotVal: sum {s in SIZES} val[s] * Qty[s];
   subject to Cap: sum {s in SIZES} s * Qty[s] <= cap;
''')


# Send subproblem data

Sub.set['SIZES'] = widths
Sub.param['cap'] = roll_width
```

Pattern Generation in Python

Generate patterns and re-solve cutting problems

```
# Alternate between master and sub solves

while True:
    Master.solve()

    Sub.param['val'].setValues(Master.con['OrderLimits'].getValues())
    Sub.solve()
    if Sub.obj['TotVal'].value() <= 1.00001:
       break

    Master.param['newPat'].setValues(Sub.var['Qty'].getValues())
    Master.eval('let nPatterns := nPatterns + 1;')
    Master.eval('let {w in WIDTHS} rolls[w, nPatterns] := newPat[w];')

# Compute integer solution

Master.option['relax_integrality'] = 0
Master.solve()
```

Pattern Generation in Python

Display solution

```
# Prepare summary data

summary = {

      'Data': dataset,
      'Obj': int(Master.obj['TotalRawRolls'].value()),
      'Waste': Master.getValue(
                  'sum {p in PATTERNS} Cut[p] * \
                     (rawWidth - sum {w in WIDTHS} w*rolls[w,p])'
                  )
}

# Retrieve patterns and solution

npatterns = int(Master.param['nPatterns'].value())
rolls = Master.param['rolls'].getValues().toDict()
cutvec = Master.var['Cut'].getValues().toDict()
```

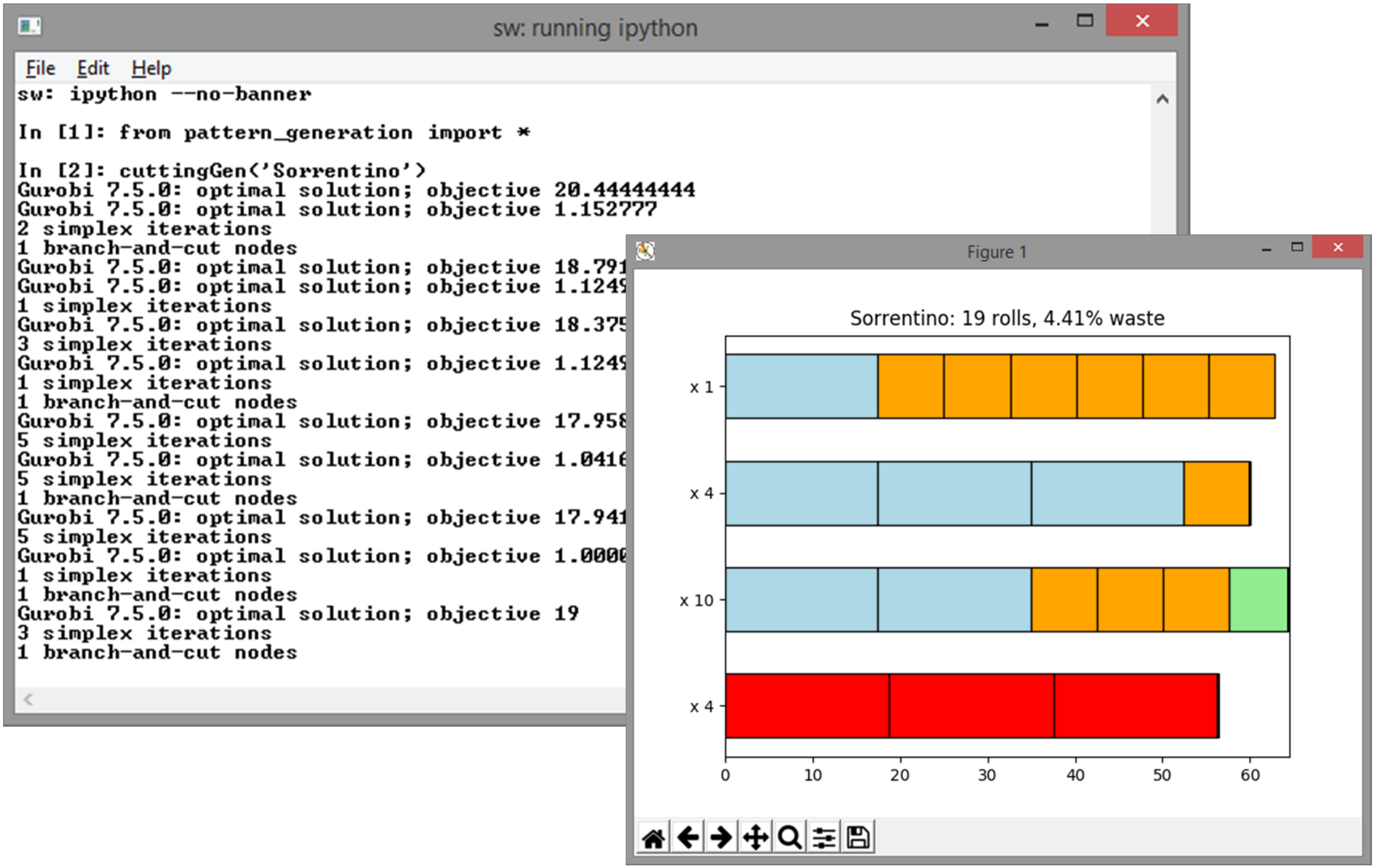
Pattern Generation in Python

Display solution

```
# Prepare solution data

solution = [
   ([int(rolls[widths[i], p+1][0])
      for i in range(len(widths))], int(cutvec[p+1][0]))
   for p in range(npatterns)
   if cutvec[p+1][0] > 0
]

# Create plot of solution

cuttingPlot(roll_width, widths, summary, solution)
```

AMPL API

Pattern Generation in Python



In practice . . .

Implement hybrid iterative schemes

- build powerful software for hard problems

Alternate between optimization & other analytics

- invoke specialized optimizers for subproblems

QuanDec

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

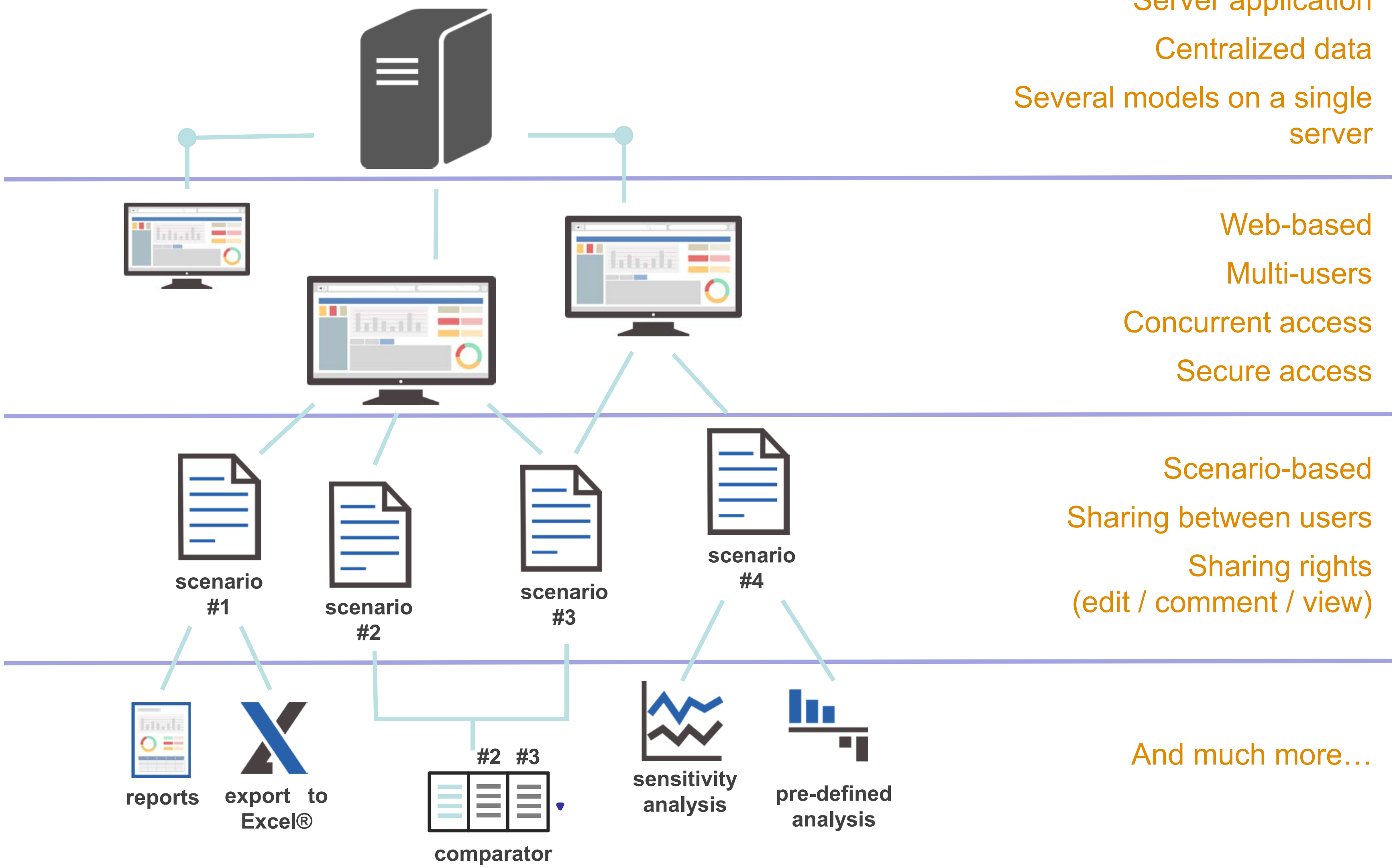
. . . developed / supported by Cassotis Consulting

The web-based graphical interface
that turns optimization models written
in AMPL into decision-making tools.

CASSOTIS consulting

AMPL

Features



Server application

Centralized data

Several models on a single server

Web-based

Multi-users

Concurrent access

Secure access

Scenario-based

Sharing between users

Sharing rights
(edit / comment / view)

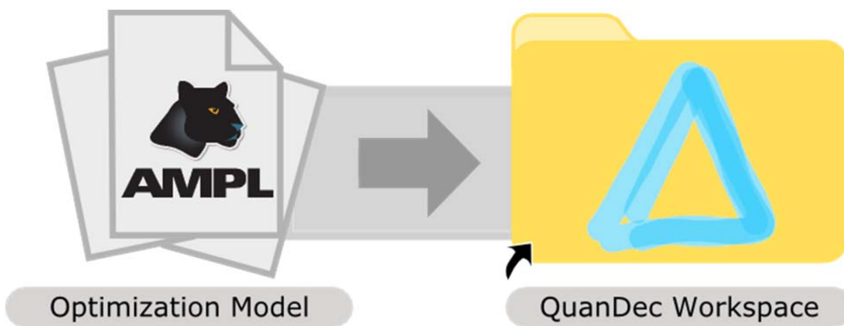
And much more…

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

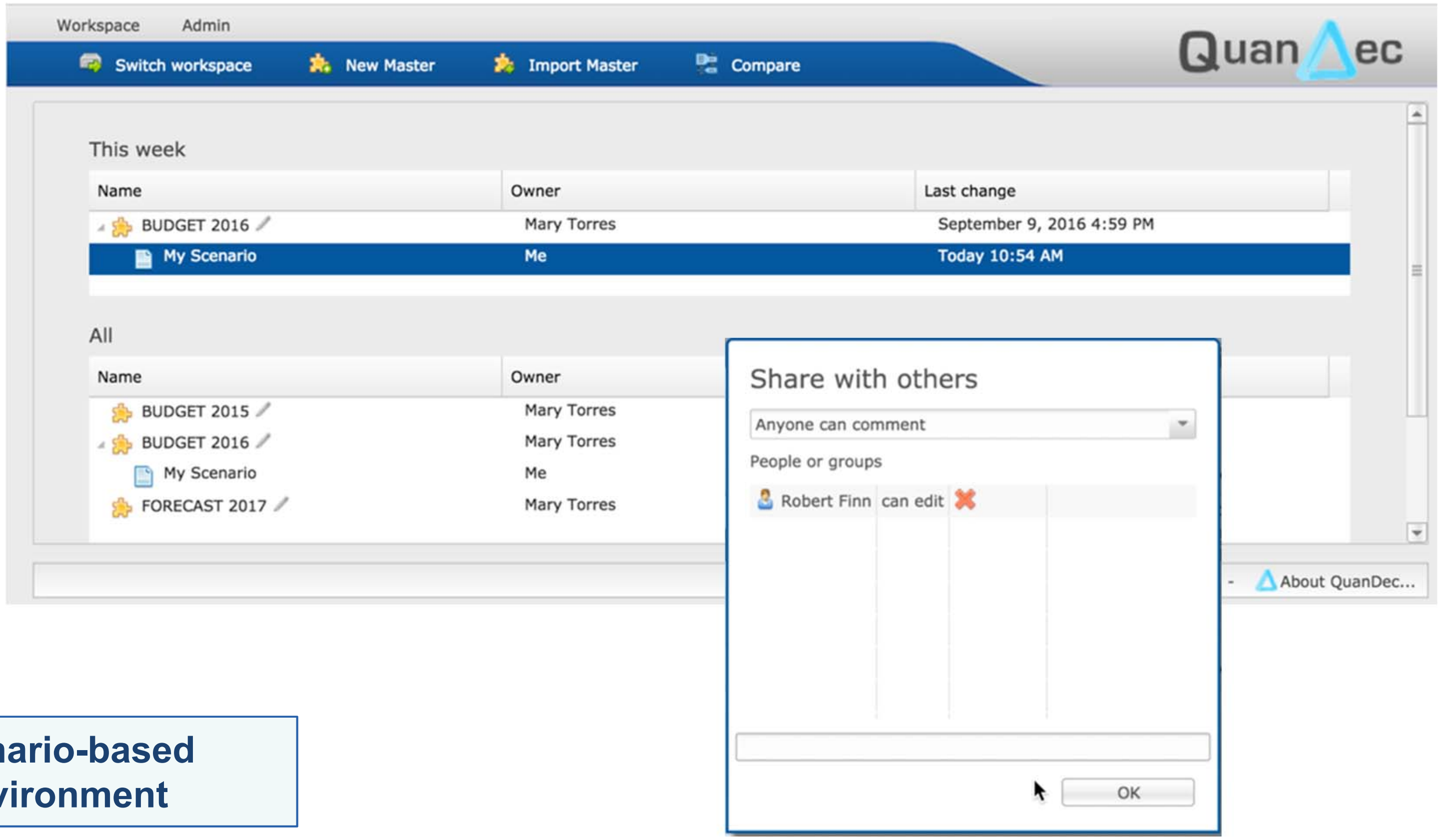


Web-application

Multi-user

Secure access

Concurrent access



Scenario-based environment

Sharing system

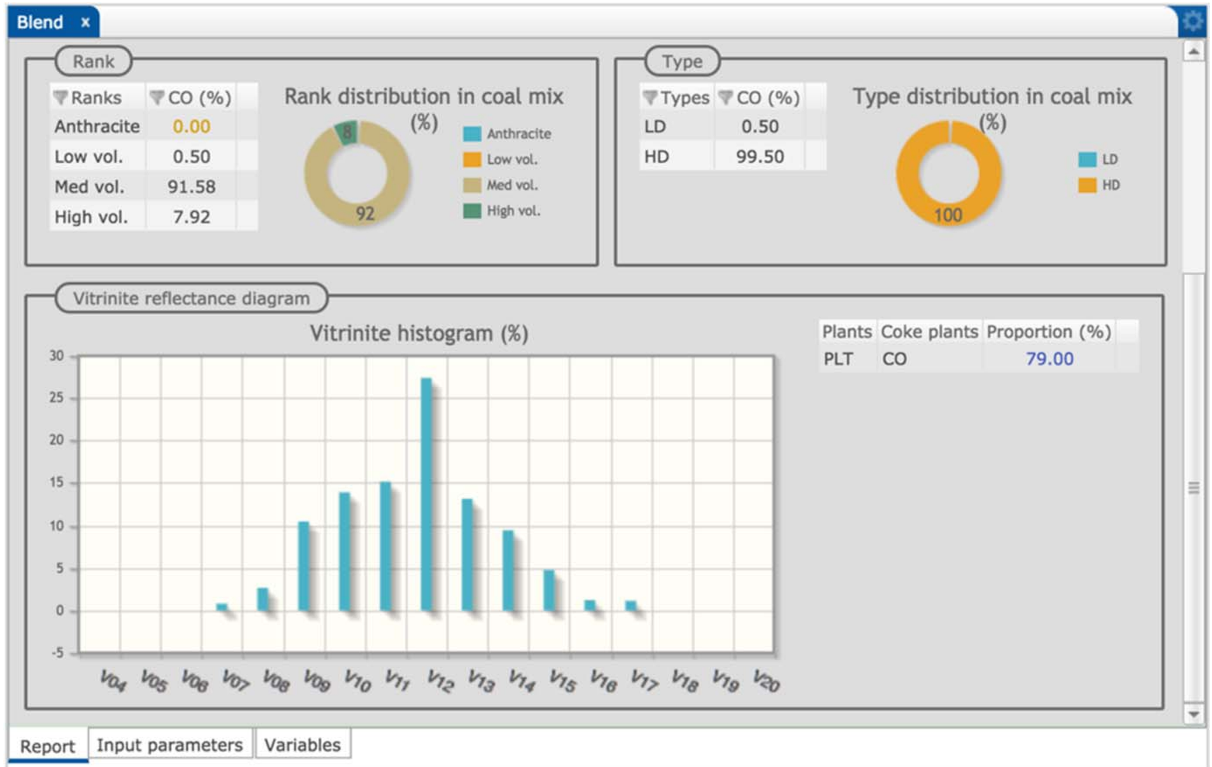
Permission:
Edit – Comment - View

3 levels:
- Report
- Input parameters
- Variables

Chart and tables

Colored values for easier analysis

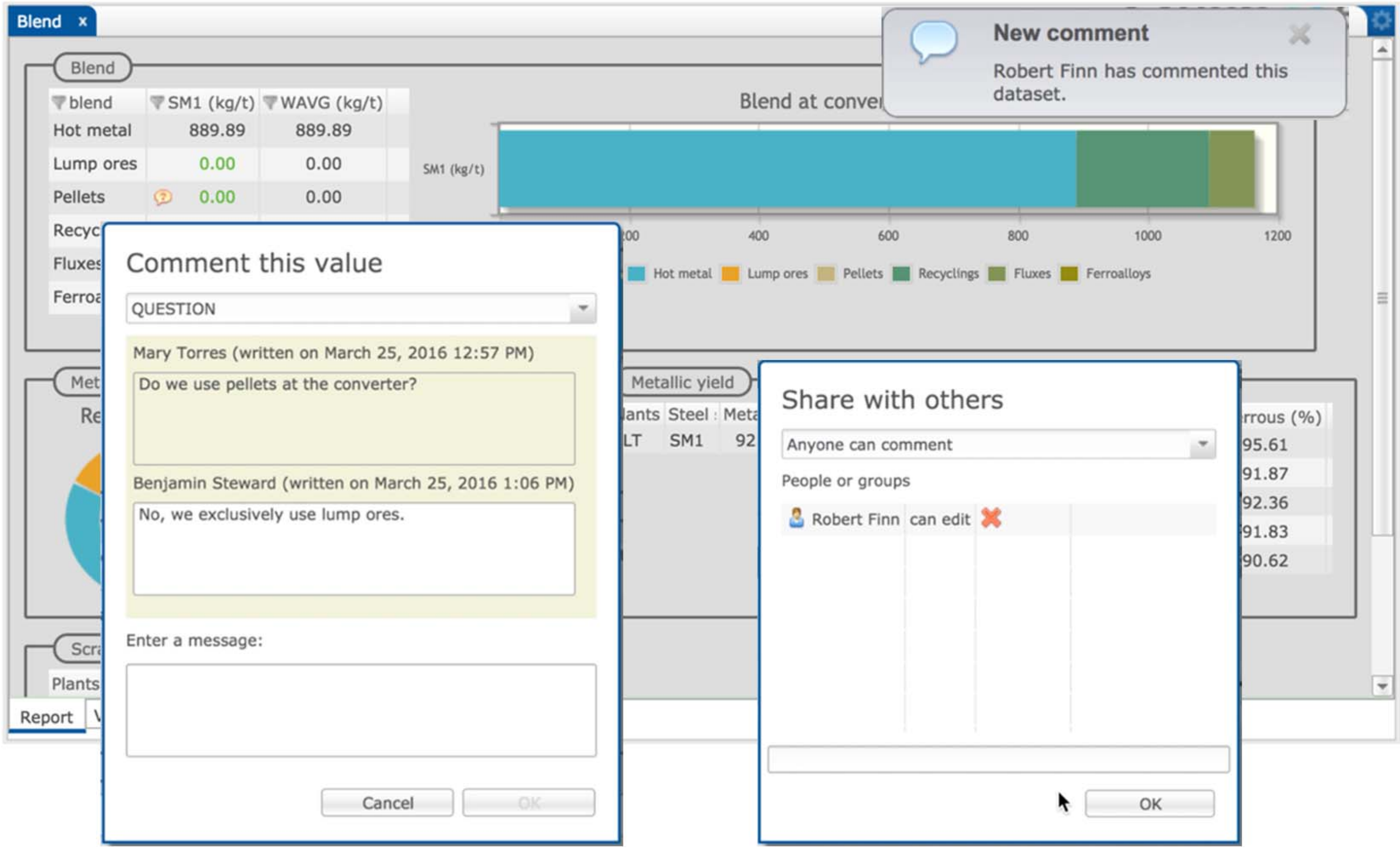
Constraint (min/max) on any variable

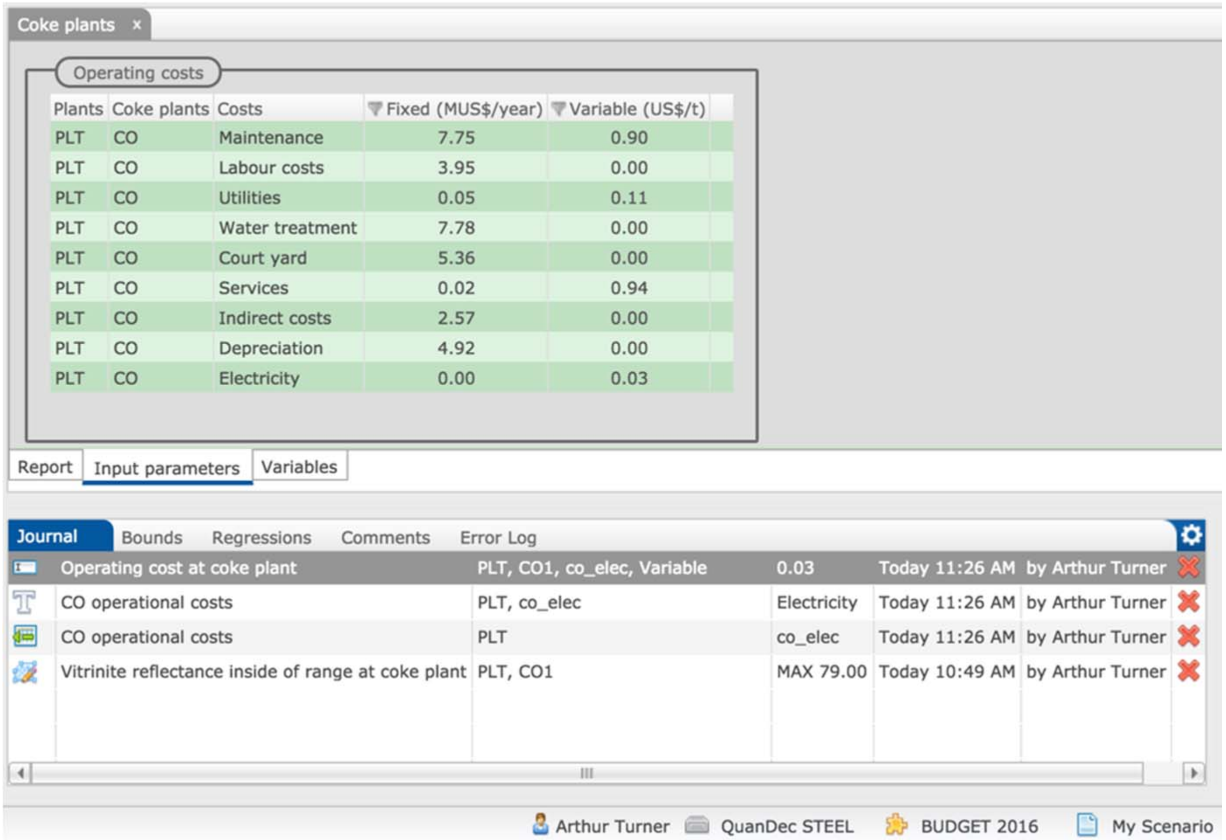


Collaborative work

Notification system

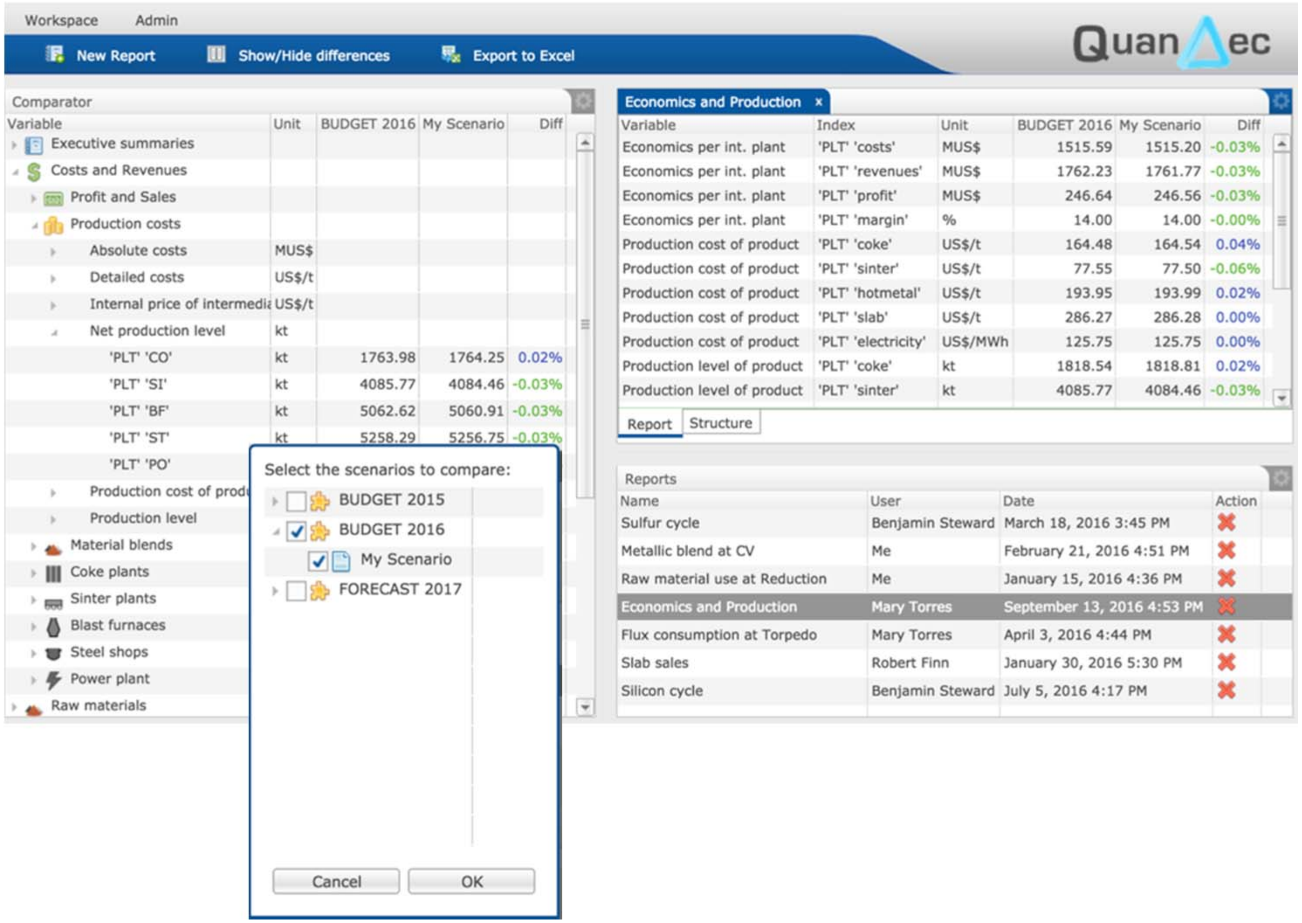
Comments between users





Scenarios with changes history

Traceability and undo system

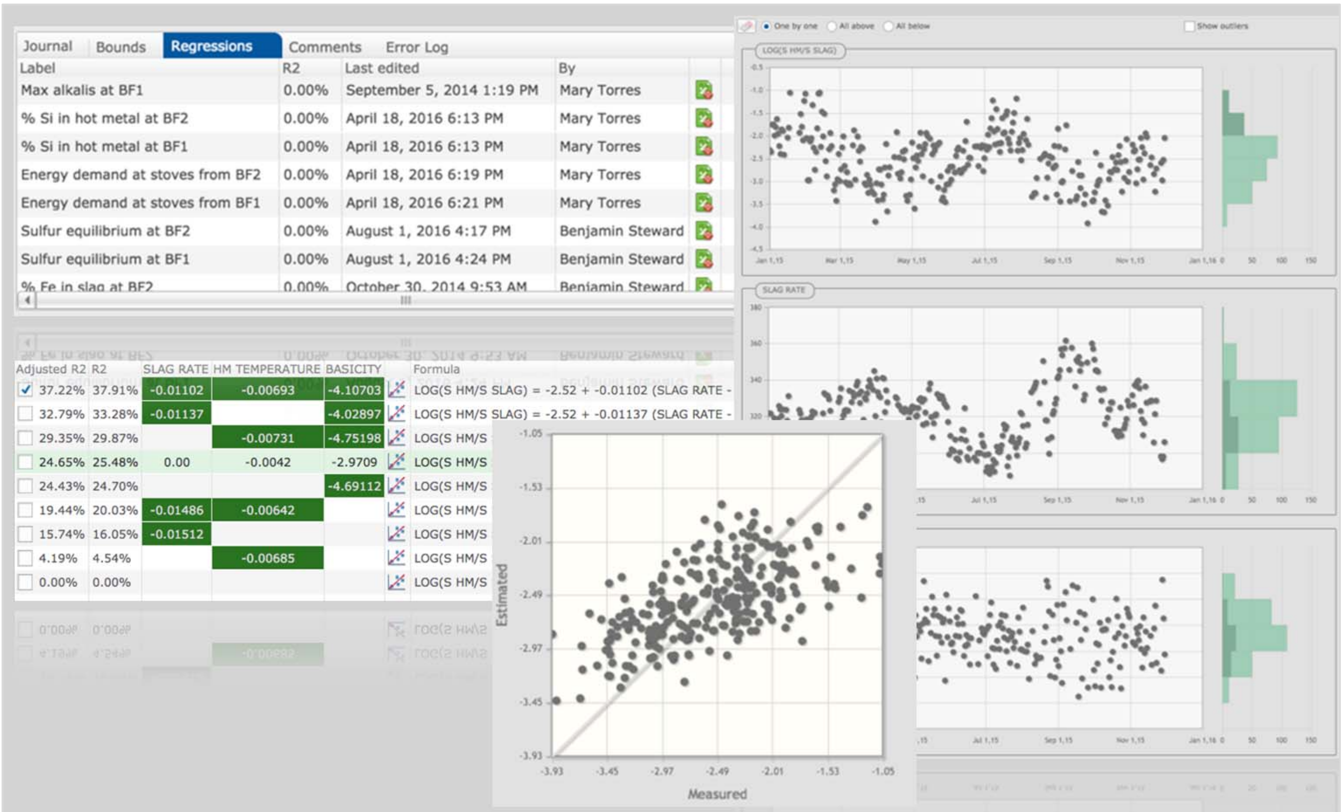


Scenario comparison

All variables can be compared

Display of relative difference

Custom reports

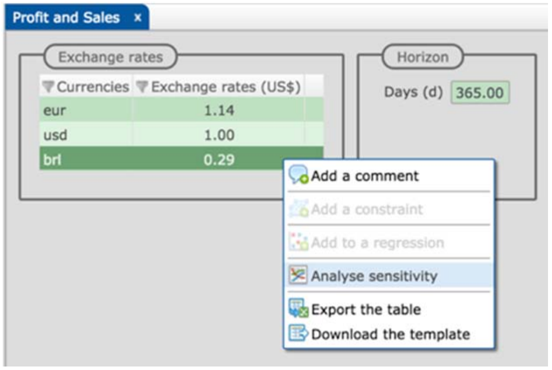


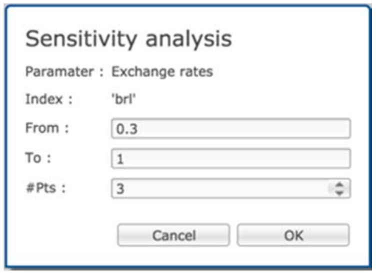
Regression tool

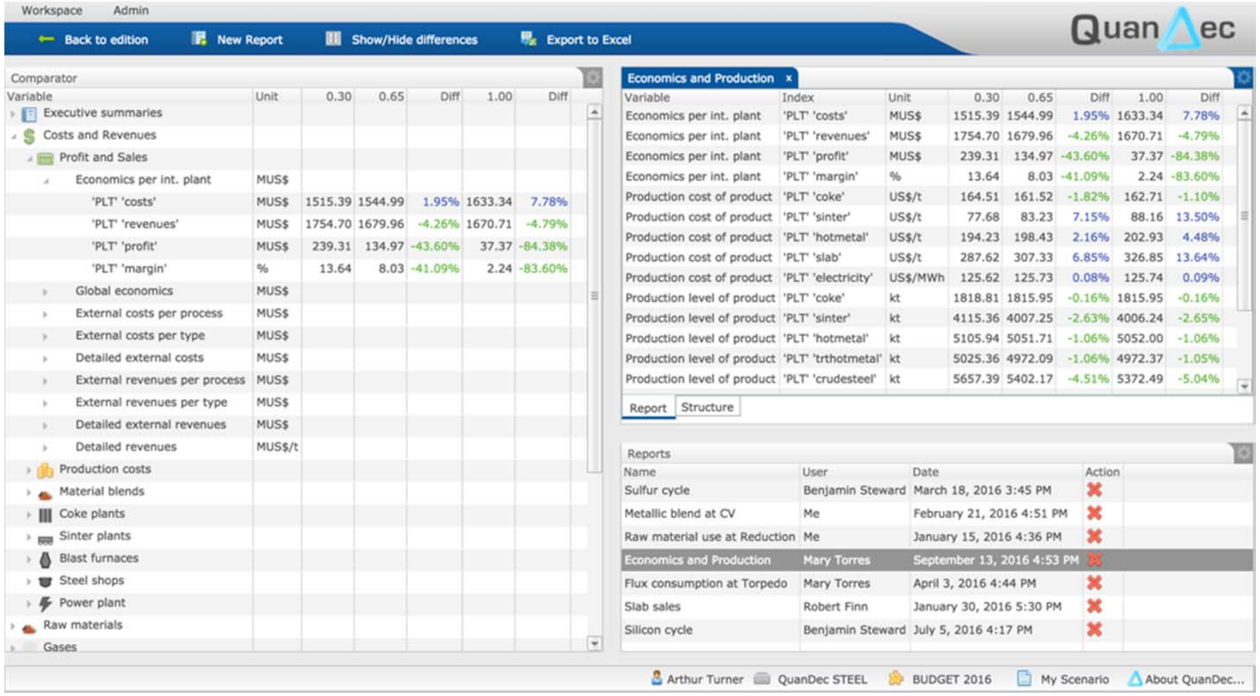
Data cleaning

Any variable can be added to a regression

Manual coefficients if no data available







Sensitivity analysis

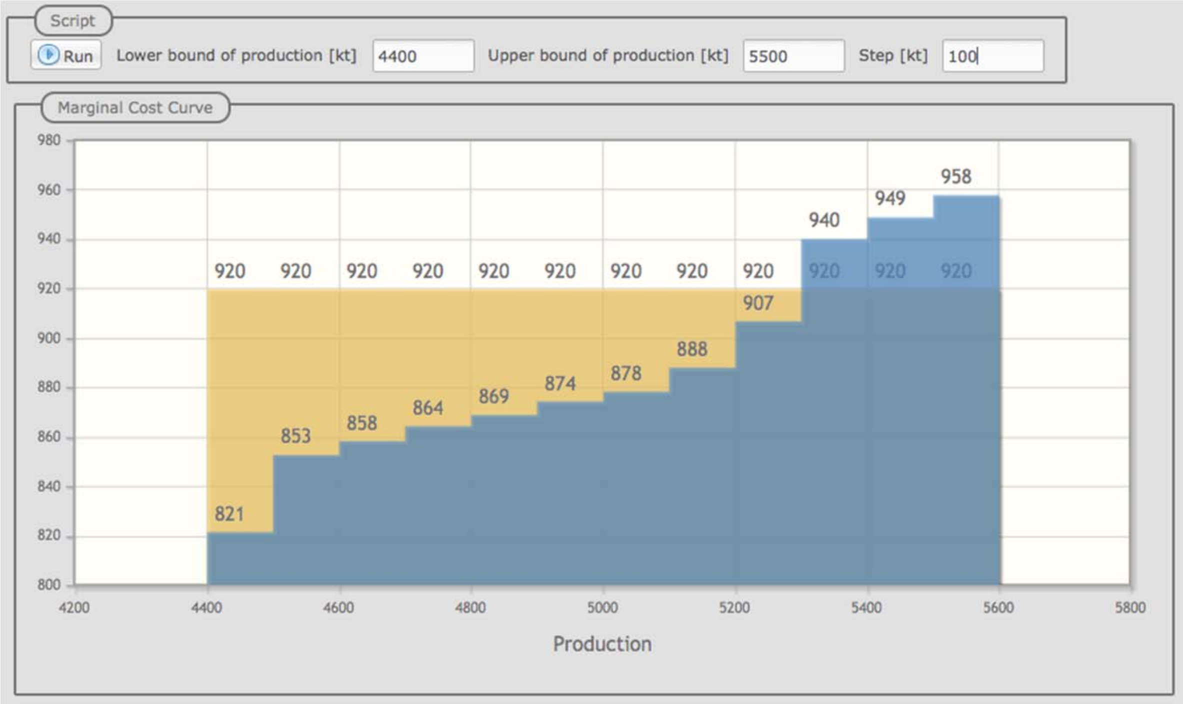
For both parameters AND variables

All variables can be compared

Display of relative difference

Predefined analyses

Script parameters



QuanDec Availability

Ready now for commercial applications

- Free trials available
- Pricing keyed to number of models & users

First year's support included

- Tailored setup support from Cassotis Consulting
- Customizations possible

. . . contact sales@ampl.com for details