Model-Based Optimization

PLAIN AND SIMPLE

From Formulation to Deployment with AMPL

AMPL Optimization Inc.

www.ampl.com — +1 773-336-AMPL

INFORMS Business Analytics Conference

Technology Workshop, 15 April 2018

Model-Based Optimization, Plain and Simple: From Formulation to Deployment with AMPL

Optimization is the most widely adopted technology of Prescriptive Analytics, but also the most challenging to implement:

- How can you prototype an optimization application fast enough to get results before the problem owner loses interest?
- How can you integrate optimization into your enterprise's decision-making systems?
- How can you deploy optimization models to support analysis and action throughout your organization?

In this presentation, we show how AMPL gets you going without elaborate training, extra programmers, or premature commitments. We start by introducing model-based optimization, the key approach to streamlining the optimization modeling cycle and building successful applications today. Then we demonstrate how AMPL's design of a language and system for model-based optimization is able to offer exceptional power of expression while maintaining ease of use.

The remainder of the presentation takes a single example through successive stages of the optimization modeling lifecycle:

- Prototyping in an interactive command environment.
- Integration via AMPL scripts and through APIs to all popular programming languages.
- Deployment with QuanDec, which turns an AMPL model into an interactive, collaborative decision-making tool.

Our example is simple enough for participants to follow its development through the course of this short workshop, yet rich enough to serve as a foundation for appreciating model-based optimization in practice.

Outline

Part 1. Model-based optimization, plain and simple

`https://ampl.com/MEETINGS/TALKS/2018_04_Baltimore_Workshop1.pdf`

- Comparison of *method-based* and *model-based* approaches
- Modeling languages for optimization
- Algebraic modeling languages: AMPL
- Solvers for broad model classes

Part 2. From formulation to deployment with AMPL

`https://ampl.com/MEETINGS/TALKS/2018_04_Baltimore_Workshop2.pdf`

- Building models: *AMPL's interactive environment*
- Developing applications: *AMPL scripts*
  - Extending script applications with Python: *pyMPL*
- Embedding into applications: *AMPL APIs*
- Creating an interactive decision-making tool: *QuanDec*

Part 2

From Formulation to Deployment with AMPL

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 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];
```

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

AMPL Formulation *(cont'd)*

Explicit data (independent of model)

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

param nPAT := 9 ;

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

Command Language

Model + data = problem instance to be solved

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

ampl: option solver cplex;

ampl: solve;

CPLEX 12.8.0.0: 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 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
```

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];
```

Roll Cutting

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
```

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;
```

Scripting

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

Scripting

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 ;
```

Scripting

Pattern Enumeration

Script (initialize)

```
model cutPAT.mod;

param dsetname symbolic;
print "Enter dataset name:";
read dsetname <-;

data (dsetname & ".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%
```

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%
```

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 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%
```

Pattern Enumeration: Observations

Parameters can serve as script variables

- Declare as in model
  - `param pattern {WIDTHS} integer >= 0;`
- Use in algorithm
  - `let pattern[curr_width] := pattern[curr_width] - 1;`
- Assign to model parameters
  - `let {w in WIDTHS} nbr[w,nPAT] := pattern[w];`

Scripts are easy to modify

- Store only every 100th pattern found

```
* if nPAT mod 100 = 0 then
     let {w in WIDTHS} nbr[w,nPAT/100] := pattern[w];
```

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

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];
```

Scripting

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;
   };
```

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];
};
```

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));
   }
```

Scripting

Pattern Generation

Results (relaxation)

```
ampl: include cutpatgen.run

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

Optimal relaxation: 17.9412 rolls

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

WASTE =  2.02%
```

Scripting

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%
```

Pattern Generation: Observations

Patterns automatically added to cutting problem

- Index variables & sums over a set
  - `var Cut {1..nPAT} integer >= 0;`
  - ```
    subj to Fulfill {i in WIDTHS}:
       sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i]
    ```
- Add patterns by expanding the set
  - `let nPAT := nPAT + 1;`

Weights automatically modified in knapsack problem

- Define objective in terms of a parameter
  - ```
    minimize Reduced_Cost:
       1 - sum {i in WIDTHS} price[i] * Use[i];
    ```
- Modify objective by changing the parameter
  - `let {i in WIDTHS} price[i] := Fill[i].dual;`

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?

- ***Extend the scripting language*** ***(pyAMPL)***
- ***Bring the modeling language to the programmer*** ***(APIs)***

“pyAMPL” *(coming soon)*

Extend AMPL’s scripting language with Python

- Execute Python code inside an AMPL script
- Generate parts of AMPL models using Python

Develop add-ons for enhanced AMPL modeling

- Piecewise-linear functions given by breakpoint and value
- Vector-packing formulations for cutting & packing
- Lot-sizing reformulations
- Subtour elimination constraints

Access solver callbacks

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

Key to R program examples

- AMPL entities
- AMPL API R objects
- AMPL API R methods
- R 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 R Data

A float, an integer, and a dataframe

```
roll_width <- 64.5

overrun <- 3

orders <- data.frame(

  width  = c( 6.77, 7.56, 17.46, 18.76 ),
  demand = c(   10,   40,    33,    10 )

)
```

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 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 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 R

Display solution

```
  # Prepare summary 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 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 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()}
```

AMPL API

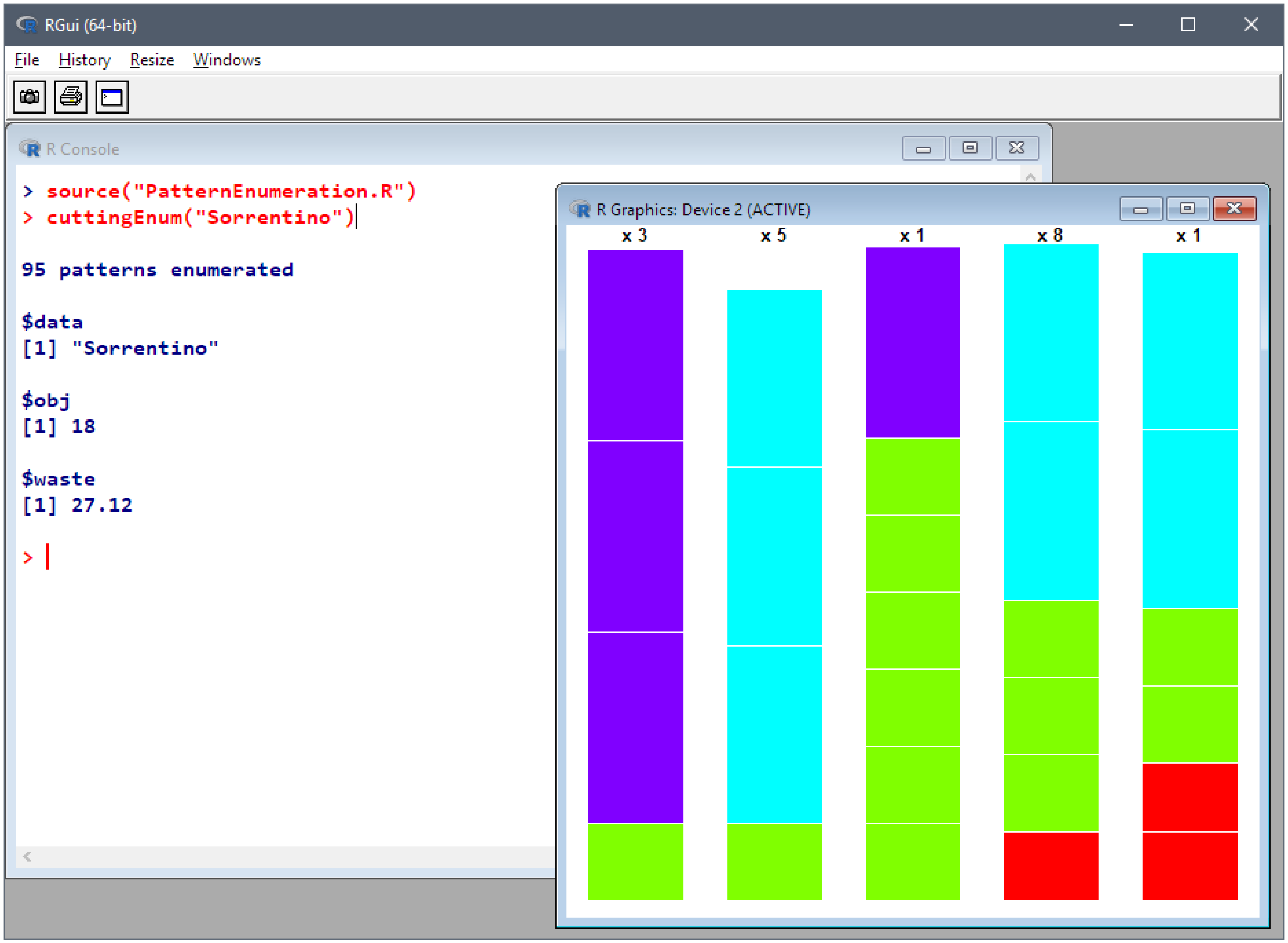
Pattern Enumeration in R

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 R



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 R

Get data, set up master problem

```
cuttingGen <- function(dataset) {
  library(rAMPL)

  # Read orders, roll_width, overrun

  source(paste(dataset, ".R", sep=""))
  widths <- sort(orders$width)

  # Set up cutting (master problem) model

  Master <- new(AMPL)
  Master$setOption("ampl_include", "models")
  Master$read("cut.mod")

  # Define a param for sending AMPL new patterns

  Master$eval("param newPat {WIDTHS} integer >= 0;")

  # Set solve options

  Master$setOption("solver", "gurobi")
  Master$setOption("relax_integrality", 1)
```

Pattern Generation in R

Send data to master problem

```
# Send scalar values

Master$getParameter("nPatterns")$set(length(widths))
Master$getParameter("overrun")$set(overrun)
Master$getParameter("rawWidth")$set(roll_width)

# Send order vector

Master$getSet("WIDTHS")$setValues(widths)
Master$getParameter("order")$setValues(orders$demand)

# Generate and send initial pattern matrix

patmat <- matrix(0, nrow=length(widths), ncol=length(widths))

for(i in 1:nrow(patmat)){
  patmat[i, i] <- floor(roll_width/widths[i])
}

df <- as.data.frame(as.table(patmat))
df[,1] <- widths[df[,1]]
df[,2] <- as.numeric(df[,2])

Master$getParameter("rolls")$setValues(df)
```

Pattern Generation in R

Set up subproblem

```
# Define knapsack subproblem

Sub <- new(AMPL)
Sub$setOption("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$getSet("SIZES")$setValues(widths)
Sub$getParameter("cap")$setValues(roll_width)
```

Pattern Generation in R

Generate patterns and re-solve cutting problems

```
# Alternate between master and sub solves

while(TRUE) {
     Master$solve()

     Sub$getParameter("val")$setValues(
         Master$getConstraint("OrderLimits")$getValues())
     Sub$solve()
     if(Sub$getObjective("TotVal")$value() <= 1.00001) {
         break
     }

     pattern <- Sub$getVariable("Qty")$getValues()
     Master$getParameter("newPat")$setValues(pattern)
     patmat <- cbind(patmat, pattern[,-1])

     Master$eval("let nPatterns := nPatterns + 1;")
     Master$eval("let {w in WIDTHS} rolls[w, nPatterns] := newPat[w];")
}

# Compute integer solution

Master$setOption("relax_integrality", 0)
Master$solve()
```

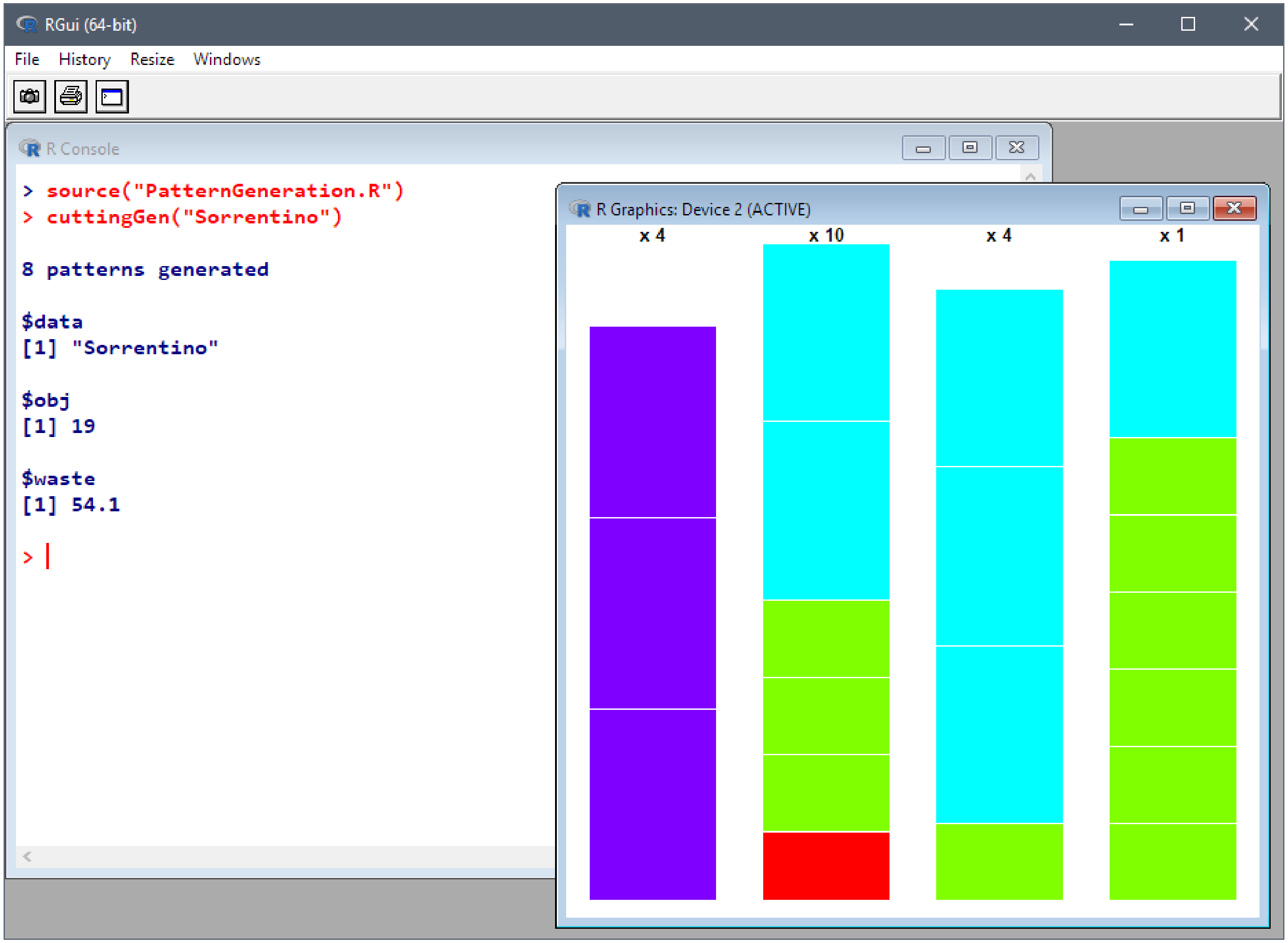
Pattern Generation in R

Display solution

```
  # Retrieve solution

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

  # Prepare summary data

  data <- dataset
  obj <- Master$getObjective("TotalRawRolls")$value()

  waste <- Master$getValue(
    "sum {p in PATTERNS} Cut[p] * (rawWidth - sum {w in WIDTHS} w*rolls[w,p])"
  )

  summary <- list(data=dataset, obj=obj, waste=waste)

  cat(sprintf("\n%d patterns generated\n\n",
      Master$getParameter("nPatterns")$value()))

  # Create plot of solution

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

AMPL API

Pattern Generation in R



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

QuanDec

Initialization

Prepare the model and data

- ❖ Add reporting variables to the model
- ❖ Select initial data in AMPL .dat format

Import to QuanDec

- ❖ Install on a server
- ❖ Read zipfile of model and data
- ❖ Create new application and first master

Configure displays

- ❖ Create data tables
- ❖ Adjust views

. . . mostly done automatically

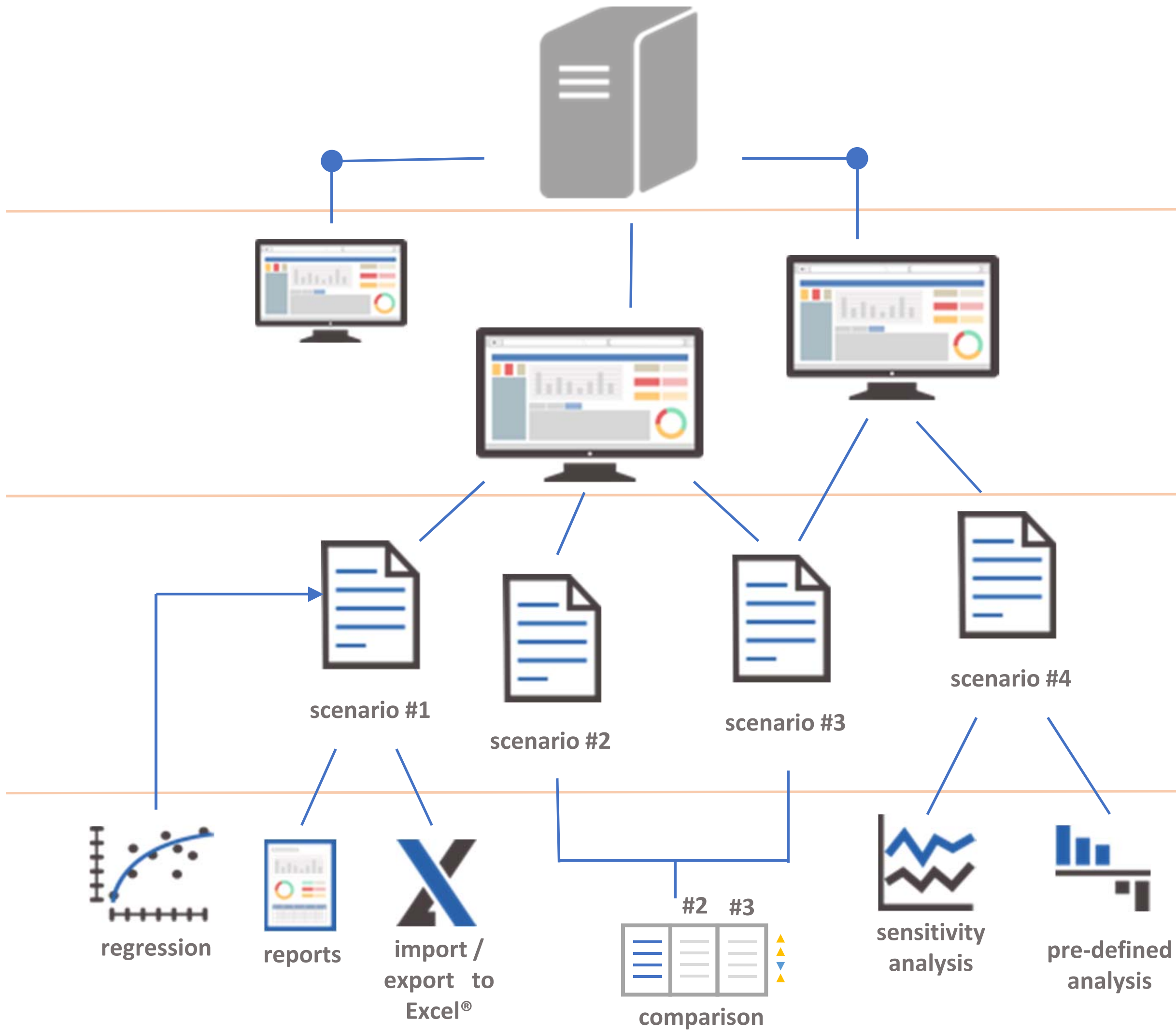
QUANDEC
Quantify your decisions

CASSOTIS
CONSULTING & SOLUTIONS IN OPTIMIZATION

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

AMPL

Features



Server application

Centralized data

Several apps on a single instance

Web-based

Multi-users

Concurrent access

Secure access

Scenario-based

Sharing between users

Sharing rights (edit / comment/ view)

And much more…

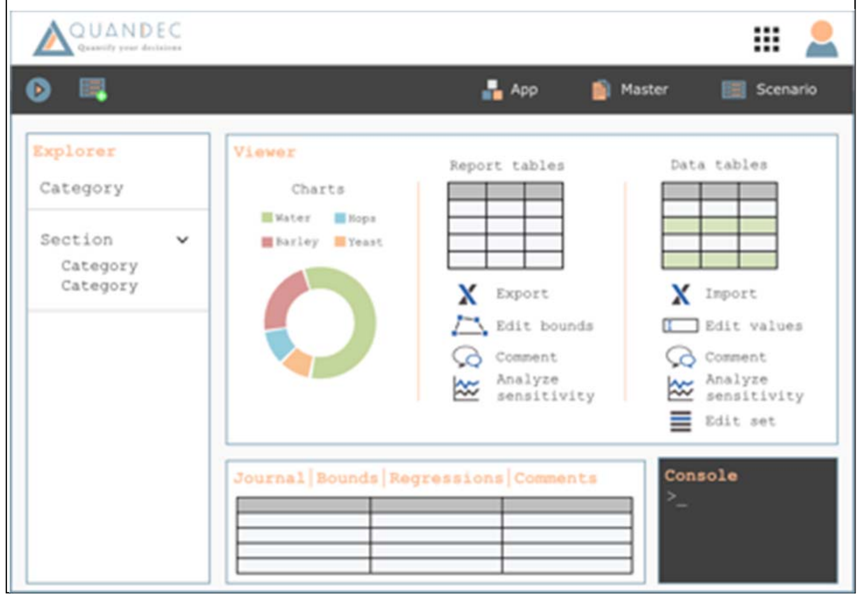
Getting started

Your
AMPL model

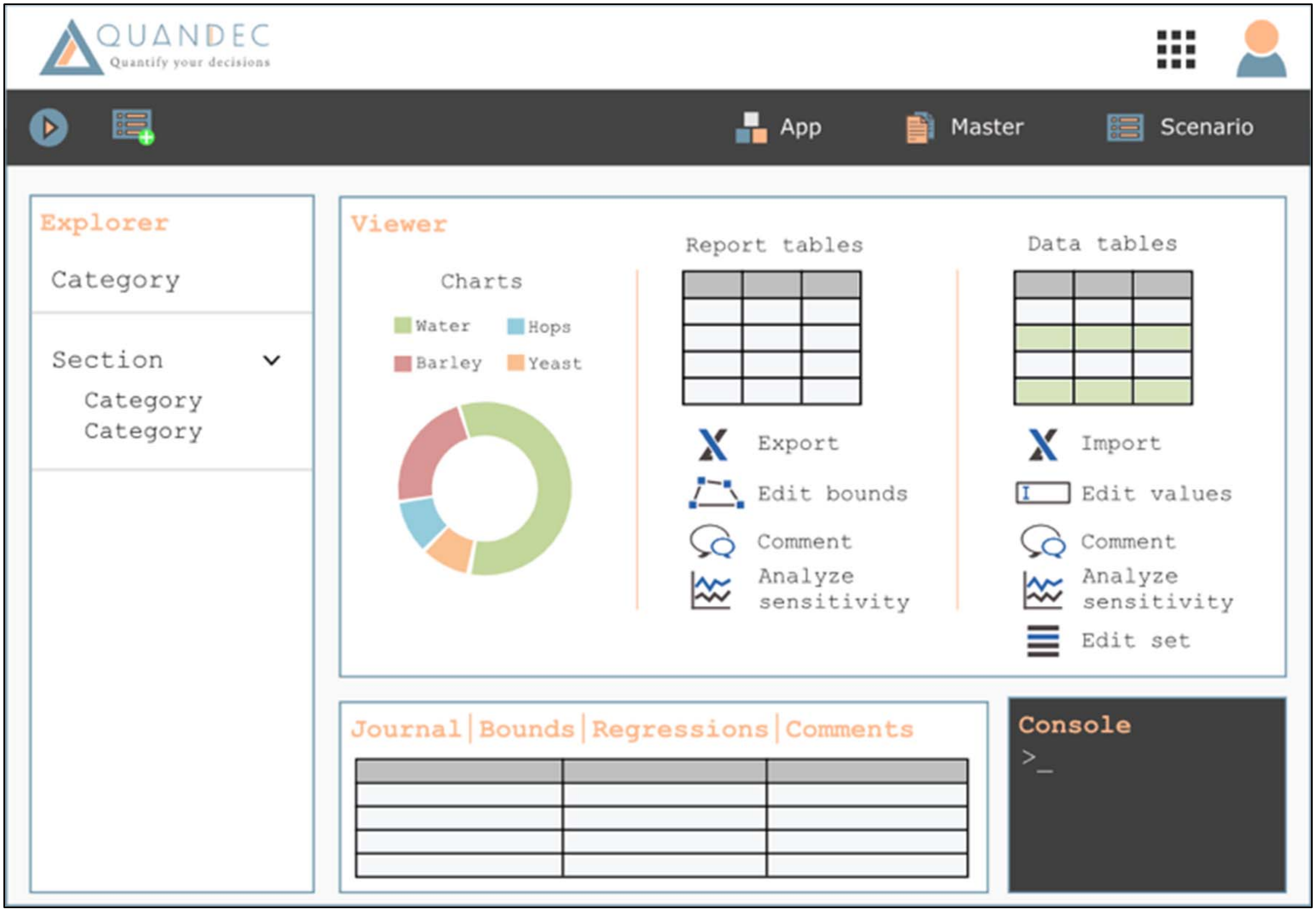
Configure how you want to display your parameters and variables (many options of tables and charts)

Zip and upload configuration and AMPL model files into

Quantify your decisions!

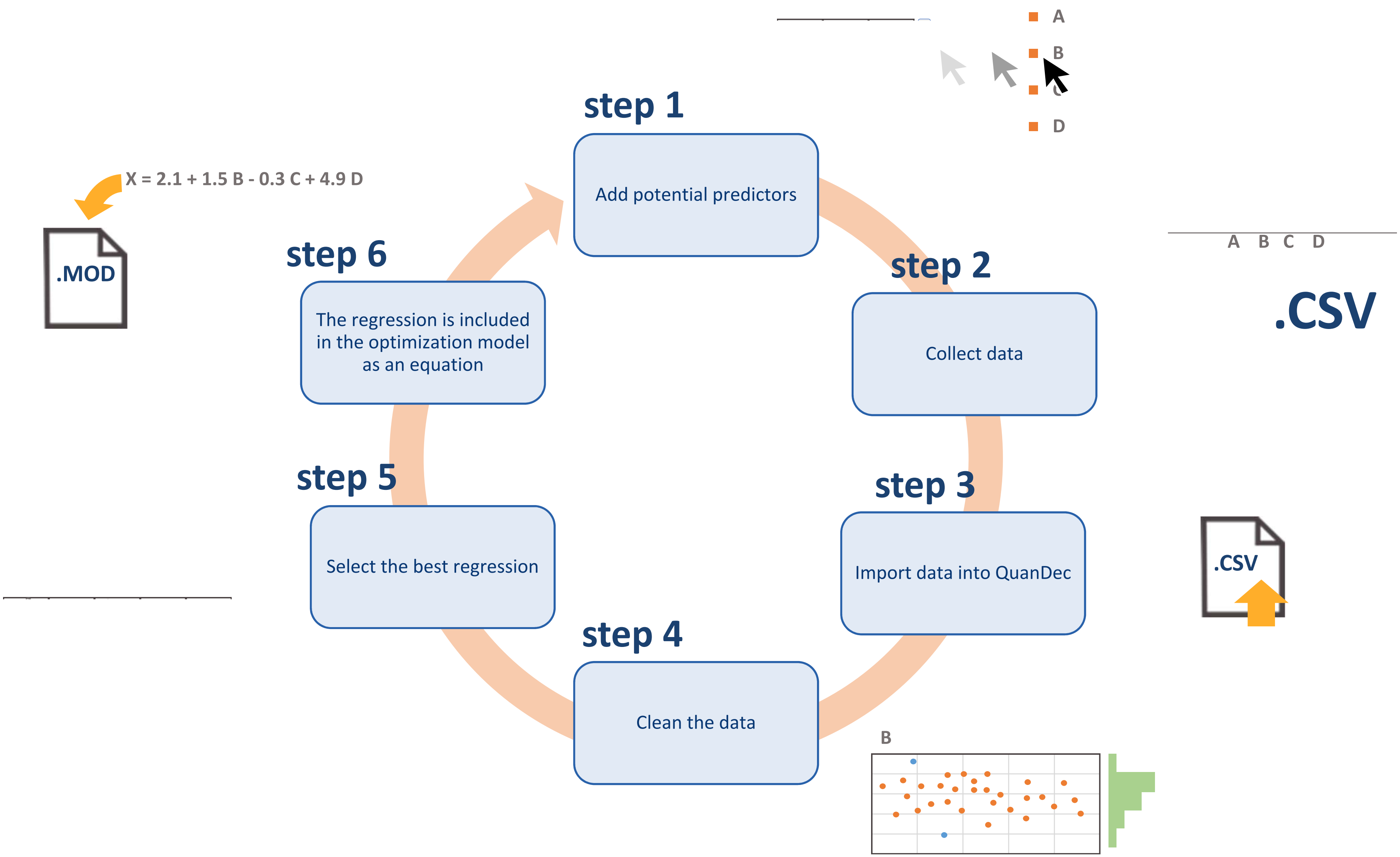


Workbench



QUANDEC
Quantify your decisions
App
Master
Scenario
Explorer
Category
Section
Category
Category
Viewer
Charts
Water
Hops
Barley
Yeast
Report tables
Export
Edit bounds
Comment
Analyze sensitivity
Data tables
Import
Edit values
Comment
Analyze sensitivity
Edit set
Journal | Bounds | Regressions | Comments
Console
>_

Regression tool

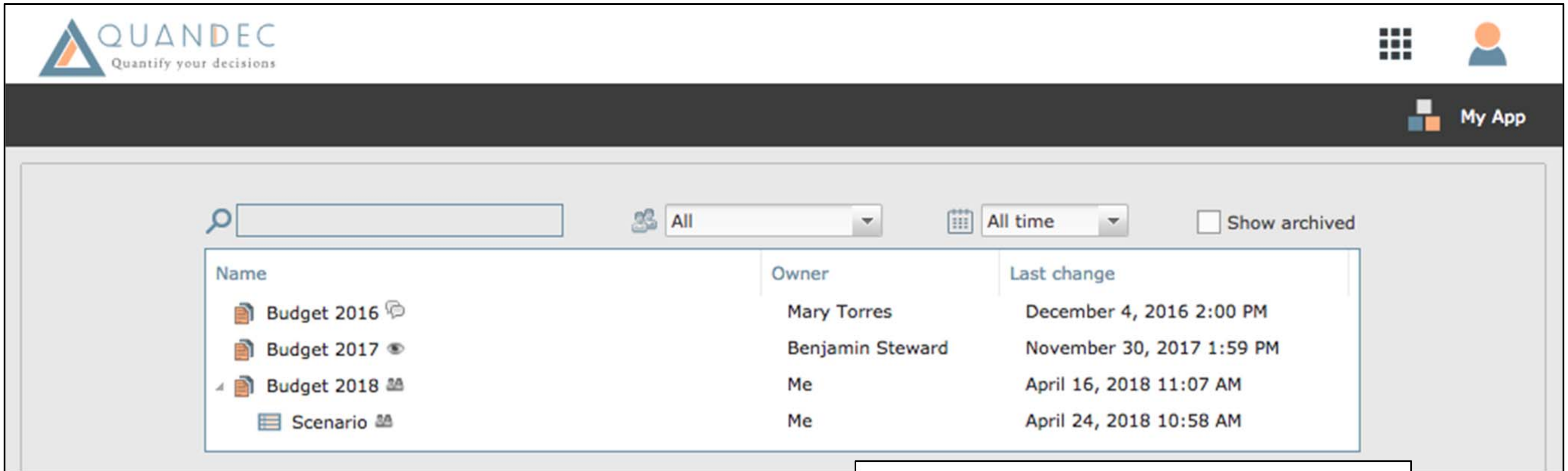


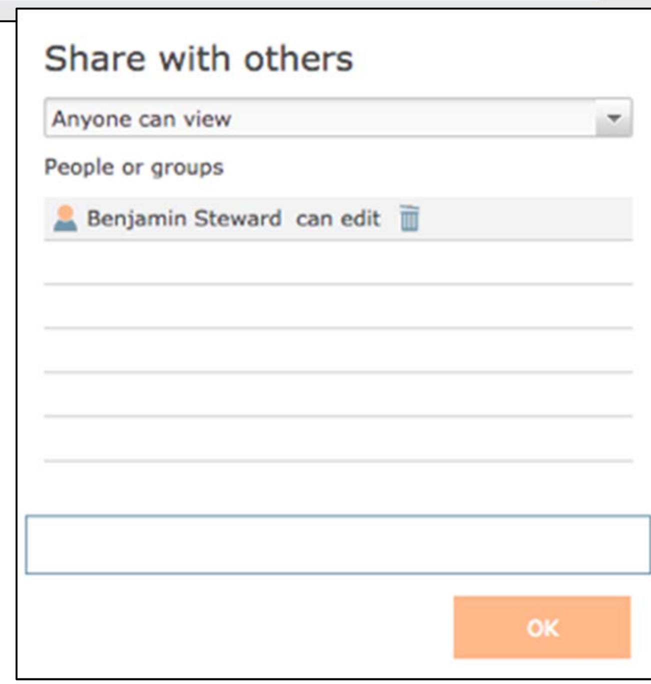
Web-application

Multi-users

Secure access

Concurrent access

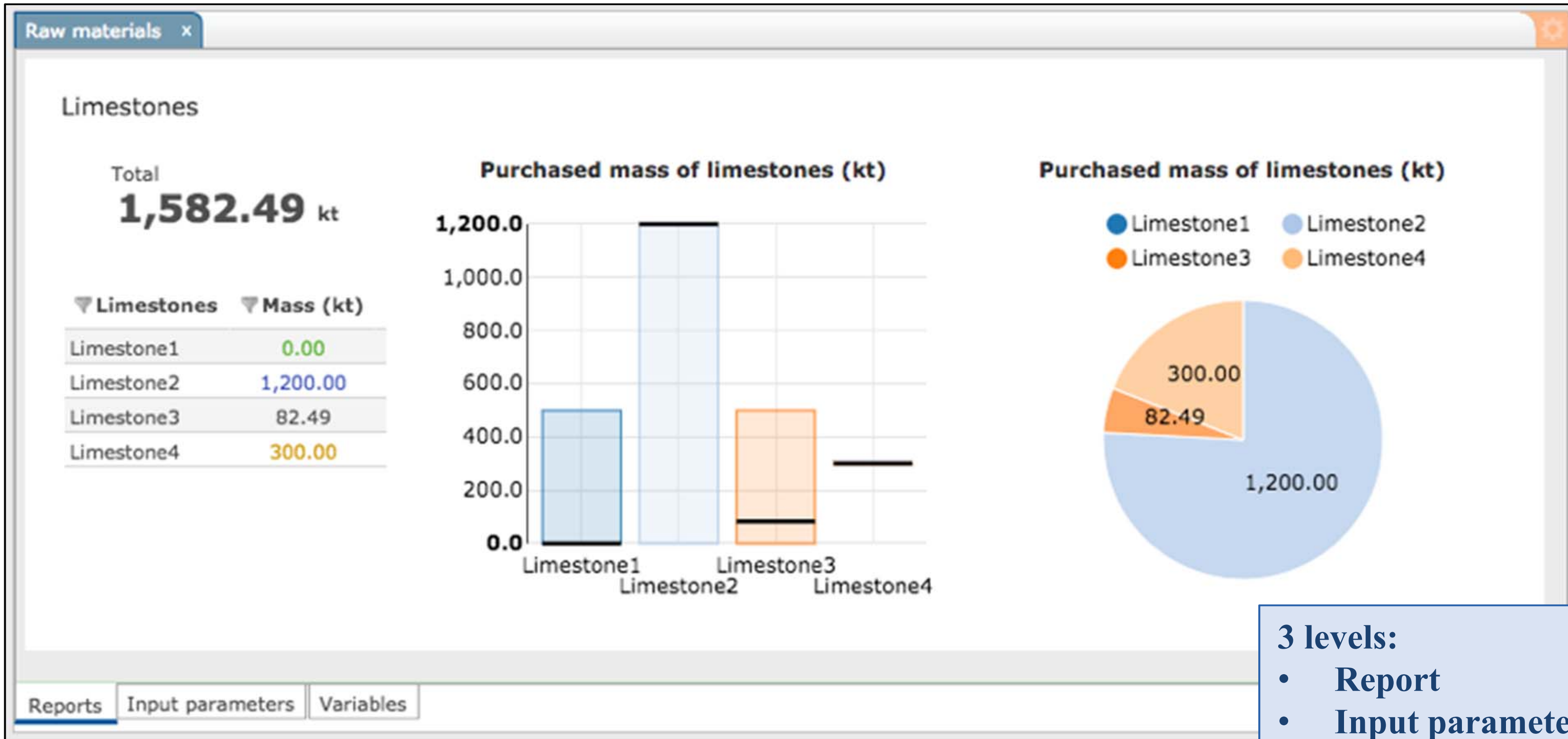




Scenario-based environment

Sharing system

Permission:
Edit – Comment - View

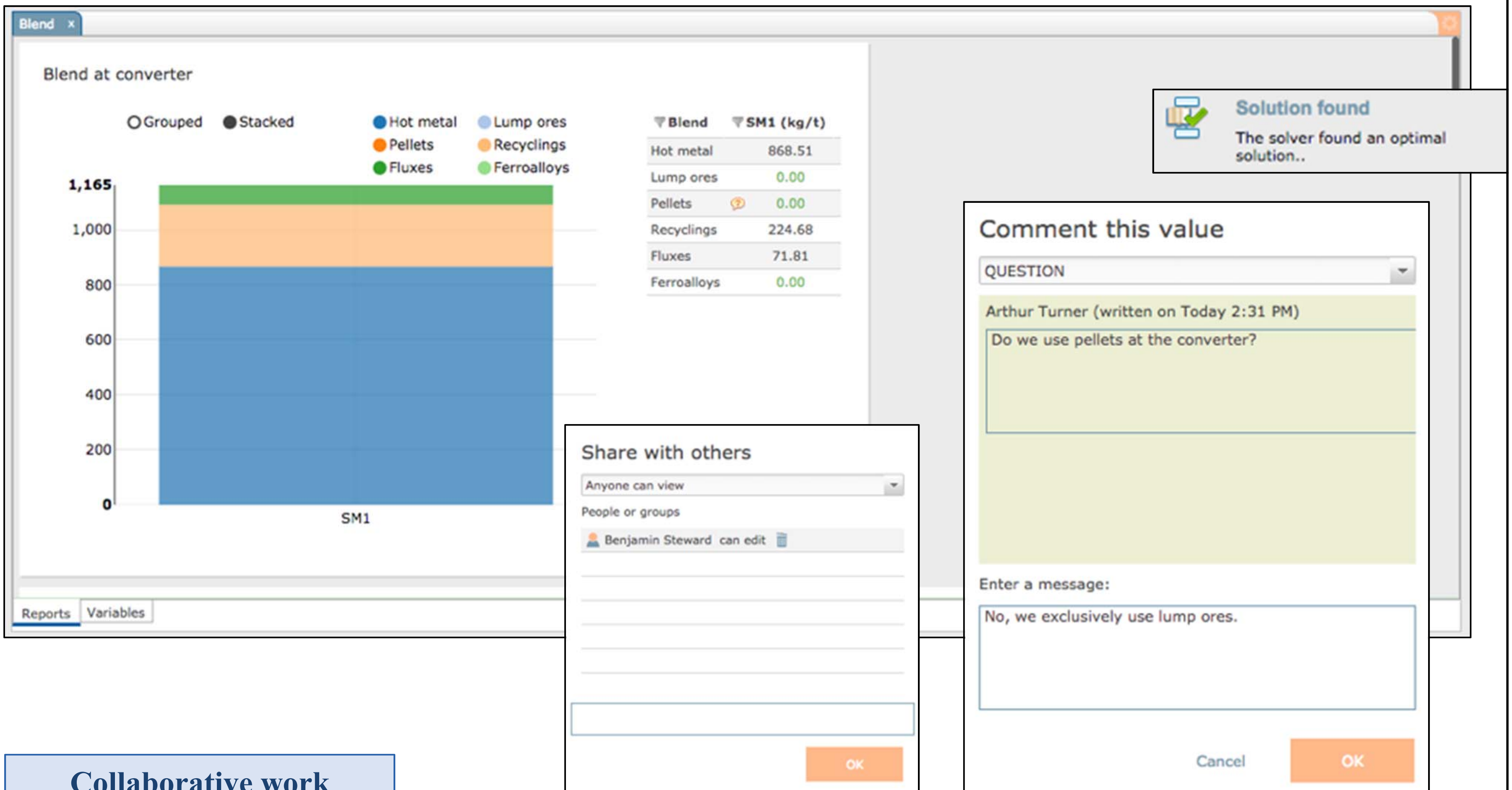


3 levels:
- **Report**
- **Input parameters**
- **Variables**

Chart and tables

Colored values for easier analysis

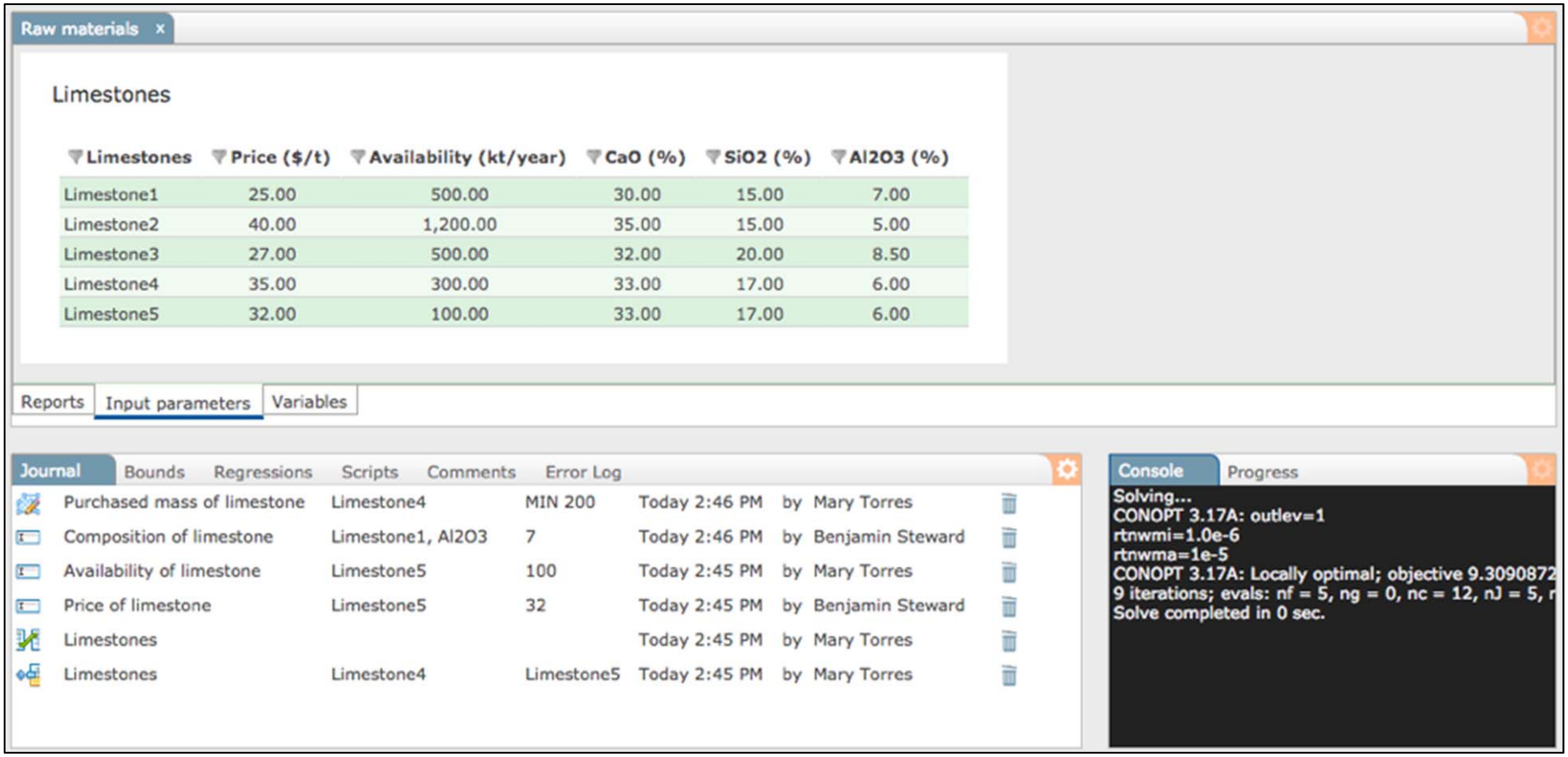
Constraint (min/max) on any variable



Collaborative work

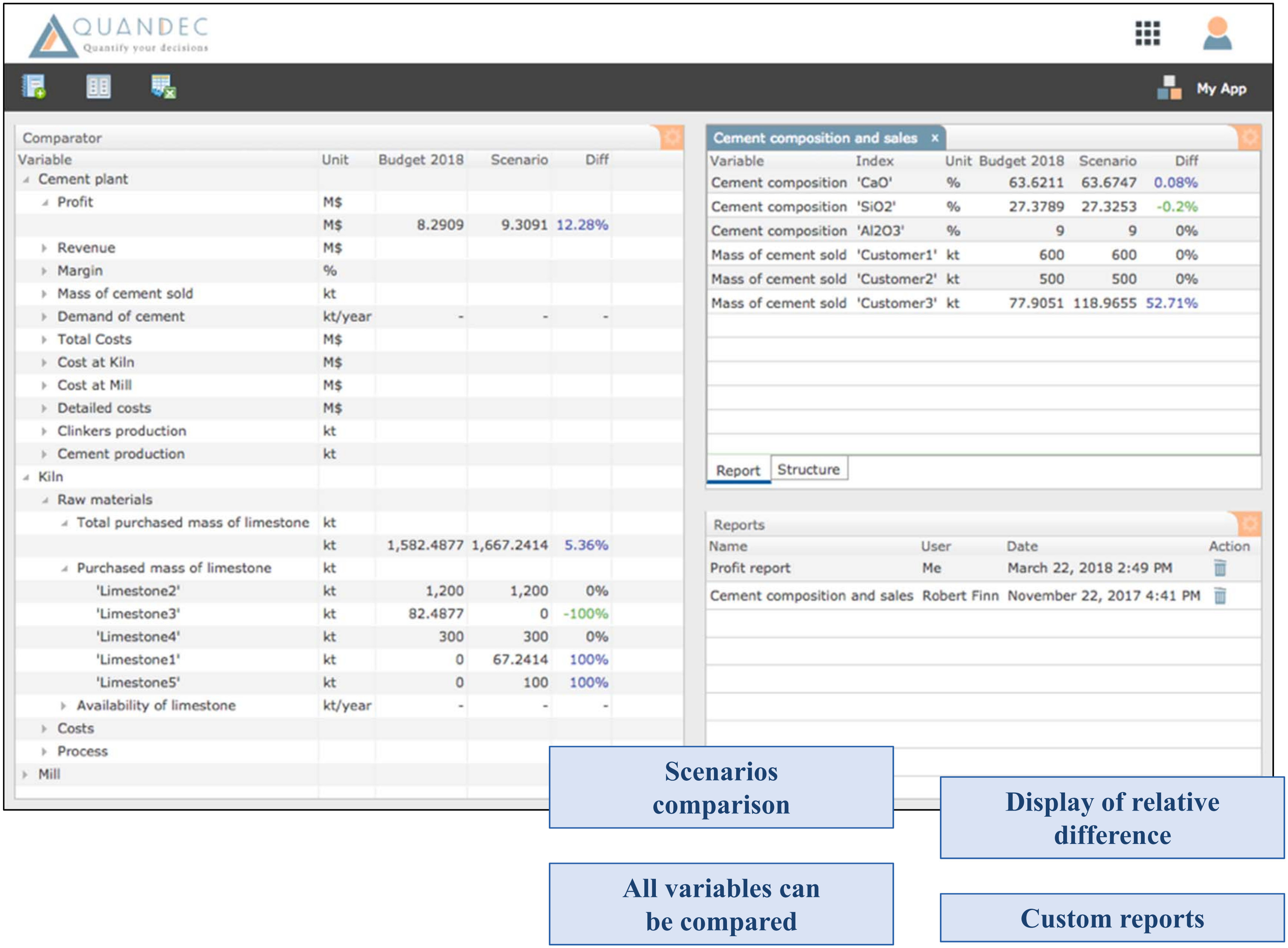
Notification system

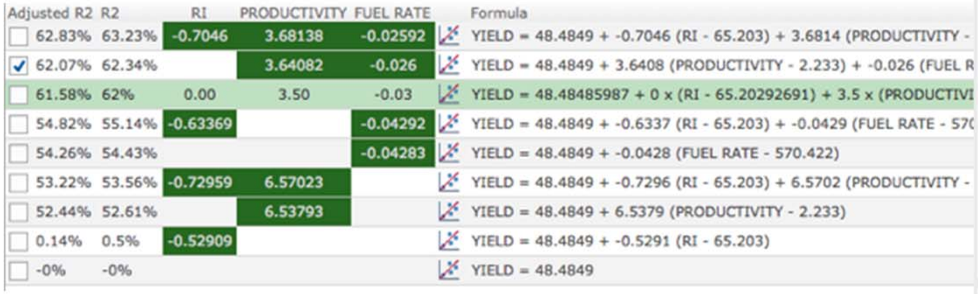
Comments between users



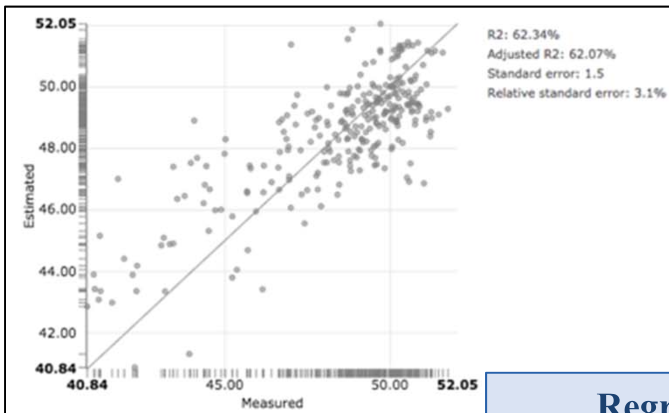
Scenarios with changes history

Traceability and undo system


QUANDEC
Quantify your decisions
My App
Comparator
Variable Unit Budget 2018 Scenario Diff
Cement plant
Profit M$
M$ 8.2909 9.3091 12.28%
Revenue M$
Margin %
Mass of cement sold kt
Demand of cement kt/year - - -
Total Costs M$
Cost at Kiln M$
Cost at Mill M$
Detailed costs M$
Clinkers production kt
Cement production kt
Kiln
Raw materials
Total purchased mass of limestone kt
kt 1,582.4877 1,667.2414 5.36%
Purchased mass of limestone kt
'Limestone2' kt 1,200 1,200 0%
'Limestone3' kt 82.4877 0 -100%
'Limestone4' kt 300 300 0%
'Limestone1' kt 0 67.2414 100%
'Limestone5' kt 0 100 100%
Availability of limestone kt/year - - -
Costs
Process
Mill
Cement composition and sales
Variable Index Unit Budget 2018 Scenario Diff
Cement composition 'CaO' % 63.6211 63.6747 0.08%
Cement composition 'SiO2' % 27.3789 27.3253 -0.2%
Cement composition 'Al2O3' % 9 9 0%
Mass of cement sold 'Customer1' kt 600 600 0%
Mass of cement sold 'Customer2' kt 500 500 0%
Mass of cement sold 'Customer3' kt 77.9051 118.9655 52.71%
Report
Structure
Reports
Name User Date Action
Profit report Me March 22, 2018 2:49 PM
Cement composition and sales Robert Finn November 22, 2017 4:41 PM
Scenarios comparison
Display of relative difference
All variables can be compared
Custom reports



Adjusted R2	R2	RI	PRODUCTIVITY	FUEL RATE	Formula
62.83%	63.23%	-0.7046	3.68138	-0.02592	YIELD = 48.4849 + -0.7046 (RI - 65.203) + 3.6814 (PRODUCTIVITY -
62.07%	62.34%		3.64082	-0.026	YIELD = 48.4849 + 3.6408 (PRODUCTIVITY - 2.233) + -0.026 (FUEL R
61.58%	62%	0.00	3.50	-0.03	YIELD = 48.48485987 + 0 x (RI - 65.20292691) + 3.5 x (PRODUCTIVI
54.82%	55.14%	-0.63369		-0.04292	YIELD = 48.4849 + -0.6337 (RI - 65.203) + -0.0429 (FUEL RATE - 570
54.26%	54.43%			-0.04283	YIELD = 48.4849 + -0.0428 (FUEL RATE - 570.422)
53.22%	53.56%	-0.72959	6.57023		YIELD = 48.4849 + -0.7296 (RI - 65.203) + 6.5702 (PRODUCTIVITY -
52.44%	52.61%		6.53793		YIELD = 48.4849 + 6.5379 (PRODUCTIVITY - 2.233)
0.14%	0.5%	-0.52909			YIELD = 48.4849 + -0.5291 (RI - 65.203)
-0%	-0%				YIELD = 48.4849

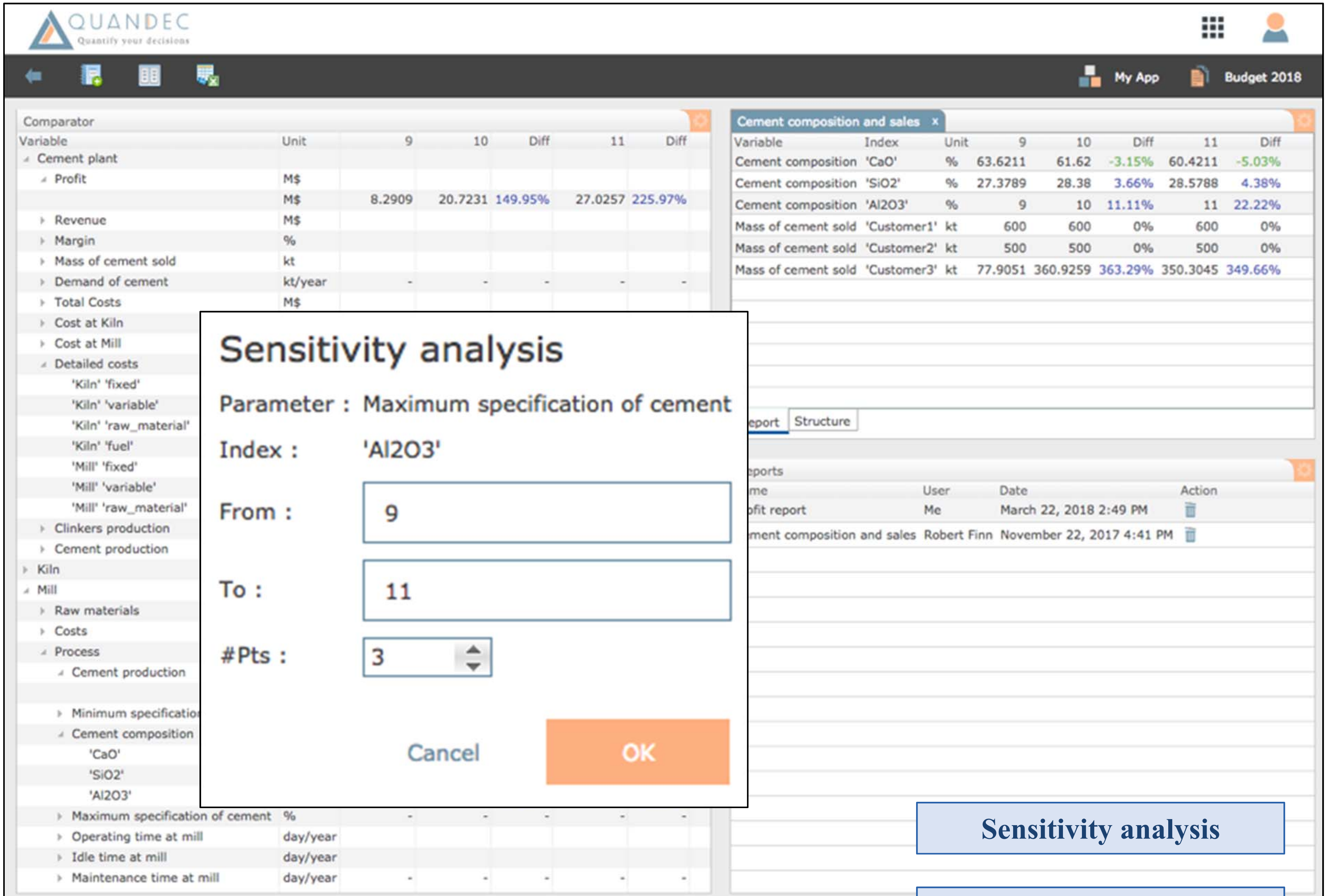


Regression tool

Data cleaning

Any variable can be added to a regression

Manuel coefficients if no data available



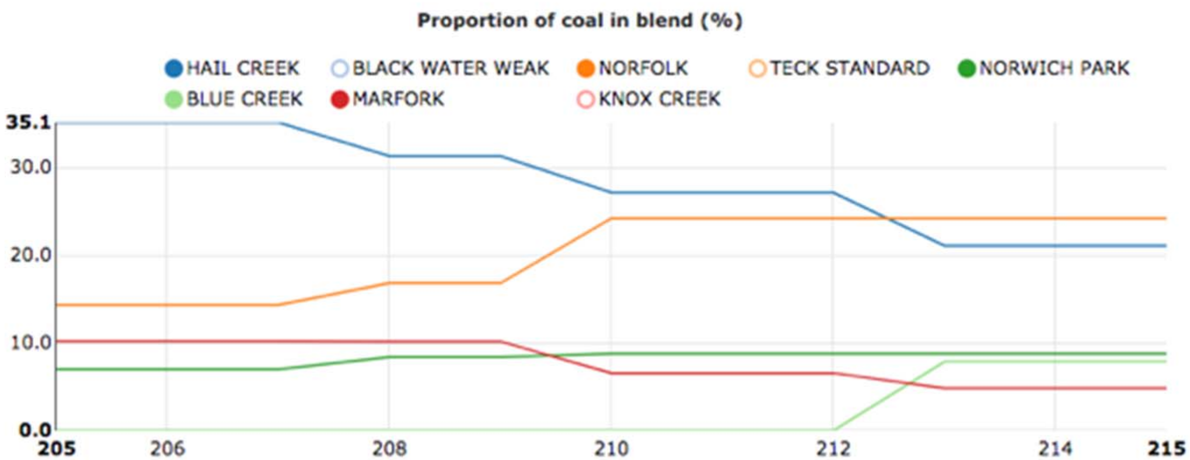
Sensitivity analysis

For both parameters AND variables

Coal blend at coke plant ×

Reports

Minimum cost (×) 205 | Maximum cost (×) 215 | Step (×) 1 | Coal selection HAIL CREEK | Integrated plant PLT | Run



Coals	205 (%)	208 (%)	210 (%)	213 (%)	215 (%)
HAIL CREEK	35.11	31.28	27.12	21.07	21.07
SHOAL CREEK	0.00	0.00	0.00	0.00	0.00
MCCLURE	0.00	0.00	0.00	0.00	0.00
BLACK WATER WEAK	13.52	13.50	13.49	13.48	13.48
GERMAN CREEK	0.00	0.00	0.00	0.00	0.00
NORFOLK	14.32	16.80	24.20	24.18	24.18
TECK STANDARD	11.99	11.98	11.97	11.96	11.96
NORWICH PARK	6.98	8.38	8.79	8.76	8.76
BLUE CREEK	0.00	0.00	0.00	7.85	7.85
MARFORK	10.17	10.16	6.56	4.83	4.83
KNOX CREEK	7.91	7.90	7.89	7.88	7.88
GREGORY	0.00	0.00	0.00	0.00	0.00

Analysis

Pre-defined analysis

Parameterized scripts

QuanDec Availability

Contact sales@ampl.com

- ❖ Free trials available
- ❖ Licensing follows AMPL licensing options

First year's support included

- ❖ Tailored setup support from CASSOTIS
- ❖ Customizations possible