How to Contact the Xpress Team

Information, Sales and Licensing

USA, CANADA AND ALL AMERICAS
Email: XpressSalesUS@fico.com

WORLDWIDE
Email: XpressSalesUK@fico.com
Tel: +44 207 940 8718
Fax: +44 870 420 3601
Xpress Optimization, FICO
FICO House
International Square
Starley Way
Birmingham B37 7GN
UK

Product Support
Email: Support@fico.com
(Please include ‘Xpress’ in the subject line)

Telephone:
NORTH AMERICA
Tel (toll free): +1 (877) 4FI-SUPP
Fax: +1 (402) 496-2224
EUROPE, MIDDLE EAST, AFRICA
Tel: +44 (0) 870-420-3777
UK (toll free): 0800-0152-153
South Africa (toll free): 0800-996-153
Fax: +44 (0) 870-420-3778
ASIA-PACIFIC, LATIN AMERICA, CARIBBEAN
Tel: +1 (415) 446-6185
Brazil (toll free): 0800-891-6146

For the latest news and Xpress software and documentation updates, please visit the Xpress website at http://www.fico.com/xpress or subscribe to our mailing list.
Robust Optimization with FICO Xpress Optimization
Usage guidelines and examples

P. Belotti, Z. Csizmadia, S. Heipcke, S. Lannez

Xpress Optimization, FICO, FICO House, Starley Way, Birmingham B37 7GN, UK
http://www.fico.com/xpress

Release 8.3
22 May, 2017

Abstract
This whitepaper gives an introduction to formulating and solving Robust Optimization problems with FICO®
Xpress Optimization.
Robust optimization is a modelling framework that can be used to model optimization problems under uncer-
tainty in the input data. The uncertainty is assumed to be defined by a set of values that the uncertain data can
take; this uncertainty set can be described by linear constraints, convex quadratic constraints, or as a discrete set
of vectors of uncertain data. The solutions generated by a robust optimization model are feasible irrespective
of the actual value that the uncertain data will take.
The introductory part explains the general concepts of Robust Optimization and which types of formulations
are available with Xpress. The remainder of this document is formed by a collection of example problems
showing typical uses of Robust Optimization in practice with a discussion of the problem implementation with
Xpress Mosel.
The examples presented in this whitepaper are included in the examples/Mosel/robust directory of the Xpress
installation.

Contents

1 Introduction ................................................................. 3
1.1 Uncertains and robust constraints .................................... 3
1.2 Types of robust constraints ............................................ 4
   1.2.1 Simple bounds on the uncertain coefficients .................. 4
   1.2.2 Linear constraints on the uncertainty sets ..................... 7
   1.2.3 Ellipsoidal uncertainty sets ..................................... 8
   1.2.4 Equality constraints and uncertain values with no uncertain constraints 9
   1.2.5 Mixing uncertainty sets and types ............................. 10
   1.2.6 Cardinality restrictions for uncertain values ................ 11
   1.2.7 Using historical data: scenarios ................................ 12
   1.2.8 Uncertainty in the objective function .......................... 13
1.3 Nominal values: centered and uncentered uncertainty ............ 13
   1.3.1 The price of robustness ......................................... 14
   1.3.2 Working with nominal values .................................. 15
   1.3.3 Using nominal values to shift the uncertainty set ............ 16
   1.3.4 Using nominal valued uncertain as coefficients ............. 17
1.4 Examples of robust models ........................................... 18
2 Robust shortest path .................................................... 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Problem description</td>
<td>19</td>
</tr>
<tr>
<td>2.2 Mathematical formulation</td>
<td>19</td>
</tr>
<tr>
<td>2.2.1 Shortest path problem</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2 Robust optimization problem</td>
<td>20</td>
</tr>
<tr>
<td>2.3 Implementation</td>
<td>21</td>
</tr>
<tr>
<td>2.4 Results</td>
<td>22</td>
</tr>
<tr>
<td>3 Production planning under demand uncertainty</td>
<td>25</td>
</tr>
<tr>
<td>3.1 Problem description</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Mathematical formulation</td>
<td>25</td>
</tr>
<tr>
<td>3.2.1 Multi-period, multi-item production planning problem</td>
<td>25</td>
</tr>
<tr>
<td>3.2.2 Robust optimization problem</td>
<td>26</td>
</tr>
<tr>
<td>3.3 Implementation</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Results</td>
<td>29</td>
</tr>
<tr>
<td>4 Robust network design</td>
<td>31</td>
</tr>
<tr>
<td>4.1 Problem description</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Mathematical formulation</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Implementation</td>
<td>32</td>
</tr>
<tr>
<td>4.4 Input Data</td>
<td>34</td>
</tr>
<tr>
<td>4.5 Results</td>
<td>34</td>
</tr>
<tr>
<td>5 Robust portfolio optimization</td>
<td>37</td>
</tr>
<tr>
<td>5.1 Problem description</td>
<td>37</td>
</tr>
<tr>
<td>5.2 Mathematical formulation</td>
<td>37</td>
</tr>
<tr>
<td>5.2.1 Highest protection</td>
<td>38</td>
</tr>
<tr>
<td>5.2.2 Budgeted protection</td>
<td>38</td>
</tr>
<tr>
<td>5.3 Implementation</td>
<td>38</td>
</tr>
<tr>
<td>5.4 Results</td>
<td>42</td>
</tr>
<tr>
<td>5.4.1 Input Data</td>
<td>42</td>
</tr>
<tr>
<td>5.4.2 Analysis</td>
<td>43</td>
</tr>
<tr>
<td>5.5 References</td>
<td>44</td>
</tr>
<tr>
<td>6 Robust unit commitment</td>
<td>45</td>
</tr>
<tr>
<td>6.1 Problem description</td>
<td>45</td>
</tr>
<tr>
<td>6.1.1 Robust against power demand variation</td>
<td>45</td>
</tr>
<tr>
<td>6.1.2 Robust against the N-k contingency</td>
<td>46</td>
</tr>
<tr>
<td>6.2 Mathematical formulation</td>
<td>46</td>
</tr>
<tr>
<td>6.2.1 Original Unit Commitment Problem</td>
<td>47</td>
</tr>
<tr>
<td>6.2.2 Load Robust Unit Commitment Problem</td>
<td>47</td>
</tr>
<tr>
<td>6.2.3 The N-k Contingency-Constrained Unit Commitment Problem</td>
<td>47</td>
</tr>
<tr>
<td>6.3 Implementation</td>
<td>48</td>
</tr>
<tr>
<td>6.3.1 The Original Unit Commitment Implementation</td>
<td>48</td>
</tr>
<tr>
<td>6.3.2 The Load Robust Unit Commitment Implementation</td>
<td>50</td>
</tr>
<tr>
<td>6.3.3 The N-k Contingency-Constrained Unit Commitment Implementation</td>
<td>50</td>
</tr>
<tr>
<td>6.4 Results</td>
<td>51</td>
</tr>
<tr>
<td>6.5 References</td>
<td>52</td>
</tr>
<tr>
<td>7 Production planning under energy supply uncertainty</td>
<td>54</td>
</tr>
<tr>
<td>7.1 Problem description</td>
<td>54</td>
</tr>
<tr>
<td>7.2 Mathematical formulation</td>
<td>54</td>
</tr>
<tr>
<td>7.3 Implementation</td>
<td>55</td>
</tr>
<tr>
<td>7.4 Input Data</td>
<td>57</td>
</tr>
<tr>
<td>7.5 Results</td>
<td>57</td>
</tr>
<tr>
<td>7.6 References</td>
<td>58</td>
</tr>
</tbody>
</table>

Bibliography
1 Introduction

Robust optimization is a modelling paradigm that offers solutions when uncertainty in the input data can be bounded within a well described region.

Robust optimization is different from stochastic optimization. While stochastic optimization usually aims to identify a solution whose expected value of the objective function is optimal with respect to the probability distribution of the uncertain data, robust optimization focuses on finding a solution that is feasible regardless of the realization of the uncertain values; hence the term robust. When uncertainty is associated with the objective of a model, robust optimization returns a solution that is optimal with respect to the worst case of all realizations of the uncertain quantities.

1.1 Uncertains and robust constraints

A model quantity or coefficient whose value is subject to uncertainty is called an uncertain.

Intuitively, an uncertain can be viewed as a unknown quantity that is not under our control, but is controlled by an opponent. The opponent makes his decision for the values of the uncertains after we have made our decision, i.e., after the solver has found an optimal value for the model variables. Hence, the values of the model variables must assume the worst case.

A model constraint that includes some uncertainty in the form of an uncertain coefficient or right hand side is called a robust constraint. A good way to visualize the concept of robust optimization is to consider a robust constraint as a two-phase expression. For the sake of argument, let us assume a less-than or equal-to constraint that contains some uncertain model quantities:

\[ a_1 x_1 + \ldots + a_l x_l + (a_k + u_k)x_k + \ldots + (a_n + u_n)x_n \leq b. \]

Here, all \( a_i \) coefficients are known. However, the values of coefficients of variables \( x_k \) to \( x_n \) are only known to some level of uncertainty, denoted by \( u_k \) to \( u_n \). As for traditional variables, we must define the values these uncertains can take.

The feasible region of the uncertains is called the uncertainty set, and is modelled by means of constraints on the uncertains. In the example above, we may know that the uncertain quantities are likely to be small and their norm might be bounded from above, hence we want a solution that is robust at a given confidence level. This implies a constraint on the uncertains of the form

\[ u^2_k + \ldots + u^2_n \leq r. \]

Having described the possible values of the uncertains, robust optimization aims to find a solution that is always feasible, thus the robust constraint is equivalent to the constraint

\[ a_1 x_1 + \ldots + a_l x_l + \max_{u^2_k + \ldots + u^2_n \leq r} [(a_k + u_k)x_k + \ldots + (a_n + u_n)x_n] \leq b. \]

A robust model may contain several robust constraints and several uncertainty sets. The solvability of a robust optimization problem depends on whether the robust constraints in the model can be transformed into a form that can be solved by the available mathematical programming solvers. The resulting transformed model that is solved is called the robust counterpart.
1.2 Types of robust constraints

A robust constraint can only contain uncertain either multiplied by a variable or at the right-hand side. The type of a robust constraint is defined by the type of uncertainty set (or sets) which define the uncertain parameters used in the constraint. We shall work with a small knapsack problem example and explore alternative formulations of the uncertainty set.

```plaintext
model Knapsack
uses "mmrobust" ! Load the robust library
parameters
NUM=5 ! Number of items
MAXVAL=100 ! Maximum value
MAXWEIGHT=80 ! Maximum weight
WTMAX=102 ! Max weight allowed for haul
end-parameters
declarations
Items=1..NUM ! Index range for items
VALUE: array(Items) of real ! Value of items
WEIGHT: array(Items) of real ! Weight of items
x: array(Items) of mpvar ! Decision variables
end-declarations
setrandseed(5);
forall(i in Items) do
VALUE(i):=50+random*MAXVAL
WEIGHT(i):=1+random*MAXWEIGHT
end-do
forall(i in Items) x(i) is_binary ! All x are 0/1
MaxVal:= sum(i in Items) VALUE(i)*x(i) ! Objective: maximize total value
WtMax:= sum(i in Items) WEIGHT(i)*x(i) <= WTMAX ! Weight restriction
maximize(MaxVal)
writeln("Solution:
 Objective: ", getobjval)
writeln("Item Weight Value")
forall(i in Items)
 writeln(i, " : ", getsol(x(i)), strfmt(WEIGHT(i),8,2), strfmt(VALUE(i),8,2))
end-model
```

This basic model without uncertainty solves to 256.601. Its execution results in the following output display.

```
Objective: 256.601
Item Weight Value
1: 0 74.37 118.04
2: 1 62.34 141.75
3: 1 27.74 114.85
4: 0 53.17 100.60
5: 0 74.02 65.82
```

1.2.1 Simple bounds on the uncertain coefficients

It is possible to impose simple box constraints on the uncertain values. This is often an immediate approach as it simply defines the range of values a coefficient can take, on a per-coefficient basis. Uncertains are introduced to the model as follows.

```plaintext
parameters
```

```plaintext
```
NUM=5  ! Number of items
MAXVAL=100  ! Maximum value
MAXWEIGHT=80  ! Maximum weight
WMAX=102  ! Max weight allowed for haul
WTPERCENT=0.3  ! Uncertainty as a percentage
end-parameters

declarations
Items=1..NUM  ! Index range for items
VALUE: array(Items) of real  ! Value of items
WEIGHT: array(Items) of real  ! Weight of items
x: array(Items) of mpvar
WeightUncertainty: array(Items) of uncertain  ! Uncertains representing
! deviation from weight
end-declarations

forall(i in Items) x(i) is_binary  ! All x are 0/1

setrandseed(5);
forall(i in Items) do
VALUE(i):=50+random*MAXVAL
WEIGHT(i):=1+random*MAXWEIGHT
end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i)  ! Objective: maximize total value
forall(i in Items) do
WeightUncertainty(i) <= WTPERCENT*WEIGHT(i)  ! Uncertainty is a percentage of
! the expected weight
WeightUncertainty(i) >= 0  ! and only expected to go up this time
end-do

WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertainty(i))*x(i) <= WTMAX  ! Weight restriction

maximize(MaxVal)

Note the uncertain type for the array WeightUncertainty. This type is used in Mosel to define a parameter that is not known a priori by the user and is not under the user’s control. The constraint WtMax that previously was of the type linctr now becomes a robustctr (the declaration of constraints is optional). The new model allows a 30% maximum deviation from the expected weight. Solving it we get 141.751.

Objective: 141.751
Item  Weight  Value
1:  0  74.37  118.04
2:  1  62.34  141.75
3:  0  27.74  114.85
4:  0  53.17  100.60
5:  0  74.02  65.82

The solution has shifted to pick the most valuable item only. Increasing the weight of the selected item by the allowed 30% it becomes 81.042. The optimal solution is robust, i.e., for any binary solution with a larger objective function there exists a vector of uncertain that would violate the robust constraint. The objective has decreased significantly because of uncertainty; the difference in objective function values between the original problem and the robust version is often referred to as the price of robustness [8504]. If a subset of the uncertainty set is used, for example with a smaller value of WTPERCENT, the effect of uncertainty decreases and we can obtain a (still robust) solution with a larger objective function. On the contrary, increasing WTPERCENT is equivalent to increasing uncertainty, which reduces the objective function value of the robust optimal solution.

Note the explicit non-negativity restrictions on the uncertain coefficients. There is no default lower bound imposed on uncertain, in contrast to traditional decision variables; this is to reflect
that uncertainty is not typically biased toward positive values.

Even if a robust modelling feature makes solving such a model simple, it is important to realize that when each individual uncertain is independently bounded by simple bounds only, the same model can be derived by modifying all coefficients in the way that restricts the constraint the most. In the knapsack example, as all variables must be nonnegative and all weights are also non-negative, this means adding the upper bound of the uncertain to the original coefficients.

```
declarations
  Items=1..NUM  ! Index range for items
  VALUE: array(Items) of real  ! Value of items
  WEIGHT: array(Items) of real  ! Weight of items
  x: array(Items) of mpvar  ! Decision variables
end-declarations

forall(i in Items) x(i) is_binary  ! All x are 0/1

setrandseed(5);
forall(i in Items) do
  VALUE(i):=50+random*MAXVAL
  WEIGHT(i):=1+random*MAXWEIGHT
end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i)  ! Objective: maximize total value
WtMax:= sum(i in Items) WEIGHT(i)*(1+WTPERCENT)*x(i) <= WTMAX  ! Weight restriction

maximize(MaxVal)
```

Indeed, the solution remains 141.751.

```
Objective: 141.751
Item Weight Value
1: 0 74.37 118.04
2: 1 62.34 141.75
3: 0 27.74 114.85
4: 0 53.17 100.60
5: 0 74.02 65.82
```

Realization of uncertain parameters Recall that an uncertain is an unknown quantity that is not under our control, but is rather chosen by a third party “opponent”. Solving a robust optimization problem amounts to solving a leader-follower game in which the user is the leader and the opponent obtains the variables from the user and chooses the uncertain parameters.

In some cases, after solving a robust optimization problem it is possible to obtain the value of the uncertain parameters that correspond to the robust solution. For this, it is necessary to have an optimal solution \( \bar{x} \) to the robust problem. Consider the robust constraint

\[
\sum_{i=1}^{n} a_i u_i x_i \leq b
\]

and assume that the uncertain \( u_i \) are within the ellipsoidal uncertainty set defined by

\[
\sum_{i=1}^{n} u_i^2 \leq 4
\]

The opponent implicitly chooses the value of the uncertain so that the left-hand side of the robust constraint is maximized (against the \( \leq \) constraint). Hence a realization of the uncertain for this robust constraint can be found by maximizing \( \sum_{i=1}^{n} a_i \bar{x}_i u_i \) subject to the quadratic
uncertainty constraint $\sum_{i=1} u_i^2 \leq 4$. It is possible to obtain the values of the uncertain parameters through the same function used to obtain the variable, getsol. The modalities for use of getsol are explained in the Mosel manual. Note that getsol can be used on uncertain only after the robust counterpart has been solved.

1.2.2 Linear constraints on the uncertainty sets

Bound constraints are only one example of the possible uncertainty sets we can model. Let us suppose that some or all of the uncertain can be aggregated and that we can write a single uncertain constraint with all of them. Let us refine the example from the simple bounds case, and require instead that the sum of the uncertain is at most 30% of the total weight.

```
parameters
NUM=5 ! Number of items
MAXVAL=100 ! Maximum value
MAXWEIGHT=80 ! Maximum weight
WTMAX=102 ! Max weight allowed for haul
WTPERCENT=0.3 ! Uncertainty as a percentage
end-parameters

declarations
Items=1..NUM ! Index range for items
VALUE: array(Items) of real ! Value of items
WEIGHT: array(Items) of real ! Weight of items
x: array(Items) of mpvar ! Decision variables
WeightUncertainty: array(Items) of uncertain ! Uncertains representing deviation from weight
end-declarations

forall(i in Items) x(i) is_binary ! All x are 0/1
setrandseed(5);
forall(i in Items) do
  VALUE(i):=50+random*MAXVAL
  WEIGHT(i):=1+random*MAXWEIGHT
end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i) ! Objective: maximize total value
forall(i in Items) do
  WeightUncertainty(i) >= 0
end-do

sum(i in Items) WeightUncertainty(i) <= WTPERCENT * sum(i in Items) WEIGHT(i)
WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertainty(i))*x(i) <= WTMAX ! Weight restriction

maximize(MaxVal)
```

While this sounds reasonable, because of the large right-hand side (30% of the total weight) the modified model is subject to a very conservative uncertainty set and will solve to a solution of all zeros.

<table>
<thead>
<tr>
<th>Objective: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
</tr>
<tr>
<td>1: 0</td>
</tr>
<tr>
<td>2: 0</td>
</tr>
<tr>
<td>3: 0</td>
</tr>
<tr>
<td>4: 0</td>
</tr>
<tr>
<td>5: 0</td>
</tr>
</tbody>
</table>
This behavior highlights how robust optimization works. When moving from the previous uncertainty set, with a small upper bound on the uncertain, to the new uncertainty set, we might in fact relax the uncertainty set: if \( \text{WTPERCENT} \times \sum (i \text{ in Items}) \ \text{WEIGHT}(i) \) is large enough, we have made the feasible space of the opponent significantly larger. This makes it possible that all uncertainty can be placed on a single item. Since each item can be too heavy even individually, the all zero solution becomes the only feasible one. This is a very important note to the behavior of robust optimization. When the uncertainty set is relaxed, the robust counterpart becomes more restrictive. Vice versa, to relax a robust constraint (and make it less conservative), its corresponding uncertainty set needs to be tightened.

A solution to the model in the example is then to restrict the feasible space of the uncertain values, that is, to tighten the uncertainty set by adding new uncertain constraints. In fact, these constraints would need to limit the amount of uncertainty per coefficient and will resemble the simple bound case in effect.

An uncertainty set defined by a set of linear constraints is often referred to as a polyhedral uncertainty set. It is important to note that polyhedral uncertainty sets have very important applications, but care must be taken when applying them as the example above has shown.

### 1.2.3 Ellipsoidal uncertainty sets

As we have seen with the previous example, the robust counterpart or opponent can take full advantage of the vertex solutions of a polyhedral uncertainty set making the model more conservative than possibly intended. This could be countered by either restricting such constraints or by refining the set, for example adding new linear constraints. It is easy to see that this is an approximation only to the underlying problem, and may be impractical in larger dimensions.

If the uncertainty set is known to be described by a quadratic constraint, for instance we know that the norm of the vector of the uncertain is bounded from above, then we can use ellipsoidal uncertainty sets. Another use for ellipsoidal uncertainty comes when the uncertainty set can be described by a confidence ellipsoid that is defined by a mean vector \( \mu \), a covariance matrix \( \Sigma \), and a level of confidence \( \alpha \). In this case, the uncertainty set is of the form \( (u - \mu)^T \Sigma (u - \mu) \leq \alpha \). In the following model, we set \( \alpha \) to \( \text{WTPERCENT} \times \sum (i \text{ in Items}) \ \text{WEIGHT}(i) \).

```plaintext
parameters
NUM=5 ! Number of items
MAXVAL=100 ! Maximum value
MAXWEIGHT=80 ! Maximum weight
WTMAX=102 ! Max weight allowed for haul
WTPERCENT=0.3 ! Uncertainty as a percentage
end-parameters

declarations
Items=1..NUM ! Index range for items
VALUE: array(Items) of real ! Value of items
WEIGHT: array(Items) of real ! Weight of items
x: array(Items) of mpvar ! Decision variables
WeightUncertainty: array(Items) of uncertain ! Uncertains representing deviation from weight
end-declarations

forall(i in Items) x(i) is_binary ! All x are 0/1
setrandseed(5);
forall(i in Items) do
VALUE(i):=50+random*MAXVAL
WEIGHT(i):=1+random*MAXWEIGHT
end-do
```
Robust Optimization with Xpress

MaxVal:= sum(i in Items) VALUE(i)*x(i) ! Objective: maximize total value

sum(i in Items) WeightUncertainty(i)^2 <= WTPERCENT * sum(i in Items) WEIGHT(i)

WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertainty(i))*x(i) <= WTMAX ! Weight restriction

maximize(MaxVal)

This model gives the desired solution as shown below. Note the value of the uncertain as obtained after maximization.

<table>
<thead>
<tr>
<th>Objective: 215.445</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1:</td>
</tr>
<tr>
<td>2:</td>
</tr>
<tr>
<td>3:</td>
</tr>
<tr>
<td>4:</td>
</tr>
<tr>
<td>5:</td>
</tr>
</tbody>
</table>

Note that polyhedral uncertainty creates a robust counterpart that is of the same class as the original problem, as it amounts to adding a set of linear constraints. This is not the case with ellipsoidal uncertainty, which creates a robust counterpart that contains one second-order conic constraint for each robust constraint subject to ellipsoidal uncertainty, and hence may change the nature of the problem: if the original problem is a Linear Programming (LP) problem, the robust counterpart is a Second Order Conic Programming (SOCP) problem; if the original problem is a Mixed Integer Linear Programming (MILP) problem, the robust counterpart is a MISOCP. While solvers for MISOCP are now more powerful, a MISOCP problem is generally more difficult to solve than a MILP.

We conclude this section with a note about the use of getsol on uncertain: the last column of the output table shows the value of the uncertain WeightUncertainty for this particular solution. It is easy to notice that the nonzero uncertain is in accordance with the nonzero of the solution. This squarely aligns with viewing robust optimization as a leader-follower game: although the robust counterpart is solved just once, it emulates the dynamics of the opponent, who chose to increase the uncertain to make the robust constraint as tight as possible, and this can only be done by increasing WeightUncertainty(3) and WeightUncertainty(4).

1.2.4 Equality constraints and uncertain values with no uncertain constraints

At this point it is worth mentioning some pitfalls of robust optimization that one needs to be careful about. First, equality robust constraints are very restrictive in robust optimization.

declarations
x, y, z : mpvar
e : uncertain
der-declarations
e <= 1
e >= 0
x + y + e*z = 10
maximize(z)

In the solution to this problem, z will always be zero. The reason is simple: whenever z is equal to a non-zero value, if the value of variables x and y are fixed, any change in the value of the uncertain e will make the equality constraint infeasible. In general, the effect of uncertainty in
equality robust constraints is to project the feasible space of the original problem to a smaller dimensional one.

Second, it is always beneficial to make sure that the uncertainty set is bounded from both below and above. The small example below shows the same effect as the case of the equality constraint.

```plaintext
declarations
x, y, z : mpvar
e : uncertain
end-declarations
x + y + e*z <= 10
maximize(z)
```

Here z will again solve to zero, as for any nonzero value of z a large enough e would make the constraint infeasible. Often, such missing bounds lead to an infeasible robust optimization problem.

### 1.2.5 Mixing uncertainty sets and types

Due to the theory of robust optimization, there are limitations on how uncertains can be used in uncertainty sets and robust constraints.

The most prominent restriction prohibits, by default, the use of the same uncertain values in different robust constraints. This restriction applies also to a model where two uncertains are used independently in two robust constraints if these two uncertains are contained in a constraint defining an uncertainty set. This restriction is enforced by virtue of how the corresponding robust counterparts are created, which allows the opponent to select its strategy, i.e. the value of an uncertain, on a per-constraint basis. Therefore, if the same uncertain appears in two robust constraints its value is allowed to take different values in each robust constraint. However, it is often convenient to make use of this behavior if the same uncertain set description is used for multiple robust constraints.

Consider the following example.

```plaintext
declarations
x, y: mpvar
e, f : uncertain
end-declarations
x*e <= 1
y*f <= 1
e >= 0
f >= 0
e+f <= 1
maximize(x+y)
```

The example will not solve using default settings, as the uncertain constraint e + f ≤ 1 connects the two robust constraints to the same uncertainty set, thereby they overlap. Setting ROBUST_UNCERTAIN_OVERLAP to true will allow for the problem to be solved.

```plaintext
declarations
x, y: mpvar
e, f : uncertain
end-declarations
```
Robust Optimization with Xpress

RobC1 := x*e <= 1
RobC2 := y*f <= 1

e >= 0
f >= 0
e+f <= 1

setparam("XPRS_PROBNAME","h")
setparam("ROBUST_UNCERTAIN_OVERLAP", true)

maximize(x+y)

However, the solution returned is x=1=y, which can easily be seen as non-optimal if e + f ≤ 1 is satisfied: the solution is assuming that e=1 to satisfy the first robust constraint and that f=1 to satisfy the second one. Allowing the overlap will result in the opponent being able to optimize its decisions independently for the two robust constraints. This is the reason why this mode is not allowed by default; although often this is exactly what the modeller wants, setting ROBUST_UNCERTAIN_OVERLAP to true allows for reusing the same uncertainty set definition multiple times without the need to repeat the uncertain constraints. However, care must be taken as the uncertain values do not take a defined value upon solving the problem and it might be different for different robust constraints.

The retrieval of uncertain values that overlap across different constraints adds a further twist: the two constraints xe ≤ 1 and yf ≤ 1 admit different combinations of the uncertain (e, f) as can be seen from the output.

\[
x = 1 \quad y = 1 \\
\text{constraint RobC1: } x*e <= 1, \text{ uncertains (e,f) = (1,0)} \\
\text{constraint RobC2: } y*f <= 1, \text{ uncertains (e,f) = (0,1)}
\]

The first constraint can only be made tighter by increasing uncertain e, while the second, whose only term is multiplied by f, can be made tight by setting the latter to its maximum value. For this reason, overlapping uncertain values can only be retrieved by calling getsol passing one of the robust constraints where they appear. The value of that uncertain will then be equal to what the opponent would choose if that were the only robust constraint.

1.2.6 Cardinality restrictions for uncertain values

Cardinality restrictions allow for limiting the number of uncertain values that are different from zero. In other words, a cardinality constraint limits the number of uncertain values that are non-zero or, more generally, not at their nominal value. Due to the way in which the robust counterpart is created, uncertain values that appear in a cardinality restriction should always have reasonable lower and upper bounds.

In the next example we show how to use uncertain values in multiple robust constraints (i.e. in uncertain values that overlap). The example is a small knapsack type problem, where only 1 of the 3 items is allowed to differ in weight from zero.

declarations
x,y,z: mpvar
ex, ey, ez: uncertain
end-declarations

x is_binary ; y is_binary ; z is_binary

20*x + 10*y + 5*z >= 20
Robust Optimization with Xpress

\[ \text{ex} \geq 0 \; ; \; \text{ex} \leq 1 \]
\[ \text{ey} \geq 0 \; ; \; \text{ey} \leq 1 \]
\[ \text{ez} \geq 0 \; ; \; \text{ez} \leq 1 \]
\[ \text{cardinality}({\text{ex,ey,ez}},1) \! \text{ Allow only one of the uncertain s to be nonzero} \]
\[ \text{setparam("ROBUST_UNCERTAIN_OVERLAP"},true) \! \text{ Allow overlaps} \]
\[ 10*(x-x*\text{ex}) + 10*(y-y*\text{ey}) + 10*(z-z*\text{ez}) \geq 10 \]
\[ 10*(x-x*\text{ex}) + 10*(y-y*\text{ey}) + 10*(z-z*\text{ez}) \leq 20 \]

maximize(x+y+z)

The solution returned is \(x=y=1\) and \(z=0\). It is important to emphasize that, as for the other constraints, the cardinality restriction is taken into consideration on a per robust constraint basis, and the model does not solve unless the ROBUST_UNCERTAIN_OVERLAP parameter is set to true.

### 1.2.7 Using historical data: scenarios

Scenarios are an efficient way of building robust optimization problems using historical realizations of the uncertain s. Suppose we have a vector \(u\) of uncertain s. We do not know an uncertainty set that would fit our model, but for the uncertain vector we have a database of records containing the value of each uncertain for each month of the past 20 years. Therefore we would like a model where one or more constraints are satisfied for each record of the uncertain vector in our database. Although we may not have a good approximation of the uncertainty set, robustness against historical data might be sufficient.

Declaring a scenario is equivalent to manually adding all the corresponding constraints for all realizations to the problem; this capability is made available to make sure that the scenario data are efficiently handled by the solver and to avoid the creation of an overly large problem.

Scenarios can be driven by the historical data, as shown in the next extension of the knapsack problem analyzed before where a number of historical measurements have been taken.

```plaintext
parameters
NUM=5 ! Number of items
MAXVAL=100 ! Maximum value
MAXWEIGHT=80 ! Maximum weight
WTMAX=102 ! Max weight allowed for haul
UNCERTAINTY_LEVEL=0.3 ! How much we are uncertain about the weight
HISTORIC_PERIODS=100 ! Number of scenarios
end-parameters
declarations
Items=1..NUM ! Index range for items
VALUE: array(Items) of real ! Value of items
WEIGHT: array(Items) of real ! Weight of items
UncertainWeight:array(Items) of uncertain
x: array(Items) of mpvar ! 1 if we take item i; 0 otherwise
historical_weights: array(range, set of uncertain) of real
end-declarations
forall(i in Items) x(i) is_binary ! All x are 0/1
setrandseed(5);
forall(i in Items) do
  VALUE(i):=50+random*MAXVAL
  WEIGHT(i):=1+random*MAXWEIGHT
end do
MaxVal:= sum(i in Items) VALUE(i)*x(i) ! Objective: maximize total value
WtMax:= sum(i in Items) (WEIGHT(i)+UncertainWeight(i))*x(i) <= WTMAX
```

---

Introduction Fair Isaac Corporation Confidential and Proprietary Information
It is beneficial to examine the deterministic equivalent to a scenario based robust model. Consider the following simple scenario based model.

declarations
  x, y : mpvar
  e, f : uncertain
end-declarations

! Load historical data for e and f
HISTDATA(1, e) := 1
HISTDATA(1, f) := 2
HISTDATA(2, e) := 3
HISTDATA(2, f) := 1
HISTDATA(3, e) := 3
HISTDATA(3, f) := 2

e*x + f * y <= 1

! Generate a solution that would be feasible for ALL historic realizations
scenario(HISTDATA)
maximize(x+y)

Its deterministic equivalent is:

declarations
  x, y : mpvar
end-declarations

Scenario1 := 1 * x + 2 * y <= 1
Scenario2 := 3 * x + 1 * y <= 1
Scenario3 := 3 * x + 2 * y <= 1
maximize(x+y)

1.2.8 Uncertainty in the objective function

When uncertain coefficients are introduced in the objective, the solution to the robust counterpart reflects the most conservative objective. More specifically, in a minimization problem where the objective function is \( f(x; u) \), where \( x \) is a vector of variables and \( u \) is a vector of uncertain, the optimal solution is one that minimizes the function \( g(x) = \max_u f(x; u) \). This is equivalent to converting the objective to the robust constraint \( z \geq f(x; u) \), where a new auxiliary variable \( z \) is used to represent the objective.

1.3 Nominal values: centered and uncentered uncertainty

The nominal value of an uncertain represents its default value, a real-valued number which is the expected or non-robust version of the uncertain. In the examples so far, all uncertainty has been treated so that the nominal value of each uncertain is zero, and in effect the uncertain has been
treated as an error term. The default value or center of an uncertain can be shifted by specifying a nominal value that is different from zero. Using nominal values has the advantage that they allow one to work on both the robust optimization model and on a deterministic one obtained by fixing the uncertains (to their nominal values) in the same model.

Before detailing the use cases and rules when working with nominal values of uncertains, let us give an example on centered and uncentered uncertains.

I. Using zero centered uncertainty, where an uncertain is modelled as a zero centered error, \( i.e. \), uncertainty with zero nominal value:

\[(5 + \text{uncertain}) x \leq 1\]

is expressed as:

```plaintext
declarations
  x : mpvar
  u : uncertain
end-declarations
(5+u)*x <= 1
```

II. Uncertainty as coefficient, where the uncertain is modelled as a coefficient, \( i.e. \) an uncertain coefficient with 5 as its nominal value:

\[(5 + \text{uncertain}) x \leq 1\]

can also be expressed as:

```plaintext
declarations
  x : mpvar
  u : uncertain
end-declarations
u:= 5  ! Setting the nominal value
u*x <= 1
```

As the example above shows, a nominal value is set using the assignment operator `:=`. It is important to distinguish this from the equal operator `=` which would fix the value of the uncertain to the given value instead.

### 1.3.1 The price of robustness

Either when using zero-centered uncertains or when setting up nominal values, it is possible to solve the model with the uncertainty removed, \( i.e. \) all uncertain coefficients being fixed to their nominal values. This makes it possible to calculate the cost of having uncertainty in the model (often referred to as the *price of robustness*), as well as to check whether the model is feasible without the added uncertainty. The next example shows how to solve with all uncertains fixed to their nominal values.

```plaintext
declarations
  x : mpvar
  u : uncertain
end-declarations
0 <= u
```
The output of the model is:

Objective at nominal values: 1
Robust objective: 0.25
Price of robustness: 0.75

When no nominal value is defined for an uncertain, the default nominal value of zero is used.

1.3.2 Working with nominal values

The workings of nominal values are defined by two simple rules.

**Rule 1: setting a nominal value is shifting the domain of the uncertain.**

Setting a nominal value for an uncertain to \( u := a \) replaces all later occurrences of \( u \) with \( u + a \).

The following simple model solves to \( x = 0.5 \):

```plaintext
declarations
  x : mpvar
  u : uncertain
end-declarations

0 <= u
  u <= 1
(1+u)*x <= 1
maximize(x)
```

as the worst case is when \( u \) takes its upper bound of 1.

Changing the nominal value of \( u \) to say 2, \( u \) is replaced by \( u + 2 \) and the model solves to 0.25 instead.

```plaintext
declarations
  x : mpvar
  u : uncertain
end-declarations

0 <= u
  u <= 1
u := 2 ! Shifts the center of the uncertain
(1+u)*x <= 1
maximize(x)
```
The uncertain \( u \) still takes its upper bound of 1, but the constraint loaded now reads 
\[(1 + u + 2)x \leq 1 \] with \( u \) taking on its upper bound of 1.

**Rule 2:** a changed nominal value only affects robust and uncertainty set constraints that are defined after it has been assigned.

This behavior is best understood when put into analogy with how real-valued parameters behave. Looking at the example of rule 1, this behavior is already observed: changing the nominal value has only affected the robust constraint but not the bounds declared before this point.

```plaintext
declarations
x : mpvar
r : real
end-declarations

r := 3
C1 := (1+r)*x <= 1
r := 5
maximize(x)
```

The model solves to \( x = 0.25 \), that is the actual value of \( r \) at the time of declaring the constraint \( C1 \) has been used (3), and redefining it later had no effect on the constraint. This behavior is reproduced when working with the nominal values of uncertain.

```plaintext
declarations
y : mpvar
u : uncertain
end-declarations

u := 3
(1+u)*y <= 1
u := 5
maximize(XPRS_NOMINAL, y)
```

The nominal version of the model where the uncertain coefficients are used solves (notice the nominal argument to maximize) to the same as when real valued coefficients are used: \( y = 0.25 \).

These rules offer significant flexibility on how to use the nominal values for uncertain, but care must be taken and the effects of rule 2 should always be kept in mind.

### 1.3.3 Using nominal values to shift the uncertainty set

Nominal values can be used to shift an uncertain coefficient while keeping its effect around the nominal values the same. Consider the following example:

```plaintext
declarations
x,y : mpvar
e,f : uncertain
end-declarations

e^2 + f^2 <= 2 ! An ellipsoidal uncertainty constraint
e*x + f*y <= 1
maximize(x + y)
```
The solution to the model is \( x=0.5, y=0.5 \). The values of uncertain \( e \) and \( f \) can take their value from a ball of radius \( \sqrt{2} \). When nominal values are added, the model becomes:

```plaintext
declarations
x, y : mpvar
e, f : uncertain
end-declarations
e^2 + f^2 <= 2
e := 1
f := 1
e*x + f*y <= 1
maximize(x + y)
```

The solution moves to \( x=0.25, y=0.25 \). To explore the reasons for this change, let us substitute the effect of the nominal values directly into the problem:

```plaintext
declarations
x, y : mpvar
e, f : uncertain
end-declarations
e^2 + f^2 <= 2
(e+1)*x + (f+1)*y <= 1
maximize(x + y)
```

The range of uncertain has remained the same, but in the robust constraint the values around which they are centered shift accordingly to the nominal values.

### 1.3.4 Using nominal valued uncertain as coefficients

Uncertains can be assigned a nominal value before the first time they are used, however they may behave in a way that is not expected. Consider three versions of the same model:

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>No nominal values</td>
<td>Nominal values are set to</td>
<td>Nominal values are set to</td>
</tr>
<tr>
<td>are set</td>
<td>1</td>
<td>different values</td>
</tr>
<tr>
<td>declarations</td>
<td>declarations</td>
<td>declarations</td>
</tr>
<tr>
<td>x, y : mpvar</td>
<td>x, y : mpvar</td>
<td>x, y : mpvar</td>
</tr>
<tr>
<td>e, f : uncertain</td>
<td>e, f : uncertain</td>
<td>e, f : uncertain</td>
</tr>
<tr>
<td>e + f &lt;= 2</td>
<td>e := 1</td>
<td>e := 2</td>
</tr>
<tr>
<td>e<em>x + f</em>y &lt;= 1</td>
<td>f := 1</td>
<td>e + f &lt;= 2</td>
</tr>
<tr>
<td>maximize(x + y)</td>
<td>e<em>x + f</em>y &lt;= 1</td>
<td>e<em>x + f</em>y &lt;= 1</td>
</tr>
<tr>
<td></td>
<td>maximize(x + y)</td>
<td>maximize(x + y)</td>
</tr>
</tbody>
</table>

All three versions of the model solve to the same solution of \( x = y = 0.5 \); The reasons behind this may not be immediately obvious but become clear if we write out the effect of rule 1 on, say, the 3rd case:
As the shifts are also applied to the uncertain constraints (including bounds), in the example they cancel each other out: the linear robust constraints and the corresponding error constraints have been shifted together.

However, there is a very real difference should the model be solved using its nominal values. To demonstrate this, let us explicitly write out the nominal equivalent for the 3 cases:

<table>
<thead>
<tr>
<th>The nominal equivalent of case 1:</th>
<th>The nominal equivalent of case 2:</th>
<th>The nominal equivalent of case 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>declarations</td>
<td>declarations</td>
<td>declarations</td>
</tr>
<tr>
<td>(x,y : \text{mpvar} )</td>
<td>(x,y : \text{mpvar} )</td>
<td>(x,y : \text{mpvar} )</td>
</tr>
<tr>
<td>(e,f : \text{uncertain} )</td>
<td>(e,f : \text{uncertain} )</td>
<td>(e,f : \text{uncertain} )</td>
</tr>
<tr>
<td>end-declarations</td>
<td>end-declarations</td>
<td>end-declarations</td>
</tr>
<tr>
<td>(e := 2)</td>
<td>(e+2 + f+5 \leq 2)</td>
<td>(e+f \leq -5)</td>
</tr>
<tr>
<td>(f := 5)</td>
<td>((e+2)*x + (f+5)*y \leq 1)</td>
<td>((e+2)*x + (f+5)*y \leq 1)</td>
</tr>
<tr>
<td>(e + f \leq 2)</td>
<td>maximize(x + y)</td>
<td>maximize(x + y)</td>
</tr>
<tr>
<td>(e<em>x + f</em>y \leq 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maximize(x + y)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

is unbounded.

solves to \(x=1\) and \(y=0\).

solves to \(x=0.5\) and \(y=0\).

1.4 Examples of robust models

In the remainder of this whitepaper we provide several full-blown examples of robust optimization applied to real-world problems. The model and data files associated with these examples are available in the FICO® Xpress Optimization distribution (subdirectory `examples/robust/Mosel`).
2 Robust shortest path

2.1 Problem description

Every day you are driving from your home to your work location. There are a number of alternative routes that you might use. You may think of these routes as a set of lines that connect nodes (=intersection points of routes). For every line (edge) we know the required travel time for getting from one end point to the other. What we don’t know is the occurrence of roadworks or similar incidents that slow down traffic and hence cause delays.

Figure 1 shows an example of a road network where the journey start and end points are marked as ‘Source’ and ‘Sink’ respectively. The label tuples (L,D) on the edges indicate the length (travel time) of an edge and the maximum delay D that may result from roadwork on this edge. In this data instance, the times in both directions for traveling along an edge are the same and the maximum delay in both senses is also the same, but in the general case the values might actually be different.

2.2 Mathematical formulation

2.2.1 Shortest path problem

The problem of finding the least cost route through a network is known as the shortest path problem. We can state this problem via binary variables use\textsubscript{a} associated with each arc a where use\textsubscript{a} = 1 if arc a is selected and 0 otherwise.

For every node in the network except the source and sink nodes we establish a flow balance constraint: the sum of incoming travel is the same as the outgoing travel. The source node only has a single outgoing used arc, the sink node has a single incoming used arc. In the following, we define an arc a via its origin ARC\textsubscript{a,1} and its destination node ARC\textsubscript{a,2}.
\[
\min \sum_{a \in \text{Arcs}} LEN_a \cdot use_a \\
\text{s.t.} \quad \sum_{a \in \text{Arcs} \mid \text{ARC}_a = \text{Source}} use_a = 1, \quad \sum_{a \in \text{Arcs} \mid \text{ARC}_a = \text{Sink}} use_a = 1 \\
\sum_{a \in \text{Arcs} \mid \text{ARC}_a = \text{Source}} use_a = 0, \quad \sum_{a \in \text{Arcs} \mid \text{ARC}_a = \text{Sink}} use_a = 0 \\
\forall n \in \text{Nodes} - \{\text{Source, Sink}\} : \quad \sum_{a \in \text{Arcs} \mid \text{ARC}_a = n} use_a = \sum_{a \in \text{Arcs} \mid \text{ARC}_a = n} use_a \\
\forall a \in \text{Arcs} : use_a \in \{0, 1\}
\]

In the model formulation above, we can add the delays caused by roadworks as an additional coefficient to the objective function so that they are taken into account by the shortest path calculation:

\[
\min \sum_{a \in \text{Arcs}} (LEN_a + \text{MAXDELAY}_a) \cdot use_a
\]

However, this means that we assume

(a) all delays take their maximum value and
(b) all roads incur delays.

These assumptions yield an overly conservative prediction. This case provides an upper bound on the total travel time estimate but it can certainly be considered as fairly unlikely to happen.

2.2.2 Robust optimization problem

The problem that we really wanted to state is

(a) delays take values up to their specified maximum value and
(b) up to \( N \) roads incur delays.

This means that the value of the delay per arc is not fixed, but uncertain. If \( delay_a \) denotes the uncertain duration of the delay associates with arc \( a \in \text{Arcs} \) we can state the cardinality constrained uncertainty set as follows:

\[
U = \{ \text{delay} : ||\text{delay}_a : \text{delay}_a > 0|| \leq N, 0 \leq \text{delay}_a \leq \text{MAXDELAY}_a \forall a \in \text{Arcs} \}
\]

The resulting robust optimization problem then has the following form.

\[
\min \sum_{a \in \text{Arcs}} (LEN_a + delay_a) \cdot use_a \\
\text{s.t.} \quad \sum_{a \in \text{Arcs} \mid \text{ARC}_a = \text{Source}} use_a = 1, \quad \sum_{a \in \text{Arcs} \mid \text{ARC}_a = \text{Sink}} use_a = 1
\]
Robust Optimization with Xpress

\[ \sum_{a \in \text{Arcs} | \text{ARC}_a = \text{Source}} \text{use}_a = 0, \quad \sum_{a \in \text{Arcs} | \text{ARC}_a = \text{Sink}} \text{use}_a = 0 \]

\[ \forall n \in \text{Nodes} - \{\text{Source, Sink}\} : \sum_{a \in \text{Arcs} | \text{ARC}_a = n} \text{use}_a = \sum_{a \in \text{Arcs} | \text{ARC}_a = n} \text{use}_a \]

\[ \forall a \in \text{Arcs} : \text{use}_a \in \{0, 1\} \]

\[ \text{delay} \in \mathcal{U} = \{\text{delay} : |\{\text{delay}_a : \text{delay}_a > 0\}| \leq N, 0 \leq \text{delay}_a \leq \text{MAXDELAY}_a \forall a \in \text{Arcs}\} \]

### 2.3 Implementation

The following Mosel model `roadworks.mos` implements the robust optimization model from the previous section. Notice the use of cardinality on the uncertain delays to limit the total number of occurrences of roadworks to at most \( N_{\text{WORK}} \) (the parameter for \( N \)) with a default value of 2. The uncertain delays that are used in such a cardinality constraint need to have explicit lower and upper bounds set. The solution reporting displays the selected arcs and the values of the uncertain delays for which a nonzero delay is chosen by the opponent.

```mosel
model "road network"
uses "mmxprs", "mmrobust"

parameters
DATAFILE="roads_9.dat"
NWORK = 2 ! Number of roadworks
end-parameters

declarations
Nodes: range ! Set of nodes
ARC: array(Arcs:range,1..2) of integer ! Arc origins/destinations
LEN,MAXDELAY: array(Arcs) of real ! Length and max delay per arc
Source,Sink: integer ! Source and sink node numbers
use: array(Arcs) of mpvar ! 1 iff arc is used
TotalLength: robctr ! Objective function
delay: array(Arcs) of uncertain ! Uncertain delay
end-declarations

!**** Input datafile ****
initializations from DATAFILE
Nodes Arcs Source Sink ARC
[LEN,MAXDELAY] as "ArcData"
end-initializations

!**** Robust problem formulation ****
forall(a in Arcs) use(a) is_binary

! Sink and source of flow
sum(a in Arcs | ARC(a,1)=Source) use(a)=1
sum(a in Arcs | ARC(a,2)=Sink) use(a)=1
sum(a in Arcs | ARC(a,2)=Source) use(a)=0
sum(a in Arcs | ARC(a,1)=Sink) use(a)=0

! Flow balance in intermediate nodes
forall(n in Nodes-{Source,Sink})
sum(a in Arcs | ARC(a,2)=n) use(a) = sum(a in Arcs | ARC(a,1)=n) use(a)

! Random construction work on NWORK arcs
forall(a in Arcs) delay(a) <= MAXDELAY(a)
forall(a in Arcs) delay(a) >= 0
cardinality(union(a in Arcs) {delay(a)}, NWORK)

! Shortest path length
TotalLength:= sum(a in Arcs) (LEN(a)+delay(a))*use(a)
```

Robust shortest path
2.4 Results

Figure 2 shows the path through the network defined by the robust solution; it has a total length of 21. The path it follows, i.e., Source → 2 → 5 → 6 → 7 → Sink, has a nominal length of 14, but if roads 2–5 and 5–6 have construction work we must add a total delay of 7. Given that these two roads have maximum delay (together with the combination of roads 2–5 and 6–7), any other combination of at most two roads with construction will result in a total duration of 21 minutes or less. The delay that we must add to the nominal length of the path is given by the sum of the solution values of the uncertain delays associated with every arc in the path. The solution values of the uncertain delays can be interpreted as providing an answer to the question ‘which is the worst possible choice of delays for the selected path?’ The resulting values of the uncertain delays place the delay on the two arcs (2,5) and (5,6) for a total delay of 7.

Delay on (2,5): 4
Delay on (5,6): 3

![Network diagram with nodes and arcs labeled, showing the path Source → 2 → 5 → 6 → 7 → Sink, with path length 21.](image)

**Figure 2:** Robust solution with 2 delays
Consider now the solution in Figure 3, obtained by ignoring the delays caused by road work. The optimal path in this case is Source → 3 → 8 → Sink, with a total duration of 11. However, this is a very optimistic solution as in the worst case the roads Sink–3 and 3–8 may have construction work, in which case the travel time increases by 13 minutes to 24, clearly worse than the robust solution of 21 minutes.

Finally, let us take a look at the solution obtained by setting all arc costs to their maximum value, i.e., a shortest path computed on a graph where all edges are assumed to have their maximum delay. As shown in Figure 4, the solution is the path Source → 2 → 4 → 7 → Sink, and has a duration of 27 minutes. Under the assumption that at most two roads have construction work, this travel time is meaningless: in fact, the nominal travel time is 13 minutes and the worst-case travel time is obtained with road work on 2–4 and 4–7, which add 10 minutes to the solution and hence yield a total travel time of 23, still worse than the robust shortest path of 21 depicted above.

We recap the above results in Table 1, which shows the worst-case travel times for the paths determined by the three approaches (robust for N=2, shortest path with nominal values, shortest path where all roads have maximum delay) in five cases: no road work, road work in N = 1, 2, or 3 edges of the graph, and road work everywhere, i.e. N is infinite.

<table>
<thead>
<tr>
<th>Table 1: Path costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>Robust</td>
</tr>
<tr>
<td>Nominal</td>
</tr>
<tr>
<td>All delays</td>
</tr>
</tbody>
</table>

The values in this table can be obtained by fixing the arc selection in the robust problem to a given path and re-solving it for different values of N (i.e. we determine the worst-case delay for a given path that satisfies all constraints of the robust problem).

Note that, under the assumption we have made at the beginning of this section, the only solution
Figure 4: Maximum-delay solution

that truly solves the problem with minimum worst-case travel time is the one that was found using the uncertain to model our knowledge, albeit limited, of the road works in this network.
3  Production planning under demand uncertainty

3.1 Problem description

As the manager of a plant producing drinking glasses you wish to plan the production for the next 12 weeks. Your plant produces six different types (V1 to V6) in batches of 1000 glasses; batches may be incomplete (fewer than 1000 glasses). For every glass type you have got demand estimate for the 12 coming weeks. The initial stock and as well as the required final stock level (in thousands) are known. Per batch of every glass type, the production and storage costs in € are given, together with the required working time for workers and machines (in hours), and the required storage space (measured in numbers of trays).

3.2 Mathematical formulation

3.2.1 Multi-period, multi-item production planning problem

Let \( PRODS \) be the set of products (glass types) and \( WEEKS = \{1, \ldots, NT\} \) the set of time periods. We write \( DEM_{pt} \) for the demand for product \( p \) in time period \( t \). We also have \( CPROD_p \) and \( CSTOCK_p \) the production and storage cost for glass type \( p \). This cost is identical for all time periods, but it would be easy to model a different cost per time period by adding an index for the time period.

\( TIMEW_p \) and \( TIMEM_p \) denote the worker and machine times respectively required per unit of product \( p \), and correspondingly, \( SPACE_p \) the storage area. The initial stock \( ISTOCK_p \) is given, as is the desired final stock level \( FSTOCK_p \) per product (see data in Table 2). We write \( CAPW \) and \( CAPM \) for the capacities of workers and machines respectively, and \( CAPS \) for the capacity of the storage area.

To solve this problem, we need variables \( produce_{pt} \) to represent the production of glass type \( p \) in time period \( t \). The variables corresponding to the stock level of every product \( p \) at the end of period \( t \) are called \( store_{pt} \). By convention, the initial stock level \( ISTOCK_p \) may be considered as the stock level at the end of time period 0 and we use the notation \( store_{p0} \) to simplify the formulation of the stock balance constraints:

\[
\forall p \in PRODS, t \in WEEKS : store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt}
\]

These stock balance constraints state that the quantity \( store_{pt} \) of product that is held in stock at the end of a time period \( t \) equals the stock level \( store_{p,t-1} \) at the end of the preceding period plus the production \( produce_{pt} \) of the time period \( t \) minus the demand \( DEM_{pt} \) of this time period.

We wish to have a certain amount of product in stock at the end of the planning period to avoid that stocks run down to zero at the end of the planning horizon. These constraints on the final stock levels are expressed by the constraints:

\[
\forall p \in PRODS : \text{store}_{p,NT} \geq FSTOCK_p
\]

We now formulate the various capacity constraints for every time period. The following constraints guarantee that the capacity limits on manpower, machine time, and storage space are kept:

\[
\forall t \in WEEKS : \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW
\]

\[
\forall t \in WEEKS : \sum_{p \in PRODS} TIMEM_p \cdot produce_{pt} \leq CAPM
\]
∀t ∈ WEEKS : \[ \sum_{p \in PRODS} SPACE_p \cdot produce_{pt} \leq CAPS \]

The cost function that is to be minimized is the sum of production and storage costs for all products and time periods.

\[
\min \sum_{p \in PRODS} \sum_{t \in WEEKS} (CPROD_p \cdot produce_{pt} + CSTORE_p \cdot store_{pt})
\]

We obtain the complete mathematical model by the non-negativity constraints for the production variables and for the stored quantities to the constraints described above.

\[
\min \sum_{p \in PRODS} \sum_{t \in WEEKS} (CPROD_p \cdot produce_{pt} + CSTORE_p \cdot store_{pt})
\]

s.t.

∀p ∈ PRODS, t ∈ WEEKS : \[ store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt} \]

∀p ∈ PRODS : \[ store_{p,NT} \geq FSTOCK_p \]

∀t ∈ WEEKS : \[ \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW \]

∀t ∈ WEEKS : \[ \sum_{p \in PRODS} TIMEM_p \cdot produce_{pt} \leq CAPM \]

∀t ∈ WEEKS : \[ \sum_{p \in PRODS} SPACE_p \cdot produce_{pt} \leq CAPS \]

∀p ∈ PRODS, t ∈ WEEKS : \[ produce_{p,t} \geq 0 \]

∀p ∈ PRODS, t ∈ WEEKS : \[ store_{p,t} \geq 0 \]

### 3.2.2 Robust optimization problem

Various assumptions behind the mathematical model that we have stated above might be questioned:

1. **Constant resource capacity**: availability of personnel most likely will not be constant over time (subject to holidays, training, sick leave etc.) and there is also a risk of scheduled (maintenance) or unscheduled (breakdown) machine outages.

2. **Exact demand quantities** are known: demand forecasts typically are estimates, most often resulting from an analysis of historical values.

**Workers’ absence**

The case of machine outage (formulated as k contingencies) is studied in Sections 7 and 6 of this whitepaper. Let us therefore here take a look at how we might capture the uncertainty in the availability of personnel.

Let \( ABSENCE_t \) be a maximum limit on the absence hours per time period \( t \) (the actual absence will take at most this value), and we also know by experience what is the average absence over a longer period of time (expressed as a percentage \( A \) of the total worker hours). We introduce uncertainties \( absent_t \) to represent the actual absence hours per time period. The worker capacity constraints from the original model are then replaced by the following:

\[
\forall t \in WEEKS : \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW - absent_t
\]

\[ absent \in U_{absent} \]
where the polyhedral uncertainty set $U_{\text{absent}}$ is characterized by

$$U_{\text{absent}} = \{\text{absent} : \sum_{t \in \text{WEEKS}} \text{absent}_t \leq A \cdot \text{CAPW} \cdot |\text{WEEKS}|, 0 \leq \text{absent}_t \leq \text{ABSENCE} \forall t \in \text{WEEKS}\}$$

### Demand scenarios

For a robust formulation of the demand, assume that we have got a number of different scenarios—obtained from historical data for comparable periods or possibly resulting from different forecasting methodologies—that describe the space of possible realizations of demand. In the place of the fixed demand quantities $DEM_{pt}$ we now work with uncertain quantities $demand_{pt}$ that are determined by the demand scenario data $SCENDEM_{pt}$ for a given set of scenarios $s \in \text{SCEN}$. The demand $s$ included in the original model formulation via the stock balance constraints.

$$\forall p \in \text{PRODS}, t \in \text{WEEKS} : store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt}$$

A naive ‘robustification’ might attempt to simply replace the fixed demand quantities by the uncertain $demand_{pt}$. However, this approach would not lead to the desired result: by introducing a single uncertain quantity in an equality constraint we do not leave any room for different realizations of the uncertain. We therefore now work with the following two sets of inequalities in the place of the stock balance constraints.

$$\forall p \in \text{PRODS}, t \in \text{WEEKS} : store_{pt} \leq store_{p,t-1} + produce_{pt} - demand_{pt}$$
$$\forall p \in \text{PRODS}, t \in \text{WEEKS} : store_{pt} \geq store_{p,t-1} + produce_{pt} - \max_{s \in \text{SCEN}} SCENDEM_{pt}$$

The first of these inequalities states that the demand must be satisfied from the production in a period and the difference between stock levels at the beginning and end of the period, which can be more easily seen in this transformed version of the inequality:

$$demand_{pt} \leq store_{p,t-1} + produce_{pt} - store_{pt}$$

The second inequality forces the final stock per period to be at least what remains after satisfying the largest possible demand.

### 3.3 Implementation

The standard deterministic model can be implemented as follows.

```plaintext
model "C-2 Glass production"
uses "mmxprs"

declarations
NT = 12 ! Number of weeks in planning period
WEEKS = 1..NT
PRODS = 1..6 ! Set of products
CAPW,CAPM: integer ! Capacity of workers and machines
CAPS: integer ! Storage capacity
DEM: array(PRODS,WEEKS) of integer ! Demand per product and per week
CPROD: array(PRODS) of integer ! Production cost per product
CSTOCK: array(PRODS) of integer ! Storage cost per product
ISTOCK: array(PRODS) of integer ! Initial stock levels
FSTOCK: array(PRODS) of integer ! Min. final stock levels
TIMEW,TIMEM: array(PRODS) of integer ! Worker and machine time per unit
```
SPACE: array(PRODS) of integer ! Storage space required by products
produce: array(PRODS,WEEKS) of mpvar ! Production of products per week
store: array(PRODS,WEEKS) of mpvar ! Amount stored at end of week
end-declarations

initializations from 'c2glass.dat'
CAPW CAPM CAPS DEM CSTOCK CPROD ISTOCK FSTOCK TIMEW TIMEM SPACE
end-initializations

! Objective: sum of production and storage costs
Cost:=
  sum(p in PRODS, t in WEEKS) (CPROD(p)*produce(p,t) + CSTOCK(p)*store(p,t))

! Stock balances
forall(p in PRODS, t in WEEKS)
  Bal(p,t):=
      store(p,t) = if(t>1, store(p,t-1), ISTOCK(p)) + produce(p,t) - DEM(p,t)

! Final stock levels
forall(p in PRODS) store(p,NT) >= FSTOCK(p)

! Capacity constraints
forall(t in WEEKS) do
  LimitW(t):= sum(p in PRODS) TIMEW(p)*produce(p,t) <= CAPW ! Workers
  LimitM(t):= sum(p in PRODS) TIMEM(p)*produce(p,t) <= CAPM ! Machines
  LimitS(t):= sum(p in PRODS) SPACE(p)*store(p,t) <= CAPS ! Storage
end-do

! Solve the problem
minimize(Cost)

! Solution printing
 writeln("Total cost: ",getobjval)
end-model

Workers’ absence

For the implementation of the more fine-grained handling of workers’ absence we introduce
uncertains absent that are bounded by the estimated maximum number of absence hours per
time period (ABSENCE) and we equally assume a limit of 5% on the total absence time across the
planning period. In the work capacity constraints the absence needs to be deduced from the
theoretically available work hours (we here assume that this value is 20% higher than the default
limit in the basic model).

declarations
  ABSENCE: array(WEEKS) of real ! Max. absence (hours)
  absent: array(WEEKS) of uncertain ! Absence of personnel
end-declarations

! Limit on total absence (uncertainty set)
  sum(t in WEEKS) absent(t) <= 0.05*CAPW*WEEKS.size
forall(t in WEEKS) absent(t) <= ABSENCE(t)
forall(t in WEEKS) absent(t) >= 0

! Uncertains occur in several constraints
setparam("ROBUST_UNCERTAIN_OVERLAP", true)

! Worker capacity constraints
forall(t in WEEKS)
  LimitW(t):= sum(p in PRODS) TIMEW(p)*produce(p,t) <= CAPW*1.2 -absent(t)

Demand scenarios

For the formulation of the demand scenarios we need to modify the definition of the constraints
Robust Optimization with Xpress

that involve demand data, that is, the stock balance constraints. Before stating the scenario constraints, we need to copy the scenario data into the form that is expected by the scenario construct, namely the array SCENDATA that is indexed by the scenario counter and the uncertain demand associated with every time period.

```plaintext
declarations
SCEN: range ! Demand scenarios
SCENDEM: array(SCEN, PRODS, WEEKS) of integer ! Demand per product & week
demand: array(PRODS, WEEKS) of uncertain ! Uncertain demand
SCENDATA: array(SCEN, set of uncertain) of real ! Aux. data structure
end-declarations

! Demand scenarios
forall(s in SCEN, p in PRODS, t in WEEKS | SCENDEM(s, p, t) > 0)
   SCENDATA(s, demand(p, t)) := SCENDEM(s, p, t)
scenario(SCENDATA)

! Stock balances
forall(p in PRODS, t in WEEKS) do
   Bal(p, t) :=
      store(p, t) <= if(t > 1, store(p, t-1), ISTOCK(p)) + produce(p, t) - demand(p, t)
   Bal2(p, t) :=
      store(p, t) >= if(t > 1, store(p, t-1), ISTOCK(p)) + produce(p, t) -
                    max(s in SCEN) SCENDEM(s, p, t)
end-do

! Uncertains occur in several constraints
setparam("ROBUST_UNCERTAIN_OVERLAP", true)
```

Both sets of uncertainties can be applied at the same time. However, for the analysis of their effect it might be preferrable to apply only one at a time.

### 3.4 Results

The original problem description and instance data are taken from [GHPS02] (Section 8.2 Production of drinking glasses). The costs and resource usage data for the six glass types are listed in Table 2.

<table>
<thead>
<tr>
<th>Production cost</th>
<th>Storage cost</th>
<th>Initial stock</th>
<th>Final stock</th>
<th>Time_{worker}</th>
<th>Time_{machine}</th>
<th>Storage space</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>100</td>
<td>25</td>
<td>50</td>
<td>10</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>V2</td>
<td>80</td>
<td>28</td>
<td>20</td>
<td>10</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>V3</td>
<td>110</td>
<td>25</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>V4</td>
<td>90</td>
<td>27</td>
<td>15</td>
<td>10</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>V5</td>
<td>200</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>V6</td>
<td>140</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

The solution of the original problem with the basic demand scenario has a total cost of €185,899. The resulting detailed production plan is displayed in Table 3. We find that the available manpower is fully used most of the time (in the first week, 351 hours are worked, in all other weeks the limit of 390 hours is reached) and the machines are used to their full capacity in certain time periods (weeks 1-3 and 5), but the available storage space is in excess of the actual needs.

If we try to introduce robustness to such a tightly constrained problem, the outcome most likely will be ‘problem infeasible’ as there is no slack to incorporate alternatives. We have therefore somewhat relaxed the original bound on the personnel hours, assuming that its original value represents a rough worst-case estimate and that the actually available working hours are 20%
Table 3: Optimal production plan for basic demand scenario

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.8</td>
<td>38.8</td>
<td>0</td>
<td>16</td>
<td>18</td>
<td>15</td>
<td>16</td>
<td>12</td>
<td>47.7</td>
<td>24.7</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>5.5</td>
<td>22.2</td>
<td>23</td>
<td>11</td>
<td>35</td>
<td>19</td>
<td>45</td>
<td>20</td>
<td>14.6</td>
<td>19.3</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>0.6</td>
<td>4.8</td>
<td>22</td>
<td>10</td>
<td>17</td>
<td>15</td>
<td>23</td>
<td>18</td>
<td>35.1</td>
<td>31.4</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>30.2</td>
<td>10.4</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>23</td>
<td>38</td>
<td>34</td>
<td>14.3</td>
<td>30.8</td>
<td>19</td>
<td>24.7</td>
</tr>
<tr>
<td>5</td>
<td>27.4</td>
<td>–</td>
<td>11</td>
<td>21</td>
<td>23</td>
<td>15</td>
<td>41</td>
<td>20</td>
<td>22.8</td>
<td>44.2</td>
<td>21</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>8.6</td>
<td>–</td>
<td>10</td>
<td>23</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>19</td>
<td>43.8</td>
<td>45</td>
<td>18</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
<td>–</td>
<td>10</td>
<td>15</td>
<td>9</td>
<td>28</td>
<td>20</td>
<td>37</td>
<td>0</td>
<td>70.8</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>–</td>
<td>10</td>
<td>23</td>
<td>10</td>
<td>12</td>
<td>19</td>
<td>27.2</td>
<td>10</td>
<td>40.75</td>
<td>30.8</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>29</td>
<td>–</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>30</td>
<td>12</td>
<td>28</td>
<td>2.8</td>
<td>39.25</td>
<td>14.3</td>
<td>29.2</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>–</td>
<td>10</td>
<td>10</td>
<td>13</td>
<td>47</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>35</td>
<td>1.1</td>
<td>5.8</td>
</tr>
<tr>
<td>11</td>
<td>28</td>
<td>–</td>
<td>10</td>
<td>–</td>
<td>13</td>
<td>–</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td>–</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Workers: 351 390 390 390 390 390 390 390 390 390 390 390
Machines: 850 850 850 850 850 850 850 850 850 786.7 711 700.3
Space: 268.8 166.2 144.8 123 218.4 180 289.8 163 157 311 325.2 330

higher. The more fine-grained estimate of absence results in the solution summarized in Table 4. The overall cost of €181,210 is slightly lower than the previous, fixed worst-case absence case.

Table 4: Summary results with robust formulation of absence

<table>
<thead>
<tr>
<th>Week</th>
<th>Workers</th>
<th>CAPW-ABSENCE</th>
<th>Machines</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>243</td>
<td>437</td>
<td>609</td>
<td>152.5</td>
</tr>
<tr>
<td>2</td>
<td>378</td>
<td>437</td>
<td>850</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>370</td>
<td>437</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>407</td>
<td>429</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>325</td>
<td>429</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>398</td>
<td>414</td>
<td>850</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>383</td>
<td>414</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>414</td>
<td>414</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>412</td>
<td>445</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>445</td>
<td>445</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>429</td>
<td>429</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>437</td>
<td>437</td>
<td>850</td>
<td>0</td>
</tr>
</tbody>
</table>

Workers: 243 378 370 407 325 398 383 414 412 445 429 437
Machines: 609 850 736 770 736 794 772.2 739.8 803 839.5 734 747.5
Space: 152.5 32 0 0 20 0 99 0 16 130 219.5 330

The scenario-based approach results in a higher total cost of €203,545. In the summary results in Table 5 we observe that both, machine capacity and workers’ hours, are most of the time used at or close to their capacity limit (850 and 429=1.1·390 respectively). Furthermore, larger quantities of products are held in stock.

Table 5: Summary results with demand scenarios

<table>
<thead>
<tr>
<th>Week</th>
<th>Workers</th>
<th>Machines</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>363</td>
<td>850</td>
<td>230.7</td>
</tr>
<tr>
<td>2</td>
<td>425</td>
<td>850</td>
<td>145.9</td>
</tr>
<tr>
<td>3</td>
<td>425</td>
<td>850</td>
<td>150.4</td>
</tr>
<tr>
<td>4</td>
<td>425</td>
<td>850</td>
<td>126.3</td>
</tr>
<tr>
<td>5</td>
<td>425</td>
<td>850</td>
<td>233.2</td>
</tr>
<tr>
<td>6</td>
<td>425</td>
<td>850</td>
<td>211.9</td>
</tr>
<tr>
<td>7</td>
<td>425</td>
<td>850</td>
<td>306.8</td>
</tr>
<tr>
<td>8</td>
<td>425</td>
<td>850</td>
<td>181.7</td>
</tr>
<tr>
<td>9</td>
<td>425</td>
<td>850</td>
<td>169.7</td>
</tr>
<tr>
<td>10</td>
<td>425</td>
<td>850</td>
<td>256.7</td>
</tr>
<tr>
<td>11</td>
<td>425</td>
<td>850</td>
<td>305.7</td>
</tr>
<tr>
<td>12</td>
<td>425</td>
<td>850</td>
<td>330</td>
</tr>
</tbody>
</table>

Machines: 850 850 850 850 850 850 850 850 786.7 711 700.3
Space: 230.7 145.9 150.4 126.3 233.2 211.9 306.8 181.7 169.7 256.7 305.7 330

For easier comparison, Table 6 shows the summary results for the original model with the same capacity limits as have been used with the robust formulation in Table 5, the total cost is in this case €181,432.

Table 6: Summary results for deterministic model with 1.1·CAPW

<table>
<thead>
<tr>
<th>Week</th>
<th>Workers</th>
<th>Machines</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>243</td>
<td>609</td>
<td>152.5</td>
</tr>
<tr>
<td>2</td>
<td>378</td>
<td>850</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>370</td>
<td>736</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>407</td>
<td>770</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>305</td>
<td>681</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>418</td>
<td>849</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>393</td>
<td>799</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>425</td>
<td>771.2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>425</td>
<td>840</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>425</td>
<td>783.2</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>425</td>
<td>721</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>425</td>
<td>721.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Workers: 243 378 370 407 305 418 393 425 425 425 425 425
Machines: 609 850 736 770 681 849 799 771.2 840 783.2 721 721.5
Space: 152.5 32 0 0 0 0 108.7 21.2 50.6 151.5 245.5 330
4 Robust network design

4.1 Problem description

Suppose we want to design a Telecommunication network. We are given a set of routers, a set of links, and a set of traffic demands (measured in MB/s) between pairs of routers; for convenience we can assume each of them sits in a city. We want to find the capacity to be installed on each link in order to satisfy all traffic demands.

There are several variants of this problem in the scientific literature, and an important subclass deals with the problem of provisioning virtual private networks, or VPNs for short.

In general, in VPN traffic demand is difficult to predict even in the short term, and hence the traffic demands are uncertain; however, there exist uncertainty models for such traffic demands. Here we consider one that has attracted considerable attention and that is realistic enough in VPNs.

We suppose that there are two parameters $s^+_i$ and $s^-_i$ which estimate the maximum total outgoing and incoming traffic, respectively, at node $i$.

The problem is hence that of finding the value of the capacity of every arc of the network (which is a multiple of a given unit capacity $C$) so as to accommodate any traffic demand $d_{hk}$ from node $h$ to node $k$, subject to uncertainty on these demands modeled by the upper bounds on the incoming and outgoing traffic for each node of the network.

4.2 Mathematical formulation

Consider a directed graph $G$ with $n$ nodes and $m$ arcs, defined as $G = (V, A)$, where $V = \{1, \ldots, n\}$ denotes the set of $n$ nodes and $A = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$ is the set of arcs. The cost of every unit of capacity (MB/s) installed at each arc $(i, j) \in A$ is denoted by $a_{ij}$, and, for each pair of nodes $(h, k)$ the (unknown) traffic demand from $h$ to $k$ is given by $d_{hk}$.

Let us introduce a binary flow variable $f_{hk}^{ij}$: this variable is one if demand $(h, k)$ is routed through arc $(i, j)$, zero otherwise, and serves to model the flow of data between each source/destination pair. We also need to introduce the integer variable $x_{ij}$, which represents the number of channels of capacity $C$ to be installed in the network.

This problem is a particular case of a multi-commodity network flow problems, with the important addition of the uncertain in the traffic demands.

Let us first model the uncertainty set: every uncertain demand $d_{hk}$ is nonnegative and the total demand leaving or entering a node $i$ is bounded from above by a given parameter:

$$\forall k \in V, \sum_{h \in V, h \neq k} d_{hk} \leq s^-_k$$

$$\forall h \in V, \sum_{k \in V, k \neq h} d_{hk} \leq s^+_h.$$  

Let us consider now the two main classes of constraints for the problem: conservation of flow for the variables $f$ and the capacity constraints. The former reads as follows: for each node $i$ and each demand $(h, k)$, if $i$ is an intermediate node for this demand, i.e., it is neither $h$ nor $k$, then the incoming flow must be equal to the outgoing flow, therefore
∀i, h, k ∈ V : h ≠ i ≠ k, \[ \sum_{j, (i, j) \in A} f_{ij}^{hk} = \sum_{j, (j, i) \in A} f_{ji}^{hk}. \]

If \( i = h \), the total balance of flow must be one, hence

\[ ∀i, h, k ∈ V : i = h, \sum_{j, (i, j) \in A} f_{ij}^{hk} - \sum_{j, (j, i) \in A} f_{ji}^{hk} = 1. \]

The case for \( i = k \) is redundant and its corresponding constraint can be omitted.

We can now write the capacity constraint: the total traffic on arc \((i, j)\), weighted by the (unknown) traffic demands, must be less than or equal to the total capacity installed on that arc:

\[ ∀(i, j) ∈ A, \sum_{h \in V} \sum_{k \in V \setminus \{h\}} d_{hk} f_{ij}^{hk} ≤ Cx_{ij}. \]

Finally, the objective function, to be minimized, is

\[ \sum_{(i, j) \in A} a_{ij} x_{ij}. \]

### 4.3 Implementation

Below is the implementation in Mosel: note that the option ROBUST_OVERLAP_UNCERTAIN is used since the capacity constraints at each arc use a subset of the uncertain demands.

```mosel
model "Robust Network"
uses "mmrobust", "mmxprs"
parameters
  vpn_data="vpn_data.dat"
end-parameters
declarations
  NODES: range ! Set of nodes
  ARCCOST: dynamic array(NODES, NODES) of real ! Per-unit cost of arc (i,j)
  DEM_IN: array(NODES) of real ! Total INCOMING demand at each node
  DEM_OUT: array(NODES) of real ! Total OUTGOING demand at each node
  UNITCAP: integer ! Per-unit capacity (independent of arc)
  NETCOST: linctr ! Objective function

  ! Decision variables
  flow: dynamic array(NODES, NODES, NODES, NODES) of mpvar
    ! flow(i,j,h,k) is 1 iff demand h→k uses arc (i,j)
  capacity: dynamic array(NODES, NODES) of mpvar ! Capacity to install on arc (i,j)

  ! Uncertain parameters
  demand: array(NODES, NODES) of uncertain ! Uncertain traffic demand
  RobustCap: array(NODES, NODES) of robustctr ! Uncertain traffic demand, formulated at each arc
end-declarations

! The following option is necessary to allow uncertain demands to be
! used across multiple capacity constraints
setparam("robust_uncertain_overlap", true)
```

Robust network design
! Set verbosity level
setparam("xprs_verbose", true)

!**** Data input ****
initializations from vpn_data
NODES ARCCOST
DEM_IN DEM_OUT
UNITCAP
end-initializations

!**** Model formulation ****
forall(i in NODES, j in NODES, h in NODES, k in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0 and h <> k) do
create(flow(i,j,h,k))
flow(i,j,h,k) is_binary
end-do
forall(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0) do
create(capacity(i,j))
capacity(i,j) is_integer
end-do

! Flow balance at intermediate nodes for each demand(h,k)
forall(i in NODES, h in NODES, k in NODES | i <> h and i <> k and k <> h)
sum(j in NODES | exists(flow(i,j,h,k))) flow(i,j,h,k) -
sum(j in NODES | exists(flow(j,i,h,k))) flow(j,i,h,k) = 0

! Flow balance at source nodes (unnecessary for sink nodes)
forall(i in NODES, h=i, k in NODES | k <> h)
sum(j in NODES | exists(flow(i,j,h,k))) flow(i,j,h,k) -
sum(j in NODES | exists(flow(j,i,h,k))) flow(j,i,h,k) = 1

! Capacity (robust) constraint
forall(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0)
RobustCap(i,j) :=
sum(h in NODES, k in NODES | h <> k) demand(h,k) * flow(i,j,h,k) <=
UNITCAP * capacity(i,j)

! Uncertainty set: all demands are nonnegative and constrained by
! total outgoing and incoming demand
forall(h in NODES, k in NODES | h <> k) demand(h,k) >= 0
forall(h in NODES) sum(k in NODES | h <> k) demand(h,k) <= DEM_OUT(h)
forall(h in NODES) sum(k in NODES | h <> k) demand(k,h) <= DEM_IN(h)

! Shortest path length
NetCost := sum(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0)
ARCCOST(i,j) * capacity(i,j)

!**** Solving ****
minimize(NetCost)
declarations
curNode: integer ! Local variable used in following the path of each demand
end-declarations

! Printing capacities
writeln("Robust solution has total cost ", getobjval)
forall(i in NODES, j in NODES | exists(capacity(i,j)) and capacity(i,j).sol > 0)
writeln("arc ", i, " -- ", j, ": ", capacity(i,j).sol, " units")
writeln("Paths:")
forall(h in NODES, k in NODES | h <> k) do
  write("Demand ", h, " --> ", k, ": ")
  curNode := h
  while(curNode <> k) do
    forall(j in NODES | flow(curNode, j, h, k).sol > 0.5) do
      write(" -- ", j)
      curNode := j
    end-do
  end-do
end-do
writeln
writeln("Active demands at each arc:")
forall(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0 and
  sum(h in NODES, k in NODES) getsol(demand(h,k), RobustCap(i,j)) > 0) do
  write("Arc (", i, ",", j, ": ")
  forall(h in NODES, k in NODES | getsol(demand(h,k), RobustCap(i,j)) > 0)
    write(h, "--", k, " (", getsol(demand(h,k), RobustCap(i,j)), "); ")
  writeln
end-do
end-model

4.4 Input Data

Consider the network depicted in Figure 5, where each link represents two arcs in opposite
directions. Table 7 below reports the parameters specified in the data file vpn_data.dat
describing the uncertainty set, i.e., the maximum incoming and outgoing traffic demands for
each node of the network.

![Figure 5: Network topology](image)

Table 8 describes instead the arc cost for every arc \((i,j)\) used in the network (a “–” means that no
connection exists).

4.5 Results

For the example provided in vpn_data.dat, which describes a network of six nodes and 18 arcs,
Table 7: Maximum incoming/outgoing demand

<table>
<thead>
<tr>
<th>Node</th>
<th>Incoming</th>
<th>Outgoing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>120</td>
<td>105</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>95</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>101</td>
<td>102</td>
</tr>
<tr>
<td>5</td>
<td>78</td>
<td>76</td>
</tr>
<tr>
<td>6</td>
<td>75</td>
<td>140</td>
</tr>
</tbody>
</table>

Table 8: Arc costs

<table>
<thead>
<tr>
<th>i/j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>9</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

we obtain a design of total cost 566. Table 9 reports the number of units of capacity to be installed on each arc, while Table 10 describes the path followed by each demand from source to destination.

Table 9: Optimal arc capacity

<table>
<thead>
<tr>
<th>i/j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The last part of the output shows, for each arc, the demands for which the uncertain values are non-zero in the worst case. Note that arcs such as (2, 3) do not admit any nonzero demand simply because they are not used by the solution—the opponent has no incentive to increase the demand on such arcs.

Active demands at each arc:
- Arc (1,2): 1-->2 (34);
- Arc (1,3): 1-->3 (35);
- Arc (1,4): 1-->4 (101); 1-->5 (4); 2-->6 (67); 3-->5 (74); 3-->6 (8);
- Arc (2,1): 2-->1 (98);
- Arc (3,1): 3-->1 (82);
- Arc (4,1): 4-->1 (102); 5-->1 (18); 6-->2 (34); 6-->3 (35);
- Arc (4,5): 1-->5 (78);
- Arc (4,6): 1-->6 (75);
- Arc (5,4): 5-->1 (76);
- Arc (8,4): 6-->1 (106); 6-->2 (34);
Table 10: Optimal routes

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Path</th>
<th>Source</th>
<th>Destination</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1→2</td>
<td>4</td>
<td>1</td>
<td>4→1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1→3</td>
<td>4</td>
<td>2</td>
<td>4→1→2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1→4</td>
<td>4</td>
<td>3</td>
<td>4→1→3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1→4→5</td>
<td>4</td>
<td>5</td>
<td>4→5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1→4→6</td>
<td>4</td>
<td>6</td>
<td>4→6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2→1</td>
<td>5</td>
<td>1</td>
<td>5→4→1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2→1→3</td>
<td>5</td>
<td>2</td>
<td>5→4→1→2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2→1→4</td>
<td>5</td>
<td>3</td>
<td>5→4→1→3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2→1→4→5</td>
<td>5</td>
<td>4</td>
<td>5→4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2→1→4→6</td>
<td>5</td>
<td>6</td>
<td>5→4→6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3→1</td>
<td>6</td>
<td>1</td>
<td>6→4→1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3→1→2</td>
<td>6</td>
<td>2</td>
<td>6→4→1→2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3→1→4</td>
<td>6</td>
<td>3</td>
<td>6→4→1→3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3→1→4→5</td>
<td>6</td>
<td>4</td>
<td>6→4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3→1→4→6</td>
<td>6</td>
<td>5</td>
<td>6→4→5</td>
</tr>
</tbody>
</table>
5 Robust portfolio optimization

In this section, we present a robust optimization formulation of the single period portfolio selection problem. In order to compare the results from the robust optimization, we also show how to quantify the robustness of a solution using Monte-Carlo simulation.

5.1 Problem description

The single-period portfolio selection problem is about selecting assets from a given list in order to create a portfolio that has the greatest expected value. The assets are bought at the market price and their value is subject to variation. The investment budget is fixed, and each selected asset is bought using a percentage of the available budget.

The market price for each asset is known at the time of buying and corresponds to the payment to be made by the trader to buy one unit. The future asset value is not known, but we assume the distribution characteristics of the random variable is known.

The trader is risk averse and wants to have some guarantees about the worst case value of the portfolio. She considers that in the worst case the downward variation of an asset value reaches 1.5 times the variance of the asset. In other words, the trader assumes it is very unlikely that the value of an asset decreases by more than 1.5 times its variance.

Example: Let’s assume that asset #20 has a market price of 100$ and its variability is +/- 10$. Then the considered worst case value of the asset is 85$.

In this context, the objective of the trader is to maximize the expected value of her portfolio, but without taking too much risk. A conservative trader would definitely spend all her budget in the asset with the highest worst case value in order to be highly protected against the worst possible outcome. Unfortunately, this strategy would dramatically reduce the expected value of the portfolio. Another extreme policy would be an opportunistic trader who spends all her budget in the asset with the highest expected value, without paying much attention to the worst case realization.

Both approaches are extreme and do not take into account the fact that it is quite unlikely to see all asset values going down (or going up) to the same degree. In practice traders often aim for portfolios with a guaranteed probability of having a value greater than a minimum target value. This is also known as chance constrained optimization. In the ‘Results’ section (5.4) we will show how robust optimization can be used to provide solutions with a large expected value without sacrificing protection against worst case realization.

5.2 Mathematical formulation

The set $S$ contains the available assets, and the index $s \in S$ represents an asset from this set. The market price of the asset $s$ is noted $PRICE_s$, and its variability range is given by $VAR_s$.

For each asset $s \in S$, the variable $x_s$ represents the fraction of the budget spent to buy asset $s$. The variable $w$ is used to represent the worst case value of the portfolio. The variable $dev_s$ describes the possible variation of the value of the asset $s$.

The value $N$ describes the worst case the trader will consider and corresponds to the maximum decrease of the asset value expressed as a multiple of its standard deviation.
5.2.1 Highest protection

A conservative trader will want to maximize the worst case value of her portfolio. If she wants to get the best protection against worst case realization, then she will try to maximize the worst case value of her portfolio. In our example, this worst case realizes when the value of every asset decreases by \( N \) times its standard deviation.

\[
\begin{align*}
\max & \quad w \\
\text{s.t.} & \quad w \leq \sum_{s \in S} (\text{PRICE}_s + N \cdot \text{dev}_s) \cdot x_s \quad \text{(with dev}_s = -\text{VAR}_s) \\
& \quad \sum_{s \in S} x_s = 1 \\
& \quad 0 \leq x_s \leq 1
\end{align*}
\]

5.2.2 Budgeted protection

As we will see in the discussion of the results (Section 5.4), achieving a high protection against worst case realization incurs an important loss of value in the average case. And hence, this strategy may not be such a clever choice because average cases are more likely to happen. The trader therefore might want to improve her model by adding a control parameter to limit the protection level.

The overall variation budget \( G \) is defined by the following formula:

\[
G = \sum_{s \in S} (N \cdot \text{VAR}_s)^2
\]

The ellipsoidal uncertainty set

Let \( k \) be the percentage value representing the protection level. The possible asset value decrease is controlled by the uncertainty set \( U(k) \) that is defined as follows:

\[
U(k) = \{ e : \sum_{s \in S} e_s^2 \leq (k \cdot G)^2 \}
\]

Robust constraints

The resulting robust optimization problem can then be stated as:

\[
\begin{align*}
\max & \quad w \\
\text{s.t.} & \quad w \leq \sum_{s \in S} (\text{PRICE}_s + e_s) \cdot x_s \quad (\forall e \in U(k)) \\
& \quad \sum_{s \in S} x_s = 1 \\
& \quad 0 \leq x_s \leq 1
\end{align*}
\]

5.3 Implementation

The following Mosel model implements the mathematical model from the previous section. Notice how the ellipsoid uncertainty set is added to the original problem to create a robust optimization problem.
Also, take a look at how the Monte-Carlo simulation method is applied to quantify the quality of the solution: in $NMC = 5000$ iterations we draw random values for the expected value of every asset by applying a normal distribution centered around its price (mean value) and using its known variance. We count the occurrence of objective function values (or more precisely, which value range the resulting solution value belongs to) to determine the probabilities of different solution qualities. This method is particularly helpful to gain some insight about the solution quality when it is difficult to determine the exact shape of the distribution function of a random variable.

```
model "Robust portfolio optimization"
uses "mmrobust", "random"
uses "mmsystem"

parameters
ZEROTOL=1e-6
SEED=12345
NMC=500000
N=1.5
! Worst case metric
end-parameters

declarations
Problems: set of mpproblem
ProtectLevel: set of real

mp_problemA: mpproblem ! Worst case optimization
mp_problemB: array(ProtectLevel) of mpproblem ! Robust optimization
NSHARES = 25 ! Max number of shares
Shares = 1..NSHARES ! Set of shares
PRICE: array(Shares) of real ! Expected return on investment
VAR: array(Shares) of real ! Uncertainty measure (deviation) per SHARE

expReturn: mpvar ! Expected portfolio value
wstReturn: mpvar ! Worst case portfolio value
frac: array(Shares) of mpvar ! Fraction of capital used per share
frac_sol: array(Shares,Problems) of real
obj: mpvar

e: array(Shares) of uncertain ! Deviation of share values
PRICEthr: array(Problems,Shares) of real ! The price threshold

end-declarations

!***************************Configuration*************************
setparam("XPRS_LOADNAMES",true)

!***************************Subroutines***************************

!**** Create the nominal model ****
procedure create_nominalmodel
  ! Nominal model
  expReturn = (sum(s in Shares) PRICE(s)*frac(s))
  wstReturn = sum(s in Shares) ( PRICE(s) - N*VAR(s) )*frac(s)
  ! Spend all the budget
  sum(s in Shares) frac(s) = 1
end-procedure

!**** Optimize for the worst case realization ****
procedure solve_det
  obj = wstReturn
  maximize(obj)
end-procedure

!**** Optimize for a given protection level ****
procedure solve_rob(k: real)
```
! The value variation domain

assert(k>=0 and k<=1)
sum(s in Shares) (e(s) / (N*VAR(s)))^2 <= (k)^2

! The robust constraint
obj <= sum(s in Shares) (PRICE(s) + e(s)) * frac(s)
maximize(obj)
end-procedure

!***************************Main model***************************

!**** Input data generation ****
setmtrandseed(12345)
cnt := 0
forall(s in Shares, cnt as counter) do
  PRICE(s) := round((100*(NSHARES-cnt+1)/NSHARES))
c := (1+sqrt((NSHARES-s)/NSHARES))/3
VAR(s) := round(PRICE(s)*c*100)/100
end-do
writeln("\n *** Input Data *** ")
writeln
writeln("Shares | Mean | S.Dev. | Worst value")
forall(s in Shares)
  writeln(strfmt(s,6), " |", strfmt(PRICE(s),5), " |", strfmt(VAR(s),7,2), " |", strfmt(PRICE(s)-N*VAR(s),6,3))
writeln

!**** Optimize the worst case ****
with mp_problemA do
  create_nominalmodel
  solve_det
  forall(s in Shares) frac_sol(s,mp_problemA) := frac(s).sol*100
  ! Price set by opponent is worst case price
  forall(s in Shares) PRICEthr(mp_problemA,s) := PRICE(s) - N*VAR(s)
end-do

!**** Optimize the 'budgeted' worst case ****
Ks := [0, ! No variation on average
  0.01, ! +/- 1% variation on the list
  0.10, ! +/- 10% variation on the list
  0.25, ! +/- 25%
  1] ! +/-100% variation on the list
forall(k in Ks) do
  create(mp_problemB(k))
  with mp_problemB(k) do
    create_nominalmodel
    solve_rob(k)
    forall(s in Shares) frac_sol(s,mp_problemB(k)) := frac(s).sol*100
    forall(s in Shares) PRICEthr(mp_problemB(k),s) := (PRICE(s) + getsol(e(s)))
  end-do
end-do

!**** Print results ****
writeln("\n *** Portfolio allocation results *** ")
write("| protection level"
write("| Shares | Wst price | Price | A |
forall(k in Ks) write("|", strfmt(k*100,3), "|% |") ; writeln
forall(s in Shares) do
  write(strfmt(s,6), " |", strfmt(PRICE(s)-N*VAR(s),10,0), " |", strfmt(PRICE(s),6), " | ")
forall(mp in Problems) do
  if (frac_sol(s,mp)>1) then
    write(strfmt(frac_sol(s,mp),3,0), "% | ")
  else
    write(" | ")
Robust Optimization with Xpress

end-if
end-do
writeln
end-do

writeln("\n *** Worst case price *** ")
write("\n | protection level")
write("\nShares | Wet price | Price | A |")
forall(k in Ks) write(" ", strfmt(k*100,3), " % | ") ; writeln
forall(s in Shares) do
write(strfmt(s,6), " |", strfmt(PRICE(s)-N*VAR(s),10,0), " |", strfmt(PRICE(s),6), " | ")
forall(mp in Problems) do
if (frac_sol(s,mp)>1) then
write(strfmt(PRICEthr(mp,s),3,0), "* | ")
else
write(strfmt(PRICEthr(mp,s),4,0), " | ")
! write(" | ")
end-if
end-do
writeln
end-do

**** Simulate results (Monte-Carlo simulation) ****
writeln("Running Monte-Carlo simulation...")
Values := {0,20,40,60,80,100}
declarations
cntV: dynamic array(Problems,Values) of real
s1: real
s2: real
end-declarations
forall(mp in Problems) do
expRev(mp) := 0.0 ; wstRev(mp) := 0.0 ; expDev(mp) := 0.0
forall(v in Values) cntV(mp,v) := 0
c := 0.0 ! Number of draws
s1 := 0 ; s2 := 0;
forall(i in 1..NMC, c as counter) do
forall(s in Shares | frac_sol(s,mp)>0) do
! draw a price
p := normal(PRICE(s),VAR(s))
! calculate share revenue
r := frac_sol(s,mp)*p/100
! calculate total revenue
totalVal += r ! summing up revenue
end-do
forall(v in Values) do
if (totalVal>v) then
cntV(mp,v) += 1
end-if
end-do
expRev(mp) += totalVal
s1 += totalVal
s2 += totalVal^2
expRev(mp) := expRev(mp) / c
expDev(mp) := sqrt(NMC*s2 - s1^2)/(NMC)
forall(v in Values) cntV(mp,v) := cntV(mp,v) / c
end-do

**** Print simulation results ****
write("\n | protection level")
write("\n | A |")
forall(k in Ks) write(" ", strfmt(k*100,3), " % | ")
write("\n Expected value |")
forall(mp in Problems) write(strfmt(round(expRev(mp)),6), " | ")
Robust Optimization with Xpress

write("\n Standard deviation |")
forall(mp in Problems) write(strfmt(round(expDev(mp)),5), " |")
writeln
forall(v in Values) do
  write("\n P (value>textfmt(v,3)=" |")
  forall(mp in Problems) write(strfmt(cntV(mp,v),5,2), " |")
end-do
end-model

5.4 Results

In order to understand how the uncertainty set impacts the solution we will solve the portfolio allocation problem for various protection levels $k$ and compare the proposed budget allocation against the conservative approach.

The trader knows that she can expect a portfolio value in the range of 0 to 100. She therefore wishes to calculate for various ranges the probability that the value of the portfolio will belong to this range. She also wants to reach a portfolio value of 60 with high probability.

5.4.1 Input Data

Table 11 shows the asset mean value which is also the market price (Mean), the standard variation of the value (S. Dev), and the considered worst case value with $N = 1.5$ (Worst value).

<table>
<thead>
<tr>
<th>Shares</th>
<th>Mean</th>
<th>S.Dev.</th>
<th>Worst value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>65.99</td>
<td>1.015</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
<td>62.69</td>
<td>1.965</td>
</tr>
<tr>
<td>3</td>
<td>92</td>
<td>59.43</td>
<td>2.855</td>
</tr>
<tr>
<td>4</td>
<td>88</td>
<td>56.22</td>
<td>3.670</td>
</tr>
<tr>
<td>5</td>
<td>84</td>
<td>53.04</td>
<td>4.440</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>49.91</td>
<td>5.135</td>
</tr>
<tr>
<td>7</td>
<td>76</td>
<td>46.83</td>
<td>5.755</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
<td>43.79</td>
<td>6.315</td>
</tr>
<tr>
<td>9</td>
<td>68</td>
<td>40.80</td>
<td>6.800</td>
</tr>
<tr>
<td>10</td>
<td>64</td>
<td>37.86</td>
<td>7.210</td>
</tr>
<tr>
<td>11</td>
<td>60</td>
<td>34.97</td>
<td>7.545</td>
</tr>
<tr>
<td>12</td>
<td>56</td>
<td>32.13</td>
<td>7.805</td>
</tr>
<tr>
<td>13</td>
<td>52</td>
<td>29.34</td>
<td>7.990</td>
</tr>
<tr>
<td>14</td>
<td>48</td>
<td>26.61</td>
<td>8.085</td>
</tr>
<tr>
<td>15</td>
<td>44</td>
<td>23.94</td>
<td>8.090</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
<td>21.33</td>
<td>8.005</td>
</tr>
<tr>
<td>17</td>
<td>36</td>
<td>18.79</td>
<td>7.815</td>
</tr>
<tr>
<td>18</td>
<td>32</td>
<td>16.31</td>
<td>7.535</td>
</tr>
<tr>
<td>19</td>
<td>28</td>
<td>13.91</td>
<td>7.135</td>
</tr>
<tr>
<td>20</td>
<td>24</td>
<td>11.58</td>
<td>6.630</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>9.33</td>
<td>6.005</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>7.18</td>
<td>5.230</td>
</tr>
<tr>
<td>23</td>
<td>12</td>
<td>5.13</td>
<td>4.305</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>3.20</td>
<td>3.200</td>
</tr>
<tr>
<td>25</td>
<td>4</td>
<td>1.33</td>
<td>2.005</td>
</tr>
</tbody>
</table>

A quick glance at the input data reveals that asset #15 would be the best choice for the
conservative trader because it has the highest worst case value, whereas asset #1 would be the preferred investment for an optimistic trader since it has the largest best case value.

5.4.2 Analysis

In order to understand how the robust optimization behaves compared to the nominal problem, we solve the nominal problem (A) and 5 robust optimization problems parameterized by the protection level \( k \) (ranging from 0 to \( 1=100\% \)). With a protection level of \( k=0 \) the deviation budget is also zero, and hence the solution is a very optimistic one.

In a second step, we use a Monte-Carlo method to simulate the actual value of the assets selected by these solutions and calculate the probability with which the portfolio value lies in the various revenue ranges.

Table 12 presents the portfolio selection suggestion for each of these problems. For the sake of simplicity we list only those assets that are selected in at least one solution. The conservative solution (A) is to allocate all budget to the asset with highest worst case value. The optimistic solution (\( k=0.0 \)) is to allocate the whole budget to the asset with the highest expected value. With an increasing protection level the suggested solutions improve the balance between assets with a high largest expected value and those with a high worst case value.

Table 12: Results of the parameterized robust optimization

<table>
<thead>
<tr>
<th>Shares</th>
<th>Wst price</th>
<th>Price</th>
<th>A</th>
<th>k=0.0%</th>
<th>k=1%</th>
<th>k=10%</th>
<th>k=25%</th>
<th>k=100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>100%</td>
<td>100%</td>
<td>60%</td>
<td>33%</td>
<td>11%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>96</td>
<td>35%</td>
<td>28%</td>
<td>11%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>92</td>
<td>5%</td>
<td>22%</td>
<td>11%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>88</td>
<td>14%</td>
<td>11%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>84</td>
<td>3%</td>
<td>11%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>80</td>
<td></td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>76</td>
<td></td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>72</td>
<td></td>
<td>9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>68</td>
<td></td>
<td>7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>64</td>
<td></td>
<td>5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>60</td>
<td></td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>44</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13 presents the results of the Monte-Carlo simulation of the portfolio value for each portfolio selection suggestion. The conservative solution (A) is the one with highest worst case value, and as expected the probability of this portfolio’s value to go above 60 is very low. From the trader’s point of view this is not a good solution, even in the average case for which she will get a value of 44.

The optimistic solution (\( k=0\% \)) and the solution with the lowest protection level (\( k=1\% \)) have the largest expected value. However, the trader cannot use these two solutions because she won’t be able to commit to achieving a minimum value of 60 since the probability of obtaining a portfolio value greater than 60 is only 73% for these two solutions.

The behavior of the other solutions (\( k \geq 10\% \)) shows that with an increasing deviation budget, the probability of achieving a portfolio value higher than 60 also increases.

The worst case value of the portfolio equally increases while the expected value decreases. The volatility of the distribution of the values against the three ranges tends to reduce.

The solution with \( k = 100\% \) protects the trader against important losses and allows her to commit to a minimum portfolio value of 60 with a likelihood of more than 90% and a total expected
Table 13: Results of the Monte-Carlo simulation

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>k=0.0%</th>
<th>k=1%</th>
<th>k=10%</th>
<th>k=25%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected value</td>
<td>44</td>
<td>100</td>
<td>100</td>
<td>98</td>
<td>95</td>
<td>83</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>24</td>
<td>66</td>
<td>66</td>
<td>45</td>
<td>32</td>
<td>17</td>
</tr>
<tr>
<td>P (value $\geq 0$)</td>
<td>97%</td>
<td>94%</td>
<td>93%</td>
<td>98%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P (value $\geq 20$)</td>
<td>84%</td>
<td>89%</td>
<td>89%</td>
<td>96%</td>
<td>99%</td>
<td>100%</td>
</tr>
<tr>
<td>P (value $\geq 40$)</td>
<td>57%</td>
<td>82%</td>
<td>82%</td>
<td>90%</td>
<td>96%</td>
<td>99%</td>
</tr>
<tr>
<td>P (value $\geq 60$)</td>
<td>25%</td>
<td>73%</td>
<td>73%</td>
<td>80%</td>
<td>86%</td>
<td>91%</td>
</tr>
<tr>
<td>P (value $\geq 80$)</td>
<td>07%</td>
<td>62%</td>
<td>62%</td>
<td>66%</td>
<td>68%</td>
<td>58%</td>
</tr>
<tr>
<td>P (value $\geq 100$)</td>
<td>01%</td>
<td>50%</td>
<td>50%</td>
<td>48%</td>
<td>44%</td>
<td>17%</td>
</tr>
</tbody>
</table>

A value of 83. A more risk-prone trader may want to reduce the protection level to say, 10% resulting in a considerably higher average outcome (98) and still having a 90% chance of achieving a portfolio value greater than 40.

5.5 References

The presented problem is inspired by the Single-Period Portfolio selection problem described in [BTEGN09].

Applications of robust optimization to portfolio optimizations models date back to as early as 2000 [BTMN00].
6 Robust unit commitment

6.1 Problem description

The unit commitment problem consists in scheduling the power generation level of a set of generators to meet the power demand. In order for a power system to operate safely, the power generation must equal the demand at any time (the so-called power balance constraints). Failing to satisfy this requirement may result in serious issues, possibly causing a blackout.

The power generation level of a power generating unit depends on its commitment. When the unit is switched off, no power is produced, only when it has been started the unit can generate electricity. Due to technical constraints, the power generation level must lie within a band describing safe operation. For the vast majority of unit types, the minimum power generation level is an important parameter when deciding whether to start a particular unit or not.

Another parameter that plays an important role in the decision whether to start a unit is its startup cost. Units with low startup cost could be started and shut down multiple times during a planning period, while units with high cost will preferably be kept running once started.

For the sake of simplicity, other considerations like ramp rate constraints, shutdown costs or the various reserve types are omitted from the present model.

Example: We shall consider the case of day-head unit commitment scheduling. This operational planning task consists in determining on a day for the next, the startup and shutdown phases of the generating units in order to be able to safely satisfy the next day’s power demand. However, the exact demand is not known for sure at the time of scheduling, and hence it is considered an uncertain parameter of the optimization problem. Common practice for dealing with this uncertainty is to commit a larger number of units than strictly required to be sure to have sufficient upward and downward power generation reserve. In the following we will show how to apply robust optimization techniques to take into account this uncertainty more efficiently.

We are going to present two robust implementations of the unit commitment problem resulting in plans for which the unit commitment decisions do not have to be changed in real time. More precisely, the suggested unit commitment schedules make sure that enough power generation capacity is available to supply the actual demand without requiring any last minute unit startups or shutdowns. The first model is based on a scenario uncertainty set, it ensures that the unit commitment status will not require any modification even if the real power demand is different from the forecast. In this problem it is assumed that the power demand realization will be similar to either the forecast or some historical power demand curve. The second model, based on a cardinality uncertainty set, guarantees that the unit commitment status will not require any modification even if up to ‘k’ generating units are simultaneously forced in outage.

6.1.1 Robust against power demand variation

Typical unit commitment robustness against power demand variation is achieved by adding constraints to bound the difference between the total power generation and the total available power generation capacity of running units. This value is known as the upward power generation level and defines the maximum power demand increase that can be safely supplied without requiring new unit startup. This empirical value is arbitrarily determined from historical power demand.

We will see how the robust optimization framework can be used to seamlessly implement similar constraints by replacing the reserve requirement by a set of robust constraints formed from historical power demand. These historical data and the power demand forecast are used to determine the real power demand. This real power demand is modeled with the scenario
Robust Optimization with Xpress

 uncertainty sets. The figure 6 presents the next day’s historical power demand of the last 6 years alongside with the forecasted power demand.

![Historical power demand and forecast](image)

**Figure 6**: Historical power demand and forecast

### 6.1.2 Robust against the $N - k$ contingency

Typical unit commitment robustness against unit forced outage is achieved by analysing the impact of a contingency resulting in loss of $k$ generating units. If the total lost generation is greater than the upward reserve from the remaining generating units, then the system is at risk and demand cannot be fully sustained. Running such a simulation is an important task and provides insights about the security of the power system. Unfortunately, if it can prove the inability of the schedule to survive $k$, the model will fail to suggest how to improve the schedule to satisfy the safety constraint.

A robust optimization formulation can be used to simulate the loss of $k$ running units and to suggest the startup and shutdown of new units to safely supply the power demand. The *cardinality uncertainty sets* will be used to model the outage state of a unit.

### 6.2 Mathematical formulation

The set $\text{Horizon}$ of consecutive time periods describes the planning horizon of the study. Time periods $t \in \text{Horizon}$ have different lengths $\text{LEN}_t$. Each time period $t \in \text{Horizon}$ is associated with a power demand $\text{DEM}_t$.

The set $\text{Units}$ denotes the set of available power generation units. The minimum power generation level of a unit is $\text{PMIN}_u$, and its maximum capacity is $\text{PMAX}_u$. The startup cost of a single unit is $\text{CSTR}_u$, and the cost of running it at the minimum power generation level is $\text{CMIN}_u$. The marginal power production cost above $\text{PMIN}_u$ for each hour is $\text{CADD}_u$.

Each unit $u$ is associated with three decision variables. The binary variable $\text{start}_{ut}$ equals 1 if the unit $u$ is starting at the beginning of time period $t$. The binary variable $\text{work}_{ut}$ equals 1 if the unit $u$ is up and running during the whole time period $t$. At last, the variable $\text{padd}_{ut}$ is set to the power generation level of the unit $u$ during time period $t$. 
6.2.1 Original Unit Commitment Problem

The objective function, to be minimized, is the expected operation cost. The operation cost is composed of the startup cost and the generation cost. It can be expressed as follows:

\[ \sum_{u \in \text{Units}, t \in \text{Horizon}} CSTR_u \cdot \text{start}_u + \text{LEN}_t \cdot (c_{\text{min}_u} \cdot \text{work}_u + c_{\text{add}_u} \cdot \text{padd}_u) \]

The startup constraints describe the time period(s) during which the unit is starting. They can expressed as follows:

\[ \text{start}_u \geq \text{work}_u - \text{work}_{u-1}, \quad \forall u \in \text{Units}, \text{if} \quad t > 1 \]
\[ \text{start}_u \geq \text{work}_{u1} - \text{work}_{uT}, \quad \forall u \in \text{Units}, \text{if} \quad t = 0 \]

The maximum power generation level constraints limit the power generation level of a unit. These constraints are stated as:

\[ \text{padd}_u \leq (\text{PMAX}_u - \text{PMIN}_u) \cdot \text{work}_u \quad \forall u \in \text{Units}, t \in \text{Horizon} \]

The power balance constraints ensure that the total power production equals the power demand at any time. They are formulated by these equations:

\[ \sum_{u \in \text{Units}} \text{PMIN}_u \cdot \text{work}_u + \text{padd}_u = \text{DEM}_u, \quad \forall t \in \text{Horizon} \]

6.2.2 Load Robust Unit Commitment Problem

The Load Robust Unit Commitment problem extends the Original Unit Commitment problem by constraining the unit commitment to be safe under power demand variation. Historical power demands for the last years are known.

The scenario uncertainty set
Let \( U_{dem} \) be the set of possible future power demands. Let \( \text{Years} \) be the set of past years that should be taken into account, and \( HDEM_{yt} \) the demand for the time period \( t \) of the year \( y \). Then the uncertainty set can be expressed as follows:

\[ U_{dem} = \{ e : \forall t \in \text{Horizon}, \exists y \in \text{Years} : e_t = HDEM_{yt} \} \]

Robust constraints

\[ \sum_{u \in \text{Units}} \text{PMAX}_u \cdot \text{work}_u \geq \text{dem}_t \quad \forall \text{dem} \in U_{dem}, t \in \text{Horizon} \]

The reformulation engine will efficiently handle the resulting, potentially large number of constraints, even with large sets of historical data.

6.2.3 The N-k Contingency-Constrained Unit Commitment Problem

The \( N - k \) Contingency-Constrained Unit Commitment problem extends the Original Unit Commitment problem as to make sure that the committed power generation units can supply the load, even if \( k \) generators are simultaneously forced in outage. The uncertain \( e_{ut} \) represents the forced outage event, and the uncertainty set is the set of units that are in forced outage.
The power generation units cannot efficiently be turned on or off at short notice. At the time of scheduling the units, the demand is not known for sure and we must make sure that the total maximum power output capacity of the committed units is greater than the highest demand forecast. Similarly, the total minimum power generation level must be lower than the lowest demand forecast.

The cardinality uncertainty set
Let $U_{\text{outage}}$ be the set describing groups of combinations of $k$ units in forced outage. This uncertainty set can be expressed as follows:

$$U_{\text{outage}} = \{e : \sum_{u \in \text{Units}} e_u \leq k\}$$

Robust constraints

$$\sum_{u \in \text{Units}} PMAX_u \cdot \text{work}_{ut} \cdot (1 - e_u) \geq DEM_t \quad \forall e \in U_{\text{outage}}, t \in \text{Horizon}$$

$$\sum_{u \in \text{Units}} PMIN_u \cdot \text{work}_{ut} \cdot (1 - e_u) \leq DEM_t \quad \forall e \in U_{\text{outage}}, t \in \text{Horizon}$$

These two robust constraints contain the same uncertain. Because uncertain is used to model worst case situation for a particular constraint, the same uncertain may have different values in each constraint.

6.3 Implementation

6.3.1 The Original Unit Commitment Implementation

The Mosel code printed below implements and solves the original formulation of the unit commitment problem.

```mosel
declarations
  NT = 7 ! Number of time periods
  TIME = 1..NT ! Time periods
  PLANTS = 1..4 ! Power generation plant
  UNITS: set of string ! Power generation units
  LEN, DEM: array(TIME) of integer ! Length and demand of time periods
  PMIN,PMAX: array(PLANTS) of integer ! Min. and max output of a generator type
  CSTART: array(PLANTS) of integer ! Start-up cost of a generator
  CMIN: array(PLANTS) of integer ! Hourly cost of gen. at min. output
  CADD: array(PLANTS) of real ! Cost/hour/MW of prod. above min. level
  PLANT: array(UNITS) of integer ! Unit generation plant
  YEARS: range ! Historical years
  HDEM: array(YEARS,TIME) of integer ! Historical demand profiles
  start: array(UNITS,TIME) of mpvar ! Is generation unit starting ?
  work: array(UNITS,TIME) of mpvar ! Is generation unit up ?
  padd: array(UNITS,TIME) of mpvar ! Production above min. output level
end-declarations

initializations from 'a6electr_ro_simple.dat'
LEN DEM PMIN PMAX CSTART CMIN CADD PLANT HDEM
end-initializations

! Create decision variables
forall(u in UNITS, t in TIME) do
end
```
Robust Optimization with Xpress

\begin{verbatim}
start(u,t) is_binary
work(u,t) is_binary

! Objective function: total daily cost
Cost:= sum(u in UNITS, t in TIME) (CSTART(PLANT(u))*start(u,t) +
LEN(t)*CMIN(PLANT(u))*work(u,t) + CADD(PLANT(u))*padd(u,t))

! Number of generators started per period and per type
forall(u in UNITS, t in TIME)
start(u,t) >= work(u,t) - if(t>1, work(u,t-1), work(u,NT))

! Limit on power production range
forall(u in UNITS, t in TIME)
padd(u,t) <= (PMAX(PLANT(u))-PMIN(PLANT(u)))*work(u,t)

! Power balance
forall(t in TIME)
sum(u in UNITS) (PMIN(PLANT(u))*work(u,t) + padd(u,t)) = DEM(t)

! Symmetry breaker
forall(t in TIME, p in PLANTS)
forall(u1, u2 in UNITS | PLANT(u1) = PLANT(u2) and u1<u2)
work(u1,t) >= work(u2,t)

**** Solving and solution reporting ****

! Solve the original problem
minimize(Cost)
writeln("=== Nominal case ===")
print_solution

! Add robust constraints
robust_demand

! Solve robust the problem
setparam("ROBUST_UNCERTAIN_OVERLAP", true)
minimize(Cost)
writeln("=== Robust against demand variation ===")
print_solution

The following output printing routines are invoked from the model shown above:

**** Print highest loss of load in case of worst case scenario realization ****

procedure print_risk
write("\n", strfmt("Loss of load",20))
forall(t in TIME) do
s:= sum(u in UNITS) (work(u,t).sol*PMAX(PLANT(u))) ! Total capacity
v:= maxlist(DEM(t), max(y in YEARS) HDEM(y,t)) ! Hist. peak demand
if(s < v) then
write(strfmt(v-s,8))
else
write(strfmt("",8))
end-if
end-do
end-procedure

**** Solution printing ****

procedure print_solution
writeln("Total cost: ", getobjval)
write(strfmt("Time period ",-20))
ct:=0
forall(t in TIME) do
write(strfmt(ct,5), ", " , strfmt(ct+LEN(t),2), "+")
end-do
end-procedure
\end{verbatim}
ct+=LEN(t)
end-do

write("\n",strfmt("Up Units",-20))
forall(t in TIME) write(strfmt(sum(u in UNITS) work(u,t).sol,8))
write("\n", strfmt("Gen. Pwr.",20))
forall(t in TIME) write(strfmt(sum(u in UNITS) (work(u,t).sol*PMIN(PLANT(u))+padd(u,t).sol),8))
write("\n", strfmt("Gen. Cap.",20))
forall(t in TIME) write(strfmt(sum(u in UNITS) (work(u,t).sol*PMAX(PLANT(u))),8))
write("\n", strfmt("Res. Dn",20))
forall(t in TIME) write(strfmt(sum(u in UNITS) (padd(u,t).sol),8))
write("\n", strfmt("Res. Up",20))
forall(t in TIME) write(strfmt(sum(u in UNITS) (work(u,t).sol*PMAX(PLANT(u)) - DEM(t),8))

print_risk
writeln
end-procedure

6.3.2 The Load Robust Unit Commitment Implementation

The extension to scenario-based robust optimization is implemented by the following subroutines.

!**** Helper routine to create scenario uncertain set ****
public procedure makescenario(a:array(r1:range,c1:range) of integer,
                            u:array(c2:range) of uncertain)
declarations
    aa:dynamic array(r0:range,U:set of uncertain) of real
end-declarations
forall(i in r1, j in c1|exists(a(i,j)) and exists(u(j)),n as counter) do
    aa(n,u(j)):=a(i,j)
end-do
scenario(aa)
end-procedure

!**** Robust against past scenario ****
public procedure robust_demand
declarations
    demand: array(TIME) of uncertain ! Uncertain power demand
end-declarations
! Add uncertainty set (time correlation)
makescenario(HDEM,demand)
! Security reserve upward
forall(t in TIME) sum(u in UNITS) PMAX(PLANT(u))*work(u,t) >= demand(t)
end-procedure

6.3.3 The N-k Contingency-Constrained Unit Commitment Implementation

Replacing the call to the procedure robust_demand by a call to the procedure robust_contingency printed below will turn the problem formulation into a N-k contingency-constrained unit commitment model.

The example also demonstrates how to use the value of the uncertain to extract meaningful information about the units which are the most critical ones for the system.
procedure robust_contingency(k: integer)
forall(t in TIME, u in UNITS) create(outage(u,t))
forall(t in TIME, u in UNITS) outage(u,t) >= 0
! If the unit is forced in outage, the total capacity
! of the generating units are lost
forall(t in TIME, u in UNITS) outage(u,t) <= 1
forall(t in TIME) sum(u in UNITS) outage(u,t) <= k
forall(t in TIME) cardinality(union(u in UNITS | exists(outage(u,t))) {outage(u,t)}, k)
forall(t in TIME)
RobCtrMaxDem(t) :=
sum(u in UNITS) PMAX(PLANT(u))*work(u,t) -
sum(u in UNITS) PMAX(PLANT(u))*work(u,t)*outage(u,t) >= DEM(t)
forall(t in TIME)
RobCtrMinDem(t) :=
sum(u in UNITS) PMIN(PLANT(u))*work(u,t) -
sum(u in UNITS) PMIN(PLANT(u))*work(u,t)*outage(u,t) <= DEM(t)
end-procedure

6.4 Results

Let us now take a look at the results generated by the three models. The Original Unit Commitment Problem is used as the baseline scenario and the results of the two robust versions are compared against it.

For all three model formulations, a feasible unit commitment which satisfies the technical constraints and supplies the load is found.

The characteristics of the power generation units used in the test instances (number of units per type, minimum and maximum generation levels, fixed cost when running at minimum level, variable cost between minimum and maximum generation levels, start-up cost) are summarized in Table 14.

<table>
<thead>
<tr>
<th>Unit type</th>
<th>Number</th>
<th>Pmin</th>
<th>Pmax</th>
<th>Fix cost</th>
<th>Add. MW cost</th>
<th>Start-up cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>750</td>
<td>1750</td>
<td>2250</td>
<td>2.7</td>
<td>5000</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1000</td>
<td>1500</td>
<td>1800</td>
<td>2.2</td>
<td>1600</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>1200</td>
<td>2000</td>
<td>3750</td>
<td>1.8</td>
<td>2400</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1800</td>
<td>3500</td>
<td>4800</td>
<td>3.8</td>
<td>1200</td>
</tr>
</tbody>
</table>

Table 15 presents results of the nominal optimization problem. For each time period, the number of started units is presented (Up Units), along with the generation power (Gen. Pwr.). The generation capacity (Gen. Cap.) is the total maximum power generation available from the started units. The columns ‘reserve down’ (Res. Dn.) and ‘reserve up’ (Res. Up) respectively show the maximum load decrease or load increase that can be safely supported without requiring startup or shutdown of any units.

For example, during the first period (0-6) the load can increase from 12000 kWh (power demand forecast) to 13250 kWh (power demand forecast + reserve up) without having to start up new
units.

The last two lines present the maximum loss-of-load in case the worst scenario realizes. For the demand variation, the worst scenario is the realization of the load profile that requires the highest power generation increase. For the contingency planning, the worst scenario occurs when the $k$ ‘critical’ units are forced in outage. The criticality of a unit depends on its power generation level and total reserve. A unit without reserve up is critical, as is a unit supplying a very large share of the total load.

Table 15: Nominal case (Total cost: 1.406 Million)

<table>
<thead>
<tr>
<th></th>
<th>0-6</th>
<th>6-9</th>
<th>9-12</th>
<th>12-14</th>
<th>14-18</th>
<th>18-22</th>
<th>22-24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up units</td>
<td>8</td>
<td>18</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Gen. Pwr.</td>
<td>12000</td>
<td>32000</td>
<td>25000</td>
<td>36000</td>
<td>25000</td>
<td>30000</td>
<td>18000</td>
</tr>
<tr>
<td>Gen. Cap.</td>
<td>13250</td>
<td>37750</td>
<td>27250</td>
<td>44750</td>
<td>27250</td>
<td>34250</td>
<td>19250</td>
</tr>
<tr>
<td>Res. Dn</td>
<td>3850</td>
<td>10050</td>
<td>8450</td>
<td>10450</td>
<td>8450</td>
<td>9850</td>
<td>6250</td>
</tr>
<tr>
<td>Res. Up</td>
<td>1250</td>
<td>5750</td>
<td>2250</td>
<td>8750</td>
<td>2250</td>
<td>4250</td>
<td>1250</td>
</tr>
</tbody>
</table>

Loss of load

<table>
<thead>
<tr>
<th>Demand variation</th>
<th>0</th>
<th>1395</th>
<th>1491</th>
<th>0</th>
<th>2592</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Contingencies</td>
<td>2750</td>
<td>1250</td>
<td>1750</td>
<td>0</td>
<td>1750</td>
<td>2750</td>
<td>2750</td>
</tr>
</tbody>
</table>

If the nominal case is applied, then in case of demand variation, there is a risk of not supplying 1395 kWh of demand during the morning peak. Common sense would be to start up a new unit in order to cover the risk for all time periods when there is a risk of loss of load. However, the results from the robust formulation for the demand variation problem presented in Table 16 show that it is possible to overcome the risk merely by changing the type of committed units.

Table 16: Robust against demand variation (Total cost: 1.409 Million)

<table>
<thead>
<tr>
<th></th>
<th>0-6</th>
<th>6-9</th>
<th>9-12</th>
<th>12-14</th>
<th>14-18</th>
<th>18-22</th>
<th>22-24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up Units</td>
<td>8</td>
<td>18</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Gen. Pwr.</td>
<td>12000</td>
<td>32000</td>
<td>25000</td>
<td>36000</td>
<td>25000</td>
<td>30000</td>
<td>18000</td>
</tr>
<tr>
<td>Gen. Cap.</td>
<td>13250</td>
<td>39250</td>
<td>28750</td>
<td>44750</td>
<td>30250</td>
<td>37250</td>
<td>19250</td>
</tr>
<tr>
<td>Res. Dn</td>
<td>3850</td>
<td>9450</td>
<td>7850</td>
<td>10450</td>
<td>7250</td>
<td>8650</td>
<td>6250</td>
</tr>
<tr>
<td>Res. Up</td>
<td>1250</td>
<td>7250</td>
<td>3750</td>
<td>8750</td>
<td>5250</td>
<td>7250</td>
<td>1250</td>
</tr>
</tbody>
</table>

Loss of load

<table>
<thead>
<tr>
<th>Demand variation</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Contingencies</td>
<td>2750</td>
<td>1250</td>
<td>1750</td>
<td>0</td>
<td>1750</td>
<td>2750</td>
<td>2750</td>
</tr>
</tbody>
</table>

The loss of load for the 2 contingencies case highlights that there is a risk of not being able to supply 2750 kWh during the initial periods. It means that about 20% of the total demand will not be covered. Indeed there is only 1250 kWh of upward reserve available, which means that even if just a single unit running at full capacity is lost (1500 kWh) then the power demand cannot be
supplied. Like in the previous case, the common sense conclusion would be to start up new units. However, the results from the robust formulation for the 2 contingencies problem presented in Table 17 show that it is possible to find a unit commitment schedule without risking load disconnection and without committing too many new units (and hence incurring higher startup costs).

6.5 References

The presented nominal unit commitment model is based on a problem described in the book 'Applications of optimization with Xpress-MP' [GHPS02].
7 Production planning under energy supply uncertainty

7.1 Problem description

A company produces liquid nitrogen and liquid oxygen (we will call them LIN and LOX in the remainder) and needs to plan production for the next $N$ periods, identified as shifts of 8 hours each. Although the resource procurement is rather trivial (these two elements compose most of the troposphere), the process of obtaining these two liquid gases requires a vast amount of electricity, both for refrigerating the stored liquid gases and for powering the plant.

Given that the energy cost is the largest component of the production cost, power suppliers offer several types of energy supply contracts; one of these is known as Interruptible Load Contracts (or ILC for short) and allows the power supplier to interrupt, with a short notice (a few minutes), the provision of electricity to a large customer such as a plant for a limited period of time; most likely the power supplier will take advantage of this in times of high electricity demand, such as hot summer days. However, the ILC requires that there be no more than $K$ interruptions throughout the planning horizon.

We are dealing therefore with a production planning problem under uncertainty in the power supply. We are given in input the production and inventory cost, the maximum production, the inventory capacity, the initial inventory, the demand in LIN and LOX for each period, and the maximum number $K$ of interruptions. The problem consists of finding the amount of LIN and LOX to be produced every day so that the demands are satisfied irrespective of the interruptions that may occur within the clause of the contract.

This optimization problem requires the application of Robust Optimization for one reason among several: among the customers of this company there are hospitals, for which the fulfilment of the demand of LOX is imperative.

7.2 Mathematical formulation

We are given the set of gases $G = \{\text{lin, lox}\}$ and the set of time periods $T = \{1, 2, \ldots, N\}$, the maximum production level $P_{\text{lin}}$ and $P_{\text{lox}}$ for each period and the inventory capacity $S_{\text{lin}}$ and $S_{\text{lox}}$ for each gas. The demand is given by the parameter $\text{dem}_{tg}$ for $t \in T, g \in G$.

The uncertainty set is defined by every vector of interruptions $(\xi_t)_{t \in T}$ such that $\xi_t \in [0, 1]$ and not more than $K$ of its elements are positive:

$$\sum_{t \in T} \xi_t \leq K.$$

Let us define the variables $\text{prod}_{tg}$ and $\text{inv}_{tg}$ representing the production and inventory level, respectively, at period $t$ for gas $g$. Without uncertainty in the input data, the model would have simple of constraints: bounds on the variables and fixing of the initial inventory level:

$$0 \leq \text{prod}_{tg} \leq P_g \quad \forall t \in \{1, 2, \ldots, N\}$$

$$0 \leq \text{inv}_{tg} \leq S_g \quad \forall t \in \{1, 2, \ldots, N\}$$

$$0 \leq \text{inv}_{0g} = I_g \quad \forall g;$$
and the production planning constraint would consist of a mass conservation constraint for every gas \( g \) and time period \( t \):

\[
inv_{t-1,g} + prod_{tg} = inv_{tg} + dem_{tg} \quad \forall t \in T, g \in G.
\]

The modeling is more involved for this case. A first attempt would be to write the robust constraint by multiplying the production variable by \( (1 - \xi_t) \), so that the actual production at time period \( t \) is zero if \( \xi_t = 1 \):

\[
inv_{t-1,g} + (1 - \xi_t) prod_{tg} = inv_{tg} + dem_{tg} \quad \forall t \in T, g \in G.
\]

However, this would yield no feasible solution: for each robust constraint related to production at time period \( t \) the uncertain \( \xi_t \) would be set to one, and no production would occur. We need to aggregate the production constraints in such a way that the balance between production and demand yields an inventory that is nonnegative at every time period. The above constraint is replaced by the following one:

\[
inv_{0,g} + \sum_{\tau \in T: \tau \leq t} ((1 - \xi_{\tau}) prod_{\tau g} - dem_{\tau g}) \geq 0 \quad \forall t \in T, g \in G.
\]

This robust constraint considers uncertainty in the interruptions but allows for a production plan that satisfies all demands even with \( K \) interruptions. In order to deal with contour conditions and take into account the initial inventory, we add a slight modification to the uncertainty set and require that no interruption happens at time period 1:

\[
\xi_1 = 0.
\]

Finally, the robust value of the inventory must take into account the situation in which, after planning the production, no interruption actually occurs: this is simply enforced by bounding the inventory at every time period as the sum of all accumulated production discounted by the accumulated demand:

\[
inv_{tg} \geq inv_{0,g} + \sum_{\tau \in T: \tau \leq t} (prod_{\tau g} - dem_{\tau g}) \quad \forall t \in T, g \in G.
\]

### 7.3 Implementation

Below is the implementation in Mosel: note that the option ROBUST_OVERLAP_UNCERTAIN is again used since the nonnegative-inventory constraints use the same set of uncertain multiple times.

```mosel
model robust_prodplan
uses "mmrobust"
parameters
  plan_data = "prodplan_robust.dat"
end-parameters
declarations
  NDAYS: integer ! Planning horizon
  PERDAY: integer ! Number of periods per day
  NPERIODS: integer ! Total number of periods
end-declarations
```
PERIODS, PERIODS0: range ! Time periods
GASES: set of string ! Set of products

PROD_CAP: array(GASES) of real ! Production capacity every day
INV_CAP: array(GASES) of real ! Inventory capacity
INV_0: array(GASES) of real ! Initial inventory

PROD_COST: real ! Production cost
INV_COST: real ! Inventory cost

DEMAND: array (PERIODS, GASES) of real ! Demand of each gas every day

MAX_NINTERR: integer ! Maximum number of interruptions ! (as per contract)

end-declarations

!**** Initialize data ****
initializations from plan_data
NDAYS PERDAY
PROD_CAP INV_CAP
PROD_COST INV_COST INV_0
DEMAND
MAX_NINTERR
end-initializations

NPERIODS := NDAYS * PERDAY
PERIODSO := 0..NPERIODS

!**** Problem formulation ****
declarations
produce: array(PERIODS, GASES) of mpvar ! Production every day
inventory: array(PERIODS0, GASES) of mpvar ! Inventory level every day,
! including initial level

interruption: array(PERIODS) of uncertain ! Is power cut at this time?
RobProd: array(PERIODS, GASES) of robustctr ! Robust constraint dealing with uncertainty
end-declarations

!**** Constraints ****

! Start inventory
forall(g in GASES)
  inventory (0,g) = INV_0 (g)

! Inventory balance
forall(t in PERIODS, g in GASES)
do
  RobProd(t,g) := inventory(0,g) + sum(tp in PERIODS | tp <= t)
  \((1 - interruption(tp)) \times produce(tp,g) - DEMAND(tp,g)\) >= 0
  inventory (0,g) + sum (tp in PERIODS | tp <= t)
  (produce(tp,g) - DEMAND(tp,g)) <= inventory(t,g)
  inventory(t,g) <= INV_CAP(g)
  produce(t,g) <= PROD_CAP(g)
dendo

! Interruptions of production
forall (t in PERIODS)
do
  interruption (t) <= 1
  interruption (t) >= 0
dendo

sum(t in PERIODS) interruption (t) <= MAX_NINTERR
interruption(1) = 0
7.4 Input Data

For this example, we consider a simple instance of the problem where the planning has a horizon of five days and three periods per day. The cost of production and inventory is 4 and 3, respectively, and the ILC contract allows for four interruptions. Table 18 below shows the production and inventory capacity and the initial inventory level for both gases.

<table>
<thead>
<tr>
<th>Gas</th>
<th>Prod. capacity</th>
<th>Inv. capacity</th>
<th>Initial inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIN</td>
<td>29</td>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>LOX</td>
<td>5.5</td>
<td>27</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 19 below lists the demand for each gas over all 15 periods.

7.5 Results

The optimal solution of this problem has cost 4727.17. Tables 20 and 21 give the optimal levels of
production required in order to satisfy the demands regardless of when the interruptions occur, if ever. Note that the production levels are higher at the beginning and then, for liquid oxygen, they equal the demand. This depends on the fact that production at the initial time periods must assume for worst-case scenarios such as $K = 4$ consecutive interruptions early in the planning horizon. The inventory levels equal the balance between demand and production as they must account for the event that no interruption occurs.

Table 20: Production of LIN

<table>
<thead>
<tr>
<th>Time</th>
<th>Dem</th>
<th>Prod</th>
<th>Inv*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>29</td>
<td>43</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>15</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>15</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>15</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 21: Production of LOX

<table>
<thead>
<tr>
<th>Time</th>
<th>Dem</th>
<th>Prod</th>
<th>Inv*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5.5</td>
<td>23.5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5.2</td>
<td>23.7</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5.2</td>
<td>25.8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5.2</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>5.2</td>
<td>24.2</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5.2</td>
<td>25.3</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>5.2</td>
<td>22.5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>5.2</td>
<td>20.7</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>5.2</td>
<td>21.8</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>5.2</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>5.2</td>
<td>26.2</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>5.2</td>
<td>26.3</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>5.2</td>
<td>22.5</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>5.2</td>
<td>20.7</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>5.2</td>
<td></td>
</tr>
</tbody>
</table>

Note that this is the result of the optimization problem solved at the beginning of the time period. Using a rolling horizon approach, i.e., re-solving the model at every time period while taking into account the interruptions that already occurred, would allow us to obtain a less expensive production plan although the planned production levels might have to change.

Finally, since the interruption of a plant at a given period affects the production of both gases, the retrieval of uncertain needs to be done for each gas and getsol takes both interruption and the robust constraint RobProd as arguments. As would be expected, most worst-case interruptions are at the initial time periods, since those are critical moments when the LIN and LOX tanks need to be built up.

Worst-case interruptions for LIN:
2 3 4 5 6 14

Worst-case interruptions for LOX:
2 3 4 5 6 8 10 15

7.6 References

This example is a simplified version of a real-world application that has been documented in [LBS13].

Bibliography


