Whitepaper

FICO

FICO^{TM} Xpress
Optimization Suite

Robust Optimization with Xpress

Usage guidelines and examples

FICO^{TM} Xpress Optimization Suite whitepaper

Last update 23 April, 2014

Robust Optimization with FICO^{TM} Xpress

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 7.7
23 April, 2014

Abstract

This whitepaper gives an introduction to formulating and solving Robust Optimization problems with FICO^{TM} Xpress.
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/robust/Mosel` directory of the Xpress installation.

Contents

1 Introduction . . . 2
1.1 Uncertains and robust constraints . . . 3
1.2 Types of robust constraints . . . 3
1.2.1 Simple bounds on the uncertain coefficients . . . 4
1.2.2 Linear constraints on the uncertainty sets . . . 6
1.2.3 Ellipsoidal uncertainty sets . . . 7
1.2.4 Equality constraints and uncertain values with no uncertain constraints 8
1.2.5 Mixing uncertainty sets and types . . . 9
1.2.6 Cardinality restrictions for uncertains . . . 10
1.2.7 Using historical data - scenarios . . . 10
1.2.8 Uncertainty in the objective . . . 12
1.3 Nominal values: centered and uncentered uncertainty . . . 12
1.3.1 The price of robustness . . . 13
1.3.2 Working with nominal values . . . 14
1.3.3 Using nominal values to shift the uncertainty set . . . 15
1.3.4 Using nominal valued uncertains as coefficients . . . 16
1.4 Examples of robust models . . . 17
2 Robust shortest path . . . 18
2.1 Problem description . . . 18
2.2 Mathematical formulation . . . 18
2.2.1 Shortest path problem . . . 18
2.2.2 Robust optimization problem . . . 19
2.3 Implementation . . . 20
2.4 Results . . . 21

3 Production planning under demand uncertainty 24
3.1 Problem description 24
3.2 Mathematical formulation 24
3.2.1 Multi-period, multi-item production planning problem 24
3.2.2 Robust optimization problem 25
3.3 Implementation 26
3.4 Results 28
4 Robust network design 30
4.1 Problem description 30
4.2 Mathematical formulation 30
4.3 Implementation 31
4.4 Input Data 33
4.5 Results 33
5 Robust portfolio optimization 35
5.1 Problem description 35
5.2 Mathematical formulation 35
5.2.1 Highest protection 36
5.2.2 Budgeted protection 36
5.3 Implementation 36
5.4 Results 39
5.4.1 Input Data 39
5.4.2 Analysis 40
5.5 References 41
6 Robust unit commitment 42
6.1 Problem description 42
6.1.1 Robust against power demand variation 42
6.1.2 Robust against the N-k contingency 43
6.2 Mathematical formulation 43
6.2.1 Original Unit Commitment Problem 44
6.2.2 Load Robust Unit Commitment Problem 44
6.2.3 The N-k Contingency-Constrained Unit Commitment Problem 44
6.3 Implementation 45
6.3.1 The Original Unit Commitment Implementation 45
6.3.2 The Load Robust Unit Commitment Implementation 47
6.3.3 The N-k Contingency-Constrained Unit Commitment Implementation 47
6.4 Results 48
6.5 References 49
7 Production planning under energy supply uncertainty 50
7.1 Problem description 50
7.2 Mathematical formulation 50
7.3 Implementation 51
7.4 Input Data 53
7.5 Results 53
7.6 References 54
Bibliography 54

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 to 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 in the model 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 for 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:

$$a_1x_1 + \ldots + a_lx_l + (a_k + u_k)x_k + \ldots + (a_n + u_n)x_n \leq b.$$

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

$$u_k^2 + \ldots + u_n^2 \leq r.$$

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

$$a_1x_1 + \ldots + a_lx_l + max_{u_k^2+\ldots+u_n^2 \leq r}(a_k + u_k)x_k + \ldots + (a_n + u_n)x_n \leq b.$$

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 uncertains either multiplied by a variable or at the right-hand side. The type of a robust constraint is defined by how the uncertainty set of affecting it is described. We shall work with a small knapsack problem example and explore alternative formulations of the uncertainty set.

```
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)
```

This basic model without uncertainty solves to 256.601.

```
 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. Introducing uncertains to the model is as follows.

```
 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
  WeightUncertanty: 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
 WeightUncertanty(i) <=  WTPERCENT*WEIGHT(i) ! Uncertainty is a percentage of
                                             ! the expected weight
 WeightUncertanty(i) >=  0                   ! and only expected to go up this time
end-do
WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertanty(i))*x(i) <= WTMAX
                                             ! Weight restriction

maximize(MaxVal)
```

The new model expects 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 uncertains that would violate the robust constraint. The objective has decreased significantly because of uncertainty; the difference in objective functions between the original problem and the robust version is often referred to as the *price of robustness* [BS04]. 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 uncertains, 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 uncertains 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
```

1.2.2 Linear constraints on the uncertainty sets

Bound constraints are only an example of the possible uncertainty sets we can model. Let us suppose that some or all of the uncertains 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 uncertains is at most 10% 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
 WeightUncertanty: 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
 WeightUncertanty(i) >=  0
end-do

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

maximize(MaxVal)
```

While this sounds reasonable, because of the large right-hand side (10% of the total weight) the modified model is subject to a very conservative uncertainty set and will solve to a solution of all zeros.

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

This behavior highlights how robust optimization works. When moving from the previous uncertainty set, with a small upper bound on the uncertains, to the new uncertainty set, we have in fact relaxed the uncertainty set and have made the feasible space of the robust opponent significantly larger, making 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 behaviour 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 uncertains 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 a, a covariance matrix Q, and a level of confidence α. In this case, the uncertainty set is of the form $(u-a)^T Q(u-a) \leq \alpha$.

```
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
 WeightUncertanty: 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

sum(i in Items) WeightUncertanty(i)^2 <= WTPERCENT * sum(i in Items) WEIGHT(i)
WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertanty(i))*x(i) <= WTMAX
                                             ! Weight restriction

maximize(MaxVal)
```

This model gives the desired solution as shown below.

```
 Objective: 215.445
Item  Weight  Value
1: 0   74.37  118.04
2: 0   62.34  141.75
3: 1   27.74  114.85
4: 1   53.17  100.60
5: 0   74.02   65.82
```

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.

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
end-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` equals 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 equality constraint's case.

```
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, it is important to note that 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 between different robust constraints. This restriction applies to all uncertains that are connected to different robust constraints through any uncertain constraints, not only if appearing directly. 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 behaviour if the same uncertain set description is to be used for multiple robust constraints.

Consider the following example.

```
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 $e + f \leq 1$ 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.

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

x*e <= 1
y*f <= 1

e >= 0
```

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

setparam("XPRS_PROBNAME","h")

setparam("ROBUST_UNCERTAIN_OVERLAP", true);
```

However, the solution returned is x=1=y, which can easily be seen as non-optimal if $e + f \leq 1$ is satisfied. What happens is that 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.

1.2.6 Cardinality restrictions for uncertains

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

In the next example we show how to use uncertains in multiple robust constraints (i.e. in uncertains 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
  S={ex,ey,ez}
end-declarations

x is_binary ; y is_binary ; z is_binary

20*x + 10*y + 5*z >= 20

ex>=0 ; ex<=1
ey>=0 ; ey<=1
ez>=0 ; ez<=1

cardinality({ex,ey,ez},1)   ! Allow only one of the uncertains to be nonzero

setparam("ROBUST_UNCERTAIN_OVERLAP",true)   ! allow overlaps

10*(x-x*ex) + 10*(y-y*ey) + 10*(z-z*ez) >= 10
10*(x-x*ex) + 10*(y-y*ey) + 10*(z-z*ez) <= 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 uncertains. Suppose we have a vector *u* of uncertains. 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 shonwn in the next extension of the knapsack problem analyzed before where a number of historical measurements have been taken.

```
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

! Generate historical data, this would be data collected from actual realizations
forall(period in 1..HISTORIC_PERIODS, i in Items)
  historical_weights(period, UncertainWeight(i)) :=
    WEIGHT(i)*UNCERTAINTY_LEVEL*random
! Generate a solution that would be feasible for ALL historic realizations
scenario(historical_weights)

maximize(MaxVal)
```

It is beneficial to examine the deterministic equaivalent to a scenario based robust model. Consider the following simple scenario based model.

```
declarations
 x, y : mpvar
 e, f : uncertain
 historical_data: array (range, set of uncertain) of real
end-declarations

! load historical data for e and f
historical_data(1, e) := 1
historical_data(1, f) := 2
historical_data(2, e) := 3
historical_data(2, f) := 1
historical_data(3, e) := 3
historical_data(3, f) := 2

e*x + f * y <= 1

! Generate a solution that would be feasible for ALL historic realizations
scenario(historical_data)

maximize(x+y)
```

Its deterministic equivalent is:

```
declarations
 x, y : mpvar
 e, f : uncertain
 historical_data: array (range, set of uncertain) of real
end-declarations


scenario1 := 1 * x + 2 * y <= 1
scenario2 := 3 * x + 1 * y <= 1
scenario3 := 3 * x + 2 * y <= 1

! Generate a solution that would be feasible for ALL historic realizations
scenario(historical_data)

maximize(x+y)
```

1.2.8 Uncertainty in the objective

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 uncertains, 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.

It is possible to work with the uncertains as actual uncertain coefficients. 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., uncertanty with zero nominal value:

$$(5 + uncertain)x \leq 1$$

is expressed as:

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

(5+u)*x <= 1
```

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

$$(5 + uncertain)x \leq 1$$

can also be expressed as:

```
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.

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

0 <= u
     u <= 3

(1+u)*x <= 1

maximize(XPRS_NOMINAL, x)
nominal_objective := getobjval
writeln("Objective at nominal values:", nominal_objective );

maximize(x)
robust_objective := getobjval
writeln("Robust objective:", robust_objective );

writeln("Price of robustness:", nominal_objective - robust_objective );
```

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:

```
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.

```
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.

Rule2: a changed nominal value only affects robust and uncertainty set constraints that are defined after it has been assigned.

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

```
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 has been used (3), and redefining it later had no effect on the constraint. This behaviour is reproduced when working with the nominal values of uncertains.

```
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 uncertains, 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:

```
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 uncertains `e` and `f` can take their value from a ball of radius $\sqrt{2}$. When nominal values are added, the model becomes:

```
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:

```
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 uncertains 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 uncertains 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:

Case 1 **No nominal values are set**	**Case 2** **Nominal values are set to 1**	**Case 3** **Nominal values are set to different values**
declarations x,y : mpvar e,f : uncertain end-declarations $e + f \leq 2$ $e*x + f*y \leq 1$ maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations e := 1 f := 1 $e + f \leq 2$ $e*x + f*y \leq 1$ maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations e := 2 f := 5 $e + f \leq 2$ $e*x + f*y \leq 1$ maximize(x + y)

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:

Case 3	becomes	which is
declarations x,y : mpvar e,f : uncertain end-declarations e := 2 f := 5 $e + f \leq 2$ $e*x + f*y \leq 1$ maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations $e+2 + f+5 \leq 2$ $(e+2)*x + (f+5)*y \leq 1$ maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations $e + f \leq -5$ $(e+2)*x + (f+5)*y \leq 1$ maximize(x + y)

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:

The nominal equivalent of case 1:	**The nominal equivalent of case 2:**	**The nominal equivalent of case 3:**
declarations x,y : mpvar e,f : uncertain end-declarations 0*x + 0*y $\leq$ 1 maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations 1*x + 1*y $\leq$ 1 maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations 2*x + 5*y $\leq$ 1 maximize(x + y)
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 Xpress 7.7 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.

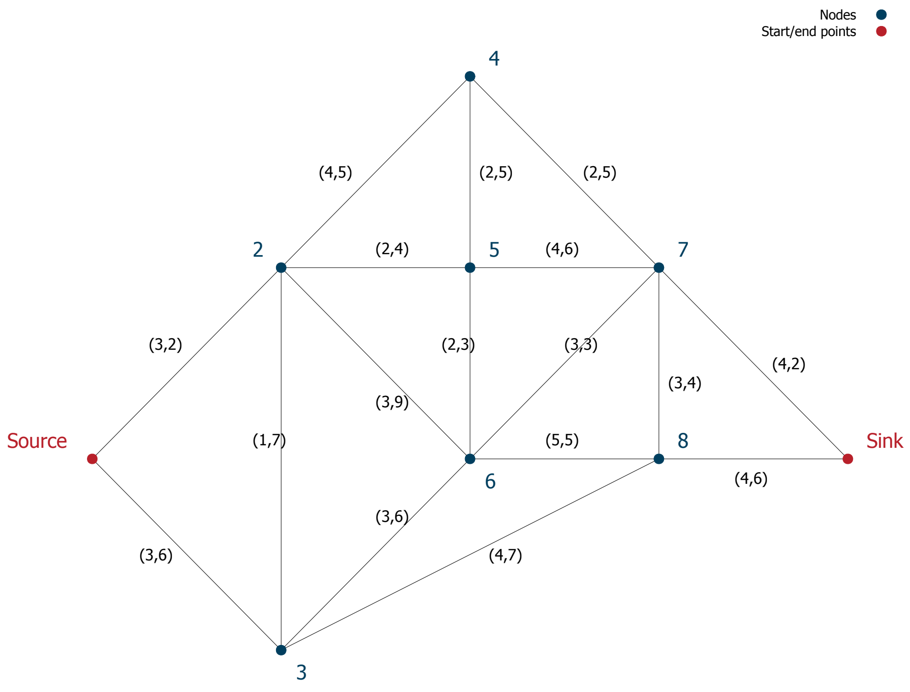

Figure 1: Road network

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_a associated with each arc a where $use_a = 1$ 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 $ARC_{a,1}$ and its destination node $ARC_{a,2}$.

$$
\begin{array}{ll}
\min & \sum_{a \in Arcs} LEN_a \cdot use_a \\
\text{s.t.} & \sum_{a \in Arcs | ARC_{a,1}=Source} use_a = 1, \quad \sum_{a \in Arcs | ARC_{a,2}=Sink} use_a = 1 \\
& \sum_{a \in Arcs | ARC_{a,2}=Source} use_a = 0, \quad \sum_{a \in Arcs | ARC_{a,1}=Sink} use_a = 0 \\
& \forall n \in Nodes - \{Source, Sink\} : \sum_{a \in Arcs | ARC_{a,2}=n} use_a = \sum_{a \in Arcs | ARC_{a,1}=n} use_a \\
& \forall a \in Arcs : use_a \in \{0, 1\}
\end{array}
$$

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:

$$
\min \quad \sum_{a \in Arcs} (LEN_a + MAXDELAY_a) \cdot use_a
$$

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 $a \in Arcs$ we can state the cardinality constrained uncertainty set as follows:

$$
U = \{delay : |\{delay_a : delay_a > 0\}| \le N, 0 \le delay_a \le MAXDELAY_a \forall a \in Arcs\}
$$

The resulting *robust optimization problem* then has the following form.

$$
\begin{array}{ll}
\min & \sum_{a \in Arcs} (LEN_a + delay_a) \cdot use_a \\
\text{s.t.} & \sum_{a \in Arcs | ARC_{a,1} = Source} use_a = 1, \quad \sum_{a \in Arcs | ARC_{a,2} = Sink} use_a = 1 \\
& \sum_{a \in Arcs | ARC_{a,2} = Source} use_a = 0, \quad \sum_{a \in Arcs | ARC_{a,1} = Sink} use_a = 0 \\
& \forall n \in Nodes - \{Source, Sink\} : \sum_{a \in Arcs | ARC_{a,2} = n} use_a = \sum_{a \in Arcs | ARC_{a,1} = n} use_a \\
& \forall a \in Arcs : use_a \in \{0, 1\} \\
& delay \in U = \{delay : |\{delay_a : delay_a > 0\}| \leq N, 0 \leq delay_a \leq MAXDELAY_a \forall a \in Arcs\}
\end{array}
$$

2.3 Implementation

The following Mosel model implements the robust optimization model from the previous section. Notice the use of `cardinality` on the uncertains `delay` to limit the total number of occurrences of roadworks to at most `NWORK`, the parameter for $N = 2$. The uncertains that are used in such a `cardinality` constraint need to have explicit lower and upper bounds set.

```
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)

!**** Solving ****
 minimize(TotalLength)

! Solution reporting
 if getprobstat=XPRS_OPT then
   writeln("Robust shortest path: ", getobjval)
   writeln("Delay: ", getobjval - sum(a in Arcs) LEN(a)*use(a).sol)
   forall(a in Arcs | use(a).sol>0)
     writeln(if(ARC(a,1)=Source, "Source", string(ARC(a,1))), " -> ",
             if(ARC(a,2)=Sink, "Sink", string(ARC(a,2))), ";  ")
 else
   writeln("No solution")
 end-if

end-model
```

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 6–7 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 5–6), any other combination of at most two roads with construction will result in a total duration of 21 minutes or less.

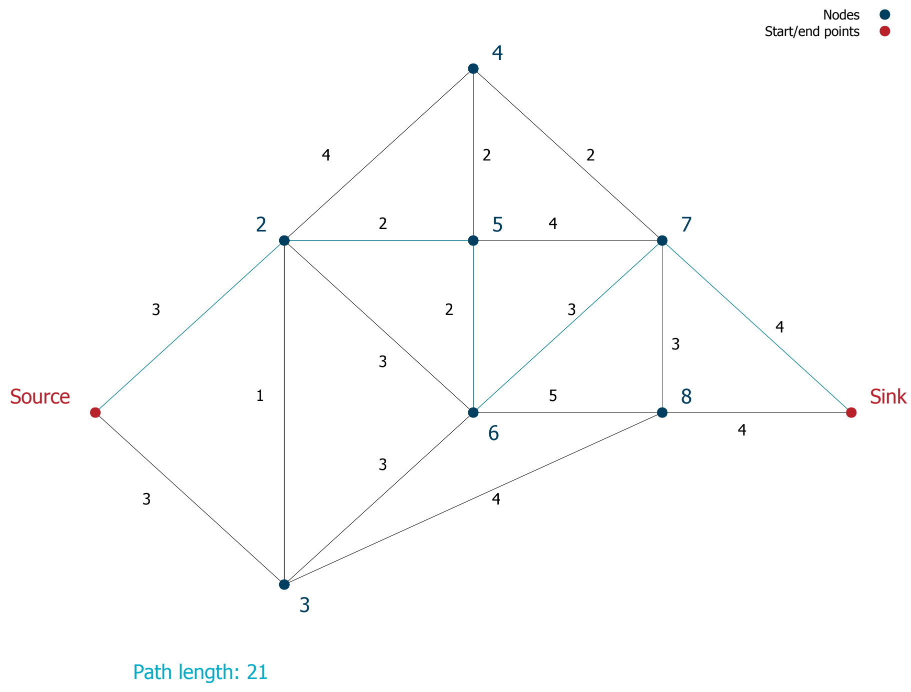

Figure 2: Robust solution

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.

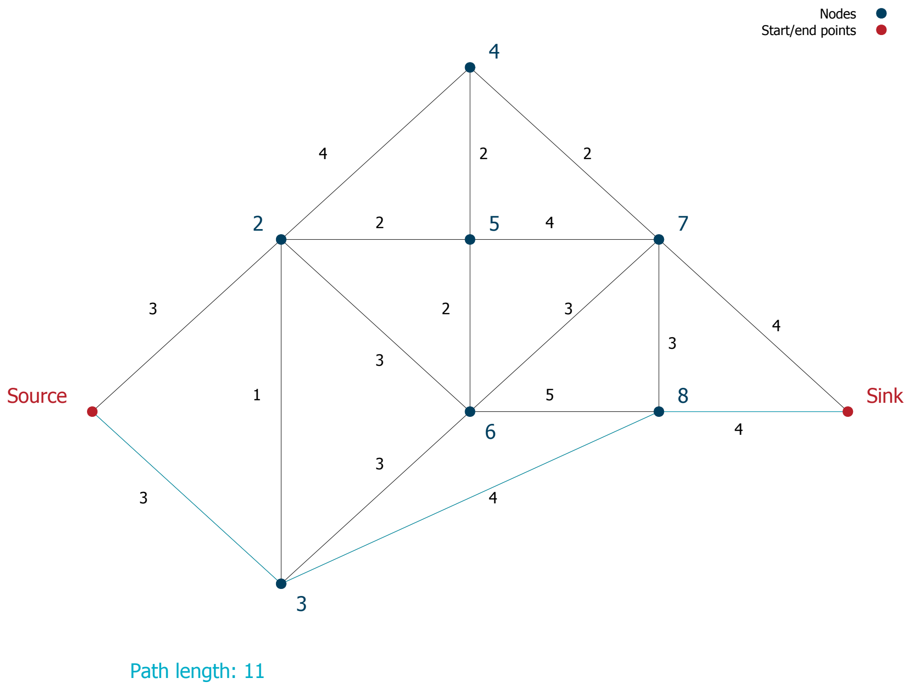

Figure 3: Zero-delay solution

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 travel time for the three approaches (robust, shortest path with nominal values, shortest path where all roads have maximum delay) in three cases: no road work, road work in $N = 2$ edges of the graph, and road work everywhere, *i.e.* N is infinite.

Table 1: Path costs

N	**0**	**2**	**+inf**
Robust	14	**21**	29
Nominal	**11**	24	30
All delays	13	23	**27**

Note that, under the assumption we have made at the beginning of this section, the only solution

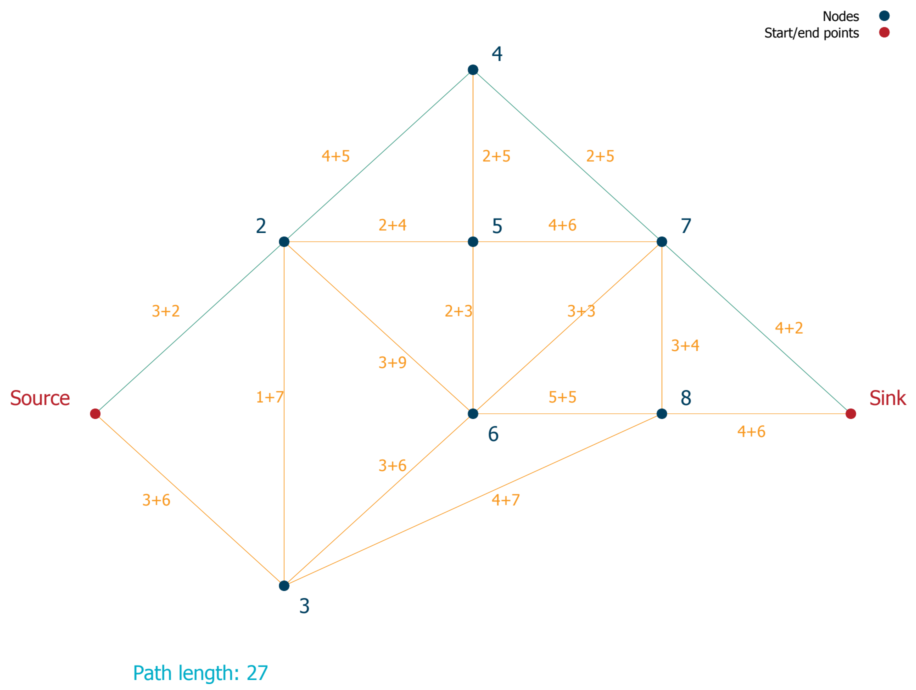

Figure 4: Maximum-delay solution

that truly solves the problem with minimum worst-case travel time is the one that was found using the uncertains 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:

$$\forall p \in PRODS : store_{p,NT} \geq FSTOCK_p$$

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:

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

$$\forall t \in WEEKS : \sum_{p \in PRODS} TIMEM_p \cdot produce_{pt} \leq CAPM$$

$$\forall t \in WEEKS : \sum_{p \in PRODS} SPACE_p \cdot produce_{pt} \leq CAPS$$

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

$$\min \sum_{p \in PRODS} \sum_{t \in WEEKS} \left(CPROD_p \cdot produce_{pt} + CSTORE_p \cdot store_{pt} \right)$$

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.

$$\begin{array}{ll}
\min & \sum_{p \in PRODS} \sum_{t \in WEEKS} \left(CPROD_p \cdot produce_{pt} + CSTORE_p \cdot store_{pt} \right) \\
\text{s.t.} & \forall p \in PRODS, t \in WEEKS : store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt} \\
& \forall p \in PRODS : store_{p,NT} \geq FSTOCK_p \\
& \forall t \in WEEKS : \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW \\
& \forall t \in WEEKS : \sum_{p \in PRODS} TIMEM_p \cdot produce_{pt} \leq CAPM \\
& \forall t \in WEEKS : \sum_{p \in PRODS} SPACE_p \cdot produce_{pt} \leq CAPS \\
& \forall p \in PRODS, t \in WEEKS : produce_{p,t} \geq 0 \\
& \forall p \in PRODS, t \in WEEKS : store_{p,t} \geq 0
\end{array}$$

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 uncertains $absent_t$ 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_{absent} is characterized by

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

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 $SCENDEM_{spt}$ for a given set of scenarios $s \in SCEN$. The demand s included in the original model formulation via the stock balance constraints.

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

A naive 'robustification' might attempt to simply replace the fixed demand quantities by the uncertains $demand_{pt}$. 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.

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

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.

```
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 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.

```
 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 2: Data for six glass types

	Production cost	Storage cost	Initial stock	Final stock	Time_{worker}	$\text{Time}_{machine}$	Storage space
V1	100	25	50	10	3	2	4
V2	80	28	20	10	3	1	5
V3	110	25	0	10	3	4	5
V4	90	27	15	10	2	8	6
V5	200	10	0	10	4	11	4
V6	140	20	10	10	4	9	9

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

	Week	1	2	3	4	5	6	7	8	9	10	11	12
V1	Prod.	8.8	5.5	0.6	30.2	27.4	8.6	23	20	29	30	28	42
	Store	*38.8*	*22.2*	*4.8*	-	*10.4*	–	–	–	–	–	–	*10*
V2	Prod.	0	16	23	20	11	10	12	34	21	23	30	22
	Store	*3*	–	–	–	–	–	–	–	–	–	–	*10*
V3	Prod.	18	35	17	10	9	21	23	15	10	0	13	27
	Store	–	–	–	–	–	–	–	–	–	–	–	*10*
V4	Prod.	16	45	24	38	41	20	19	37	28	12	30	47
	Store	–	–	–	–	–	–	–	–	–	–	–	*10*
V5	Prod.	47.7	14.6	35.1	14.3	23.5	22.8	43.8	0	26.5	2.8	0	0
	Store	*24.7*	*19.3*	*31.4*	*30.8*	*44.2*	*45*	*70.8*	*40.75*	*39.25*	*35*	*20*	*10*
V6	Prod.	12	18	20	19	18	35	0.8	27.2	12	49	29.2	5.8
	Store	–	–	–	–	–	–	*0.75*	–	–	*19*	*27.25*	*10*
Workers		351	390	390	390	390	390	390	390	390	390	390	390
Machines		850	850	850	753.2	850	836.8	790	675.2	742.5	650.2	641.2	641.8
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

Week	1	2	3	4	5	6	7	8	9	10	11	12
Workers	243	378	370	407	325	398	383	414	412	445	429	437
CAPW-ABSENCE	*437*	*437*	*429*	*429*	*414*	*398*	*398*	*414*	*445*	*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

Week	1	2	3	4	5	6	7	8	9	10	11	12
Workers	363	425	425	425	425	425	425	425	425	425	425	425
Machines	850	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 wit 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

Week	1	2	3	4	5	6	7	8	9	10	11	12
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 $V = \{1, \dots, n\}$ denotes the set of n nodes and $A = \{(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m)\}$ 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_{ij}^{hk}: 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

$$\forall i, h, k \in V : h \neq i \neq k, \sum_{j:(i,j)\in A} f_{ij}^{hk} = \sum_{j:(j,i)\in A} f_{ji}^{hk}.$$

If $i = h$, the total balance of flow must be one, hence

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

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:

$$\forall (i, j) \in A, \sum_{h\in V} \sum_{k\in V:k\neq h} d_{hk} f_{ij}^{hk} \leq C x_{ij}.$$

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.

```
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
  end-declarations

  ! The following option is necessary to allow uncertain demands to be
  ! used across multiple capacity constraints

  setparam("robust_uncertain_overlap", true)

  ! 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)
  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
          write(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
          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.

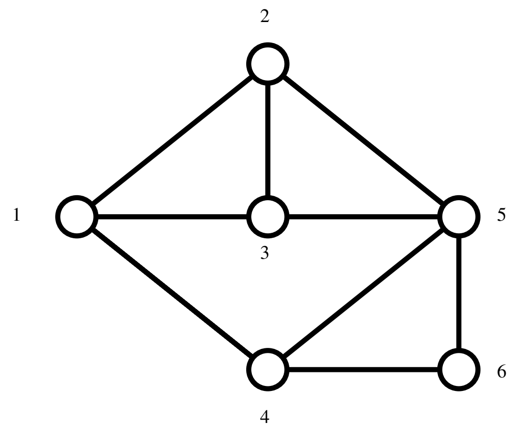

Figure 5: Network topology

Table 7: Maximum incoming/outgoing demand

Node	Incoming	Outgoing
1	120	105
2	34	95
3	35	82
4	101	102
5	78	76
6	75	140

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, we obtain a design of total cost 566. Table 9 reports the number of units of capacity to be

Table 8: Arc costs

i/j	1	2	3	4	5	6
1	–	10	9	12	–	–
2	8	–	10	–	12	–
3	10	7	–	–	9	–
4	10	–	–	–	12	5
5	–	9	8	11	–	12
6	–	–	–	10	9	–

installed on each arc, while Table 10 describes the path followed by each demand from source to destination.

Table 9: Optimal arc capacity

i/j	1	2	3	4	5	6
1	–	2	2	13	–	–
2	5	–	0	–	0	–
3	5	0	–	–	0	–
4	10	–	–	–	4	4
5	–	0	0	4	–	0
6	–	–	–	7	0	–

Table 10: Optimal routes

Source	Destination	Path	Source	Destination	Path
1	2	1→2	4	1	4→1
1	3	1→3	4	2	4→1→2
1	4	1→4	4	3	4→1→3
1	5	1→4→5	4	5	4→5
1	6	1→4→6	4	6	4→6
2	1	2→1	5	1	5→4→1
2	3	2→1→3	5	2	5→4→1→2
2	4	2→1→4	5	3	5→4→1→3
2	5	2→1→4→5	5	4	5→4
2	6	2→1→4→6	5	6	5→4→6
3	1	3→1	6	1	6→4→1
3	2	3→1→2	6	2	6→4→1→2
3	4	3→1→4	6	3	6→4→1→3
3	5	3→1→4→5	6	4	6→4
3	6	3→1→4→6	6	5	6→4→5

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.

$$\begin{array}{ll} \max & w \\ \text{s.t.} & w \leq \sum_{s \in S} (PRICE_s + N \cdot dev_s) \cdot x_s \text{ (with } dev_s = -VAR_s) \\ & \sum_{s \in S} x_s = 1 \\ & 0 \leq x_s \leq 1 \end{array}$$

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 improuve her model by adding a control parameter to limit the protection level.

Let *k* be the percentage value representing the protection level, and *G* the overall variation budget as defined by the following formula.

$$G = k \sum_{s \in S} VAR_s^2$$

The ellipsoidal uncertainty set
The possible asset value decrease is controlled by the uncertainty set $U(G)$ that is defined as follows:

$$U(G) = \{e : \sum_{s \in S} VAR_s \cdot e_s^2 \leq G\}$$

Robust constraints
The resulting robust optimization problem can then be stated as:

$$\begin{array}{ll} \max & w \\ \text{s.t.} & w \leq \sum_{s \in S} (PRICE_s + VAR_s \cdot e_s) \cdot x_s \quad (\forall e \in U(G)) \\ & \sum_{s \in S} x_s = 1 \\ & 0 \leq x_s \leq 1 \end{array}$$

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"

  parameters
    ZEROTOL=1e-6
    SEED=12345
  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       ! Estimated return in 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

    N=1.5                              ! Worst case metric
  end-declarations

!**************************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
    G := k*sum(s in Shares) VAR(s)^2 + ZEROTOL
    sum(s in Shares) (VAR(s)*e(s))^2 <= G
    ! The robust constraint
    obj <= sum(s in Shares) (PRICE(s) + N*VAR(s)*e(s))*frac(s)

    maximize(obj)
  end-procedure

!***************************Main model***************************
  !**** Input data generation ****
  setrandseed(12345)
  forall(s in Shares | s<>NSHARES) do
    PRICE(s) := round(30+s*4+random*(2+s))
    VAR(s) := round((PRICE(s)/4)*random)
  end-do
  PRICE(NSHARES) := round(1.01*(max(s in Shares | s<>NSHARES) PRICE(s)))
  VAR(NSHARES) := round(PRICE(NSHARES)*0.99/N)

  writeln("Shares | Mean | S.Dev. | Worst value")
  forall(s in Shares)
    writeln(strfmt(s,6), " |", strfmt(PRICE(s),5), " |", strfmt(VAR(s),7),
      " |", strfmt(PRICE(s)-N*VAR(s),6,0))
  write("\n\n")

  !**** 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
  end-do

  !**** Optimize the 'budgeted' worst case ****
  Ks := [0,         ! No variation on average
         0.001,     ! +/-  2% variation on the list
         0.01,      ! +/-  5% variation on the list
         0.05]      ! +/- 15% 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
    end-do
  end-do

  !**** Print results ****
  write("\n\nShares | Wst price | Price |   A    |")
  forall(k in Ks) write(" k=", strfmt(k*100,2,1), "% |" ) ; 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),5,0), "% | ")
      else
        write("       | ")
      end-if
    end-do
    writeln
  end-do

  !**** Simulate results (Monte-Carlo simulation) ****
  forall(mp in Problems) do
    NMC:=5000
    expRev(mp) := 0.0 ; wstRev(mp) := 0.0
    cntBelow110(mp) := 0.0 ; cntAbove130(mp) := 0.0
    c := 0.0
    forall(i in 1..NMC, c as counter) do
      totalVal := 0.0
```

```
        forall(s in Shares | frac_sol(s,mp)>0) do
          v := maxlist(normal(PRICE(s),VAR(s)),0)
          totalVal += v*frac_sol(s,mp)/100
          expRev(mp) += v*frac_sol(s,mp)/100
          while(v >PRICE(s)-N*VAR(s)) v := maxlist(normal(PRICE(s),VAR(s)),0)
          wstRev(mp) += v*frac_sol(s,mp)/100
        end-do
        if (totalVal<110) then cntBelow110(mp) += 1
        elif (totalVal>130) then cntAbove130(mp) += 1
        end-if
      end-do
      expRev(mp) := expRev(mp) / c
      wstRev(mp) := wstRev(mp) / c
      cntBelow110(mp) := cntBelow110(mp) / c
      cntAbove130(mp) := cntAbove130(mp) / c
    end-do

    !**** Print simulation results ****
    write("\n                           |   A    |")
    forall(k in Ks) write(" k=", strfmt(k*100,2,1), "% |" )
    write("\n        Expected value     |")
    forall(mp in Problems) write(strfmt(expRev(mp),7,1), " |")
    write("\n        Worst case value   |")
    forall(mp in Problems) write(strfmt(wstRev(mp),7,1), " |")
    write("\n\n        P (value<110)      |")
    forall(mp in Problems) write(strfmt(cntBelow110(mp),7,2) , " |")
    write("\n        P (110<=value<130) |")
    forall(mp in Problems) write(strfmt(1-cntBelow110(mp)-cntAbove130(mp),7,2),     " |")
    write("\n        P (value>130)      |")
    forall(mp in Problems) write(strfmt(cntAbove130(mp),7,2), " |")
    writeln

  end-model
```

5.4 Results

In order to understand how the uncertainty set impacts the solution we will solve the porfolio allocation problem for various protection levels *k* and compare the suggestions against the conservative approach.

The trader knows that she can expect a portfolio value in the range of 110 to 130. She therefore wishes to calculate for each of the three ranges 0–110, 110–130, 130–infinity the probability that the value of the portfolio will belong to this range.

5.4.1 Input Data

The 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).

A quick glance at the input data reveals that asset #22 would the best choice for the conservative trader because it has the highest worst case value, whereas asset #25 would be the preferred investment for an optimistic trader since it has the largest best case value for its expected value.

Table 11: Asset value distribution

Shares	Mean	S.Dev.	Worst value
1	35	2	32
2	42	8	30
3	47	11	31
4	49	3	45
5	51	12	33
6	60	1	59
7	61	8	49
8	64	16	40
9	72	10	57
10	81	12	63
11	85	6	76
12	83	5	76
13	83	15	61
14	91	2	88
15	102	8	90
16	97	1	96
17	109	18	82
18	107	12	89
19	126	30	81
20	132	19	104
21	128	11	112
22	124	4	118
23	136	22	103
24	130	12	112
25	137	90	2

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 4 robust optimization problems parameterized by the protection level *k*. When the protection level is k=0.0, then the deviation budget is also zero, 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 5 solutions and calculate the probability with which the portfolio value lies in one of the three value ranges that have previously been determined by the trader.

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 high largest expected value and high worst case.

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 drop under 110 is zero. However, from the trader's point of view this may not be judged a good solution as it is very unlikely that the portfolio value will rise beyond 130.

Table 12: Results of the parameterized robust optimization

Shares	Wst price	Price	A	k=0.0%	k=0.1%	k=1.0%	k=5.0%
19	81	126					9%
20	104	132				18%	16%
21	112	128				6%	12%
22	118	124	100%				7%
23	103	136			43%	31%	20%
24	112	130				12%	14%
25	2	137		100%	57%	34%	21%

The optimistic solution has the largest expected value, but also a substantial risk (38%) of achieving a portfolio value that lies below the trader's target of 110. It is also important to notice the high volatility of the portfolio value.

The behavior of the other solutions ($k > 0$) shows that as the deviation budget 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.

Table 13: Results of the Monte-Carlo simulation

	A	k=0.0%	k=0.1%	k=1.0%	k=5.0%
Expected value	124.0	139.5	138.3	135.5	133.1
Worst case value	116.3	0.1	40.3	64.6	76.3
P (value $\leq$ 110)	0.00	0.38	0.30	0.22	0.13
P (110 $\leq$ value $\leq$ 130)	0.93	0.08	0.14	0.21	0.33
P (value $\leq$ 130)	0.07	0.53	0.56	0.57	0.54

The solution with k = 5% seems to present a good balance between expected value and worst case value. Moreover, this solution has a great chance (87%) of producing a portfolio value that is greater than 110.

5.5 References

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

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 black out.

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

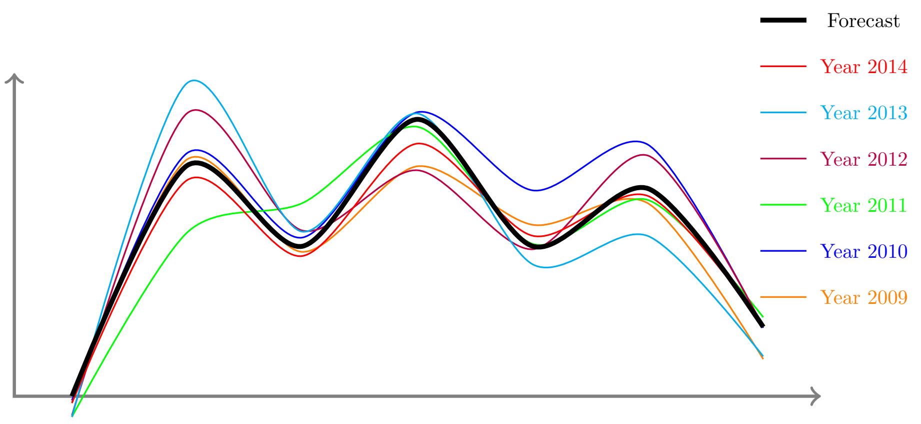

Figure 6: Historical power demand and forecast

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 *Horizon* of consecutive time periods describes the planning horizon of the study. Time periods $t \in Horizon$ have different lengths LEN_t. Each time period $t \in Horizon$ is associated with a power demand DEM_t.

The set *Units* denotes the set of available power generation units. The minimum power generation level of a unit is $PMIN_u$, and its maximum capacity is $PMAX_u$. The startup cost of a single unit is $CSTR_u$, and the cost of running it at the minimum power generation level is $CMIN_u$. The marginal power production cost above $PMIN_u$ for each hour is $CADD_u$.

Each unit u is associated with three decision variables. The binary variable $start_{ut}$ equals 1 if the unit u is starting at the beginning of time period t. The binary variable $work_{ut}$ equals 1 if the unit u is up and running during the whole time period t. At last, the variable $padd_{ut}$ 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:

$$\sum_{u \in Units, t \in Horizon} CSTR_u \cdot start_{ut} + LEN_t \cdot (cmin_u work_{ut} + cadd_u \cdot padd_{ut})$$

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

$$start_{ut} \geq work_{ut} - work_{ut-1}, \quad \forall u \in Units, if \quad t > 1$$
$$start_{u1} \geq work_{u1} - work_{uT}, \quad \forall u \in Units, if \quad t = 0$$

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

$$\leq (PMAX_u - PMIN_u) \cdot work_{ut} \quad \forall u \in Units, t \in Horizon$$

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

$$\sum_{u \in Units} PMIN_u \cdot work_{ut} + padd_{ut} = DEM_u, \quad \forall t \in Horizon$$

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 *Udem* be the set of possible future power demands. Let *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:

$$Udem = \{e : \forall t \in Horizon, \exists y \in Years : e_t = HDEM_{yt}\}$$

Robust constraints

$$\sum_{u \in Units} PMAX_u \cdot work_{ut} \geq dem_t \quad \forall dem \in Udem, t \in Horizon$$

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 cardinality uncertainty set
Let *Uoutage* be the set describing groups of combinations of *k* units in forced outage. This uncertainty set can be expressed as follows:

$$Uoutage = \{e : \sum_{u \in Units} e_u \leq k\}$$

Robust constraints

$$\sum_{u \in Units} PMAX_u * work_{ut} \cdot (1 - e_u) \geq DEM_t \quad \forall e \in Uout, t \in Horizon$$

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.

```
 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
  start(u,t) is_binary
  work(u,t) is_binary
 end-do

! 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("\n=== Nominal case ===\n")
 print_solution

! Add robust constraints
 robust_demand
! Solve robust the problem
 minimize(Cost)

 writeln("\n=== Robust against demand variation ===\n")
 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), "")
     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.

```
!**** Ensure enough power production capacity allows to
!**** satisfy load even if 'k' units are forced in outage.
 procedure robust_contingency(k: integer)
   declarations
     outage: array(UNITS,TIME) of uncertain ! uncertain event
   end-declarations

   forall(t in TIME, u in UNITS) outage(u,t) >= 0
   ! In forced outage, loosing 100% of
   ! the unit generation capacity
   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) {outage(u,t)}, k)

   forall(t in TIME) 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)
   ! sum(u in UNITS) PMAX(PLANT(u))*work(u,t)*(1 - 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 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 14: Description of power generators

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

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 periods (0-6) the load can increase from 12000 kWh (power demand forecast) to 13250 kWh (power demand forecast + reserve up) without starting 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 *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: 1.40679 Million)

	0-6	6-9	9-12	12-14	14-18	18-22	22-24
Up units	8	18	15	20	15	17	11
Gen. Pwr.	12000	32000	25000	36000	25000	30000	18000
Gen. Cap.	13250	37750	27250	44750	27250	34250	19250
Res. Dn	3850	10050	8450	10450	8450	9850	6250
Res. Up	1250	5750	2250	8750	2250	4250	1250
Loss of load							
Demand variation	0	1395	1491	0	2592	0	0
2 Contingencies	2750	1250	1750	0	1750	2750	2750

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

	0-6	6-9	9-12	12-14	14-18	18-22	22-24
Up Units	8	18	15	20	15	17	11
Gen. Pwr.	12000	32000	25000	36000	25000	30000	18000
Gen. Cap.	13250	39250	28750	44750	30250	37250	19250
Res. Dn	3850	9450	7850	10450	7250	8650	6250
Res. Up	1250	7250	3750	8750	5250	7250	1250
Loss of load	0	0	0	0	0	0	0

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 17: Robust against 2 contingencies (Total cost: 1.420 Million)

	0-6	6-9	9-12	12-14	14-18	18-22	22-24
Up Units	9	18	16	20	15	17	11
Gen. Pwr.	12000	32000	25000	36000	25000	30000	18000
Gen. Cap.	19750	39250	32250	44750	33250	38750	25250
Res. Dn	850	9450	6050	10450	6050	8050	3850
Res. Up	850	9450	6050	10450	6050	8050	3850
Loss of load	0	0	0	0	0	0	0

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

The uncertainty set is defined by every vector of interruptions $(\xi)_{i=1..n}$ such that $\xi_i \in [0, 1]$ and not more than K of its elements are positive:

$$\sum_{i=1}^{n} \xi_i \leq K.$$

Let us define the variables $prod_{tg}$ and 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:

$$0 \leq prod_{tg} \leq P_g \quad \forall g, \forall t \in \{1, 2, \ldots, N\}$$

$$0 \leq inv_{tg} \leq S_g \quad \forall g, \forall t \in \{1, 2, \ldots, N\}$$

$$0 \leq inv_{0g} = I_g \quad \forall g;$$

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 ξ_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:

$$inv_{0,g} + sum_{\tau \in T:\tau \le t}((1 - \xi_\tau)prod_{\tau g} - dem_{\tau g}) \ge 0 \quad \forall t \in T, g \in G.$$

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:

$$inv_{tg} \ge inv_{0,g} + sum_{\tau \in T:\tau \le t}(prod_{\tau g} - dem_{\tau g}) \quad \forall t \in T, g \in G.$$

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 uncertains multiple times.

```
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

    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 interruption
                                           ! (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
  PERIODS0 := 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?
  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

    inventory(0,g) + sum(tp in PERIODS | tp <= t)
      ((1 - interruption(tp)) * 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)

  end-do

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

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

!**** Solving ****
  setparam("robust_uncertain_overlap", true)

  ! Set verbosity level
  setparam("xprs_verbose", true)

  ! Objective function: total cost of production and storage
  minimize(sum (t in PERIODS, g in GASES)
             (PROD_COST * produce (t,g) + INV_COST * inventory(t,g)))

!**** Soution reporting ****
  writeln("\nNumber of interruptions: ", MAX_NINTERR)
  writeln("\nOptimal solution has cost ", getobjval)

  COLWIDTH := 6

  forall(g in GASES) do

    writeln("\nProduction of ", g)

    write(strfmt ("Time",-COLWIDTH))
    forall(t in PERIODS0) write (strfmt(t,COLWIDTH))

    write("\n", strfmt("Dem",-2*COLWIDTH))
    forall(t in PERIODS) write(strfmt(DEMAND(t,g),COLWIDTH,1))

    write("\n", strfmt("Prod",-2*COLWIDTH))
    forall(t in PERIODS) write(strfmt (produce(t,g).sol,COLWIDTH,1))

    write("\n", strfmt("Inv*",-COLWIDTH))
    forall(t in PERIODS0)
      write (strfmt(inventory(0,g).sol +
            sum (tp in PERIODS | tp <= t)
              (produce(tp,g).sol - DEMAND(tp,g)),COLWIDTH,1))
    writeln("")

  end-do

end-model
```

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 18: Production and inventory capacity

Gas	Prod. capacity	Inv. capacity	Initial inv.
LIN	29	60	20
LOX	5.5	27	20

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

Table 19: Customer demand

Gas/Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LIN	6	14	10	8	11	15	10	9	10	9	10	12	11	15	9
LOX	2	5	3	4	8	4	8	7	5	4	3	3	5	9	7

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

Time	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Dem	6	14	10	8	11	15	10	9	10	9	10	12	11	15	9
Prod	29	15	15	15	15	15	10	9	10	9	10	12	11	15	9
Inv*	43	44	49	56	60	60	60	60	60	60	60	60	60	60	60

Table 21: Production of LOX

Time	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Dem	2	5	3	4	8	4	8	7	5	4	3	3	5	9	7
Prod	5.5	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5	5.2	5.2	5.2	5.2	5.2	5.2
Inv*	23.5	23.7	25.8	27	24.2	25.3	22.5	20.7	20.7	21.8	24	26.2	26.3	22.5	20.7

Finally, 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.

7.6 References

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

Bibliography

[BS04] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics, 2009.

[GHPS02] C. Guéret, S. Heipcke, C. Prins, and M. Sevaux. *Applications of Optimization with Xpress-MP*. Dash Optimization, Blisworth, UK, 2002.

[LBS13] C. Latifoglu, P. Belotti, and L.V. Snyder. Models for production planning under power interruptions. *Naval Research Logistics (NRL)*, 60(5):413–431, 2013.