

## Special Ordered Sets

© Copyright Dash Optimization, 12 December 2002

Special ordered sets are a feature available in MIP optimization to help model certain situations in a way that can be solved efficiently by the MIP optimization. Special ordered sets can help you model mutually exclusive decisions and non-linear functions.

There are two types of Special Ordered Sets (SOS) available in Xpress-MP. This note explains what they are, what you can use them for, and how you can specify them in Xpress-Mosel and Xpress-BCL.

### Special Ordered Sets of Type 1 (SOS1)

A SOS1 is a set of decision variables in which at most one variable may be positive at an integer feasible solution. Typically these would correspond to mutually exclusive decisions, for example, choosing the month in which to start a major project. Associated with each variable in the set must be a number which orders the variables: the ordering or reference values. The ordering values must be all different: two elements in one set cannot have the same ordering value.

For example, suppose  $x(t)$  is a variable in our model which will take the solution value one if we are to start a project in month  $t$ , for  $t=1, \dots, NT$ . Then we shall use the month index  $t$  itself to order the variables in the set, and define:

$$\{ x(1), x(2), \dots, x(T) \}$$

to be a SOS1 where the ordering value for element in  $x(t)$  the set is  $t$ .

The ordering values order the set: we can say that  $x(4)$  comes before  $x(9)$  in the set, and that  $x(2)$  and  $x(3)$  are adjacent in the set (in the sense that there is no other element of the set between them) whilst  $x(2)$  and  $x(4)$  are not adjacent in the set because  $x(3)$  lies between them.

We can define this SOS1 in Mosel as follows:

```
DefMySOS1 := sum(t in 1..NT) t*x(t) is_sos1
```

where the variables  $x(t)$  have been declared to be of type mpvar, the constraint object `DefMySOS1` has been to be of type linctr, and `NT` is an integer.

Notice how the ordering values are specified as coefficients of the variables in the set: this is simply a convenience in Mosel, as Mosel provides many facilities for defining and manipulating linear expressions. (The expression that defines the SOS is sometimes referred to as the reference constraint or reference row, because historically a normal constraint (or row) in the problem was used to define the SOS.) However, in Mosel the value of this linear expression is not relevant to the definition of the SOS1 or to the problem formulation in general.

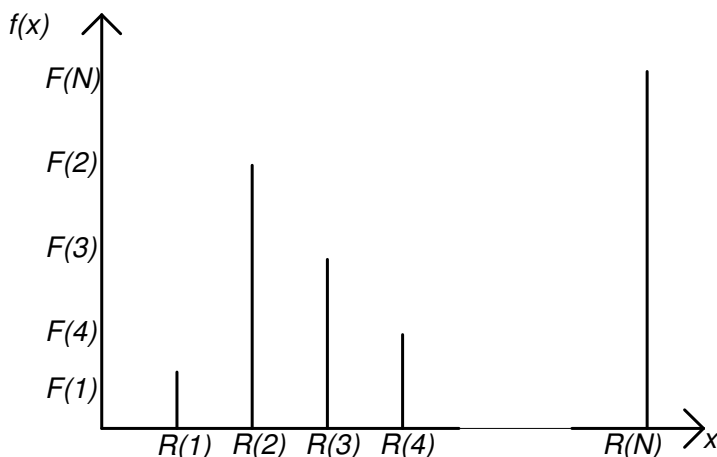
Returning to the example, in this case we also need a convexity constraint to ensure that at least one of the  $x(t)$  variables is non-zero, and that the non-zero  $x(t)$  equals one. This can be written in Mosel as a standard constraint:

```
Convex := sum(t in 1..NT) x(t) = 1
```

Although a convexity constraint is normally associated with a SOS, strictly speaking it is not required and occasions do arise in which it makes sense to define a SOS without a convexity constraint.

## Special Ordered Sets of Type 2 (SOS2)

A SOS2 is a set of variables in which at most two variables may be positive at an integer feasible solution, and moreover, any positive variables must be adjacent in the ordering specified by the ordering values. Their main use is in non-linear programming and we shall illustrate it by modeling a non-linear function of a single argument,  $f(x)$ .



Let  $R(1), R(2), \dots, R(N)$  be a given grid of suitable values for  $x$ , not necessarily evenly spaced. Also let  $F(1), F(2), \dots, F(N)$  be the corresponding calculated values of  $F$  at each grid point, i.e.  $F(i) = F(R(i))$ . If we define  $w(1), w(2), \dots, w(N)$  to be weight variables associated with each grid point then we may approximate the value of  $f(x)$  by the weighted combination of the  $F(x)$  grid values,  $F(i)$ :

$$fx = \text{sum}(i \text{ in } 1..N) F(i) * w(i)$$

provided that the values of the weight variables are chosen so that:

- the value of  $x$  is given the weighted combination of the  $x$  grid values,  $R(i)$ :

$$x = \text{sum}(i \text{ in } 1..N) R(i) * w(i)$$

- they add up to unity:

$$1 = \text{sum}(i \text{ in } 1..N) w(i)$$

- and at most two adjacent weight variables are positive.

The final condition may be guaranteed by the cost structure of the model, for example if we were maximizing the value of a concave function. However, if it is not, then a method of ensuring that the  $w(i)$  still yield a valid piecewise linear approximation to  $f$  is to stipulate that they form a SOS2. The  $w(i)$  must be ordered by the  $x$  grid values,  $R(i)$ . Two weight variables  $w(i)$  and  $w(j)$  are then adjacent if no other weight variable  $w(k)$  exists with  $R(i) \leq R(k) \leq R(j)$ .

We can define this SOS2 in Mosel as follows:

```
DefMySOS2 := sum(i in 1..N) R(i)*w(i) is_sos2
```

together with the three constraints given above.

## Specifying Special Ordered Sets in Mosel

### Method 1

We have already seen how to specify special ordered sets using the `is_sos1` or `is_sos2` operators.

```
DefMySOS1 := sum(t in 1..NT) t*x(t) is_sos1
DefMySOS2 := sum(i in 1..N) R(i)*w(i) is_sos2
```

All constraints related to the SOS, such as the convexity constraint and the grid constraints given above, must be specified in the normal way. See below for a more complete example of defining an SOS1 using `is_sos1`.

However, it is impossible to specify an ordering value of zero using this method. This is because if a coefficient of zero is used in the linear expression used to define an SOS, e.g.,  $0*w(i)$ , then Mosel evaluates the term to be zero, and it is dropped from the linear expression defining the SOS definition. If you want to specify an ordering value of zero using this approach, it is necessary to use a very small value such as  $1e-20$  instead. (The Optimizer will treat such a small value as zero in any case.)

### Method 2

An alternative method, available from Xpress-MP 2003, is to use the `makesos1` and `makesos2` procedures.

For example, the two SOS described above can be specified using:

```
makesos1(union(t in 1..NT) {x(t)}, sum(t in 1..NT) t*x(t))
makesos2(union(i in 1..N) {w(i)}, sum(i in 1..N) R(i)*w(i))
```

with all constraints again specified in the normal way. See below for a more complete example of defining an SOS2 using `makesos2`.

These procedures generate a SOS containing all of the decision variables specified in the first argument, using the ordering values given by the coefficients in the linear expression supplied in the second argument.

Using this method it does not matter if one variable in a SOS has a zero coefficient in the linear expression: provided the variable is included in the first argument, it will be included in the SOS. If the variable has a zero coefficient in the linear expression, or equivalently it does not appear in the linear expression at all, then the variable will be given an ordering value of zero. Of course, at most one variable may have an ordering value of zero because the ordering values of all variables in a single SOS must be distinct.

### Ordering Values

A common mistake when defining a SOS is to define two ordering values to be identical. Ordering values are treated as identical if they differ by no more than 0.001 (the value is given by the Optimizer control variable `XPRS_SOSREFTOL`). If two ordering values are found to be identical, an error will be reported that the problem could not be loaded into the Optimizer:

```
XPRS: Error when loading the problem.
```

Further information can be obtained by enabling Optimizer message output and loading the entity names into the Optimizer using the following statements in your model:

```
setparam('XPRS_VERBOSE', true)
setparam('XPRS_LOADNAMES', true)
```

and including debugging information in the compiled model (this happens by default when using Xpress-IVE).

Running the model again gives the more specific error message

```
? 56 Error: Reference row entries too close for set SET1 member w(3)
XPRS: Error when loading the problem.
Mosel: Error located at line 19 of 'MySOS2.mos'.
```

which allows you to identify the second variable in the set that has an ordering (reference row) value the same as another variable, and the line number in the Mosel model at which the error occurs.

### Complete SOS1 Example

Here is a complete (well, almost!) model showing how an SOS1 may be defined.

```
model MySOS1
  uses "mmxprs"

  declarations
    NT = 10
    R: array(1..NT) of real
    w: array(1..NT) of mpvar    ! weights
  end-declarations

  DefMySOS1 := sum(t in 1..NT) t*x(t) is_sos1
  Convex    := sum(t in 1..NT) x(t) = 1

  ...
end-model
```

### Complete SOS2 Example

Here is a complete (well, almost!) model showing how an SOS2 may be defined.

```
model MySOS2
  uses "mmxprs"

  declarations
    N = 10
    R, F: array(1..N) of real
    w: array(1..N) of mpvar    ! weights
    x, fval: mpvar             ! x and f values
  end-declarations

  ! Define an quadratic function to approximate
  forall(i in 1..N) do
    R(i) := i
    F(i) := R(i)^2
  end-do

  makesos2(union(i in 1..N) {w(i)}, sum(i in 1..N) R(i)*w(i))

  Xdef := x = sum(i in 1..N) R(i)*w(i)
  Fdef := fx = sum(i in 1..N) F(i)*w(i)
  Convex := 1 = sum(i in 1..N) w(i)

  ...
end-model
```

## Specifying Special Ordered Sets in BCL

Here is a fragment of a project planning model, written using BCL in C. Variable  $x[p][t]$  is one if we start project  $p$  in time period  $t$  and for each  $p$  we form a SOS1 from all the  $x[p][t]$  using  $t$  as the ordering value.

```
#include "xprb.h"

#define NProj 3          /* Number of projects */
#define NTime 6         /* Number of months to plan for */

XPRBvar x[NProj][NTime]; /* 1 if project p starts in month t */

int main(void)
{
    XPRBprob prob;
    XPRBsos sos1;
    int p, t, s;

    XPRBinit("");
    prob = XPRBnewprob("PPlan");

    /**** Create the Variables ****/
    /* Define variables x[p][t] with bounds [0, 1] */
    for(p = 0; p < NProj; p++)
        for(t = 0; t < NTime; t++)
            x[p][t] = XPRBnewvar(prob, XPRB_PL, "x", 0, 1);

    /**** Form the SOS1 Sets ****/
    /* The set object 'sos1' is reused - we don't need it again*/
    for(p = 0; p < NProj; p++) {
        sos1 = XPRBnewsos(prob, "set", XPRB_S1);
        for(t = 1; t <= NTime; t++)
            /* Add x[p][t] to the set, using t to order */
            XPRBaddsosel(prob, sos1, x[p][t-1], t);
    }

    return 0;
}
```