Global entities

Lorenzo Castelli, Università degli Studi di Trieste.

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Global entities

In principle, only continuous and binary variables are needed to formulate any MIP problem. However, it is convenient to extend the set of modelling entities (**mpvar** in XPress)

- Integer variables
- Partial integer variables
- Semi-continuous variables (SC)
- Semi-continuous integer variables (SI)
- Special Ordered Sets of type 1 (SOS1)
- Special Ordered Sets of type 2 (SOS2)

Integer variables

is_integer

An integer variable is a variable that must take only an integer (whole number) value

Any integer variable can be written as a function of binary variables. As an example, let v be a variable that can take any integer value between 0 and 10. It can also be written as

$$v = b_1 + 2b_2 + 4b_3 + 8b_4$$

and

$$b_1 + 2b_2 + 4b_3 + 8b_4 \leq 10.$$

where $b_1, b_2, b_3, b_4 \in \{0, 1\}$. But we now have four global entities rather just one.

Integer variables

In addition, a particular integer value can be achieved by many fractional settings, e.g., $\boldsymbol{v} = \mathbf{6}$ ca be achieved as

$$\boldsymbol{b_1} = \boldsymbol{b_2} = \boldsymbol{b_3} = \mathbf{0}, \boldsymbol{b_4} = \mathbf{0.75} \text{ or } \boldsymbol{b_1} = \mathbf{0}, \boldsymbol{b_2} = \mathbf{1}, \boldsymbol{b_3} = \mathbf{1}, \boldsymbol{b_4} = \mathbf{0}$$

and many other possibilities. Thus even though $\boldsymbol{v}$ is integer it can be achieved by fractional values of the binary variables and these fractional values would force our integer programming code to do more branching.

Since all commercial integer programming codes allow one to use general integer variables, and with the disadvantage of expanding integers in terms of binary variables, it is usually silly to use binary variables where we really have an underlying integer.

Partial integer variables

is_partint

A partial integer variable is a variable that must take an integer value if it is less than a certain user-specified limit L, or any value above that limit.

- It is up to the decision maker to decide the limit L below which it is acceptable to round a fractional answer to a nearby integer.
- The modelling system allows the decision maker to specify a different L for each partial integer variable, if so desired.

Partial integer variables

The natural way to express a partial integer $\boldsymbol{p}$

$$\boldsymbol{p} = \boldsymbol{v} + \boldsymbol{x}$$

where variable $\boldsymbol{v}$ is an integer that must be less than or equal to $\boldsymbol{L}$ (i.e. $\mathbf{0} \leq \boldsymbol{v} \leq \boldsymbol{L}$) and $\boldsymbol{x}$ is a real variable.

We could also use a binary variable $\boldsymbol{b}$ to express the fact that if $\boldsymbol{v}$ is strictly less than $\boldsymbol{L}$ then $\boldsymbol{x}$ must be zero. Suppose that we know some value $\boldsymbol{X_{max}}$ which $\boldsymbol{x}$ is guaranteed not to exceed, i.e. an upper bound for $\boldsymbol{x}$. Then we can introduce the following constraints

$$\begin{aligned}&\boldsymbol{x} \leq \boldsymbol{X_{max}} \cdot \boldsymbol{b}\\&\boldsymbol{b} \leq \boldsymbol{v}/\boldsymbol{L} \text{ (or, equivalently, } \boldsymbol{v} \geq \boldsymbol{L} \cdot \boldsymbol{b}\text{)}\end{aligned}$$

Partial integer variables

If $\boldsymbol{b} = \mathbf{0}$ then $\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{v} \geq \mathbf{0}$. If $\boldsymbol{b} = \mathbf{1}$ then $\boldsymbol{x} \leq \boldsymbol{X_{max}}$ and $\boldsymbol{v} \geq \boldsymbol{L}$, which combined with the fact that $\boldsymbol{v} \leq \boldsymbol{L}$ means $\boldsymbol{v} = \boldsymbol{L}$. So if $\boldsymbol{b} = \mathbf{1}$, $\boldsymbol{v}$ is at its maximum, and we can then start having non-zero amounts of $\boldsymbol{x}$. However, introducing the extra binary variable will never make the branch and bound tree search quicker and may often make the tree search worse. Consider the following example.

- Suppose that $\boldsymbol{L} = \mathbf{10}$ and that the LP solution gives the optimum value of $\boldsymbol{v} = \mathbf{3}$ and $\boldsymbol{x} = \mathbf{0}$, but the binary variable $\boldsymbol{b}$ is at $\mathbf{0.3}$. We can see that this satisfies all the inequalities we have given but that the binary variable $\boldsymbol{b}$ is fractional and we will have to branch on $\boldsymbol{b}$ to satisfy integrality.
- But $\boldsymbol{v}$ is feasible as far as partial integrality is concerned, so we have a case where partial integer variables enable us to say that we are integer feasible but the introduction of the binary variable results in it not being integer feasible and we have to branch further.
- In models with many partial integer variables this can lead to a great expansion of the branch and bound tree and a lengthening of the search.

Semi-continuous variables

is_semcont

A semi-continuous variable(SC) is a decision variable that can take either the value $\mathbf{0}$, or a value between some user specified lower limit $\boldsymbol{L}$ and upper limit $\boldsymbol{U}$.

SCs variables are useful where, if some variable is to be used at all, its value must be no less than some minimum amount. For example, in a blending problem we might be constrained by practical reasons either to use none of a component or at least $\boldsymbol{20kg}$ of it. This type of constraint occurs very frequently in practical problems.

Semi-continuous variables

If we introduce a binary variable $\boldsymbol{b}$ then we can represent the semi-continuity of $\boldsymbol{s}$ by the following pair of constraints

$$\boldsymbol{L} \cdot \boldsymbol{b} \leq \boldsymbol{s}$$
$$\boldsymbol{s} \leq \boldsymbol{U} \cdot \boldsymbol{b}$$

There are two cases to consider.

- Either $\boldsymbol{b} = \mathbf{0}$, in which case $\boldsymbol{s}$ is constrained to be $\mathbf{0}$
- $\boldsymbol{b} = \mathbf{1}$, in which case $\boldsymbol{s}$ is constrained to lie between $\boldsymbol{L}$ and $\boldsymbol{U}$.

These are exactly the two conditions that we want to impose.

- Why are semi-continuous variables useful? It seems a very small penalty just to have to replace a SC variable by one extra binary variable.
- The answer is that the semi-continuous property may be satisfied by the variable $\boldsymbol{s}$, but if we introduce a binary variable the integrality of that binary variable may not be satisfied.

Semi-continuous variables

- Consider the case where s either has to take on the value $\mathbf{0}$ or it has to take on a real value in the range between $\mathbf{5}$ and $\mathbf{10}$.
- Suppose that we solve the LP relaxation and we get a value for s of $\mathbf{7.5}$ and we get a value of b of $\mathbf{0.75}$. This certainly satisfies the two inequalities but the branch and bound search would note that the binary variable was not satisfied (it is not either 0 or 1) and so we would have to go on branching and bounding. But the underlying semi-continuity of s is satisfied; s does indeed lie between its lower bound $\mathbf{5}$ and its upper bound $\mathbf{10}$ and so we would not have to branch any further on the s.
- Thus not having semi-continuous variables in the modelling language or in the optimizer can lead to more branching and bounding in the branch and bound search, and indeed if you need to use semi-continuous variables you should look for an optimizer that has these built into it.

Semi-continuous integer variables

is_semint

Semi-continuous integer variables (SI): decision variables that can take either the value $\mathbf{0}$, or an integer value between some lower limit and upper limit. SIs help model situations where if a variable is to be used at all, it has to be used at some minimum level, and has to be integer.

SIs often arise in shift planning modelling. If we are going to set up a machine, then we have to run it for a whole number of shifts, and in any case for at least some minimum number of shifts. For example, it might be possible only to run the machine for $\mathbf{0}$, or $\mathbf{12, 13, 14, \ldots}$ shifts, with the case of running it for $\mathbf{1, 2, \ldots, 11}$ shifts making no economic sense.

Special Ordered Sets of type 1

is_sos1

Special Ordered Sets of type 1 (SOS1): an ordered set of non-negative variables at most one of which can take a non-zero value.

SOS1 are often used in modelling choice problems, where we have to select at most one thing from a set of items. The choice may be from such sets as: the time period in which to start a job; one of a finite set of possible sizes for building a factory; which machine type to process a part on, etc.

Special Ordered Sets of type 1

Many models contain special ordered sets of type 1 (SOS1) of the form

$$\sum_{j=1}^{k} x_j = 1$$

with $x_j \in \{0, 1\}$ for $j = 1, \ldots, k$.

- If the linear programming solution x^* has some of the variables $x_1^*, \ldots, x_k^*$ fractional, then the standard branching rule is to impose $S_1 = S \cap \{x : x_j = 0\}$ and $S_2 = S \cap \{x : x_j = 1\}$ for some $j \in \{1, \ldots, k\}$.
- However, because of the SOS1 constraint, $\{x : x_j = 0\}$ leaves $k - 1$ possibilities $\{x : x_j = 1\}_{i \neq j}$ whereas $\{x : x_j = 1\}$ leaves only one possibility.
- So S_1 is typically a much larger set than S_2, and the tree is unbalanced.

Special Ordered Sets of type 1

SOS1 branching is designed to provide a balanced division of S into S_1 and S_2. Specifically, the user specify an ordering of the variables in the SOS1 set $j_1, \ldots, j_k$, and the branching scheme is then to set

$$S_1 = S \cap \{x : x_{j_i} = 0 \ i = 1, \ldots, r\} \text{ and}$$
$$S_2 = S \cap \{x : x_{j_i} = 0 \ i = r+1, \ldots, k\}$$

where $r = \min\{t : \sum_{i=1}^{t} x^*_{j_i} \geq \frac{1}{2}\}$. In many cases such a branching scheme is much more effective than the standard scheme, and the number of nodes in the tree is significantly reduced.

Special Ordered Sets of type 1

- Suppose that we are considering building a factory and we have options S for a small factory, M for a medium sized factory, B for a big factory and N which is no factory at all.
- We can model this by introducing four binary variables b_S, b_M, b_B and b_N and model the fact that we can only choose one of these options by the constraint

$$b_N + b_S + b_M + b_B = 1$$

- Clearly since these are binary variables adding up to 1, only one of them can be 1 and one of them must be 1. So the SOS1 property applies.

Special Ordered Sets of type 1

- Suppose that the capacity of the various options for the factories are
  - Small: 10000 tonnes
  - Medium: 20000 tonnes
  - Big: 38000 tonnes
  - No factory: 0 tonnes

  ,
- One of the things we will be interested in is the selected capacity of the factory that we will be building, and somewhere in the model we would have a decision variable size and an equation

$$\boldsymbol{size} = \mathbf{0} \cdot \boldsymbol{b_N} + \mathbf{10000} \cdot \boldsymbol{b_S} + \mathbf{20000} \cdot \boldsymbol{b_M} + \mathbf{38000} \cdot \boldsymbol{b_B}.$$

Special Ordered Sets of type 1

- We can use the coefficients in this equation (which we call the reference row) to **order** the variables.
- The order emerging is $\boldsymbol{b_N, b_S, b_M, b_B}$ where we have ordered them by their coefficients in the Reference Row.
- Now suppose that we have solved the linear programming relaxation and got the following solution:

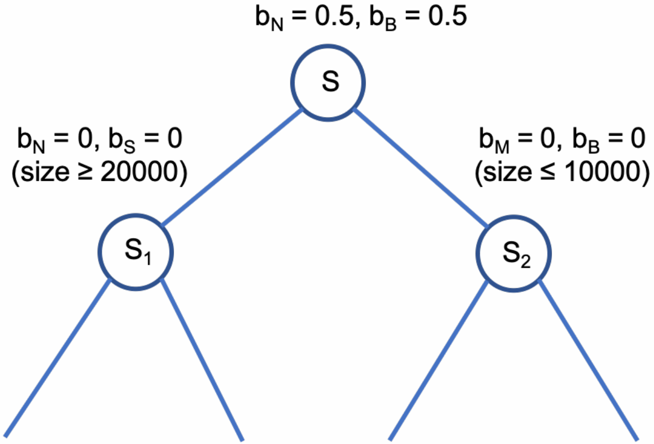
$$\boldsymbol{b_N = 0.5, b_S = 0, b_M = 0, b_B = 0.5}$$

indicating that we are going to build half of the 0 sized factory and half of the big factory (remember these are the values of the LP relaxation where we have relaxed the integrality of the binary variables).

Special Ordered Sets of type 1

- The usual branch and bound search will look at that solution and say that there are only two fractional variables $\boldsymbol{b_N}$ and $\boldsymbol{b_B}$ and so it will branch on one or other of those variables.
- But there is a lot more information: we are being told by the LP solution that the tonnage that we require from that factory is a half of the biggest one, i.e., 19000 tonnes.
- Now consider branching on, say, $\boldsymbol{b_N}$. Either we are forcing the factory to have zero capacity if we branched on $\boldsymbol{b_N = 1}$, or we are forcing the factory to be open (i.e., $\boldsymbol{b_N = 0}$ and then $\boldsymbol{b_S = 1}$ or $\boldsymbol{b_M = 1}$ or $\boldsymbol{b_B = 1}$)
- It would be much better to have some branching scheme that understood and tried to exploit the fact that the indicated capacity is about 19000 tonnes.
- This is precisely what the special ordered set property indicates to the branch and bound tree, and **the branching occurs on sets of variables** rather than on the individual variables which form the Special Ordered Set.

Special Ordered Sets of type 1



Special Ordered Sets of type 1

- It is, however, an observed fact that modelling these selection constraints in terms of Special Ordered Sets generally gives a faster branch and bound search.
- Again as with the case of general integer variables, the solution we get will be exactly the same whichever way we model. The 'only' benefit of using Special Ordered Sets is that the search procedure will generally be noticeably faster.

Special Ordered Sets of type 2

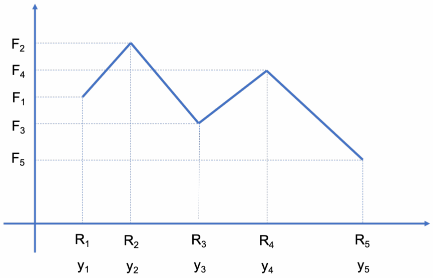
is_sos2

Special Ordered Sets of type 2 (SOS2): an ordered set of non-negative variables, of which at most two can be non-zero, and if two are non-zero these must be consecutive in their ordering.

SOS2 are typically used to model non-linear functions of a variable, as piece-wise function of a single variable.

Piece-wise linear functions

We wish to represent $\boldsymbol{f}$ as a function of $\boldsymbol{x}$ and we are prepared to consider four line segments between five points. The five points are $(\boldsymbol{R_1}, \boldsymbol{F_1}), (\boldsymbol{R_2}, \boldsymbol{F_2}), (\boldsymbol{R_3}, \boldsymbol{F_3}), (\boldsymbol{R_4}, \boldsymbol{F_4})$ and $(\boldsymbol{R_5}, \boldsymbol{F_5})$ and associated with each point $\boldsymbol{i}$ is a weight variable $\boldsymbol{y_i}$.



Piece-wise linear functions

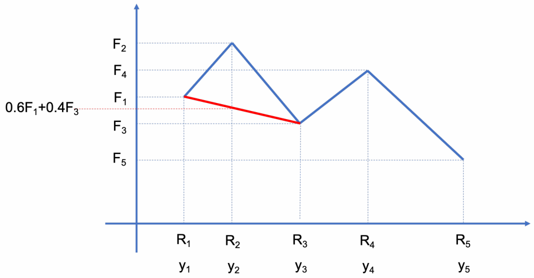
The formulation is

$$
\begin{aligned}
\boldsymbol{x} &= \sum_{i=1}^{5} \boldsymbol{R_i y_i} \text{ reference row} \\
\boldsymbol{f} &= \sum_{i=1}^{5} \boldsymbol{F_i y_i} \\
\sum_{i=1}^{5} \boldsymbol{y_i} &= \mathbf{1} \text{ convexity row}
\end{aligned}
$$

where there are at most two non-zero $\boldsymbol{y_i}$, and if there are two non-zero then they must be adjacent.

Adjacency condition

- The adjacency condition must be respected to be always on the piece-wise linear function.
- In case it is not, e.g., $\boldsymbol{y_1 = 0.6, y_3 = 0.4}$ (all the others $\mathbf{0}$), at $\boldsymbol{x = R_2}$ the function value is not $\boldsymbol{F_2}$ but $\boldsymbol{0.6F_1 + 0.4F_3}$, considerably lower.



Adjacency condition

- The adjacency condition is enforced by introducing a binary variable b_i associated with each of the intervals $(R_1, R_2), (R_2, R_3), (R_3, R_4)$ and (R_4, R_5), and 6 constraints.
- b_i is 1 if the value of x lies between R_i and R_{i+1}.

Consider the constraints

$$\begin{aligned} y_1 &\leq b_1 \\ y_2 &\leq b_1 + b_2 \\ y_3 &\leq b_2 + b_3 \\ y_4 &\leq b_3 + b_4 \\ y_5 &\leq b_4 \\ b_1 + b_2 + b_3 + b_4 &= 1 \end{aligned}$$

Adjacency condition

- The last constraint ensures that one and only one b_i is 1 in an acceptable solution.
- For instance, if b_3 is 1, the constraints on the y_i ensure that $y_1 = y_2 = y_5 = 0$, leaving only the adjacent y_3 and y_4 to be (possibly) non-zero.
- The other case to consider is when one of the ‘end’ b_i is 1, say b_1. Then $y_3 = y_4 = y_5 = 0$, and y_1 and y_2, which are adjacent, can be non-zero.
- So the constraints ensure the SOS2 property, at the expense of introducing $N - 1$ additional binary variables and $N + 1$ constraints, if there are N points.

Non-positive variables

- If we wish to have a variable $\boldsymbol{x}$ that is non-positive, i.e., $\boldsymbol{x} \leq \mathbf{0}$
- $\boldsymbol{x}$ **is_free**
- $\boldsymbol{x} <= \mathbf{0}$
- Since a variable is by default non-negative (i.e., $\geq \mathbf{0}$) by only adding $\leq \mathbf{0}$, we force it to be $= \mathbf{0}$, then the **is_free** condition is necessary.