Mixed Integer, Semi, and SOS Programming

There are a number of features within GAMS that are designed for use in formulating and solving mixed integer, semi integer, semi continuous and specially ordered set (SOS) programming problems all of which require a mixed integer or MIP solver.

Specify types of variables ................................................................. 1
  Binary variables.............................................................................. 2
  Integer variables ........................................................................... 2
  Specially ordered set variables of type 1 (SOS1)............................. 3
  Specially ordered set variables of type 2 (SOS2).............................. 5
  Semi-continuous variables............................................................ 6
  Semi-integer variables .................................................................. 7

Imposing priorities ........................................................................... 7

GAMS options and model attributes .................................................. 8
  Modelname.Cheat = x; ................................................................. 8
  Modelname.Cutoff = x; ............................................................... 9
  Modelname.Nodlim = x; ............................................................. 9
  Modelname.Optca = x; ............................................................... 10
    Option Optca=X; ........................................................................ 10
  Modelname.Optcr = x; ............................................................... 10
    Option Optcr=X; ....................................................................... 10
  Modelname.Optfile = 1; ............................................................. 10
  Modelname.Prioropt = 1; ........................................................... 11
  Modelname.Tryint = x; .............................................................. 12

Branch and bound output ............................................................... 12

Nonlinear MIPs ............................................................................... 13

Identifying the solver ..................................................................... 13
  MINLP ..................................................................................... 13
  MIP ......................................................................................... 13
  RMIP ....................................................................................... 14
  RMINLP ................................................................................... 14

Model termination conditions and actions .................................... 14

Things to watch out for ................................................................. 14
  Default bounds........................................................................... 14
  Ending with a gap – big default for Optcr (10%)............................ 15
  The nonending quest................................................................. 15

Alphabetic list of features .............................................................. 16

Specify types of variables

The following types of variables fall into the mixed integer programming category in GAMS

- Binary variables (Binary). These can only take on values of 0 or 1.
- Integer variables (Integer). These can take on integer values between a lower and an upper bound. By default these variables are bounded in GAMS to the interval 0 to 100.

- Specially Ordered Sets Type 1 (SOS1). Groups of variables where only one member in each group can have a nonzero value in the solution.

- Specially Ordered Sets Type 2 (SOS2). Groups of variables where only two variables in the group can have nonzero solution levels and they must be adjacent.

- Semi-continuous variables (Semicont). Variables that must either be zero or can take on a continuous value above a threshold value.

- Semi-integer variables (Semiint). Variables that must be must either be zero or can take on an integer value above a threshold limit.

Each is discussed below.

**Binary variables**

These can take on values of 0 or 1 only. Binary variables are declared as follows

```gams
Binary Variable s1(i), t1(k,j), w1(i,k,j);
```

**Example**

```gams
POSITIVE VARIABLE X1
INTEGER VARIABLE X2
BINARY VARIABLE X3
VARIABLE OBJ
EQUATIONS OBJF
X1X2
X1X3;

OBJF.. 7*X1-3*X2-10*X3 =E= OBJ;
X1X2.. X1-2*X2 =L=0;
X1X3.. X1-20*X3 =L=0;
option optcr=0.01;
MODEL IPTEST /ALL/;
SOLVE IPTEST USING MIP MAXIMIZING OBJ;
```

**Notes**

- The lower bound of zero and upper bound of one restrictions do not need to be added as they are automatically generated.

- Often such variables are used in generating logical conditions such as imposing mutual exclusivity, complementarity, or other types of phenomenon as discussed in McCarl and Spreen

**Integer variables**
These can take on integer values between negative and a specified upper bound. Integer variables are declared as follows,

```
Integer Variable s1(i), t1(k,j), w1(i,k,j) ;
```

**Example**

basint.gms

```
POSITIVE VARIABLE       X1
INTEGER VARIABLE        X2
BINARY VARIABLE         X3
VARIABLE                OBJ
EQUATIONS               OBJF
X1X2
X1X3;
OBJF..     7*X1-3*X2-10*X3 =E= OBJ;
X1X2..     X1-2*X2 =L=0;
X1X3..     X1-20*X3 =L=0;
x2.up=125;
option optcr=0.01;
MODEL IPTEST /ALL/;
SOLVE IPTEST USING MIP MAXIMIZING OBJ;
```

**Notes**

- These variables are automatically bounded by GAMS so they have a default upper bound of 100. If the user wishes the integer variables to take on values greater than 100, a larger bound must be specified.
- A lower bound of zero is automatically generated. This may also be changed.

**Specially ordered set variables of type 1 (SOS1)**

At most one variable within a specially ordered set of type 1 (SOS1) can have a non-zero value. This variable can take any positive value. SOS1 variables are declared as follows:

```
SOS1 Variable s1(i), t1(k,j), w1(i,k,j) ;
```

The members of the right-most index for each named item are defined as belonging to the SOS1 group or set of variables of which at most one of which can be non zero.

For example, in the SOS1 variables defined above,

- s1 forms one group of mutually exclusive SOS1 variables which contains elements for each member of the set i and thus only one variable for one of the cases of i can be nonzero with the rest being zero.
- t1 defines a separate SOS1 set for each element of k and within each of those sets the variables indexed by j are SOS1 or mutually exclusive.
- w1 a separate SOS1 set for each pair of elements in i and k and within each of
those sets the variables indexed by \( j \) are SOS1 or mutually exclusive.

Example

PRODSCHX.gms from the GAMS model library shows formulations with binary, SOS1 and SOS2 sets.

Notes

- By default each SOS1 variable can range from 0 to infinity. As with any other variable, the user may set these bounds to whatever is required.
- One is required to utilize a mixed integer (MIP) solver to solve any model containing SOS1 variables. However, the SOS1 variables do not have to take on integer solution levels.
- The MIP solver is required because the solution process needs to impose mutual exclusivity and to do this it implicitly defines an additional set of zero one integer variables, then solves the problem as a MIP.
- The user can provide additional constraints say requiring the sum to the SOS1 variables in a set to be less than or equal to a quantity (often 1 for convexity). Consider the following example,

```plaintext
SOS1 Variable s1(i) ;
Equation defsoss1 ;
defsoss1.. sum(i,s1(i)) =l= 3.5 ;
```

Here the equation defsoss1 defines the largest non-zero value that one of the elements of the SOS1 variable \( s1 \) can take.

- A special case of SOS1 variables is when exactly one of the elements of the set has to be nonzero and equal to a number. In this case, the defSoss1 equation will be

```plaintext
defSoss1.. sum(i,s1(i)) =e= 10 ;
```

A common use of the use of this type of restriction is for the case where the right hand side in the equation above is 1. In such cases, the SOS1 variable is effectively a binary variable. In such a case, the SOS1 variable could just have been binary and the solution provided by the solver would be indistinguishable from the SOS1 case.

- Not all MIP solvers allow SOS1 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. A model that contains these variables may not be perfectly transferable among solvers. You should verify how the solver you are using handles SOS1 variables by checking the relevant section of the solver manual.
Specially ordered set variables of type 2 (SOS2)

At most two variables within a specially ordered set of type 2 (SOS2) can take on non-zero values. The two non-zero values have to be for adjacent variables in that set. Specially ordered sets of type 2 variables are declared as follows:

\[
\text{SOS2 Variable } s_2(i), t_2(k,j), w_2(i,j,k) ;
\]

The members of the right-most index for each named item are defined as belonging to a special (SOS2) group or set of variables of which at most one of which can be non zero.

For example, in the SOS1 variables defined above,

- s2 forms one group of SOS2 variables of which at most 2 can be non zero and they must be adjacent in terms of the set i. The adjacency means if the set i has elements /a,b,c,d,f,g/ that one could have any 2 variables like the ones associated with set elements a and b but never a and c since the set elements are not adjacent. This means the sets used must be ordered as discussed in the Sets chapter.

- t2 defines a separate SOS2 set for each element of k and within each of those sets no more than 2 variables can be non zero. Further, they must be adjacent in terms of the set j. The adjacency means if the set j has elements /j1,j2,j3,j4,j5,j6/ that one could have any 2 variables like j3 and j4 but never j1 and j6 since the set elements are not adjacent. This means the set j must be ordered as discussed in the Sets chapter.

- w2 defines a separate SOS2 set for each pair of elements in i and k. Within each of those sets no more than 2 variables can be non zero and they must be adjacent in terms of the set j. The adjacency means if the set j has elements /j1,j2,j3,j4,j5,j6/ that one could have any 2 variables like j3 and j4 but never j2 and j4 since the set elements are not adjacent. This means the set j must be ordered as discussed in the Sets chapter.

Example

PRODSCHX.gms from the model library shows formulations with binary, SOS1 and SOS2 sets.

Notes

- The most common use of SOS2 sets is to model piece-wise linear approximations to nonlinear functions using separable programming.

- One must use a mixed integer (MIP) solver to solve any model containing SOS2 variables. But, the SOS2 variables do not have to take on integer solution levels.

- The MIP solver is required because the solution process needs to impose both adjacency restrictions and the restrictions that no more than 2 nonzero level
values can be present and to do this the solvers implicitly defines an additional set of zero one variables, then solves the problem as a MIP.

- The default bounds for SOS2 variables are 0 to plus infinity. As with any other variable, the user may set these bounds to whatever is required.
- Not all MIP solvers allow SOS2 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. Thus a model that contains these variables may not be perfectly transferable among solvers. Please verify how the solver you are using handles SOS2 variables by checking the relevant section of the Solver Manual.

**Semi-continuous variables**

Semi-continuous variables are restricted, if non-zero, to take on a level above a given minimum and below given maximum. This can be expressed algebraically as:

Either

\[ x = 0 \]

or

\[ x \in [a, b) \]

By default, the lower bound (a) is 1.0 and the variable is upper bounded at infinity. The lower and upper bounds are set through the .lo and .up variable attributes as discussed in the Variables, Equations, Models and Solves chapter. In GAMS, a semi-continuous variable is declared using the reserved phrase Semicon variable. The following example illustrates its use.

```gams
semicont variable x;
.x.lo = 1.5; x.up = 23.1;
```

The above code declares the variable x to be a semi-continuous variable that can either be 0, or can behave as a continuous variable between 1.5 and 23.1.

**Notes**

- One is required to utilize a mixed integer (MIP) solver to solve any model containing Semi-continuous variables. However, these variables do not have to take on integer solution levels.
- The MIP solver is required because the solution process needs to impose the discontinuous jump between zero and the threshold value. To do this solvers implicitly define an additional zero one variable, and then solve the problem as a MIP.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than 0. GAMS will flag an error if it finds that this is not the case.
- Not all MIP solvers allow semi-continuous variables. Please verify that the solver
you are using can handle semi-continuous variables by checking the solver manual.

Semi-integer variables

Semi-integer variables are restricted, if non-zero, to take on an integer level above a given minimum level. This can be expressed algebraically as:

Either

\[ x = 0 \]

or

\[ x \geq a \text{ and integer} \]

By default, the lower bound (a) is set to 1.0 and the variable is upper bounded at 100. The lower and upper bounds are set through the .lo and .up variable attributes as discussed in the Variables, Equations, Models and Solves chapter.

In GAMS, a semi-integer variable is declared using the reserved phrase Semiint variable. The following example illustrates its use.

```gams
semiint variable x;
    x.lo = 2 ; x.up = 23 ;
```

The above declares the variable x to be a semi-continuous variable that can either be 0, or can behave as an integer variable between 2 and 23.

Notes

- One is required to utilize a mixed integer (MIP) solver to solve this problem type.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than 0. GAMS will flag an error if it finds that this is not the case.
- The variables are upper bounded at 100. If one wants larger bounds then they need to be specified.
- The bounds for semiint variables have to be set at integer values. GAMS will flag an error during model generation if it finds that this is not the case.
- Not all MIP solvers allow semi-continuous variables. Please verify that the solver you are interested in can handle semi-continuous variables by checking the Solver Manual.

Imposing priorities

In MIP models users can specify an order for picking variables to branch on during a branch and bound search. This is done through the use of priorities. Without priorities,
the MIP algorithm will internally determine which variable is the most suitable to branch on.

Priorities are set for individual variables through the use of the .prior variable attribute as discussed in the Variables, Equations, Models and Solves chapter. The closer to one the setting of the .prior variable attribute, then the higher the priority the variable is given when it is one of the eligible candidates for branching upon in the MIP solver. Priorities can be set to any real value. The default value is 1.0. The most important variables should be given the highest priority and this means they should have the closest to one nonzero values of the prior attribute.

For priorities to be used the user must activate them through use of the GAMS model attribute statement

```
mymodel.prioropt = 1;
```

where mymodel is the name of the model specified in the model statement for the problem to be solved as discussed in the Model Attributes chapter. The default value is 0 in which case priorities will not be used.

**Example**

The following example illustrates its use,

```
mymodel.prioropt = 1;
z.prior(i,'small') = 3;
z.prior(i,'medium') = 2;
z.prior(i,'large') = 1;
```

In the above example, z(i,'large') variables are branched on before z(i, 'small') variables.

**Notes**

1. The higher the value given to the .prior suffix, the lower the priority for branching.
2. Note that there is a prior variable attribute for each individual component of a multidimensional variable.
3. All members of any SOS1 or SOS2 set should be given the same priority value.

**GAMS options and model attributes**

GAMS has a number of options and model attributes that can be used to influence MIP solver performance. They are invoked as discussed below or in the Model Attributes chapter.

```
Modelname.Cheat = x;
```
The cheat value requires each new integer solution to be at least $x$ better than the previous one. This can reduce the number of nodes that the MIP solver examines and can improve problem solving efficiency. However, setting this option at a positive value (zero is the default) can cause some integer solutions, including the true integer optimum, to be missed. When a model has been solved with the cheat parameter set at a nonzero level than all one is able to say is that the optimum solution is within the cheat parameter or less of the solution found. The cheat parameter is specified in absolute terms (like the Optca option). Certain solver options override the cheat setting.

Use of this parameter is done using a command like (basint.gms)

```
  ipatest.cheat=0.1;
```

where the model being solved is named ipatest and cheat is set to 0.1.

Integer programming solver option file parameters like the CPLEX option objdif can override the cheat attribute value.

**Modelname.Cutoff = x;**

As the branch and bound search proceeds, the parts of the tree with an objective worse than the cutoff value $x$ are ignored. This can speed up the initial phase of the branch and bound algorithm (before the first integer solution is found). However, setting this option at a positive value (zero is the default) can cause some integer solutions, including the true integer optimum, to be missed if in a maximization, it's value is above the cutoff. In fact, if you set cutoff below the optimum, you get no solution -- not just some missed integer solutions. Cutoff may also cause one to miss finding an initial feasible integer solution.

The cutoff parameter is specified in absolute terms (like the Optca option).

Use of this parameter is done using a command like (basint.gms)

```
  ipatest.cutoff=12;
```

where the model being solved is named ipatest and cutoff is set to 12.

**Modelname.Nodlim = x;**

This attribute specifies the maximum number of nodes to process in the branch and bound tree for a MIP problem. This can stop solutions that are exhibiting "excessive" iterations and if the limit is reached causes the algorithm to terminate, without reaching optimality. The Nodlim parameter is specified as an integer.

Use of this parameter is done using a command like (secur.gms)
security.nodlim=10000;

where the model being solved is named security and nodlim is set to 10000.

**Modelname.Optca = x;**  
**Option Optca=X;**

This specifies the absolute optimality criterion for a MIP problem. In general, GAMS tells the solvers to stop trying to improve upon the integer solution and stop calling the solution close enough to optimal when

\[ (|BP - BF|) < \text{Optca}, \]

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution.

This reduces solution time as the solver stops not looking for better solutions. However, setting this option at a positive value (zero is the default) can cause the true integer optimum to be missed if it's value is within Optca of the best solution on hand when the problem stops. The final solution could be the best, but is guaranteed only to be within the tolerance of the "true optimal".

Optca is specified in absolute terms relative to the objective value. Thus a value of 100 means the objective value will be within the 100 units of the true objective value.

Use of this parameter involves a command like *(basint.gms)*

```
 iptest.optca=12;
```

or

```
 Option optca=12;
```

where the model being solved is named iptest and optca is set to 12.

**Modelname.Optcr = x;**  
**Option Optcr=X;**

This specifies the relative optimality criterion for a MIP problem. In general GAMS tells the solvers to stop trying to improve upon the integer solution when

\[ (|BP - BF|)/(|BF|) < \text{Optcr}, \]

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. However some solvers, in particular CPLEX use slightly different definitions. The Optcr option is used in CPLEX to stop when \( (|BP - BF|)/(1.0e-10 + |BF|) < \text{Optcr} \).
In turn the solver stops after finding a solution proven to be "close enough" (within the Optcr tolerance) to optimal. This reduces solution time as the solver stops not looking for better solutions. However, setting this option at a positive value (0.1 is the default) can cause the true integer optimum to be missed if it's value is within Optcr of the best solution on hand when the problem stops. The final solution could be the best but is guaranteed only to be within the tolerance of the "true optimal".

The Optcr parameter is specified in proportional terms relative to the objective value thus a value of 0.10 means the objective value will be within the 10% of the true objective value.

Use of this parameter is done using a command like (basint.gms)

    iptest.optcr=0.012;

or

    Option optcr=0.012;

where the model being solved is named iptest and optcr is set to 0.012 or 1.2%. The default value for Optcr is large being 0.10 or 10%.

**Modelname.Optfile = 1;**

Instructs the mixed integer solver to read an options file as discussed in the OptionFile chapter. The name of the option file is solvername.opt (ie cplex.opt or xa.opt or osl.opt). Solver options allow one to manipulate the way solvers work affecting a number of solver functions including choice of the branch and bound tree handing strategies. The solver manuals cover the allowable options.

**Modelname.Prioropt = 1;**

Instructs the mixed integer solver to use priority branching information passed by GAMS through the variable.prior attributes. If used priorities should reflect knowledge of the problem. Variables with higher priorities – lower values of the .prior attribute -- will be branched upon before variables of lower priorities. This information indicates user specified preferred directions for the internal branch and bound tree search and can dramatically reduce the number of nodes searched.

Problem knowledge may indicate what could be considered first. For example, consider a problem with a binary variable representing a yes/no decision to build a factory, and other binary variables representing equipment selections within that factory. You would naturally want to explore whether or not the factory should be built before considering what specific equipment to be purchased within the factory so you would set the priority values lower for the build variables. By assigning a higher priority – lower value of prior- to the build/nobuild decision variable, you can force this logic into the tree search and speed up computation time by not exploring uninteresting portions of the tree.
Note priorities are not needed and the branch and bound codes used sophisticated branching criteria involving potential of the variable to affect the objective function value.

**Modelname.Tryint = x;**

Causes the mixed integer solver to try to make use of the available initial integer solution. The exact form of implementation depends on the solver and can be in part controlled by solver settings or options. See the solver manuals for details.

**Branch and bound output**

When the model `secur.gms` was solved with an earlier version of CPLEX it yielded output like the following

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Integer</th>
<th>Best Node</th>
<th>ItCnt</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2.2303e+007</td>
<td>24</td>
<td>2.2303e+007</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>93</td>
<td>2.2303e+007</td>
<td>1</td>
<td>2.2303e+007</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>193</td>
<td>2.2303e+007</td>
<td>1</td>
<td>2.2303e+007</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 280+</td>
<td>266</td>
<td>2.2303e+007</td>
<td>0</td>
<td>2.2303e+007</td>
<td>53 0.00%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>270</td>
<td>2.2303e+007</td>
<td>1</td>
<td>2.2303e+007</td>
<td>53 0.00%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 390</td>
<td>65</td>
<td>2.2303e+007</td>
<td>0</td>
<td>2.2303e+007</td>
<td>104 0.00%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fixing integer variables, and solving final LP...

**MIP Solution** : 22303062.100023 (104 iterations, 391 nodes)

**Final LP** : 22303062.100023 (0 iterations)

Best integer solution possible : 22303113.765793

Absolute gap : 51.6658

Relative gap : 2.31653e-006

This output will differ across solvers but generally contains the same types of information showing the branch and bound approach in action. Namely the columns by label are

- **Node** is number of branch and bound problems examined so far.
- **Nodes left** is number of problems created during the branching process that are yet to be examined.
- **IInf** tells number of integer variables with non-integer solution levels.
- **Objective** gives the current objective function value.
- **Best node** gives the current lower bound on the solution. Similarly the column
- **Best integer** gives the incumbent solution. Note the last solution in that column is not necessarily global best.
- **Gap** gives max percentage difference from theoretical optimum.

Here we see, no solution is found for a while (indicated by blank entry in Best Integer until iteration 280), then one found and another. This shows the common phenomena that MIP solves usually end with a gap between the solution found and the best possible. This
is controlled by iteration limits, resource limits, solver options, and model attributes like optcr/optca.

## Nonlinear MIPs

Modelers may wish to impose integer restrictions on nonlinear formulations. Today GAMS contains the DICOPT and SBB solvers that permit this. They tie together other solvers. For example both can use CONOPT to solve the nonlinear sub-problems. DICOPT also uses MIP solvers on the integer part of the problem while SBB contains an internal integer solution algorithm.

For example suppose we impose restrictions in a portfolio problem that a minimum of 10 shares be bought if any and that we buy integer numbers of shares (INTEV.gms)

```plaintext
Integer VARIABLES INVEST(STOCKS) MONEY INVESTED IN EACH STOCK
binary variables mininv(stocks) at least 10 shares bought
VARIABLE OBJ NUMBER TO BE MAXIMIZED ;
EQUATIONS OBJJ OBJECTIVE FUNCTION
    INVESTAV INVESTMENT FUNDS AVAILABLE
    minstock(stocks) at least 10 units to be bought
    maxstock(stocks) Set up indicator variable ;
OBJJ.. OBJ =E= SUM(STOCKS, MEAN(STOCKS) * INVEST(STOCKS))
       - RAP*(SUM(STOCK, SUM(STOCKS, INVEST(STOCK) * COVAR(STOCK,STOCKS) * INVEST(STOCKS))));
    INVESTAV.. SUM(STOCKS, PRICES(STOCKS) * INVEST(STOCKS)) =L= FUNDS;
    minstock(stocks).. invest(stocks) =g= 10*mininv(stocks);
    maxstock(stocks).. invest(stocks)=l=1000*mininv(stocks);
MODEL EVPORTFOL /ALL/ ;
SOLVE EVPORTFOL USING MINLP MAXIMIZING OBJ ;
```

When using DICOPT and SBB it is very important to have the constraints represent to the full extent possible the link between continuous and integer variables.

## Identifying the solver

MIP problems are sometimes hard to solve and sometimes involve trying out different alternatives. Option statements are involved as discussed in the options chapter.

### MINLP

This option specifies what solver GAMS will use when it needs to solve a MINLP type of model. This option is used by setting

```plaintext
Option MINLP=solvername;
```

where the solver must be MINLP capable.

### MIP

This option specifies what solver GAMS will use when it needs to solve a MIP type of
model. This option is used by setting

    Option MIP=solvername;

where the solver must be MIP capable.

**RMIP**

This option specifies what solver GAMS will use when it needs to solve a RMIP type of model. This option is used by setting

    Option RMIP=solvername;

Where the solver must be RMIP capable.

**RMINLP**

This option specifies what solver GAMS will use when it needs to solve a RMINLP type of model. This option is used by setting

    Option RMINLP=solvername;

Where the solver must be RMINLP capable.

**Model termination conditions and actions**

The following termination conditions can occur

<table>
<thead>
<tr>
<th>Value of Modelstat</th>
<th>Indicated Model Solution Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Integer solution found</td>
</tr>
<tr>
<td>9</td>
<td>Solver terminated early with a non-integer solution found(only in MIPs)</td>
</tr>
<tr>
<td>10</td>
<td>No feasible integer solution could be found</td>
</tr>
</tbody>
</table>

When a number 9 occurs one may need to consider whether the gap is satisfactory or whether the model has to be run for a longer time. When 10 occurs there truly may be no feasible MIP solution and this can be hard to diagnose. In such cases one should certainly make sure that the RMIP has a feasible solution, then try to fix in an integer solution that should be feasible and find out why not.

**Things to watch out for**

There are some problems one may have either due to GAMS settings or problem characteristics. I summarize three of these below.
Default bounds

One needs to be aware that the GAMS default bound limits the maximum value of the integer variable to 100. One needs to reset this if the values are allowed to be higher as illustrated above.

Ending with a gap – big default for Optcr (10%)

MIPs solves often end with a gap between the solution found and the best possible. This is controlled by optcr/optca or by non convergence. The default value of Optcr is relatively large being 0.10 or a 10% gap as discussed above. Users may want to reduce this to a smaller value. The other cause of a gap is discussed just below.

The nonending quest

Integer programming is a quite desirable formulation technique. But, integer problems can be hard to solve due to search nature of solution process

Three approaches can help

- Reformulate reflecting as much problem knowledge in the formulation as possible improving the depiction of the
  - Way the integer variables are tied together by the constraints. For example, entering constraints that reflect that if one size of machine is chosen in the first stage of an assembly line that it must be matched with a comparable machine in the second stage.
  - Way the integer and continuous are tied together by the constraints. For example, one could achieve benefits by entering constraints indicating that the sum of the continuous variables depicting volume through a warehouse whose construction is depicted by integer variable representing warehouse must be no more than capacity times the integer variable and no less than say 75% of capacity times the integer variable.
  - Restricting the values of the integer variables eliminating “unnecessary” cases of integer variables. For example, in a warehouse location problem if only one warehouse can practically be built require the sum across the variables to be one. Similarly if experience says at least 2 machines are necessary but no more than 5 then enter such constraints.

- Use MIP solver features through options and GAMS. Sometimes spectacular reductions in solution time can be achieved in very little time by changing solver branch and bound procedures.
- Expand available resources or allowable iterations.
Also these solves are generally slower so one must be patient

### Alphabetic list of features

<table>
<thead>
<tr>
<th>Binary variables</th>
<th>Variables that can take on values of 0 or 1 only.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.Cheat</td>
<td>Model attribute requiring each new integer solution</td>
</tr>
<tr>
<td></td>
<td>to be at least a tolerance better than the previous</td>
</tr>
<tr>
<td></td>
<td>one.</td>
</tr>
<tr>
<td>.Cutoff</td>
<td>Model attribute causing the MIP solver to disregard</td>
</tr>
<tr>
<td></td>
<td>parts of the tree with an objective worse than a</td>
</tr>
<tr>
<td></td>
<td>value</td>
</tr>
<tr>
<td>DICOPT</td>
<td>Non linear mixed integer solver</td>
</tr>
<tr>
<td>Integer variables</td>
<td>Variables that must take on integer values</td>
</tr>
<tr>
<td>MINLP</td>
<td>A type of optimization problem commonly called a</td>
</tr>
<tr>
<td></td>
<td>mixed integer non linear problem</td>
</tr>
<tr>
<td>MINLP</td>
<td>A keyword used in solve statements to invoke use</td>
</tr>
<tr>
<td></td>
<td>of a MINLP solver</td>
</tr>
<tr>
<td>MINLP</td>
<td>An option keyword used to define the currently</td>
</tr>
<tr>
<td></td>
<td>active MINLP solver</td>
</tr>
<tr>
<td>MINLP</td>
<td>An command line keyword used to define the</td>
</tr>
<tr>
<td></td>
<td>currently active MINLP solver</td>
</tr>
<tr>
<td>MINLP</td>
<td>A customization parameter used to identify default</td>
</tr>
<tr>
<td></td>
<td>MINLP solver</td>
</tr>
<tr>
<td>MIP</td>
<td>A type of optimization problem commonly called a</td>
</tr>
<tr>
<td></td>
<td>mixed integer problem</td>
</tr>
<tr>
<td>MIP</td>
<td>A keyword used in solve statements to invoke use</td>
</tr>
<tr>
<td></td>
<td>of a MIP solver</td>
</tr>
<tr>
<td>MIP</td>
<td>An option keyword used to define the currently</td>
</tr>
<tr>
<td></td>
<td>active MIP solver</td>
</tr>
<tr>
<td>MIP</td>
<td>An command line keyword used to define the</td>
</tr>
<tr>
<td></td>
<td>currently active MIP solver</td>
</tr>
<tr>
<td>MIP</td>
<td>A customization parameter used to identify default</td>
</tr>
<tr>
<td></td>
<td>MIP solver</td>
</tr>
<tr>
<td>.NodLim</td>
<td>Model attribute limiting the maximum number of</td>
</tr>
<tr>
<td></td>
<td>nodes that can be examined in a MIP solution</td>
</tr>
<tr>
<td>Nonlinear MIPs</td>
<td>Integer programming problems with nonlinear terms</td>
</tr>
<tr>
<td>Optca</td>
<td>Model attribute or option command telling the</td>
</tr>
<tr>
<td></td>
<td>solver to stop when best solution is no more than</td>
</tr>
<tr>
<td></td>
<td>a given amount away from the best solution</td>
</tr>
<tr>
<td>Optcr</td>
<td>Model attribute or option command telling the</td>
</tr>
<tr>
<td></td>
<td>solver to stop when best solution is no more than</td>
</tr>
<tr>
<td></td>
<td>a given proportion of the best solution away</td>
</tr>
<tr>
<td></td>
<td>from the best solution</td>
</tr>
<tr>
<td>Optcr</td>
<td>Default value</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.OptFile</td>
<td>Model attribute activating option files for MIP solvers</td>
</tr>
<tr>
<td>.Prior</td>
<td>Variable attribute specifying priority for a variable – the lower the value the higher the priority</td>
</tr>
<tr>
<td>Priorities</td>
<td>Way of specifying an order for picking variables to branch on during a MIP branch and bound solution.</td>
</tr>
<tr>
<td>.Prioropt</td>
<td>Model attribute activating MIP priorities</td>
</tr>
<tr>
<td>RMINLP</td>
<td>Mixed integer non linear problem with integer variables made continuous over their feasible region</td>
</tr>
<tr>
<td>RMIP</td>
<td>Mixed integer problem with integer variables made continuous over their feasible region</td>
</tr>
<tr>
<td>SBB</td>
<td>Non linear mixed integer solver</td>
</tr>
<tr>
<td>Semicont variables</td>
<td>Variables that are zero or continuous above a threshold value</td>
</tr>
<tr>
<td>Semiint variables</td>
<td>Variables that are zero or are integer above a threshold value</td>
</tr>
<tr>
<td>SOS1 variables</td>
<td>Variables in groupings where only one variable in the group can be nonzero.</td>
</tr>
<tr>
<td>SOS2 variables</td>
<td>Variables in groupings where only two variables in the group can be nonzero they must be adjacent</td>
</tr>
<tr>
<td>.TryInt</td>
<td>Model attribute causing MIP solvers to make use of current variable values when solving a MIP problem.</td>
</tr>
</tbody>
</table>