SBB

Contents

Release Notes

- April 30, 2002: Level 009
  - NLP solvers sometimes have difficulty proving the optimality of a good point. The way they report that solution is with solver status "Terminated by Solver" and model status "Intermediate Nonoptimal". SBB's default behavior is to ignore such a solution (potentially go into failseq). With the new option `acceptnonopt` these solutions are accepted for further action.
  - SBB offers node selections that switch between DFS and best bound/best estimate selection. In DFS mode SBB usually switches back to best bound/estimate if the DFS search resulted in a pruned or infeasible node or a new integer solution was found. In these cases it can be advantageous to search the close neighborhood of that node also in a DFS fashion. With the new option `dfsstay` SBB is instructed to do some more DFS nodes even after a switch to best bound/estimate has been requested.
- January 11, 2002: Level 008
  - Maintenance release
- December 11, 2001: Level 007
  - NLP solvers sometimes have difficulty solving particular nodes and using up all the resources in this node. SBB provides options (see `subres, subiter`) to overcome these instances, but these options must be set in advance. SBB now keeps track of how much time is spent in the nodes, builds an average over time, and automatically controls the time spend in each node. The option `avgresmult` allows the user to customize this new feature.
- November 13, 2001: Level 006
  - Maintenance release
- May 22, 2001: Level 005
  - SBB derives an implicit, absolute termination tolerance if the model has a `discrete` objective row. This may speed up the overall time if the user has tight termination tolerances (optca, optcr).
  - SBB passes indices of rows with domain violations back to the LST file. All domain violation from the root node and from all sub nodes are reported, and the user can take advantage of this information to overcome these violations.

- March 21, 2001: Level 004
  - Pseudo Costs are available in SBB. Check section SBB with Pseudo Costs
  - A mix of DFS/Best Bound/Best Estimate node selection schemes are available through the `nodesel` option.
  - The `tryint` option now works for integer variables.
  - Additional information about node and variable selection can be printed to the Log file using the `printbbinfo` option.
- December 22, 2000: Level 003
  - MINOS and SNOPT are available as NLP subsolvers.
- November 19, 2000: Level 002
  - The model and solver status returned by SBB are synchronized with return codes reported by DICOPT.
- October 16, 2000: Level 001
  - SBB introduced to GAMS distribution 19.5.
  - CONOPT and CONOPT2 are the available NLP subsolvers.

1 Introduction

SBB is a new GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. It is based on a combination of the standard Branch and Bound (B&B) method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. Currently, SBB can use

- CONOPT
- MINOS
- SNOPT

as solvers for submodels.

SBB supports all types of discrete variables supported by GAMS, including:

- Binary
- Integer
- Semicont
- Semiint
- Sos1
- Sos2

2 The Branch and Bound Algorithm

The Relaxed Mixed Integer Nonlinear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails (see option `infeasseq` and `failseq` below for an exception). If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch and Bound procedure will start.

During the Branch and Bound process, the feasible region for the discrete variables is subdivided, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP submodel is solved starting from the optimal solution to the previous looser submodel. The objective function values from the NLP submodel is assumed to be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a submodel, it is usually assumed that there is no feasible solution to the submodel, even though the infeasibility only has been determined locally (see option `infeasseq` below for an exception). If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.

3 SBB with Pseudo Costs

Over the last decades quite a number of search strategies have been successfully introduced for mixed integer linear programming (for details see e.g. J.T. Linderoth and M.W.P. Savelsbergh, A Computational Study of Search Strategies for Mixed Integer Programming, INFORMS Journal on Computing, 11(2), 1999). Pseudo costs are key elements of sophisticated search strategies. Using pseudo costs, we can estimate the degradation of the objective function if we move a fractional variable to a close integer value. Naturally, the variable selection can be based on pseudo costs (see SBB option `varsel`). Node selection can also make use of pseudo cost: If we can estimate the change of the objective for moving one fractional variable to the closed integer value, we can then aggregate this change for all fractional variables, to estimate the objective of the best integer solution reachable from a particular node (see SBB option `nodesel`).

Unfortunately, the computation of pseudo cost can be a substantial part of the overall computation. Models with a large number of fractional variables in the root node are **not** good candidates for search strategies which require pseudo costs (`varsel 3, nodesel 3,5,6`). The impact (positive or negative) of using pseudo cost depends significantly on the particular model. At this stage, general statements are difficult to make.

Selecting pseudo cost related search strategies (`varsel 3, nodesel 3,5,6`) may use computation time which sometimes does not pay off. However, we encourage the user to try these options for difficult models which require a large number of branch-and-bound nodes to solve.

4 The SBB Options

SBB works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are `iterlim, reslim, nodlim, optca, optcr, optfile, cheat`, and `cutoff`. A description of all available GAMS options can be found in Chapter "Basic Solver Usage". GAMS options `prioropt` and `tryint` are also accepted by SBB.

SBB uses the `var.prior` information to select the fractional variable with the smallest priority during the variable selection process. SBB uses the `tryint` information to set the branching direction in the B&B algorithm. At the beginning, SBB looks at the levels of the discrete variables provided by the user and if $\text{abs}(\text{round}(x.l) - x.l) < m.\texttt{tryint}$, SBB will branch on that variable in the direction of $\text{round}(x.l)$. For example, $x.l = 0.9$ and $m.\texttt{tryint} = 0.2$. We have $\text{abs}(\text{round}(0.9) - 0.9) = 0.1 < 0.2$, so when SBB decides to branch on this variable (because it is fractional, lets say with value 0.5), the node explored next will have the additional constraint $x \geq 1$ (the node with $x \leq 0$ will be explored later). If everything goes well (there is the chance that we end up in a different local optima in the subsolves for non-convex problems), SBB should reproduce a preset incumbent solution in a couple of nodes.

If you specify "`<modelname>.optfile = 1;`" before the SOLVE statement in your GAMS model, SBB will then look for and read an option file with the name *sbb.opt* (see "Using Solver Specific Options" for general use of solver option files). Unless explicitly specified in the SBB option file, the NLP subsolvers will not read an option file. The syntax for the SBB option file is

```
optname value
```

with one option on each line.

For example,

```
rootsolver conopt.1
subsolver snopt
loginterval 10
```

The first two lines determine the NLP subsolvers for the Branch and Bound procedure. CONOPT with the option file *conopt.opt* will be used for solving the root node. SNOPT with no option file will be used for the remaining nodes. The last option determines the frequency for log line printing. Every 10th node, and each node with a new integer solution, causes a log line to be printed. The following options are implemented:

Option	Description	Default
rootsolver	solver[.n] Solver is the name of the GAMS NLP solver that should be used in the root node, and n is the integer corresponding to optfile for the root node. If .n is missing, the optfile treated as zero (i.e., the NLP solver) will not look for an options file. This SBB option can be used to overwrite the default that uses the NLP solver specified with an Option NLP = *solver*; statement or the default GAMS solver for NLP.	GAMS NLP solver
subsolver	solver[.n] Similar to rootsolver but applied to the subnodes.	GAMS NLP solver
solvelink	This option defines the solvelink used for the NLP solver: 1: Call NLP solver via script 2: Call NLP and MIP solver via module 5: Call NLP and MIP solver in memory	5
loginterval	The interval (number of nodes) for which log lines are written.	1
loglevel	The level of log output: 0: only SBB log lines with one line every loginterval nodes 1: NLP solver log for the root node plus SBB loglines as 0 2: NLP solver log for all nodes plus SBB log lines as 0	1
subres	The default for subres passed on through reslim. Sets the time limit in seconds for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure and the node is ignored, or the solvers in the failseq are called.	reslim
subiter	The default for subiter passed on through iterlim. Similar to subres but sets the iteration limit for solving a node in the B&B tree.	iterlim
failseq	solver1[.n1] solver2[.n2] ... where solver1 is the name of a GAMS NLP solver to be used if the default solver fails, i.e., if it was not stopped by an iteration, resource, or domain limit and does not return a locally optimal or locally infeasible solution. n1 is the value of optfile passed to the alternative NLP solver. If .n1 is left blank it is interpreted as zero. Similarly, solver2 is the name of a GAMS NLP solver that is used if solver1 fails, and n2 is the value of optfile passed to the second NLP solver. If you have a difficult model where solver failures are not unlikely, you may add more solver.n pairs. You can use the same solver several times with different options files. failseq conopt conopt.2 conopt.3 means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file *conopt.op2*, and if it again fails, try CONOPT with options file *conopt.op3*. If all solver and options file combinations fail the node will be labeled "ignored" and the node will not be explored further. The default is to try only one solver (the rootsolver or subsolver) and to ignore nodes with a solver failure.	None
infeasseq	level solver1[.n1] solver2[.n2] ... The purpose of infeasseq is to avoid cutting parts of the search tree that appear to be infeasible but really are feasible. If the NLP solver labels a node "Locally Infeasible" and the model is not convex a feasible solution may actually exist. If SBB is high in the search tree it can be very drastic to prune the node immediately. SBB is therefore directed to try the solver/option combinations in the list as long as the depth in the search tree is less than the integer value <level>. If the list is exhausted without finding a feasible solution, the node is assumed to be infeasible. The default is to trust that Locally Infeasible nodes are indeed infeasible and to remove them from further consideration.	None
acceptnonopt	If this option is set to 1 and the subsolver terminates with solver status "Terminated by Solver" and model status "Intermediate Nonoptimal" SBB takes this as a good solution and keeps on going. In default mode such a return is treated as a subsolver failure and the failseq is consulted.	0

Option	Description	Default
avgresmult	Similar to subres, this option allows the user to control the time limit spend in a node. SBB keeps track of how much time is spent in the nodes, and builds an average over time. This average multiplied by the factor avgresmult is set as a time limit for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure: the node is ignored or the solvers in the failseq are called. The default multiplier avgresmult is 5. Setting avgresmult to 0 will disable the automatic time limit feature. A multiplier is not very useful for very small node solution times; therefore, independent of each node, SBB grants the solver at least 5 seconds to solve the node. The competing option subres overwrites the automatically generated resource limit.	5
nodesel	Node selection: 0: automatic 1: Depth First Search (DFS) 2: Best Bound (BB) 3: Best Estimate (BE) 4: DFS/BB mix 5: DFS/BE mix 6: DFS/BB/BE mix	0
dfsstay	If the node selection is a B*/DFS mix, SBB switches frequently to DFS node selection mode. It switches back into B* node selection mode, if no subnodes were created (new int, pruned, infeasible, fail). It can be advantageous to search the neighborhood of the last node also in a DFS manner. Setting dfsstay to n instructs SBB to stay in DFS mode for another n nodes.	0
varsel	Variable selection: 0: automatic 1: maximum integer infeasibility 2: minimum integer infeasibility 3: pseudo costs	0
epint	The integer infeasibility tolerance.	1.0e-5
memnodes	The maximum number of nodes SBB can have in memory. If this number is exceeded, SBB will terminate and return the best solution found so far.	10000
printbbinfo	Additional info of log output: 0: no additional info 1: the node and variable selection for the current node are indicated by a two letter code at the end of the log line. The first letter represents the node selection: D for DFS, B for Best Bound, and E for Best Estimate. The second letter represents the variable selection: X for maximum infeasibility, N for minimum infeasibility, and P for pseudo cost.	0
intsollim	maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.	99999

5 The SBB Log File

The SBB Log file (usually directed to the screen) can be controlled with the loginterval and loglevel options in SBB. It will by default first show the iteration output from the NLP solver that solves the root node. This is followed by output from SBB describing the search tree. An example of this search tree output follows:

```
Root node solved locally optimal.
   Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
      0     0   0    8457.6878   3            -       8457.6878       -
      1     1   1    8491.2869   2            -       8457.6878       -
      2     2   2    8518.1779   1            -       8457.6878       -
*     3     3   3    9338.1020   0    9338.1020       8457.6878  0.1041
      4     2   1       pruned   -    9338.1020       8491.2869  0.0997
```

```
Solution satisfies optcr
Statistics:
   Iterations    :                90
   NLP Seconds   :          0.110000
   B&B nodes     :                 3
   MIP solution  :       9338.101979 found in node 3
   Best possible :       8491.286941
   Absolute gap  :        846.815039     optca :  0.000000
   Relative gap  :          0.099728     optcr :  0.100000
   Model Status  :                 8
   Solver Status :                 1

NLP Solver Statistics
   Total Number of NLP solves  :          7
   Total Number of NLP failures:          0
   Details:         conopt
     # execs             7
     # failures          0
Terminating.
```

The fields in the log are:

Field	Description
`Node`	The number of the current node. The root node is node 0.
`Act`	The number of active nodes defined as the number of subnodes that have not yet been solved.
`Lev`	The level in the search tree, i.e., the number of branches needed to reach this node.
`Objective`	The objective function value for the node. A numerical value indicates that the node was solved and the objective was good enough for the node to not be ignored. ”pruned” indicates that the objective value was worse than the Best Integer value, ”infeasible” indicates that the node was Infeasible or Locally Infeasible, and ”ignored” indicates that the node could not be solved (see under failseq above).
`IInf`	The number of integer infeasibilities, i.e. the number of variables that are supposed to be binary or integer that do not satisfy the integrality requirement. Semi continuous variables and SOS variables may also contribute to `IInf`.
`Best Int`	The value of the best integer solution found so far. A dash (-) indicates that an integer solution has not yet been found. A star (*) in column one indicates that the node is integer and that the solution is better than the best yet found.
`Best Bound`	The minimum value of ”Objective” for the subnodes that have not been solved yet (maximum for maximization models). For convex models, Best Bound will increase monotonically. For nonconvex models, Best Bound may decrease, indicating that the Objective value for a node was not a valid lower bound for that node.
`Gap`	The relative gap between the Best Integer solution and the Best Bound.

The remaining part of the Log file displays various solution statistics similar to those provided by the MIP solvers. This information can also be found in the Solver Status area of the GAMS listing file.

The following Log file shows cases where the NLP solver fails to solve a subnode. The text ”ignored” in the Objective field shows the failure, and the values in parenthesis following the Gap field are the Solve and Model status returned by the NLP solver:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
   1     1   1   infeasible   -            -       6046.0186       -
   2     0   1    6042.0995  10            -       6042.0995       -
   3     1   2      ignored   -            -       6042.0995       - (4,6)
   4     0   2    5804.5856   8            -       5804.5856       -
```

```
   5     1   3      ignored   -            -       5804.5856       - (4,7)
```

The next Log file shows the effect of the `infeasseq` and `failseq` options on the model above. CONOPT with options file *conopt.opt* (the default solver and options file pair for this model) considers the first subnode to be locally infeasible. CONOPT1, MINOS, and SNOPT, all with no options file, are therefore tried in sequence. In this case, they all declare the node infeasible and it is considered to be infeasible.

In node 3, CONOPT fails but CONOPT1 finds a Locally Optimal solution, and this solution is then used for further search. The option file for the following run would be:

```
rootsolver conopt.1
subsolver conopt.1
infeasseq conopt1 minos snopt
```

The log looks as follows:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
conopt.1 reports locally infeasible
Executing conopt1
conopt1 reports locally infeasible
Executing minos
minos reports locally infeasible
Executing snopt
   1     1    1  infeasible   -            -       6046.0186       -
   2     0    1   6042.0995  10            -       6042.0995       -
conopt.1 failed. 4 TERMINATED BY SOLVER, 7 INTERMEDIATE NONOPTIMAL
Executing conopt1
   3     1    2   4790.2373   8            -       6042.0995       -
   4     2    3   4481.4156   6            -       6042.0995       -
conopt.1 reports locally infeasible
Executing conopt1
conopt1 reports locally infeasible
Executing minos
minos failed. 4 TERMINATED BY SOLVER, 6 INTERMEDIATE INFEASIBLE
Executing snopt
   5     3    4  infeasible   -            -       6042.0995       -
   6     2    4   4480.3778   4            -       6042.0995       -
```

The Log file shows a solver statistic at the end, summarizing how many times an NLP was executed and how often it failed:

```
NLP Solver Statistics
   Total Number of NLP solves  :   45
   Total Number of NLP failures:   13
   Details:     conopt     minos   snopt
     # execs        34         3       8
     # failures      4         3       6
```

The solutions found by the NLP solver to the subproblems in the Branch and Bound may not be the global optima. Therefore, the objective can improve even though we restrict the problem by tightening some bounds. These *jumps* of the objective in the *wrong* direction which might also have an impact on the best bound/possible are reported in a separate statistic:

```
Non convex model!
   # jumps in best bound       :           2
   Maximum jump in best bound  :   20.626587 in node 13
   # jumps to better objective :           2
   Maximum jump in objective   :   20.626587 in node 13
```

6 Comparison of DICOPT and SBB

Until recently, MINLP models could only be solved with the DICOPT solver. DICOPT is based on the outer approximation method. Initially, the RMINLP model is solved just as in SBB. The model is then linearized around this point and a linear MIP model is solved. The discrete variables are then fixed at the optimal values from the MIP model, and the resulting NLP model is solved. If the NLP model is feasible, we have an integer feasible solution.

The model is linearized again and a new MIP model with both the old and new linearized constraints is solved. The discrete variables are again fixed at the optimal values, and a new NLP model is solved.

The process stops when the MIP model becomes infeasible, when the NLP solution becomes worse, or, in some cases, when bounds derived from the MIP model indicate that it is safe to stop.

DICOPT is based on the assumption that MIP models can be solved efficiently while NLP models can be expensive and difficult to solve. The MIP models try to approximate the NLP model over a large area and solve it using cheap linear technology. Ideally, only a few NLPs must be solved.

DICOPT can experience difficulties solving models, if many or all the NLP submodels are infeasible. DICOPT can also have problems if the linearizations used for the MIP model create ill-conditioned models. The MIP models may become very difficult to solve, and the results from the MIP models may be poor as initial values for the NLP models. The linearized constraint used by DICOPT may also exclude certain areas of the feasible space from consideration.

SBB uses different assumptions and works very differently. Most of the work in SBB involves solving NLP models. Since the NLP submodels differ only in one or a few bounds, the assumption is that the NLP models can be solved quickly using a good restart procedure. Since the NLP models differ very little and good initial values are available, the solution process will be fairly reliable compared to the solution process in DICOPT, where initial values of good quality seldom are available. Because search space is reduced based on very different grounds than in DICOPT, other solutions may therefore be explored.

Overall, DICOPT should perform better on models that have a significant and difficult combinatorial part, while SBB may perform better on models that have fewer discrete variables but more difficult nonlinearities (and possibly also on models that are fairly non convex).