GAMS/Cplex 6.5 User Notes

Table of Contents

1 Introduction

GAMS/Cplex is a GAMS solver that allows users to combine the high level modeling capabilities of GAMS with the power of Cplex optimizers. Cplex optimizers are designed to solve large, difficult problems quickly and with minimal user intervention. Access is provided (subject to proper licensing) to Cplex solution algorithms including the Linear Optimizer and the Barrier and Mixed Integer Solvers. While numerous solving options are available, GAMS/Cplex automatically calculates and sets most options at the best values for specific problems.

All Cplex options available through GAMS/Cplex are summarized at the end of this document.

2 How to run a model with Cplex

The following statement can be used inside your GAMS program to specify using Cplex

```
    Option LP = Cplex ;   { or RMIP or MIP }
```

The above statement should appear before the Solve statement. The MIP capability is separately licensed, so you may not be able to use Cplex for MIP problems on your system. If Cplex was specified as the default solver during GAMS installation, the above statement is not necessary.

3 Overview of Cplex

3.1 Linear Programming

Cplex solves LP problems using several alternative algorithms. The majority of LP problems solve best using Cplex's state of the art modified primal simplex algorithm. Certain types of problems benefit from using the alternative dual simplex algorithm, the network optimizer, or the barrier algorithm. The algorithm that Cplex should use is specified with option lpalg.

Solving linear programming problems is memory intensive. Even though Cplex manages memory very efficiently, insufficient physical memory is one of the most common problems when running large LPs. When memory is limited, Cplex will automatically make adjustments which may negatively impact performance. If you are working with large models, study the section entitled Physical Memory Limitations carefully.

Cplex is designed to solve the majority of LP problems using default option settings. These settings usually provide the best overall problem optimization speed and reliability. However, there are occasionally reasons for changing option settings to improve performance, avoid numerical difficulties, control optimization run duration, or control output options.

Some problems solve faster with the dual simplex algorithm rather than the default primal simplex algorithm. In particular, highly degenerate problems with little variability in the right-hand-side coefficients but significant variability in the cost coefficients often solve much faster using dual simplex. Also, very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying dual simplex if numerical problems occur while using primal simplex.

Cplex has a very efficient algorithm for network models. Network constraints have the following property:

- each non-zero coefficient is either a +1 or a -1
- each column appearing in these constraints has exactly 2 nonzero entries, one with a +1 coefficient and one with a -1 coefficient

Cplex can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

The barrier algorithm is an alternative to the simplex method for solving linear programs. It employs a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. Specifying the barrier algorithm may be advantageous for large, sparse problems.

GAMS/Cplex also provides access to the Cplex Infeasibility Finder. The Infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Cplex reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing.

3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with integer variables, Cplex uses a branch and bound algorithm (with cuts) which solves a series of LP subproblems. Because a single mixed integer problem generates many LP subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

`Option IterLim = n;`	Sets the iteration limit. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. Cplex handles the iteration limit for MIP problems differently than some other GAMS solvers. For MIP problems, controlling the length of the solution run by limiting the execution time (ResLim) is preferable. IterLim is not used at all when solving an LP with the barrier algorithm. By default, Cplex will use an iteration limit based on problem characteristics. To specify a barrier iteration limit, use Cplex parameter baritlim.
`OPTION ResLim = x;`	Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution.
`Option OptCA = x;`	Absolute optimality criterion for a MIP problem.
`Option OptCR = x ;`	Relative optimality criterion for a MIP problem. Notice that Cplex uses a slightly different definition than GAMS normally uses. The OptCR option signals Cplex to stop when

	`(|BP-BF|)/(1.0+|BP|) < OptCR`
	
	where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. The GAMS definition is:
	
	`(|BP-BF|)/(|BP|) < OptCR`

`Option Bratio = x;`	Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Cplex not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed. Note that using an advanced basis will disable the Cplex presolve. Especially for mixed integer problems, the presolve may be more beneficial than an advanced basis.
`Option SysOut = On;`	Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.
ModelName`.OptFile = 1;`	Instructs Cplex to read the option file. The name of the option file is *cplex.opt*.
ModelName`.Cheat = x;`	Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OptCA option). The Cplex option objdif overrides the GAMS cheat parameter.
ModelName`.Cutoff = x;`	Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.
ModelName`.PriorOpt = 1;`	Instructs Cplex to use priority branching information passed by GAMS through the *variable*.prior parameters.
ModelName`.TryInt = x;`	Causes GAMS/Cplex to make use of current variable values when solving a MIP problem. If a variable value is within x of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound. Moving the value to the bound allows the variable to be part of an initial integer solution when using parameter mipstart. The preferred branching direction will only be effective when priorities are used.

5 Summary of Cplex Options

The various Cplex options are listed here by category, with a few words about each to indicate its function. The options are listed again, in alphabetical order and with detailed descriptions, in the last section of this document.

5.1 Preprocessing and General Options

advind	advanced basis use
aggfill	aggregator fill parameter
aggind	aggregator on/off
coeredind	coefficient reduction on/off
depind	dependency checker on/off
lpalg	algorithm to use for solving an LP
predual	give dual problem to the optimizer
preind	turn presolver on/off
prepass	number of presolve applications to perform
printoptions	write values of all options to the GAMS listing file
relaxpreind	presolve for initial relaxation on/off
rerun	rerun problem if presolve infeasible or unbounded
scaind	matrix scaling on/off
tilim	overrides the GAMS ResLim option

5.2 Simplex Algorithmic Options

craind	crash strategy (used to obtain starting basis)
dpriind	dual simplex pricing
epper	perturbation constant
iis	run the IIS finder if the problem is infeasible
iisind	IIS finder method to use
netfind	attempt network extraction
netfinishalg	algorithm to finish solution with after solving the network
perind	force initial perturbation
perlim	number of stalled iterations before perturbation
ppriind	primal simplex pricing
pricelim	pricing candidate list
reinv	refactorization frequency
simthreads	number of threads for parallel dual simplex algorithm

5.3 Simplex Limit Options

itlim	iteration limit
objllim	objective function lower limit
objulim	objective function upper limit
singlim	limit on singularity repairs

5.4 Simplex Tolerance Options

epmrk	Markowitz pivot tolerance
epopt	optimality tolerance

eprhs	feasibility tolerance

5.5 Barrier Specific Options

baralg	algorithm selection
barcolnz	dense column handling
barepcomp	convergence tolerance
bargrowth	unbounded face detection
baritlim	iteration limit
barmaxcor	maximum correction limit
barobjrng	maximum objective function
barorder	row ordering algorithm selection
barthreads	number of threads for parallel barrier algorithm
barvarup	variable upper limit
crossoveralg	crossover method

5.6 MIP Algorithmic Options

bbinterval	best bound interval
bndstrenind	bound strengthening
brdir	set branching direction
bttol	backtracking limit
cliques	clique cut generation
covers	cover cut generation
cutlo	lower cutoff for tree search
cutsfactor	cut limit
cutup	upper cutoff for tree search
flowcovers	flow cover cut generation
gubcovers	GUB cover cut generation
heurfreq	heuristic frequency
heuristic	initial integer solution heuristic
implbd	implied bound cut generation
miphybalg	crossover type when solving subproblems with barrier
mipordind	priority list on/off
mipordtype	priority order generation
mipstart	use mip starting values
mipthreads	number of threads for parallel mip algorithm
nodefiledir	node storage directory
nodefileind	node storage file indicator
nodefilelim	node file size limit
nodelim	maximum number of nodes to process

nodesel	node selection strategy
probe	perform probing before solving a MIP
startalg	algorithm for initial LP
strongcandlim	size of the candidates list for strong branching
strongitlim	limit on iterations per branch for strong branching
strongthreadlim	number of threads for strong branching
subalg	algorithm for subproblems
varsel	variable selection strategy at each node

5.7 MIP Limit Options

intsollim	maximum number of integer solutions
nodelim	maximum number of nodes to solve
trelim	maximum space in memory for tree

5.8 MIP Tolerance Options

epagap	absolute stopping tolerance
epgap	relative stopping tolerance
epint	integrality tolerance
objdif	over-rides GAMS Cheat parameter
relobjdif	relative cheat parameter

5.9 Output Options

bardisplay	progress display level
mipdisplay	progress display level
mipinterval	progress display interval
netdisplay	network display level
simdisplay	simplex display level
writebas	produce a Cplex basis file
writemps	produce a Cplex MPS file
writelp	produce a Cplex LP file
writesav	produce a Cplex binary problem file
writesos	produce a Cplex sos file

5.10 The GAMS/Cplex Options File

The GAMS/Cplex options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs). Anything after the value will be ignored.

Following is an example options file *cplex.opt.*

```
lpalg dual
simdisplay 2
```

It will cause Cplex to solve an LP with the dual simplex algorithm instead of the default primal algorithm. The iteration log will have an entry for each iteration instead of an entry for each refactorization.

6 Special Notes

6.1 Physical Memory Limitations

For the sake of computational speed, Cplex should use only available physical memory rather than virtual or paged memory. When Cplex recognizes that a limited amount of memory is available it automatically makes algorithmic adjustments to compensate. These adjustments almost always reduce optimization speed. Learning to recognize when these automatic adjustments occur can help to determine when additional memory should be added to the computer.

On virtual memory systems, if memory paging to disk is observed, a considerable performance penalty is incurred. Increasing available memory will speed the solution process dramatically.

Cplex performs an operation called refactorization at a frequency determined by the reinv option setting. The longer Cplex works between refactorizations, the greater the amount of memory required to complete each iteration. Therefore, one means for conserving memory is to increase the refactorization frequency. Since refactorizing is an expensive operation, increasing the refactorization frequency by reducing the reinv option setting generally will slow performance. Cplex will automatically increase the refactorization frequency if it encounters low memory availability. This can be seen by watching the iteration log. The default log reports problem status at every refactorization. If the number of iterations between iteration log entries is decreasing, Cplex is increasing the refactorization frequency. Since Cplex might increase the frequency to once per iteration, the impact on performance can be dramatic. Providing additional memory should be beneficial.

6.2 Using Special Ordered Sets

For some models a special structure can be exploited. GAMS allows you to declare SOS1 and SOS2 variables (Special Ordered Sets of type 1 and 2).

In Cplex the definition for SOS1 variables is:

> A set of variables for which at most one variable may be non-zero.

The definition for SOS2 variables is:

> A set of variables for which at most two variables may be non-zero. If two variables are non-zero, they must be adjacent in the set.

6.3 Running Out of Memory for MIP problems

The most common difficulty when solving MIP problems is running out of memory. This problem arises when the branch and bound tree becomes so large that insufficient memory is available to solve an LP subproblem. As memory gets tight, you may observe frequent warning messages while Cplex attempts to navigate through various operations within limited memory. If a solution is not found shortly the solution process will be terminated with an unrecoverable integer failure message.

The tree information saved in memory can be substantial. Cplex saves a basis for every unexplored node. When utilizing the best bound method of node selection, the list of such nodes can become very long for large or difficult problems. How large the unexplored node list can become is entirely dependent on the actual amount of physical memory available and the actual size of the problem. Certainly increasing the amount of memory available extends the problem solving capability. Unfortunately, once a problem has failed because of insufficient memory, you can neither project how much further the process needed to go nor how much memory would be required to ultimately solve it.

Memory requirements can be limitted by using the trelim option with the nodefileind option. Setting nodefileind to 2 or 3 will cause Cplex to store portions of the branch and bound tree on disk whenever it grows to larger than the size specified by option trelim. That size should be set to something less than the amount of physical memory available.

Another approach is to modify the solution process to utilize less memory.

- Set option bttol to a number closer to 1.0 to cause more of a depth-first-search of the tree.
- Set option nodesel to use a best estimate strategy or, more drastically a depth-first-search. Depth first search rarely generates a large unexplored node list since Cplex will be diving deep into the branch and bound tree rather than jumping around within it.
- Set option varsel to use strong branching. Strong branching spends extra computation time at each node to choose a better branching variable. As a result it generates a smaller tree. It is often faster overall, as well.
- On some problems, a large number of cuts will be generated without a correspondingly large benefit in solution speed. Cut generation can be turned off using options cliques, covers, flowcovers, gubcovers, and implbd.

6.4 Failing to Prove Integer Optimality

One frustrating aspect of the branch and bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. Remember that the branch and bound tree may be as large as 2^n nodes, where n equals the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes! If no other stopping criteria have been set, the process might continue ad infinitum until the search is complete or your computer's memory is exhausted.

In general you should set at least one limit on the optimization process before beginning an optimization. Setting limits ensures that an exhaustive tree search will terminate in reasonable time. Once terminated, you can rerun the problem using some different option settings. Consider some of the shortcuts described previously for improving performance including setting the options for mip gap, objective value difference, upper cutoff, or lower cutoff.

7 GAMS/Cplex Log file

Cplex reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the primal simplex algorithm, the iteration log starts with the iteration number followed by the scaled infeasibility value. Once feasibility has been attained, the objective function value is listed instead. At the default value for option simdisplay there is a log line for each refactorization. The screen log has the following appearance:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.
Using conservative initial basis.

Iteration log . . .
Iteration:     1    Scaled infeas =        193998.067174
Iteration:    29    Objective     =         -3484.286415
Switched to devex.
Iteration:    98    Objective     =         -1852.931117
Iteration:   166    Objective     =          -349.706562

Optimal solution found.

Objective :         901.161538
```

The iteration log for the dual simplex algorithm is similar, but the dual infeasibility and dual objective are reported instead of the corresponding primal values:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.

Iteration log . . .
Iteration:     1   Scaled dual infeas =            3.890823
Iteration:    53   Dual objective     =         4844.392441
Iteration:   114   Dual objective     =         1794.360714
Iteration:   176   Dual objective     =         1120.183325
Iteration:   238   Dual objective     =          915.143030
Removing shift (1).

Optimal solution found.

Objective :         901.161538
```

The log for the network algorithm adds statistics about the extracted network and a log of the network iterations. The optimization is finished by one of the simplex algorithms and an iteration log for that is produced as well.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.
Extracted network with 25 nodes and 116 arcs.
Extraction time =   -0.00 sec.
Iteration log . . .
Iteration:     0   Infeasibility     =          1232.378800 (-1.32326e+12)

Network - Optimal:  Objective =    1.5716820779e+03
Network time =    0.01 sec.  Iterations = 26 (24)

Iteration log . . .
Iteration:     1    Scaled infeas =        212696.154729
Iteration:    62    Scaled infeas =         10020.401232
Iteration:   142    Scaled infeas =          4985.200129
Switched to devex.
Iteration:   217    Objective     =         -3883.782587
Iteration:   291    Objective     =         -1423.126582

Optimal solution found.

Objective :         901.161538
```

The log for the barrier algorithm adds various algorithm specific statistics about the problem before starting the iteration log. The iteration log includes columns for primal and dual objective values and infeasibility values. A special log follows for the crossover to a basic solution.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.02 sec.
Number of nonzeros in lower triangle of A*A' = 6545
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.01 sec.
Summary statistics for Cholesky factor:
  Rows in Factor            = 243
  Integer space required    = 578
  Total non-zeros in factor = 8491
  Total FP ops to factor    = 410889
 Itn      Primal Obj        Dual Obj  Prim Inf Upper Inf  Dual Inf
   0  -1.2826603e+06   7.4700787e+08  2.25e+10  6.13e+06  4.00e+05
   1  -2.6426195e+05   6.3552653e+08  4.58e+09  1.25e+06  1.35e+05
   2  -9.9117854e+04   4.1669756e+08  1.66e+09  4.52e+05  3.93e+04
   3  -2.6624468e+04   2.1507018e+08  3.80e+08  1.04e+05  1.20e+04
   4  -1.2104334e+04   7.8532364e+07  9.69e+07  2.65e+04  2.52e+03
```

```
    5  -9.5217661e+03   4.2663811e+07  2.81e+07  7.67e+03  9.92e+02
    6  -8.6929410e+03   1.4134077e+07  4.94e+06  1.35e+03  2.16e+02
    7  -8.3726267e+03   3.1619431e+06  3.13e-07  6.84e-12  3.72e+01
    8  -8.2962559e+03   3.3985844e+03  1.43e-08  5.60e-12  3.98e-02
    9  -3.8181279e+03   2.6166059e+03  1.58e-08  9.37e-12  2.50e-02
   10  -5.1366439e+03   2.8102021e+03  3.90e-06  7.34e-12  1.78e-02
   11  -1.9771576e+03   1.5960442e+03  3.43e-06  7.02e-12  3.81e-03
   12  -4.3346261e+02   8.3443795e+02  4.99e-07  1.22e-11  7.93e-04
   13   1.2882968e+02   5.2138155e+02  2.22e-07  1.45e-11  8.72e-04
   14   5.0418542e+02   5.3676806e+02  1.45e-07  1.26e-11  7.93e-04
   15   2.4951043e+02   6.5911879e+02  1.73e-07  1.43e-11  5.33e-04
   16   2.4666057e+02   7.6179064e+02  7.83e-06  2.17e-11  3.15e-04
   17   4.6820025e+02   8.1319322e+02  4.75e-06  1.78e-11  2.57e-04
   18   5.6081604e+02   7.9608915e+02  3.09e-06  1.98e-11  2.89e-04
   19   6.4517294e+02   7.7729659e+02  1.61e-06  1.27e-11  3.29e-04
   20   7.9603053e+02   7.8584631e+02  5.91e-07  1.91e-11  3.00e-04
   21   8.5871436e+02   8.0198336e+02  1.32e-07  1.46e-11  2.57e-04
   22   8.8146686e+02   8.1244367e+02  1.46e-07  1.84e-11  2.29e-04
   23   8.8327998e+02   8.3544569e+02  1.44e-07  1.96e-11  1.71e-04
   24   8.8595062e+02   8.4926550e+02  1.30e-07  2.85e-11  1.35e-04
   25   8.9780584e+02   8.6318712e+02  1.60e-07  1.08e-11  9.89e-05
   26   8.9940069e+02   8.9108502e+02  1.78e-07  1.07e-11  2.62e-05
   27   8.9979049e+02   8.9138752e+02  5.14e-07  1.88e-11  2.54e-05
   28   8.9979401e+02   8.9139850e+02  5.13e-07  2.18e-11  2.54e-05
   29   9.0067378e+02   8.9385969e+02  2.45e-07  1.46e-11  1.90e-05
   30   9.0112149e+02   8.9746581e+02  2.12e-07  1.71e-11  9.61e-06
   31   9.0113610e+02   8.9837069e+02  2.11e-07  1.31e-11  7.40e-06
   32   9.0113661e+02   8.9982723e+02  1.90e-07  2.12e-11  3.53e-06
   33   9.0115644e+02   9.0088083e+02  2.92e-07  1.27e-11  7.35e-07
   34   9.0116131e+02   9.0116262e+02  3.07e-07  1.81e-11  3.13e-09
   35   9.0116154e+02   9.0116154e+02  4.85e-07  1.69e-11  9.72e-13
Barrier time =    0.39 sec.

Primal crossover.
  Primal:  Fixing 13 variables.
       12 PMoves:  Infeasibility  1.97677059e-06  Objective  9.01161542e+02
        0 PMoves:  Infeasibility  0.00000000e+00  Objective  9.01161540e+02
  Primal:  Pushed 1, exchanged 12.
  Dual:  Fixing 3 variables.
        2 DMoves:  Infeasibility  1.28422758e-36  Objective  9.01161540e+02
        0 DMoves:  Infeasibility  1.28422758e-36  Objective  9.01161540e+02
  Dual:  Pushed 3, exchanged 0.
Using devex.
Total crossover time =    0.02 sec.

Optimal solution found.

Objective :         901.161540
```

For MIP problems, during the branch and bound search, Cplex reports the node number, the number of nodes left, the value of the Objective function, the number of integer variables that have fractional values, the current best integer solution, the best relaxed solution at a node and an iteration count. The last column show the current optimality gap as a percentage.

```
Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 99 rows, 76 columns, and 419 nonzeros.
Presolve time =    0.00 sec.
```

```
Iteration log . . .
Iteration:     1   Dual objective     =             0.000000
Root relaxation solution time =    0.01 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0        0.0000    24                      0.0000       40
*     0+    0        6.0000     0        6.0000        0.0000       40  100.00%
*    50+   50        4.0000     0        4.0000        0.0000      691  100.00%
    100    99        2.0000    15        4.0000        0.4000     1448   90.00%
Fixing integer variables, and solving final LP..
Tried aggregator 1 time.
LP Presolve eliminated 100 rows and 77 columns.
All rows and columns eliminated.
Presolve time =    0.00 sec.

Solution satisfies tolerances.

MIP Solution    :            4.000000    (2650 iterations, 185 nodes)
Final LP                :            4.000000    (0 iterations)

Best integer solution possible :           1.000000
Absolute gap                    :                   3
Relative gap                    :                 1.5
```

8 Detailed Descriptions of Cplex Options

These options should be entered in the options file after setting the GAMS ModelName.OptFile parameter to 1. The name of the option file is 'cplex.opt'. The option file is case sensitive and the keywords should be given in full.

advind *(integer)*

Use an Advanced Basis. GAMS/Cplex will automatically use an advanced basis from a previous solve statement. The GAMS Bratio option can be used to specify when not to use an advanced basis. The Cplex option *advind* can be used to ignore a basis passed on by GAMS (it overrides the Bratio option). Note that using an advanced basis will disable the Cplex presolve.
(default is determined by GAMS Bratio)

0	do not use advanced basis
1	use advanced basis if available

aggfill *(integer)*

Aggregator fill limit. If the net result of a single substitution is more non-zeros than the setting of the AGGFILL parameter, the substitution will not be made.
(default = 10)

aggind *(integer)*

This option, when set to a nonzero value, will cause the Cplex aggregator to use substitution where possible to reduce the number of rows and columns in the problem. If set to a positive value, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs and an unlimited number of times for mixed integer problems.
(default = -1)

-1	once for LP, unlimited for MIP
0	do not use

baralg *(integer)*

Selects which barrier algorithm to use. The default setting of 0 uses the infeasibility-estimate start algorithm for MIP subproblems and the standard barrier algorithm, option 3, for other cases. The standard barrier algorithm is almost always fastest. The alternative algorithms, options 1 and 2, may eliminate numerical difficulties related to infeasibility, but will generally be slower.
(default = 0)

0	Same as 1 for MIP subproblems, 3 otherwise
1	Infeasibility-estimate start
2	Infeasibility-constant start
2	standard barrier algorithm

barcolnz *(integer)*

Determines whether or not columns are considered dense for special barrier algorithm handling. At the default setting of 0, this parameter is determined dynamically. Values above 0 specify the number of entries in columns to be considered as dense.
(default = 0)

bardisplay *(integer)*

Determines the level of progress information to be displayed while the barrier method is running.
(default = 1)

0	No progress information
1	Display normal information
2	Display diagnostic information

barepcomp *(real)*

Determines the tolerance on complementarity for convergence of the barrier algorithm. The algorithm will terminate with an optimal solution if the relative complementarity is smaller than this value.
(default = 1.0 e-8)

bargrowth *(real)*

Used by the barrier algorithm to detect unbounded optimal faces. At higher values, the barrier algorithm will be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem does have an unbounded face.
(default = 1.0 e6)

baritlim *(integer)*

Determines the maximum number of iterations for the barrier algorithm. The default value of -1 allows Cplex to automatically determine the limit based on problem characteristics.
(default = -1)

barmaxcor *(integer)*

Specifies the maximum number of centering corrections that should be done on each iteration. Larger values may improve the numerical performance of the barrier algorithm at the expense of computation time. The default of -1 means the number is automatically determined.
Range - [-1, 10]
(default = -1)

barobjrng *(real)*

Determines the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.
(default = 1.0 e20)

barorder *(integer)*

Determines the ordering algorithm to be used by the barrier method. By default, Cplex attempts to choose the most effective of the available alternatives. Higher numbers tend to favor better orderings at the expense of longer ordering runtimes.
(default = 0)

0	Automatic
1	Approximate Minimum Degree (AMD)
2	Approximate Minimum Fill (AMF)
3	Nested Dissection (ND)

barstartalg *(integer)*

This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default starting point is satisfactory for most problems. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the predual parameter.
(default = 1)

1	default primal, dual is 0
2	default primal, estimate dual
3	primal average, dual is 0
4	primal average, estimate dual

barthreads *(integer)*

This option sets a limit on the number of threads available to the parallel barrier algorithm. The actual number of processors used will be the minimum of this number and the number of available processors. The parallel option is separately licensed.
(default = the number of threads licensed)

barvarup *(real)*

Determines an upper bound for all variables that have no finite upper bound. This number is used by the barrier algorithm to detect unbounded optimal faces.
(default = 1.0 e20)

bbinterval *(integer)*

Set interval for selecting a best bound node when doing a best estimate search. Active only when *nodesel* is 2 (best estimate). Decreasing this interval may be useful when best estimate is finding good solutions but making little progress in moving the bound. Increasing this interval may help when the best estimate node selection is not finding any good integer solutions. Setting the interval to 1 is equivalent to setting *nodesel* to 1.
(default = 7)

bndstrenind *(integer)*

Use bound strengthening when solving mixed integer problems. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during the branch and bound algorithm. This reduction is usually beneficial, but occasionaly, due to its iterative nature, takes a long time.
(default = -1)

-1	Determine automatically
0	Don't use bound strengthening
1	Use bound strengthening

brdir *(integer)*

Used to decide which branch (up or down) should be taken first at each node.
(default = 0)

-1	Down branch selected first
0	Algorithm decides
1	Up branch selected first

bttol ***(real)***

This option controls how often backtracking is done during the branching process. At each node, Cplex compares the objective function value or estimated integer objective value to these values at parent nodes; the value of the *bttol* parameter dictates how much relative degradation is tolerated before backtracking. Lower values tend to increase the amount of backtracking, making the search more of a pure best-bound search. Higher values tend to decrease the amount of backtracking, making the search more of a depth-first search. This parameter is used only once a first integer solution is found or when a cutoff has been specified.
Range: [0.0, 1.0]
(default = 0.01)

cliques ***(integer)***

Determines whether or not clique cuts should be generated during optimization.
(default = 0)

-1	Do not generate clique cuts
0	Determined automatically
1	Generate clique cuts at the root node only
2	Generate clique cuts throughout the tree

coeredind ***(integer)***

Coefficient reduction is a technique used when presolving mixed integer programs. The benefit is to improve the objective value of the initial (and subsequent) linear programming relaxations by reducing the number of non-integral vertices. However, the linear programs generated at each node may become more difficult to solve.
(default = 2)

0	Do not use coefficient reduction
1	Reduce only to integral coefficients
2	Reduce all potential coefficients

covers ***(integer)***

Determines whether or not cover cuts should be generated during optimization.
(default = 0)

-1	Do not generate cover cuts
0	Determined automatically
1	Generate cover cuts at the root node only
2	Generate cover cuts throughout the tree

craind ***(integer)***

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.
(default = 1)

Primal:

0	ignore objective coefficients during crash
-1, 1	alternate ways of using objective coefficients

Dual:

0, -1	aggressive starting basis
1	default starting basis

crossoveralg ***(string)***

Specifies the method to be used for creating a basis after solving with the barrier algorithm.
(default = primal)

primal	primal crossover
dual	dual crossover
none	no crossover; solution is non-basic

cutlo ***(real)***

Lower cutoff value for tree search in maximization problems. This is used to cut off any nodes that have an objective value below the lower cutoff value. This option overrides the GAMS Cutoff setting. A too restrictive value may result in no integer solutions being found.
(default = -1 e +75)

cutsfactor ***(real)***

This option limits the number of cuts that can be added. The number of rows in the problem with cuts added is limited to *cutsfactor* times the original (after presolve) number of rows.
(default = 4.0)

cutup ***(real)***

Upper Cutoff value for tree search in minimization problems. This is used to cut off any nodes that have an objective value above the upper cutoff value. This option overrides the GAMS Cutoff setting. A too restrictive value may result in no integer solutions being found.

(default = 1 e +75)

depind ***(integer)***

This option determines if the dependency checker will be used.
(default = 0)

0	Do not use the dependency checker
1	Use the dependency checker

dpriind ***(integer)***

Pricing strategy for dual simplex method. Consider using dual steepest-edge pricing. Dual steepest-edge is particularly efficient and does not carry as much computational burden as the primal steepest-edge pricing.
(default = 0)

0	Determined automatically
1	Standard dual pricing
2	Steepest-edge pricing
3	Steepest-edge pricing in slack space
4	Steepest-edge pricing, unit initial norms

epagap ***(real)***

Absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the epagap setting, the optimization is stopped. This option overrides GAMS OptCA which provides its initial value.
(default = GAMS OptCA)

epgap ***(real)***

Relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the epgap setting, the mixed integer optimization is stopped. Note the difference in the Cplex definition of the relative tolerance with the GAMS definition. This option overrides GAMS OptCR which provides its initial value.
Range: [0.0, 1.0]
(default = GAMS OptCR)

epint ***(real)***

Integrality Tolerance. This specifies the amount by which an integer variable can be different than an integer and still be considered feasible.
Range: [1.0e-9, 1.0]
(default = 1.0e-5)

epmrk ***(real)***

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.
Range - [0.0001, 0.99999]
(default = 0.01)

epopt *(real)*

The optimality tolerance influences the reduced-cost tolerance for optimality. This option setting governs how closely Cplex must approach the theoretically optimal solution.
Range - [1.0e-9, 1.0e-4]
(default = 1.0e-6)

epper *(integer)*

Perturbation setting. Highly degenerate problems tend to stall optimization progress. Cplex automatically perturbs the variable bounds when this occurs. Perturbation expands the bounds on every variable by a small amount thereby creating a different but closely related problem. Generally, the solution to the less constrained problem is easier to solve. Once the solution to the perturbed problem has advanced as far as it can go, Cplex removes the perturbation by resetting the bounds to their original values.

If the problem is perturbed more than once, the perturbation constant is probably too large. Reduce the *epper* option to a level where only one perturbation is required. Any non-negative values are valid.
(default = 1.0e-6)

eprhs *(real)*

Feasibility tolerance. This specifies the degree to which a problem's basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a lower value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, Cplex may falsely conclude that a problem is infeasible.
Range - [1.0e-9, 1.0e-4]
(default = 1.0e-6)

flowcovers *(integer)*

Determines whether or not flow cover cuts should be generated during optimization.
(default = 0)

-1	Do not generate flow cover cuts
0	Determined automatically
1	Generate flow cover cuts at the root node only
2	Generate flow cover cuts throughout the tree

gubcovers *(integer)*

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization.
(default = 0)

-1	Do not generate GUB cover cuts
0	Determined automatically
1	Generate GUB cover cuts at the root node only
2	Generate GUB cover cuts throughout the tree

heurfreq ***(integer)***

This option specifies how often to apply the node heuristic.
(default = 0)

-1	Do not use the node heuristic
0	Determined automatically

heuristic ***(integer)***

Affects the use of heuristics in finding an initial integer feasible solution.
(default = -1)

-1	Do not use heuristics
0	Determined automatically
1	Use a heuristic at node 0

iis ***(string)***

Find an IIS (Irreducably Inconsistent Set of constraints) and write an IIS report to the GAMS solution listing if the model is found to be infeasible. Legal option values are yes or no.
(default = yes)

iisind ***(integer)***

This option specifies which method to use for finding an IIS (Irreducably Inconsistent Set of constraints). Note that the iis option must be set to cause an IIS computation. This option just specifies the method.
(default = 0)

0	Method with minimum computation time
1	Method generating minimum size for the IIS

implbd ***(integer)***

Determines whether or not implied bound cuts should be generated during optimization.
(default = 0)

-1	Do not generate implied bound cuts
0	Determined automatically
1	Generate implied bound cuts at the root node only
2	Generate implied bound cuts throughout the tree

intsollim ***(integer)***

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.
(default = 2,100,000,000)

itlim ***(integer)***

The iteration limit option sets the maximum number of iterations before the algorithm terminates, without reaching optimality. This Cplex option overrides the GAMS IterLim option. Any non-negative integer value is valid.
(default = GAMS IterLim)

lpalg ***(string)***

The algorithm to use when solving an LP.
(default = primal)

primal	primal simplex
dual	dual simplex
network	network algorithm followed by primal or dual simplex as specified by option netfinishalg.
barrier	barrier algorithm with optional crossover to a basic solution as specified by option crossoveralg.

mipdisplay ***(integer)***

The amount of information displayed during MIP solution increases with increasing values of this option.
(default = 4)

0	No display
1	Display integer feasible solutions.
2	Displays nodes under mipinterval control.
3	Same as 2 but adds information on cuts.
4	Same as 3 but adds LP display for the root node.
5	Same as 3 but adds LP display for all nodes.

miphybalg ***(integer)***

Determines the type of crossover to use when the barrier algorithm is used for solving MIP subproblems.

(default = 1)

1	primal
2	dual

mipinterval *(integer)*

The MIP interval option value determines the frequency of the node logging when the mipdisplay option is set to 2 or 3. If the value of the MIP interval option setting is n, then only every nth node, plus all integer feasible nodes, will be logged. This option is useful for selectively limiting the volume of information generated for a problem that requires many nodes to solve. Any non-negative integer value is valid.
(default = 100)

mipordind *(integer)*

Use priorities. Priorities should be assigned based on your knowledge of the problem. Variables with higher priorities will be branched upon before variables of lower priorities. This direction of the tree search can often dramatically reduce the number of nodes searched. 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 purchased within the factory. By assigning a higher priority to the build/nobuild decision variable, you can force this logic into the tree search and eliminate wasted computation time exploring uninteresting portions of the tree. When set at 0 (default), the *mipordind* option instructs Cplex not to use priorities for branching. When set to 1, priority orders are utilized.

Note: Priorities are assigned to discrete variables using the *.prior* suffix in the GAMS model. Lower *.prior* values mean higher priority. The *.prioropt* model suffix has to be used to signal GAMS to export the priorities to the solver.
(default = 1)

0	Do not use priorities for branching
1	Priority orders are utilized

mipordtype *(integer)*

This option is used to select the type of generic priority order to generate when no priority order is present.
(default = 0)

0	None
1	decreasing cost magnitude
2	increasing bound range
3	increasing cost per coefficient count

mipstart ***(integer)***

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization.
(default = 0)

0	do not use the values
1	use the values

mipthreads ***(integer)***

This option sets a limit on the number of threads available to the parallel mip algorithm. The actual number of processors used will be the minimum of this number and the number of available processors. The parallel option is separately licensed.
(default = the number of threads licensed)

netdisplay ***(integer)***

This option controls the log for network iterations.
(default = 2)

0	No network log.
1	Displays true objective values.
2	Displays penalized objective values.

netfind ***(integer)***

Specifies the level of network extraction to be done. Note that option lpalg must be used to cause the network algorithm to execute. This option only specifies the level of network extraction.
(default = 2)

0	Cplex automatically determines the level of network extraction to be done.
1	Extract pure network only
2	Try reflection scaling
3	Try general scaling

netfinishalg ***(string)***

When using the network algorithm, either the primal or dual simplex method can be used to finish the solution.
(default = primal)

primal	Use primal simplex.
dual	Use dual simplex.

nodefiledir ***(string)***

Directory for use in storing node files. This parameter can be overridden by using an environment variable (TMPDIR under Unix and TMP under Windows 95 or NT).
(default = GAMS scratch directory)

nodefileind ***(integer)***

Node files are used when the tree memory limit (option trelim) is reached. At the default value of 0, optimization is terminated if trelim is reached. Otherwise a group of nodes is removed from memory and transferred to a node file. The group of nodes is returned to memory as needed. Using node files is a much better option than using swap space.
(default = 0)

0	No node files
1	Node files in memory and compressed
2	Node files on disk
3	Node files on disk and compressed

nodefilelim ***(real)***

The maximum amount of disk space the node files can consume before the optimization is terminated.
(default = 1.0e+75)

nodelim ***(integer)***

The maximum number of nodes solved before the algorithm terminates, without reaching optimality.

nodesel ***(integer)***

This option is used to set the rule for selecting the next node to process when backtracking.
(default = 1)

0	Depth-first search. This chooses the most recently created node.
1	Best-bound search. This chooses the unprocessed node with the best objective function for the associated LP relaxation.
2	Best-estimate search. This chooses the node with the best estimate of the integer objective value that would be obtained once all integer infeasibilities are removed.
3	Alternate best-estimate search.

objdif ***(real)***

A means for automatically updating the cutoff to more restrictive values. Normally the most recently found integer feasible solution objective value is used as the cutoff for subsequent nodes. When this option is set to a positive value, the value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. The option can be adjusted to

improve problem solving efficiency by limiting the number of nodes; however, setting this option at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. Negative values for this option will result in some integer solutions that are worse than or the same as those previously generated, but will not necessarily result in the generation of all possible integer solutions. This option overrides the GAMS Cheat parameter.
(default = 0.0)

objllim *(real)*

Setting a lower objective function limit will cause Cplex to halt the optimization process once the minimum objective function value limit has been exceeded.
(default = -1.0e+75)

objulim *(real)*

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.
(default = 1.0e+75)

perind *(integer)*

Perturbation Indicator. If a problem automatically perturbs early in the solution process, consider starting the solution process with a perturbation by setting perind to 1. Manually perturbing the problem will save the time of first allowing the optimization to stall before activating the perturbation mechanism, but is useful only rarely, for extremely degenerate problems.
(default = 0)

0	not automatically perturbed
1	automatically perturbed

perlim *(integer)*

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.
(default = 0)

ppriind *(integer)*

Pricing algorithm. Likely to show the biggest impact on performance. Look at overall solution time and the number of Phase I and total iterations as a guide in selecting alternate pricing algorithms. If you are using the dual Simplex method use *dpriind* to select a pricing algorithm. If the number of iterations required to solve your problem is approximately the same as the number of rows in your problem, then you are doing well. Iteration counts more than three times greater than the number of rows suggest that improvements might be possible.
(default = 0)

-1	Reduced-cost pricing. This is less compute intensive and may be preferred if the problem is small or easy. This option may also be advantageous for dense problems (say 20 to 30 nonzeros per column).
0	Hybrid reduced-cost and Devex pricing.
1	Devex pricing. This may be useful for more difficult problems which take many iterations to complete Phase I. Each iteration may consume more time, but the reduced number of total iterations may lead to an overall reduction in time. Tenfold iteration count reductions leading to threefold speed improvements have been observed. Do not use devex pricing if the problem has many columns and relatively few rows. The number of calculations required per iteration will usually be disadvantageous.
2	Steepest edge pricing. If devex pricing helps, this option may be beneficial. Steepest-edge pricing is computationally expensive, but may produce the best results on exceptionally difficult problems.
3	Steepest edge pricing with slack initial norms. This reduces the computationally intensive nature of steepest edge pricing.
4	Full pricing

predual ***(integer)***

Solve the dual. Some linear programs with many more rows than columns may be solved faster by explicitly solving the dual. The *predual* option will cause Cplex to solve the dual while returning the solution in the context of the original problem. This option is ignored if presolve is turned off.
(default = 0)

0	do not give dual to optimizer
1	give dual to optimizer

preind ***(integer)***

Perform Presolve. This helps most problems by simplifying, reducing and eliminating redundancies. However, if there are no redundancies or opportunities for simplification in the model, if may be faster to turn presolve off to avoid this step. On rare occasions, the presolved model, although smaller, may be more difficult than the original problem. In this case turning the presolve off leads to better performance. Specifying 0 turns the aggregator off as well.
(default = 1)

0	Do not presolve the problem
1	Perform the presolve

prepass ***(integer)***

Number of MIP presolve applications to perform. By default, Cplex determines this automatically. Specifying 0 turns off the presolve but not the aggregator. Set preind to 0 to turn both off.
(default = -1)

-1	Determined automatically
0	No presolve

pricelim ***(integer)***

Size for the pricing candidate list. Cplex dynamically determines a good value based on problem dimensions. Only very rarely will setting this option manually improve performance. Any non-negative intege values are valid.
(default = 0, in which case it is determined automatically)

printoptions ***(string)***

Write the values of all options to the GAMS listing file. Valid values are no or yes.
(default = no)

probe ***(integer)***

Determines the amount of probing performed on a MIP. Probing can be both very powerful and very time consuming. Setting the value to 1 can result in dramatic reductions or dramatic increases in solution time depending on the particular model.
(default = 0)

-1	no probing
0	automatic
1	full probing

reinv ***(integer)***

Refactorization Frequency. This option determines the number of iterations between refactorizations of the basis matrix. The default should be optimal for most problems. Cplex's performance is relatively insensitive to changes in refactorization frequency. Only for extremely large, difficult problems should reducing the number of iterations between refactorizations be considered. Any non-negative integer value is valid.
(default = 0, in which case it is determined automatically)

relaxpreind ***(integer)***

This option will cause the Cplex presolve to be invoked for the initial relaxation of a mixed integer program (according to the other presolve option settings). Sometimes, additional reductions can be made beyond any MIP presolve reductions that may already have been done.
(default = 0)

0	do not presolve initial relaxation
1	use presolve on initial relaxation

relobjdif ***(real)***

The relative version of the objdif option. Ignored if objdif is non-zero.
(default = 0)

rerun ***(string)***

The Cplex presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Cplex can, in order to get better diagnostic information, rerun the problem with the presolve and aggregator turned off. The GAMS solution listing will then mark variables and equations as infeasible or unbounded according to the final solution returned by the simplex algorithm. The iis option can be used to get even more diagnostic information. The *rerun* option controls this behavior. Valid values are yes or no.
(default = yes)

scaind ***(integer)***

This option influences the scaling of the problem matrix.
(default = 0)

-1	No scaling
0	Standard scaling - An equilibration scaling method is implemented which is generally very effective.
1	Modified, more aggressive scaling method that can produce improvements on some problems. This scaling should be used if the problem is observed to have difficulty staying feasible during the solution process.

simdisplay ***(integer)***

This option controls what Cplex reports (normally to the screen) during optimization. The amount of information displayed increases as the setting value increases.
(default = 1)

0	No iteration messages are issued until the optimal solution is reported.
1	An iteration log message will be issued after each refactorization. Each entry will contain the iteration count and scaled infeasibility or objective value.
2	An iteration log message will be issued after each iteration. The variables, slacks and artificials entering and leaving the basis will also be reported.

simthreads ***(integer)***

This option sets a limit on the number of threads available to the parallel simplex algorithm. The actual number of processors used will be the minimum of this number and the number of available processors. The parallel option is separately licensed.
(default = the number of threads licensed)

singlim ***(integer)***

The singularity limit setting restricts the number of times Cplex will attempt to repair the basis when singularities are encountered. Once the limit is exceeded, Cplex replaces the current basis with the best factorizable basis that has been found. Any non-negative integer value is valid.
(default = 10)

startalg ***(integer)***

Selects the algorithm to use for the initial relaxation of a MIP.
(default = 2)

1	primal simplex
2	dual simplex
3	network followed by dual simplex
4	barrier with crossover
5	dual simplex to iteration limit, then barrier
6	barrier without crossover

strongcandlim *(integer)*

Limit on the length of the candidate list for strong branching (varsel = 3).
(default = 10)

strongitlim *(integer)*

Limit on the number of iterations per branch in strong branching (varsel = 3). The default value of 0 causes the limit to be chosen automatically which is normally satisfactory. Try reducing this value if the time per node seems excessive. Try increasing this value if the time per node is reasonable but Cplex is making little progress.
(default = 0)

strongthreadlim *(integer)*

Controls the number of parallel threads available for strong branching (varsel = 3). This option does nothing if option mipthreads is greater than 1.
(default = 1)

subalg *(integer)*

Strategy for solving linear sub-problems at each node.
(default = 2)

1	primal simplex
2	dual simplex
3	network optimizer followed by dual simplex
4	barrier with crossover
5	dual simplex to iteration limit, then barrier
6	barrier without crossover

tilim *(real)*

The time limit setting determines the amount of time in seconds that Cplex will continue to solve a problem. This Cplex option overrides the GAMS ResLim option. Any non-negative value is valid.
(default = GAMS ResLim)

trelim ***(real)***

Maximum amount of space in megabytes that the branch and bound tree can consume in memory. Any positive value is valid. Consider allocating half of a machine's memory to the tree to a maximum of about 32 megabytes. Larger values provide only marginal improvement relative to the use of node files (option nodefilind).
(default = 1.0e+75)

varsel ***(integer)***

This option is used to set the rule for selecting the branching variable at the node which has been selected for branching. The default value of 0 allows Cplex to select the best rule based on the problem and its progress.
(default = 0)

-1	Branch on variable with minimum infeasibility. This rule may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.
0	Branch variable automatically selected.
1	Branch on variable with maximum infeasibility. This rule forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal integer solution.
2	Branch based on pseudo costs. Generally, the pseudo-cost setting is more effective when the problem contains complex trade-offs and the dual values have an economic interpretation.
3	Strong Branching. This setting causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is most promising. This is often effective on large, difficult problems.
4	Branch based on pseudo reduced costs.

writebas ***(character string)***

Write a basis file.

writelp ***(character string)***

Write a file in Cplex LP format.

writemps ***(character string)***

Write an MPS problem file.

writesav ***(character string)***

Write a binary problem file.

writesos ***(character string)***

Write a file containing the SOS structure. For models with SOS variables only.

Appendix: Cplex Licensing

The Cplex library, upon which GAMS/Cplex is based, uses a license manager to control use of the library. This is in addition to the normal GAMS license.

GAMS/Cplex licensing operations are performed using a set of commands described below. Using these commands (instead of running the Cplex licensing utility directly) will ensure that the license is set up in a standard way. This will avoid potential conflicts with other products and will make support easier if problems do arise.

The same set of licensing commands is used on PCs and Unix machines. The differences in how they operate are described in the next couple of sections. For both platforms, the command files reside in the cplex subdirectory of the GAMS system directory. When they are run, though, the current directory should be the GAMS system directory. The commands therefore look like cplex\update on a PC, or cplex/update on a Unix machine.

PC Licensing

If no evidence is seen of another Cplex license, a GAMS/Cplex demonstration license will be installed automatically at GAMS installation (gamsinst) time.

A GAMS/Cplex license is installed in a standard location. This location depends on the machine configuration and the operating system that is being used, but that information is determined automatically. An environment variable, CPLEXLICDIR, will be automatically defined whenever one of the licensing commands is being run as well as when GAMS/Cplex is actually solving a model. CPLEXLICDIR should not be set manually unless requested by GAMS support.

Unix Licensing

A GAMS/Cplex demonstration license is not automatically installed on Unix systems. That is because, under Unix, there is no good choice for a standard location for the license. The location must be specified by the person doing the installation.

The location for the license is specified by creating a license pointer file that contains the absolute path of a license directory. The pointer file must be created in a standard place and the directory that it points to should not exist yet (it will be created by the Cplex licensing utility). The location must be the cplex subdiretory of the GAMS system directory and its name should be cplexlicfile. The suggested name for the directory is cplexlicdir, but that is not enforced.

The location for cplexlicdir should be picked so that it will not have to be moved. Moving it, or even saving and restoring it from tape, will corrupt the Cplex license. It should not be kept under, or in a subdirectory of, the GAMS system directory. Otherwise, you may inadvertently delete the license when upgrading to a new version of GAMS.

An environment variable, CPLEXLICENSE, will be automatically defined whenever one of the licensing commands is being run as well as when GAMS/Cplex is actually solving a model. CPLEXLICENSE should not be set manually unless requested by GAMS support.

Licensing Commands

A Cplex license is installed and maintained by running a few commands. These commands automate defining the required environment variables, provide some GAMS specific checks, and call the Cplex license utility. All commands should be run from the command prompt. The current directory must be the GAMS system directory.

The command descriptions that follow call for using e-mail to support@gams.com. Please include the contents of both gamslice.txt and cpxlicen.log (PC) or cpxlicense.log (Unix) in the body of your e-mail. This is the preferred method to avoid transcription errors and to provide an electronic record. If e-mail cannot be used, a fax is the second choice. Transcribing license codes by phone can be done if necessary.

instdemo

Installs a demonstration license. GAMS/Cplex will work but will enforce size restrictions. This can be installed for testing purposes even if an unrestricted license will be installed later. In that case, however, the demonstration license will have to be be deleted with *delete* before getting an initial license code by running *initial.*

initial

Produces an initial license code that should be sent to support@gams.com. A copy of the file gamslice.txt from the GAMS system directory should be included with the e-mail to avoid confusion about which GAMS system is being licensed. This requires that a GAMS/Cplex license does not already exist on the machine.

update

Installs a permanent (or evaluation) license. The code is obtained from GAMS by first running *initial* and sending the initial license code to support@gams.com. GAMS/Cplex should be operational after running *update.*

delete

Deletes both the GAMS/Cplex license and the license directory. This is usually only required if a demonstration license is to be replaced with an evaluation or permanent license.

transfer

Deletes the GAMS/Cplex license and produces a license code to be taken to another machine. Requires an initial license code from the other machine as input.

restore

Restores a corrupted license. This should be run only when requested by GAMS support. It requires a restore code as input.

Installing a New License

The following steps must be followed to license a new GAMS/Cplex system.

1. If a demo license has been installed, it should be deleted using the *delete* command.
2. On Unix machines, the license directory pointer file, cplexlicfile, must be created. See **Unix Licensing** above.
3. An initial Cplex license code should be obtained by running the *initial* command.
4. Copies of gamslice.txt and cpxlicen.log (cpxlicense.log for Unix) should be e-mailed to support@gams.com with a request for a permanent or an evaluation license code.
5. A new license code will be returned by e-mail -- usually later the same day. It should be installed by running the *update* command.

Updating an Existing License

The method used to install an update depends on how the old license was installed.

PC Using cplex\update

If the existing license was installed using cplex\update, it can be updated the same way. Contact support@gams.com for an update code. Don't forget to include the contents of gamslice.txt and cpxlicen.log (cpxlicense.log for Unix).

The cplex\update command can be used to find out if the existing license was installed the same way. If it was, output similar to the following will be seen:

```
A CPLEX license already exists on this machine.
The current license code is:
     30-C5-4E-F4-1C-EB-D5-4A-99-39-92-09-DC-BE

Environment variable settings:
  CPLEXLICDIR='C:\WINDOWS\gmscplic'

Type 'quit' or enter a license in the form:
   XX-XX-XX-XX-XX-XX-XX-XX-XX-XX-XX-XX-XX-XX
```

Otherwise a message will be seen that says "No CPLEX license was found on this machine..."

PC Using CPLEXLICTYPE=DEFAULT

Some existing GAMS/Cplex systems were licensed by putting

```
        SET CPLEXLICTYPE=DEFAULT
```

in the autoexec.bat file and running cpxlicen directly. For those systems, set up the new license using the *initial* and *update* commands. After the new license is working, delete the old license by 1) making sure the CPLEXLICDIR environment variable is not set and 2) running cpxlicen -d. You will need to send the delete confirmation code to support@gams.com.

To test if the existing license was installed with CPLEXLICTYPE=DEFAULT, simply type SET at the command prompt to see if CPLEXLICTYPE is defined.

PC Using a Hardware Key

Cplex no longer supports licensing with a hardware key. To switch to keyless licensing, set up the new license using the *initial* and *update* commands. You will need to return the key to GAMS Development after testing the new license.

Unix

Simply make cplex/cplexlicfile point to the existing license and ask support@gams.com for a code to use with the *update* command.

Last modified April 28, 1999 by PES