CPLEX 12

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 for linear, quadratically constrained and mixed integer programming problems. 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 QCP, MIP, MIQCP, RMIP or RMIQCP }
```

The above statement should appear before the Solve statement. The MIP and QCP capabilities are separately licensed, so you may not be able to use Cplex for those problem types 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 dual simplex algorithm. Certain types of problems benefit from using the primal simplex algorithm, the network optimizer, the barrier algorithm, or the sifting algorithm. The concurrent option will allow solving with different algorithms in parallel. The solution is returned by the first to finish.

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 primal simplex algorithm rather than the default dual simplex algorithm. Very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying primal simplex if numerical problems occur while using dual 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.

Cplex provides a sifting algorithm which can be effective on problems with many more varaibles than equations. Sifting solves a sequence of LP subproblems where the results from one subproblem are used to select columns from the original model for inclusion in the next subproblem.

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. IIS is available for LP problems only.

3.2   Quadratically Constrained Programming

Cplex can solve models with quadratic contraints. These are formulated in GAMS as models of type QCP. QCP models are solved with the Cplex Barrier method.

QP models are a special case that can be reformuated to have a quadratic objective function and only linear constraints. Those are automatically reformulated from GAMS QCP models and can be solved with any of the Cplex QP methods (Barrier, Primal Simplex or Dual Simplex).

For QCP models, Cplex returns a primal only solution to GAMS. Dual values are returned for QP models.

3.3   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 cut algorithm which solves a series of LP, subproblems. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS and GAMS/Cplex support Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

Cplex can also solve problems of GAMS model type MIQCP. As in the continuous case, if the base model is a QP the Simplex methods can be used and duals will be available at the solution. If the base model is a QCP, only the Barrier method can be used for the nodes and only primal values will be available at the solution.

3.4   Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Cplex is known as FeasOpt (for Feasible Optimization) and turned on by parameter feasopt in a CPLEX option file. More details can be found in the section entitled Using the Feasibility Relaxation.

3.5   Solution Pool: Generating and Keeping Multiple Solutions

This chapter introduces the *solution pool* for storing multiple solutions to a mixed integer programming problem (MIP and MIQCP). The chapter also explains techniques for generating and managing those solutions.

The solution pool stores multiple solutions to a mixed integer programming (MIP and MIQCP) model. With this feature, you can direct the algorithm to generate multiple solutions in addition to the optimal solution. For example, some constraints may be difficult to formulate efficiently as linear expressions, or the objective may be difficult to quantify exactly. In such cases, obtaining multiple solutions will help you choose one which best fits all your criteria, including the criteria that could not be expressed easily in a conventional MIP or MIQCP model. For example,

- You can collect solutions within a given percentage of the optimal solution. To do so, apply the solution pool gap parameters `solnpoolagap` and `solnpoolgap`.

- You can collect a set of diverse solutions. To do so, use the solution pool replacement parameter `SolnPool-Replace` to set the solution pool replacement strategy to 2. In order to control the diversity of solutions even more finely, apply a *diversity filter*.

- In an advanced application of this feature, you can collect solutions with specific properties. To do so, see the use of the *incumbent filter*.

- You can collect all solutions or all optimal solutions to model. To do so, set the solution pool intensity parameter `SolnPoolIntensity` to its highest value.

Filling the Solution Pool

There are two ways to fill the solution pool associated with a model: You can *accumulate* successive incumbents or generate alternative solutions by *populating* the solution pool. The method is selected with the parameter `SolnPoolPop`:

- The regular optimization procedure automatically adds incumbents to the solution pool as they are discovered (`SolnPoolPop=1`).

- Cplex also provides a procedure specifically to generate multiple solutions. You can invoke this procedure by setting option `SolnPoolPop=2`. You can also invoke this procedure many times in a row in order to explore the solution space differently. In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory. This is done by specifying a GAMS program (option `SolnPoolPopRepeat`) that inspects the solutions. In case this GAMS program terminates normally, i.e. no execution or compilation error, the exploration for alternative solutions proceeds.

The option `SolnPoolReplace` designates the strategy for replacing a solution in the solution pool when the solution pool has reached its capacity. The value 0 replaces solutions according to a first-in, first-out policy. The value 1 keeps the solutions with the best objective values. The value 2 replaces solutions in order to build a set of diverse solutions.

If the solutions you obtain are too similar to each other, try setting `SolnPoolReplace` to 2.

The replacement strategy applies only to the subset of solutions created in the current call of populate. Solutions already in the pool are not affected by the replacement strategy. They will not be replaced, even if they satisfy the criterion of the replacement strategy. So with every repeated call of the populate procedure the solution pool will be extended by the newly found solution. After the GAMS program specified in `SolnPoolPopRepeat` determined to continue the search for alternative solutions, the file specified by option `SolnPoolPopDel` option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

Details can be found in the model `solnpool` in the GAMS model library.

Enumerating All Solutions

With the solution pool, you can collect all solutions to a model. To do so, set the solution pool intensity parameter `SolnPoolIntensity` to its highest value, 4 and set `SolnPoolPop=2`.

You can also enumerate all solutions that are valid for a specific criterion. For example, if you want to enumerate all alternative optimal solutions, do the following:

- Set the pool absolute gap parameter `SolnPoolAGap=0.0`.

- Set the pool intensity parameter `SolnPoolIntensity=4`.

- Set the populate limit parameter `PopulateLim` to a value sufficiently large for your model; for example, 2100000000.

- Set the pool population parameter `SolnPoolPop=2`.

Beware, however, that, even for small models, the number of possible solutions is likely to be huge. Consequently, enumerating all of them will take time and consume a large quantity of memory.

There may be an infinite number of possible values for a continuous variable, and it is not practical to enumerate all of them on a finite-precision computer. Therefore, populate gives only one solution for each set of binary

and integer variables, even though there may exist several solutions that have the same values for all binary and integer variables but different values for continuous variables.

Likewise, for the same reason, the populate procedure does not generate all possible solutions for unbounded models. As soon as the proof of unboundedness is obtained, the populate procedure stops.

Cplex uses numerical methods of finite-precision arithmetic. Consequently, the feasibility of a solution depends on the value given to tolerances. Two parameters define the tolerances that assess the feasibility of a solution:

- the integrality tolerance `EpInt`

- the feasibility tolerance `EpRHS`

A solution may be considered feasible for one pair of values for these two parameters, and infeasible for a different pair. This phenomenon is especially noticeable in models with numeric difficulties, for example, in models with Big M coefficients.

Since the definition of a feasible solution is subject to tolerances, the total number of solutions to a model may vary, depending on the approach used to enumerate solutions, and on precisely which tolerances are used. In most models, this tolerance issue is not problematic. But, in the presence of numeric difficulties, Cplex may create solutions that are slightly infeasible or integer infeasible, and therefore create more solutions than expected.

Filtering the Solution Pool

Filtering allows you to control properties of the solutions generated and stored in the solution pool. Cplex provides two predefined ways to filter solutions.

If you want to filter solutions based on their difference as compared to a reference solution, use a *diversity filter*. This filter is practical for most purposes. However, if you require finer control of which solutions to keep and which to eliminate, use the *incumbent filter*.

Diversity Filter

A diversity filter allows you to generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables using dot option `divflt` and lower and upper bounds `divfltlo` and `divfltup`. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution. If you need more than one diversity filter, for example, to generate solutions that share the characteristics of several different solutions, additional filters can be specified through a Cplex Filter File using parameter `ReadFLT`. Details can be found in the example model `solnpool` in the GAMS model library.

Incumbent Filter

If you need to enforce more complex constraints on solutions (e.g. if you need to enforce nonlinear constraints), you can use the incumbent filtering. The incumbent checking routine is part of the GAMS BCH Facility. It will accept or reject incumbents independent of a solution pool. During the populate or regular optimize procedure, the incumbent checking routine specified by the parameter `userincbcall` is called each time a new solution is found, even if the new solution does not improve the objective value of the incumbent. The incumbent filter allows your application to accept or reject the new solution based on your own criteria. If the GAMS program specified by `userincbcall` terminates normally, the solution is rejected. If this program returns with a compilation or execution error, the incumbent is accepted.

Accessing the Solution Pool

The GAMS/Cplex link produces, if properly instructed, a GDX file with name specified in `SolnPool` that contains a set `Index` with elements `file1`, `file2`, ... The associated text of these elements contain the file names of the

individual GDX solution file. The name is constructed using the prefix `soln` (which can be specified differently by option `SolnPoolPrefix`), the name of the model and a sequence number. For example `soln_loc_p1.gdx`. GAMS/Cplex will overwrite existing GDX files without warning. The set `Index` allows us to conveniently walk through the different solutions in the solution pool:

```
...
solve mymodel min z using mip;

set soln            possible solutions in the solution pool /file1*file1000/
    solnpool(soln) actual solutions;
file fsol;

execute_load 'solnpool.gdx', solnpool=Index;
loop(solnpool(soln),
  put_utility fsol 'gdxin' / solnpool.te(soln):0:0;
  execute_loadpoint;
  display z.l;
);
```

4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

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.

Option IterLim = n;

> Sets the simplex iteration limit. Simplex algorithms 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. The iteration limit is applied per node instead of as a total over all nodes. For MIP problems, controlling the length of the solution run by limiting the execution time (ResLim) is preferable.

> Simlarly, when using the sifting algorithm, the iteration limit is applied per sifting iteration (ie per LP). The number of sifting iterations (LPs) can be limited by setting Cplex parameter siftitlim. It is the number of sifting iterations that is reported back to GAMS as iterations used.

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 SysOut = On;

> Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.

***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.NodLim = x;

> Maximum number of nodes to process for a MIP problem.

ModelName.OptCA = x;

> Absolute optimality criterion for a MIP problem.

ModelName.OptCR = x;

> Relative optimality criterion for a MIP problem. Notice that Cplex uses a different definition than GAMS normally uses. The OptCR option asks Cplex to stop when
>
> $$(|BP - BF|)/(1.0e - 10 + |BF|) < \text{OptCR}$$
>
> 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:
>
> $$(|BP - BF|)/(|BP|) < \text{OptCR}$$

ModelName.OptFile = 1;

> Instructs Cplex to read the option file. The name of the option file is *cplex.opt*.

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. The preferred branching direction will only be effective when priorities are used. Priorities and tryint are sometimes not very effective and often outperformed by GAMS/CPLEX default settings. Supporting GAMS/CPLEX with knowledge about a known solution can be passed on by different means, please read more about this in section entitled Starting from a MIP Solution.

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
clocktype	clock type for computation time
coeredind	coefficient reduction on/off
depind	dependency checker on/off
feasopt	computes a minimum-cost relaxation to make an infeasible model feasible
feasoptmode	Mode of FeasOpt
.feaspref	feasibility preference
interactive	allow interactive option setting after a Control-C
lpmethod	algorithm to be used for LP problems
memoryemphasis	Reduces use of memory
names	load GAMS names into Cplex
numericalemphasis	emphasizes precision in numerically unstable or difficult problems

objrng	do objective ranging
parallelmode	parallel optimization mode
predual	give dual problem to the optimizer
preind	turn presolver on/off
prelinear	linear reduction indicator
prepass	number of presolve applications to perform
printoptions	list values of all options to GAMS listing file
qpmethod	algorithm to be used for QP problems
reduce	primal and dual reduction type
relaxpreind	presolve for initial relaxation on/off
rerun	rerun problem if presolve infeasible or unbounded
rhsrng	do right-hand-side ranging
rngrestart	write GAMS readable ranging information file
scaind	matrix scaling on/off
solutiontarget	type of solution when solving a nonconvex continuous quadratic model
threads	global default thread count
tilim	overrides the GAMS ResLim option
tuning	invokes parameter tuning tool
tuningdisplay	level of information reported by the tuning tool
tuningmeasure	measure for evaluating progress for a suite of models
tuningrepeat	number of times tuning is to be repeated on perturbed versions
tuningtilim	tuning time limit per model or suite
workdir	directory for working files
workmem	memory available for working storage

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
netfind	attempt network extraction
netppriind	network simplex pricing
perind	force initial perturbation
perlim	number of stalled iterations before perturbation
ppriind	primal simplex pricing
pricelim	pricing candidate list
reinv	refactorization frequency

5.3   Simplex Limit Options

itlim	iteration limit
netitlim	iteration limit for network simplex
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
netepopt	optimality tolerance for the network simplex method
neteprhs	feasibility tolerance for the network simplex method

5.5   Barrier Specific Options

baralg	algorithm selection
barcolnz	dense column handling
barcrossalg	barrier crossover method
barepcomp	convergence tolerance
bargrowth	unbounded face detection
baritlim	iteration limit
barmaxcor	maximum correction limit
barobjrng	maximum objective function
barorder	row ordering algorithm selection
barqcpepcomp	convergence tolerance for the barrier optimizer for QCPs
barstartalg	barrier starting point algorithm

5.6   Sifting Specific Options

siftalg	sifting subproblem algorithm
siftitlim	limit on sifting iterations

5.7   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
cuts	default cut generation
cutsfactor	cut limit
cutup	upper cutoff for tree search
disjcuts	disjunctive cuts generation
divetype	MIP dive strategy
eachcutlim	Sets a limit for each type of cut
flowcovers	flow cover cut generation
flowpaths	flow path cut generation
fpheur	feasibility pump heuristic
fraccuts	Gomory fractional cut generation
gubcovers	GUB cover cut generation

heurfreq	heuristic frequency
implbd	implied bound cut generation
lbheur	local branching heuristic
mcfcuts	multi-commodity flow cut generation
mipemphasis	MIP solution tactics
mipkappastats	MIP kappa computation
mipordind	priority list on/off
mipordtype	priority order generation
mipsearch	search strategy for mixed integer programs
mipstart	use mip starting values
miqcpstrat	MIQCP relaxation choice
mircuts	mixed integer rounding cut generation
nodefileind	node storage file indicator
nodesel	node selection strategy
preslvnd	node presolve selector
probe	perform probing before solving a MIP
qpmakepsdind	adjust MIQP formulation to make the quadratic matrix positive-semi-definite
relaxfixedinfeas	access small infeasibilties in the solve of the fixed problem
repeatpresolve	reapply presolve at root after preprocessing
rinsheur	relaxation induced neighborhood search frequency
solvefinal	switch to solve the problem with fixed discrete variables
startalg	MIP starting algorithm
strongcandlim	size of the candidates list for strong branching
strongitlim	limit on iterations per branch for strong branching
subalg	algorithm for subproblems
submipnodelim	limit on number of nodes in an RINS subMIP
symmetry	symmetry breaking cuts
varsel	variable selection strategy at each node
zerohalfcuts	zero-half cuts

5.8 MIP Limit Options

aggcutlim	aggrigation limit for cut generation
auxrootthreads	number of threads for auxiliary tasks at the root node
cutpass	maximum number of cutting plane passes
fraccand	candidate limit for generating Gomory fractional cuts
fracpass	maximum number of passes for generating Gomory fractional cuts
intsollim	maximum number of integer solutions
nodelim	maximum number of nodes to solve
polishafterepagap	Absolute MIP gap before starting to polish a feasible solution
polishafterepgap	Relative MIP gap before starting to polish a solution
polishafternode	Nodes to process before starting to polish a feasible solution
polishafterintsol	MIP integer solutions to find before starting to polish a feasible solution
polishaftertime	Time before starting to polish a feasible solution
probetime	time spent probing
repairtries	try to repair infeasible MIP start
trelim	maximum space in memory for tree

5.9 MIP Solution Pool Options

divfltup	upper bound on diversity

divfltlo   lower bound on diversity
.divflt   solution pool range filter coefficients
populatelim   limit of solutions generated for the solution pool by populate method
readflt   reads Cplex solution pool filter file
solnpool   solution pool file name
solnpoolagap   absolute tolerance for the solutions in the solution pool
solnpoolcapacity limits of solutions kept in the solution pool
solnpoolgap   relative tolerance for the solutions in the solution pool
solnpoolintensity solution pool intensity for ability to produce multiple solutions
solnpoolpop   methods to populate the solution pool
solnpoolpopdel   file with solution numbers to delete from the solution pool
solnpoolpoprepeat   method to decide if populating the solution should be repeated
solnpoolprefix   file name prefix for GDX solution files
solnpoolreplace   strategy for replacing a solution in the solution pool
userincbcall   The GAMS command line to call the incumbent checking program

5.10   MIP Tolerance Options

epagap   absolute stopping tolerance
epgap   relative stopping tolerance
epint   integrality tolerance
objdif   overrides GAMS Cheat parameter
relobjdif   relative cheat parameter

5.11   Output Options

bardisplay   progress display level
clonelog   enable clone logs
mipdisplay   progress display level
mipinterval   progress display interval
mpslongnum   MPS file format precision of numeric output
netdisplay   network display level
quality   write solution quality statistics
siftdisplay   sifting display level
simdisplay   simplex display level
writebas   produce a Cplex basis file
writeflt   produce a Cplex solution pool filter file
writelp   produce a Cplex LP file
writemps   produce a Cplex MPS file
writemst   produce a Cplex mst file
writeord   produce a Cplex ord file
writeparam   produce a Cplex parameter file with all active options
writepre   produce a Cplex LP/MPS/SAV file of the presolved problem
writesav   produce a Cplex binary problem file

5.12   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).

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

```
scaind 1
simdisplay 2
```

It will cause Cplex to use a more aggressive scaling method than the default. 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. Also consider option memoryemphasis to conserve memory where possible.

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 Using Semi-Continuous and Semi-Integer Variables

GAMS allows the declaration of semi-continous and semi-integer variables. These variable types are directly supported by GAMS/Cplex. For example:

```
SemiCont Variable x;
```

```
    x.lo = 3.2;
    x.up = 8.7;

    SemiInt Variable y;
    y.lo = 5;
    y.up = 10;
```

Variable x will be allowed to take on a value of 0.0 or any value between 3.2 and 8.7. Variable y will be allowed to take on a value of 0 or any integral value between 5 and 10.

Note that Cplex requires a finite upper bound for semi-continous and semi-integer variables.

6.4   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 limited by using the workmem, 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 workmem. 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 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 option cuts.

6.5   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.

6.6   Starting from a MIP Solution

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution. When you provide such a starting solution, you may invoke relaxation induced neighborhood search (RINS heuristic) or solution polishing to improve the given solution. This first integer solution may include continuous and discrete variables of various types, such as semi-continuous variables or special ordered sets.

If you specify values for all discrete variables, GAMS/CPLEX will check the validity of the values as an integer-feasible solution; if you specify values for only a portion of the discrete variables, GAMS/CPLEX will attempt to fill in the missing values in a way that leads to an integer-feasible solution. If the specified values do not lead directly to an integer-feasible solution, GAMS/CPLEX will apply a quick heuristic to try to repair the MIP Start. The number of times that GAMS/CPLEX applies the heuristic is controlled by the repair tries parameter (RepairTries). If this process succeeds, the solution will be treated as an integer solution of the current problem.

A MIP start will only be used by GAMS/CPLEX if the MipStart parameter is set to 1.

6.7   Using the Feasibility Relaxation

The feasibility relaxation is enabled by the FeasOpt parameter in a CPLEX solver option file.

With the FeasOpt option CPLEX accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candiates for relaxation and weigthed equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the `.feaspref` option. A negative or zero preference means that the associated bound or constraint is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a CPLEX solver option file. The syntax is:

   (variable or equation).`feaspref` *(value)*

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the *cplex.opt* file can be specified by:

```
feasopt 1
v.feaspref            1
v.feaspref('i1',*)    2
v.feaspref('i1','j2') 0

e.feaspref(*,'j1')    0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxtion on. Futhermore, we specify that all variables `v(i,j)` have preference of 1, except variables over set element `i1`, which have a preference of 2. The variable over set element `i1` and `j2` has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated

above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations). Please check description of parameter FeasOpt FeasOptMode for details. Also check example models `feasopt*` in the GAMS Model library.

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. CPLEX logs an asterisk (*) in the left-most column for any node where it finds an integer-feasible solution or new incumbent. The + denotes an incumbent generated by the heuristic.

```
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 options file is *'cplex.opt'*. The options file is case insensitive 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 *Bratio*).
>
> *(default = determined by GAMS Bratio)*
>
> > 0 Do not use advanced basis
> >
> > 1 Use advanced basis if available
> >
> > 2 Crash an advanced basis if available (use basis with presolve)

aggcutlim (*integer*)

> Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes, the default will be satisfactory.
>
> *(default = 3)*

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

auxrootthreads (*integer*)

Partitions the number of threads for CPLEX to use for auxiliary tasks while it solves the root node of a problem. On a system that offers N processors or N global threads, if you set this parameter to n, where $N > n > 0$ then CPLEX uses at most n threads for auxiliary tasks and at most $N - n$ threads to solve the root node. See also the parameter threads.

You cannot set n, the value of this parameter, to a value greater than or equal to N, the number of processors or global threads offered on your system. In other words, when you set this parameter to a value other than its default, that value must be strictly less than the number of processors or global threads on your system. Independent of the auxiliary root threads parameter, CPLEX will never use more threads than those defined by the global default thread count parameter. CPLEX also makes sure that there is at least one thread available for the main root tasks. For example, if you set the global threads parameter to 3 and the auxiliary root threads parameter to 4, CPLEX still uses only two threads for auxiliary root tasks in order to keep one thread available for the main root tasks. At its default value, 0 (zero), CPLEX automatically chooses the number of threads to use for the primary root tasks and for auxiliary tasks. The number of threads that CPLEX uses to solve the root node depends on several factors: 1) the number of processors available on your system; 2) the number of threads available to your application on your system (for example, as a result of limited resources or competition with other applications); 3) the value of the global default thread count parameter threads.

(*default = -1*)

   -1 Off: do not use additional threads for auxiliary tasks

    0 Automatic: let CPLEX choose the number of threads to use

$N > n > 0$ Use n threads for auxiliary root tasks

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

    3 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*)

barcrossalg (*integer*)

Selects which, if any, crossover method is used at the end of a barrier optimization.

(*default = 0*)

-1 No crossover

 0 Automatic

 1 Primal crossover

 2 Dual crossover

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 = 1e-008*)

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 = 1e+012*)

baritlim (*integer*)

Determines the maximum number of iterations for the barrier algorithm. When set to 0, no Barrier iterations occur, but problem *setup* occurs and information about the setup is displayed (such as Cholesky factorization information). When left at the default value, there is no explicit limit on the number of iterations.

(*default = large*)

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.

(*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 = 1e+020*)

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)

barqcpepcomp (*real*)

Range: [1e-012,1e+075]

(default = 1e-007)

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

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 occasionally, 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: [0,1]

(default = 0.9999)

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 moderately

2 Generate clique cuts aggressively

3 Generate clique cuts very aggressively

clocktype (*integer*)

Decides how computation times are measured for both reporting performance and terminating optimization when a time limit has been set. Small variations in measured time on identical runs may be expected on any computer system with any setting of this parameter. The default setting 0 (zero) allows CPLEX to choose wall clock time when other parameters invoke parallel optimization and to choose CPU time when other parameters enforce sequential (not parallel) optimization.

(default = 0)

0 Automatic

1 CPU time

2 Wall clock time

clonelog (*integer*)

The clone logs contain information normally recorded in the ordinary log file but inconvenient to send through the normal log channel in case of parallel execution. The information likely to be of most interest to you are special messages, such as error messages, that result from calls to the LP optimizers called for the subproblems. The clone log files are named cloneK.log, where K is the index of the clone, ranging from 0 (zero) to the number of threads minus one. Since the clones are created at each call to a parallel optimizer and discarded when it exits, the clone logs are opened at each call and closed at each exit. The clone log files are not removed when the clones themselves are discarded.

(default = 0)

-1 Clone log files off

0 Automatic

1 Clone log files on

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 = -1)

-1 Automatic

0 Do not use coefficient reduction

1 Reduce only to integral coefficients

2 Reduce all potential coefficients

3 Reduce aggressively with tilting

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 moderately

2 Generate cover cuts aggressively

3 Generate cover cuts very aggressively

craind (*integer*)

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.

(default = 1)

-1 Primal: alternate ways of using objective coefficients. Dual: aggressive starting basis

0 Primal: ignore objective coefficients during crash. Dual: aggressive starting basis

1 Primal: alternate ways of using objective coefficients. Dual: default starting basis

cutlo (*real*)

Sets the lower cutoff tolerance. When the problem is a maximization problem, CPLEX cuts off or discards solutions that are less than the specified cutoff value. If the model has no solution with an objective value greater than or equal to the cutoff value, then CPLEX declares the model infeasible. In other words, setting the lower cutoff value c for a maximization problem is similar to adding this constraint to the objective function of the model: $obj \geq c$.

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

(default = -1e+075)

cutpass (*integer*)

Sets the upper limit on the number of passes that will be performed when generating cutting planes on a mixed integer model.

(default = 0)

-1 None

0 Automatically determined

>0 Maximum passes to perform

cuts (*string*)

Allows generation setting of all optional cuts at once. This is done by changing the meaning of the default value (0: automatic) for the various Cplex cut generation options. The options affected are cliques, covers, disjcuts, flowcovers, flowpaths, fraccuts, gubcovers, implbd, mcfcuts, mircuts, and symmetry.

(default = 0)

-1 Do not generate cuts

 0 Determined automatically

 1 Generate cuts moderately

 2 Generate cuts aggressively

 3 Generate cuts very aggressively

 4 Generate cuts highly aggressively

 5 Generate cuts extremely aggressively

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*)

cutup (*real*)

Sets the upper cutoff tolerance. When the problem is a minimization problem, CPLEX cuts off or discards any solutions that are greater than the specified upper cutoff value. If the model has no solution with an objective value less than or equal to the cutoff value, CPLEX declares the model infeasible. In other words, setting an upper cutoff value c for a minimization problem is similar to adding this constraint to the objective function of the model: $obj \leq c$.

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

(*default = 1e+075*)

depind (*integer*)

This option determines if and when the dependency checker will be used.

(*default = -1*)

-1 Automatic

 0 Turn off dependency checking

 1 Turn on only at the beginning of preprocessing

 2 Turn on only at the end of preprocessing

 3 Turn on at the beginning and at the end of preprocessing

disjcuts (*integer*)

Determines whether or not to generate disjunctive cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

(*default = 0*)

-1 Do not generate disjunctive cuts

 0 Determined automatically

 1 Generate disjunctive cuts moderately

 2 Generate disjunctive cuts aggressively

 3 Generate disjunctive cuts very aggressively

divetype (*integer*)

The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The default (automatic) setting chooses when to perform a probing dive, and the other two settings direct Cplex when to perform probing dives: never or always.

(*default = 0*)

    0 Automatic

    1 Traditional dive

    2 Probing dive

    3 Guided dive

divfltup (*real*)

Please check option .divflt for general information on a diversity filter.

If you specify an upper bound on diversity *divfltup*, Cplex will look for solutions similar to the reference values. In other words, you can say, Give me solutions that are close to this one, within this set of variables.

(default = maxdouble)

divfltlo (*real*)

Please check option .divflt for general information on a diversity filter.

If you specify a lower bound on the diversity using *divfltlo*, Cplex will look for solutions that are different from the reference values. In other words, you can say, Give me solutions that differ by at least this amount in this set of variables.

(default = mindouble)

.divflt (*real*)

A diversity filter for a solution pool (see option solnpool) allows you generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution.

A diversity filter drives the search for multiple solutions toward new solutions that satisfy a measure of diversity specified in the filter. This diversity measure applies only to binary variables. Potential new solutions are compared to a reference set. This reference set is specified with this dot option. If no reference set is specified, the difference measure will be computed relative to the other solutions in the pool. The diversity measure is computed by summing the pair-wise absolute differences from solution and the reference values.

(default = 0)

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

    5 Devex pricing

eachcutlim (*integer*)

This parameter allows you to set a uniform limit on the number of cuts of each type that Cplex generates. By default, the limit is a large integer; that is, there is no effective limit by default.

Tighter limits on the number of cuts of each type may benefit certain models. For example, a limit on each type of cut will prevent any one type of cut from being created in such large number that the limit on the total number of all types of cuts is reached before other types of cuts have an opportunity to be created. A setting of 0 means no cuts.

This parameter does not influence the number of Gomory cuts. For means to control the number of Gomory cuts, see also the fractional cut parameters: fraccand, fraccuts, and fracpass.

(default = 2100000000)

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: [0,1]

(*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: [0,0.5]

(*default = 1e-005*)

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: [1e-009,0.1]

(*default = 1e-006*)

epper (*real*)

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 value greater than or equal to 1.0e-8 is valid.

(*default = 1e-006*)

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 higher 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: [1e-009,0.1]

(*default = 1e-006*)

feasopt (*integer*)

With *Feasopt* turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option feasoptmode.

Feasible relaxations are available for all problem types with the exception of quadratically constraint problems.

(*default = 0*)

    0 Turns Feasible Relaxation off

    1 Turns Feasible Relaxation on

feasoptmode (*integer*)

The parameter *FeasOptMode* allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter *FeasOptMode* indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations).

(*default = 0*)

    0 Minimize sum of relaxations. Minimize the sum of all required relaxations in first phase only

    1 Minimize sum of relaxations and optimize. Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations

    2 Minimize number of relaxations. Minimize the number of constraints and bounds requiring relaxation in first phase only

    3 Minimize number of relaxations and optimize. Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations

    4 Minimize sum of squares of relaxations. Minimize the sum of squares of required relaxations in first phase only

    5 Minimize sum of squares of relaxations and optimize. Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref (*real*)

You can express the costs associated with relaxing a bound or right hand side value during a `feasopt` run through the `.feaspref` option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

(*default = 1*)

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 moderately

    2 Generate flow cover cuts aggressively

flowpaths (*integer*)

Determines whether or not flow path cuts should be generated during optimization. At the default of 0, generation is continued only if it seems to be helping.

(default = 0)

-1 Do not generate flow path cuts

0 Determined automatically

1 Generate flow path cuts moderately

2 Generate flow path cuts aggressively

fpheur (*integer*)

Controls the use of the feasibility pump heuristic for mixed integer programming (MIP) models.

(default = 0)

-1 Turns Feasible Pump heuristic off

0 Automatic

1 Apply the feasibility pump heuristic with an emphasis on finding a feasible solution

2 Apply the feasibility pump heuristic with an emphasis on finding a feasible solution with a good objective value

fraccand (*integer*)

Limits the number of candidate variables for generating Gomory fractional cuts.

(default = 200)

fraccuts (*integer*)

Determines whether or not Gomory fractional cuts should be generated during optimization.

(default = 0)

-1 Do not generate Gomory fractional cuts

0 Determined automatically

1 Generate Gomory fractional cuts moderately

2 Generate Gomory fractional cuts aggressively

fracpass (*integer*)

Sets the upper limit on the number of passes that will be performed when generating Gomory fractional cuts on a mixed integer model. Ignored if parameter fraccuts is set to a nonzero value.

(default = 0)

0 0 Automatically determined

>0 Maximum passes to perform

gubcovers (*integer*)

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization. The default of 0 indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.

(default = 0)

-1 Do not generate GUB cover cuts

0 Determined automatically

1 Generate GUB cover cuts moderately

2 Generate GUB cover cuts aggressively

heurfreq (*integer*)

This option specifies how often to apply the node heuristic. Setting to a positive number applies the heuristic at the requested node interval.

(*default = 0*)

-1 Do not use the node heuristic

0 Determined automatically

iis (*integer*)

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. IIS is available for LP problems only.

(*default = 0*)

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 moderately

2 Generate implied bound cuts aggressively

interactive (*integer*)

When set to yes, options can be set interactively after interrupting Cplex with a Control-C. Options are entered just as if they were being entered in the `cplex.opt` file. Control is returned to Cplex by entering `continue`. The optimization can be aborted by entering `abort`. This option can only be used when running from the command line.

(*default = 0*)

intsollim (*integer*)

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.

(*default = large*)

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*)

lbheur (*integer*)

This parameter lets you control whether Cplex applies a local branching heuristic to try to improve new incumbents found during a MIP search. By default, this parameter is off. If you turn it on, Cplex will invoke a local branching heuristic only when it finds a new incumbent. If Cplex finds multiple incumbents at a single node, the local branching heuristic will be applied only to the last one found.

(*default = 0*)

0 Off

1 Apply local branching heuristic to new incumbent

lpmethod (*integer*)

Specifies which LP algorithm to use. If left at the default value (0 for automatic), and a primal-feasible basis is available, primal simplex will be used. If no primal-feasible basis is available, and threads is equal to 1, dual simplex will be used. If threads is greater than 1 and no primal-feasible basis is available, the concurrent option will be used.

Sifting may be useful for problems with many more variables than equations.

The concurrent option runs multiple methods in parallel. The first thread uses dual simplex. The second thread uses barrier. The next thread uses primal simplex. Remaining threads are used by the barrier run. The solution is returned by first method to finish.

(*default = 0*)

    0 Automatic

    1 Primal Simplex

    2 Dual Simplex

    3 Network Simplex

    4 Barrier

    5 Sifting

    6 Concurrent

mcfcuts (*integer*)

Specifies whether Cplex should generate multi-commodity flow (MCF) cuts in a problem where Cplex detects the characteristics of a multi-commodity flow network with arc capacities. By default, Cplex decides whether or not to generate such cuts. To turn off generation of such cuts, set this parameter to -1. Cplex is able to recognize the structure of a network as represented in many real-world models. When it recognizes such a network structure, Cplex is able to generate cutting planes that usually help solve such problems. In this case, the cuts that Cplex generates state that the capacities installed on arcs pointing into a component of the network must be at least as large as the total flow demand of the component that cannot be satisfied by flow sources within the component.

(*default = 0*)

   -1 Do not generate MCF cuts

    0 Determined automatically

    1 Generate MCF cuts moderately

    2 Generate MCF cuts aggressively

memoryemphasis (*integer*)

This parameter lets you indicate to Cplex that it should conserve memory where possible. When you set this parameter to its non default value, Cplex will choose tactics, such as data compression or disk storage, for some of the data computed by the barrier and MIP optimizers. Of course, conserving memory may impact performance in some models. Also, while solution information will be available after optimization, certain computations that require a basis that has been factored (for example, for the computation of the condition number Kappa) may be unavailable.

(*default = 0*)

    0 Do not conserve memory

    1 Conserve memory where possible

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

mipemphasis (*integer*)

This option controls the tactics for solving a mixed integer programming problem.

(*default = 0*)

    0 Balance optimality and feasibility

    1 Emphasize feasibility over optimality

    2 Emphasize optimality over feasibility

    3 Emphasize moving the best bound

    4 Emphasize hidden feasible solutions

mipkappastats (*integer*)

MIP kappa summarizes the distribution of the condition number of the optimal bases CPLEX encountered during the solution of a MIP model. That summary may let you know more about the numerical difficulties of your MIP model. Because MIP kappa (as a statistical distribution) requires CPLEX to compute the condition number of the optimal bases of the subproblems during branch-and-cut search, you can compute the MIP kappa only when CPLEX solves the subproblem with its simplex optimizer. In other words, in order to obtain results with this parameter, you can not use the sifting optimizer nor the barrier without crossover to solve the subproblems. See the parameters startalg and subalg.

Computing the kappa of a subproblem has a cost. In fact, computing MIP kappa for the basis matrices can be computationally expensive and thus generally slows down the solution of a problem. Therefore, the setting 0 (automatic) tells CPLEX generally not to compute MIP kappa, but in cases where the parameter numericalemphasis is turned on, CPLEX computes MIP kappa for a sample of subproblems. The value 1 (sample) leads to a negligible performance degradation on average, but can slow down the branch-and-cut exploration by as much as 10% on certain models. The value 2 (full) leads to a 2% performance degradation on average, but can significantly slow the branch-and-cut exploration on certain models. In practice, the value 1 (sample) is a good trade-off between performance and accuracy of statistics. If you need very accurate statistics, then use value 2 (full).

In case CPLEX is instructed to compute a MIP kappa distribution, the parameter quality is automatically turned on.

(*default = -1*)

  -1 No MIP kappa statistics; default

   0 Automatic: let CPLEX decide

   1 Compute MIP kappa for a sample of subproblems

   2 Compute MIP kappa for all subproblems

mipinterval (*integer*)

Controls the frequency of node logging when the parameter mipdisplay is set higher than 1 (one). Frequency must be an integer; it may be 0 (zero), positive, or negative. By default, CPLEX displays new information in the node log during a MIP solve at relatively high frequency during the early stages of solving a MIP model, and adds lines to the log at progressively longer intervals as solving continues. In other words, CPLEX logs information frequently in the beginning and progressively less often as it works. When the value is a positive integer n, CPLEX displays new incumbents, plus it displays a new line in the log every n nodes. When the value is a negative integer n, CPLEX displays new incumbents, and the negative value determines how much processing CPLEX does before it displays a new line in the node log. A negative value close to zero means that CPLEX displays new lines in the log frequently. A negative value far from

zero means that CPLEX displays new lines in the log less frequently. In other words, a negative value of this parameter contracts or dilates the interval at which CPLEX displays information in the node log.

(default = 0)

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/no build 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 = GAMS PriorOpt)

    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

mipsearch (*integer*)

Sets the search strategy for a mixed integer program. By default, Cplex chooses whether to apply dynamic search or conventional branch and cut based on characteristics of the model.

(default = 0)

    0 Automatic

    1 Apply traditional branch and cut strategy

    2 Apply dynamic search

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

miqcpstrat (*integer*)

This option controls how MIQCPs are solved. For some models, the setting 2 may be more effective than 1. You may need to experiment with this parameter to determine the best setting for your model.

(default = 0)

    0 Automatic

    1 QCP relaxation. Cplex will solve a QCP relaxation of the model at each node.

    2 LP relaxation. Cplex will solve a LP relaxation of the model at each node.

mircuts (*integer*)

Determines whether or not to generate mixed integer rounding (MIR) cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

(*default = 0*)

    -1 Do not generate MIR cuts

    0 Determined automatically

    1 Generate MIR cuts moderately

    2 Generate MIR cuts aggressively

mpslongnum (*integer*)

Determines the precision of numeric output in the MPS file formats. When this parameter is set to its default value 1 (one), numbers are written to MPS files in full-precision; that is, up to 15 significant digits may be written. The setting 0 (zero) writes files that correspond to the standard MPS format, where at most 12 characters can be used to represent a value. This limit may result in loss of precision.

(*default = 1*)

    0 Use limited MPS precision

    1 Use full-precision

names (*integer*)

This option causes GAMS names for the variables and equations to be loaded into Cplex. These names will then be used for error messages, log entries, and so forth. Setting names to no may help if memory is very tight.

(*default = 1*)

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

netepopt (*real*)

This optimality tolerance influences the reduced-cost tolerance for optimality when using the network simplex method. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: [1e-011,0.1]

(*default = 1e-006*)

neteprhs (*real*)

This feasibility tolerance determines the degree to which the network simplex algorithm will allow a flow value to violate its bounds.

Range: [1e-011,0.1]

(*default = 1e-006*)

netfind (*integer*)

Specifies the level of network extraction to be done.

(*default = 2*)

   1 Extract pure network only

   2 Try reflection scaling

   3 Try general scaling

netitlim (*integer*)

Iteration limit for the network simplex method.

(*default = large*)

netppriind (*integer*)

Network simplex pricing algorithm. The default of 0 (currently equivalent to 3) shows best performance for most problems.

(*default = 0*)

   0 Automatic

   1 Partial pricing

   2 Multiple partial pricing

   3 Multiple partial pricing with sorting

nodefileind (*integer*)

Specifies how node files are handled during MIP processing. Used when parameter workmem has been exceeded by the size of the branch and cut tree. If set to 0 when the tree memory limit is reached, optimization is terminated. Otherwise a group of nodes is removed from the in-memory set as needed. By default, Cplex transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node *files* in compressed form in memory. At settings 2 and 3, the node files are transferred to disk. They are stored under a directory specified by parameter workdir and Cplex actively manages which nodes remain in memory for processing.

(*default = 1*)

   0 No node files

   1 Node files in memory and compressed

   2 Node files on disk

   3 Node files on disk and compressed

nodelim (*integer*)

The maximum number of nodes solved before the algorithm terminates, without reaching optimality. This option overrides the GAMS NodLim model suffix. When this parameter is set to 0 (this is only possible through an option file), Cplex completes processing at the root; that is, it creates cuts and applies heuristics at the root. When this parameter is set to 1 (one), it allows branching from the root; that is, nodes are created but not solved.

(*default = GAMS NodLim*)

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

numericalemphasis (*integer*)

This parameter lets you indicate to Cplex that it should emphasize precision in numerically difficult or unstable problems, with consequent performance trade-offs in time and memory.

(*default = 0*)

0 Off

1 Exercise extreme caution in computation

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*)

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 = -1e+075*)

objrng (*string*)

Calculate sensitivity ranges for the specified GAMS variables. Unlike most options, *objrng* can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS variable named. Specifying `all` will cause range information to be produced for all variables. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option rngrestart is specified.

(*default = no objective ranging is done*)

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 = 1e+075*)

parallelmode (*integer*)

Sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic.

In this context, deterministic means that multiple runs with the same model at the same parameter settings on the same platform will reproduce the same solution path and results. In contrast, opportunistic implies that even slight differences in timing among threads or in the order in which tasks are executed in different threads may produce a different solution path and consequently different timings or different solution vectors during optimization executed in parallel threads. When running with multiple threads, the opportunistic setting entails less synchronization between threads and consequently may provide better performance.

In deterministic mode, Cplex applies as much parallelism as possible while still achieving deterministic results. That is, when you run the same model twice on the same platform with the same parameter settings, you will see the same solution and optimization run.

More opportunities to exploit parallelism are available if you do not require determinism. In other words, Cplex can find more opportunities for parallelism if you do not require an invariant, repeatable solution path and precisely the same solution vector. To use all available parallelism, you need to select the opportunistic parallel mode. In this mode, Cplex will utilize all opportunities for parallelism in order to achieve best performance.

However, in opportunistic mode, the actual optimization may differ from run to run, including the solution time itself. A truly parallel deterministic algorithm is available only for MIP optimization. Only opportunistic parallel algorithms (barrier and concurrent optimizers) are available for continuous models. (Each of the simplex algorithms runs sequentially on a continuous model.) Consequently, when parallel mode is set to deterministic, both barrier and concurrent optimizers are restricted to run only sequentially, not in parallel.

A GAMS/Cplex run will use deterministic mode unless explicitely specified.

If *parallelmode* is explicitly set to 0 (automatic) the settings of this parallel mode parameter interact with settings of the threads parameter. Let the result number of threads available to Cplex be n (note that negative values for the threads parameter are possible to exclude work on some cores).

$n=0$: Cplex uses maximum number of threads (determined by the computing platform) in deterministic mode unless *parallelmode* is set to -1 (opportunistic).

$n=1$: Cplex runs sequential.

$n > 1$: Cplex uses maximum number of threads (determined by the computing platform) in opportunistic mode unless *parallelmode* is set to 1 (deterministic).

Here is is list of possible value:

(default = 1)

   -1 Enable opportunistic parallel search mode

    0 Automatic

    1 Enable deterministic parallel search mode

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)

polishafterepagap (*real*)

Solution polishing can yield better solutions in situations where good solutions are otherwise hard to find. More time-intensive than other heuristics, solution polishing is actually a variety of branch-and-cut that works after an initial solution is available. In fact, it requires a solution to be available for polishing, either a solution produced by branch-and-cut, or a MIP start supplied by a user. Because of the high cost entailed by solution polishing, it is not called throughout branch-and-cut like other heuristics. Instead, solution polishing works in a second phase after a first phase of conventional branch-and-cut. As an additional step after branch-and-cut, solution polishing can improve the best known solution. As a kind of branch-and-cut algorithm itself, solution polishing focuses solely on finding better solutions. Consequently, it may not prove optimality, even if the optimal solution has indeed been found. Like the RINS heuristic, solution polishing explores neighborhoods of previously found solutions by solving subMIPs.

Sets an absolute MIP gap (that is, the difference between the best integer objective and the objective of the best node remaining) after which CPLEX stops branch-and-cut and begins polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

(*default = 0*)

polishafterepgap (*real*)

Sets a relative MIP gap after which CPLEX will stop branch-and-cut and begin polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

(*default = 0*)

polishafternode (*integer*)

Sets the number of nodes processed in branch-and-cut before CPLEX starts solution polishing, if a feasible solution is available.

(*default = 2100000000*)

polishafterintsol (*integer*)

Sets the number of integer solutions to find before CPLEX stops branch-and-cut and begins to polish a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

(*default = 2100000000*)

polishaftertime (*real*)

Tells CPLEX how much time in seconds to spend during mixed integer optimization before CPLEX starts polishing a feasible solution. The default value is such that CPLEX does not start solution polishing by default.

(*default = 0*)

populatelim (*integer*)

Limits the number of solutions generated for the solution pool during each call to the populate procedure. Populate stops when it has generated *PopulateLim* solutions. A solution is counted if it is valid for all filters (see .divflt and consistent with the relative and absolute pool gap parameters (see solnpoolgap and solnpoolagap), and has not been rejected by the incumbent checking routine (see userincbcall), whether or not it improves the objective of the model. This parameter does not apply to MIP optimization generally; it applies only to the populate procedure.

If you are looking for a parameter to control the number of solutions stored in the solution pool, consider the parameter solnpoolcapacity instead.

Populate will stop before it reaches the limit set by this parameter if it reaches another limit, such as a time or node limit set by the user.

(*default = 20*)

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*)

    -1 do not give dual to optimizer

    0 automatic

    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*)

prelinear (*integer*)

If only linear reductions are performed, each variable in the original model can be expressed as a linear form of variables in the presolved model.

(*default = 1*)

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

preslvnd (*integer*)

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective.

(*default = 0*)

    -1 No node presolve

    0 Automatic

    1 Force node presolve

    2 Perform probing on integer-infeasible variables

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 integer values are valid.

(default = 0, in which case it is determined automatically)

printoptions (*integer*)

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

(default = 0)

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 Limited probing

   2 More probing

   3 Full probing

probetime (*real*)

Limits the amount of time in seconds spent probing.

(default = 1e+075)

qpmakepsdind (*integer*)

Determines whether Cplex will attempt to adjust a MIQP formulation, in which all the variables appearing in the quadratic term are binary. When this feature is active, adjustments will be made to the elements of a quadratic matrix that is not nominally positive semi-definite (*PSD*, as required by Cplex for all QP formulations), to make it PSD, and will also attempt to tighten an already PSD matrix for better numerical behavior. The default setting of 1 means yes but you can turn it off if necessary; most models should benefit from the default setting.

(default = 1)

   0 Off

   1 On

qpmethod (*integer*)

Specifies which QP algorithm to use.

At the default of 0 (automatic), barrier is used for QP problems and dual simplex for the root relaxation of MIQP problems.

(default = 0)

   0 Automatic

   1 Primal Simplex

   2 Dual Simplex

   3 Network Simplex

   4 Barrier

   5 Sifting

   6 Concurrent dual, barrier, and primal

quality (*integer*)

Write solution quality statistics to the listing file. If set to yes, the statistics appear after the Solve Summary and before the Solution Listing.

(*default = 0*)

readflt (*string*)

The GAMS/Cplex solution pool options cover the basic use of diversity and range filters for producing multiple solutions. If you need multiple filters, weights on diversity filters or other advanced uses of solution pool filters, you could produce a Cplex filter file with your favorite editor or the GAMS Put Facility and read this into GAMS/Cplex using this option.

reduce (*integer*)

Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. It is occasionally advisable to do only one or the other when diagnosing infeasible or unbounded models.

(*default = 3*)

  0 No primal or dual reductions

  1 Only primal reductions

  2 Only dual reductions

  3 Both primal and dual reductions

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*)

relaxfixedinfeas (*integer*)

Sometimes the solution of the fixed problem of a MIP does not solve to optimality due to small (dual) infeasibilities. The default behavior of the GAMS/Cplex link is to return the primal solution values only. If the option is set to 1, the small infeasibilities are ignored and a full solution including the dual values are reported back to GAMS.

(*default = 0*)

  0 Off

  1 On

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 = -1*)

  -1 Automatic

  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*)

repairtries (*integer*)

This parameter lets you indicate to Cplex whether and how many times it should try to repair an infeasible MIP start that you supplied. The parameter has no effect if the MIP start you supplied is feasible. It has no effect if no MIP start was supplied.

(*default = 0*)

-1 None: do not try to repair

0 Automatic

>0 Maximum tries to perform

repeatpresolve (*integer*)

This integer parameter tells Cplex whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

(*default = -1*)

-1 Automatic

0 Turn off represolve

1 Represolve without cuts

2 Represolve with cuts

3 Represolve with cuts and allow new root cuts

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 presolve 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 auto, yes, no and nono. The value of auto is equivalent to no if names are successfully loaded into Cplex and option iis is set to no. In that case the Cplex messages from presolve help identify the cause of infeasibility or unboundedness in terms of GAMS variable and equation names. If names are not successfully loaded, rerun defaults to yes. Loading of GAMS names into Cplex is controlled by option names. The value of nono only affects MIP models for which Cplex finds a feasible solution in the branch-and-bound tree but the fixed problem turns out to be infeasible. In this case the value nono also disables the rerun without presolve, while the value of no still tries this run. Feasible integer solution but an infeasible fixed problem happens in few cases and mostly with badly scaled models. If you experience this try more aggressive scaling (scaind) or tightening the integer feasibility tolerance epint. If the fixed model is infeasible only the primal solution is returned to GAMS. You can recognize this inside GAMS by checking the marginal of the objective defining constraint which is always nonzero.

(*default = yes*)

auto Automatic

yes Rerun infeasible models with presolve turned off

no Do not rerun infeasible models

nono Do not rerun infeasible fixed MIP models

rhsrng (*string*)

Calculate sensitivity ranges for the specified GAMS equations. Unlike most options, *rhsrng* can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS equation named. Specifying `all` will cause range information to be produced for all equations. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option rngrestart is specified.

(*default = no right-hand-side ranging is done*)

rinsheur (*integer*)

Cplex implements a heuristic known a Relaxation Induced Neighborhood Search (RINS) for MIP and MIQCP problems. RINS explores a neighborhood of the current incumbent to try to find a new, improved incumbent. It formulates the neighborhood exploration as a MIP, a subproblem known as the subMIP, and truncates the subMIP solution by limiting the number of nodes explored in the search tree.

Parameter *rinsheur* controls how often RINS is invoked. A value of 100, for example, means that RINS is invoked every hundredth node in the tree.

(*default = 0*)

-1 Disable RINS

0 Automatic

rngrestart (*string*)

Write ranging information, in GAMS readable format, to the file named. Options objrng and rhsrng are used to specify which GAMS variables or equations are included.

(*default = ranging information is printed to the listing file*)

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. This method 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.

siftalg (*integer*)

Sets the algorithm to be used for solving sifting subproblems.

(*default = 0*)

0 Automatic

1 Primal simplex

2 Dual simplex

3 Network simplex

4 Barrier

siftdisplay (*integer*)

Determines the amount of sifting progress information to be displayed.

(*default = 1*)

0 No display

1 Display major iterations

2 Display LP subproblem information

siftitlim (*integer*)

Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

(*default = large*)

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.

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)

solnpool (*string*)

The solution pool enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file name contains the information about the different solutions generated by Cplex. Inside your GAMS program you can process the GDX file and read the different solution point files. Please check the GAMS/Cplex solver guide document and the example model `solnpool.gms` from the GAMS model library.

solnpoolagap (*real*)

Sets an absolute tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the objective of the incumbent solution according to this measure are not kept in the solution pool.

Values of the solution pool absolute gap and the solution pool relative gap solnpoolgap may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and also within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool absolute gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

(default = 1e+075)

solnpoolcapacity (*integer*)

Limits the number of solutions kept in the solution pool. At most, *solnpoolcapacity* solutions will be stored in the pool. Superfluous solutions are managed according to the replacement strategy set by the solution pool replacement parameter solnpoolreplace.

The optimization (whether by MIP optimization or the populate procedure) will not stop if more than *solnpoolcapacity* are generated. Instead, stopping criteria are regular node and time limits and populatelim, solnpoolgap and solnpoolagap.

(default = 2100000000)

solnpoolgap (*real*)

Sets a relative tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool.

Values of the solution pool absolute gap solnpoolagap and the solution pool relative gap may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute

gap and within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool relative gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

(default = 1e+075)

solnpoolintensity (*integer*)

Controls the trade-off between the number of solutions generated for the solution pool and the amount of time or memory consumed. This parameter applies both to MIP optimization and to the populate procedure.

Values from 1 to 4 invoke increasing effort to find larger numbers of solutions. Higher values are more expensive in terms of time and memory but are likely to yield more solutions.

(default = 0)

0 Automatic. Its default value, 0 , lets Cplex choose which intensity to apply.

1 Mild: generate few solutions quickly. For value 1, the performance of MIP optimization is not affected. There is no slowdown and no additional consumption of memory due to this setting. However, populate will quickly generate only a small number of solutions. Generating more than a few solutions with this setting will be slow. When you are looking for a larger number of solutions, use a higher value of this parameter.

2 Moderate: generate a larger number of solutions. For value 2, some information is stored in the branch and cut tree so that it is easier to generate a larger number of solutions. This storage has an impact on memory used but does not lead to a slowdown in the performance of MIP optimization. With this value, calling populate is likely to yield a number of solutions large enough for most purposes. This value is a good choice for most models.

3 Aggressive: generate many solutions and expect performance penalty. For value 3, the algorithm is more aggressive in computing and storing information in order to generate a large number of solutions. Compared to values 1 and 2, this value will generate a larger number of solutions, but it will slow MIP optimization and increase memory consumption. Use this value only if setting this parameter to 2 does not generate enough solutions.

4 Very aggressive: enumerate all practical solutions. For value 4, the algorithm generates all solutions to your model. Even for small models, the number of possible solutions is likely to be huge; thus enumerating all of them will take time and consume a large quantity of memory.

solnpoolpop (*integer*)

Regular MIP optimization automatically adds incumbents to the solution pool as they are discovered. Cplex also provides a procedure known as *populate* specifically to generate multiple solutions. You can invoke this procedure either as an alternative to the usual MIP optimizer or as a successor to the MIP optimizer. You can also invoke this procedure many times in a row in order to explore the solution space differently (see option solnpoolpoprepeat). In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory.

(default = 1)

1 Just collect the incumbents found during regular optimization

2 Calls the populate procedure

solnpoolpopdel (*string*)

After the GAMS program specified in solnpoolpoprepeat determined to continue the search for alternative solutions, the file specified by this option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

solnpoolpoprepeat (*string*)

After the termination of the populate procedure (see option solnpoolpop). The GAMS program specified in this option will be called which can examine the solutions in the solution pool and can decide to run the populate procedure again. If the GAMS program terminates normally (not compilation or execution time error) the search for new alternative solutions will be repeated.

solnpoolprefix (*string*)

(*default = soln*)

solnpoolreplace (*integer*)

(*default = 0*)

    0 Replace the first solution (oldest) by the most recent solution; first in, first out

    1 Replace the solution which has the worst objective

    2 Replace solutions in order to build a set of diverse solutions

solutiontarget (*integer*)

This parameter specifies the type of solution when solving a nonconvex, continuous quadratic model. This parameter affects the behavior only when CPLEX uses the barrier algorithm without crossover to solve a nonconvex continuous quadratic model (QP); that is, the variables of the model are continuous, the objective function includes a quadratic term, and the objective function is not positive semi-definite (PSD).

(*default = 0*)

    0 Automatic. CPLEX first attempts to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX will return with an error (Q is not PSD).

    1 Search for a globally optimal solution to a convex model

    2 Search for a solution that satisfies first-order optimality conditions no optimality guarantee. CPLEX first attempt to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX searches for a solution that satisfies first-order optimality conditions but is not necessarily globally optimal.

solvefinal (*integer*)

Sometimes the solution process after the branch-and-cut that solves the problem with fixed discrete variables takes a long time and the user is interested in the primal values of the solution only. In these cases, `solvefinal` can be used to turn this final solve off. Without the final solve no proper marginal values are available and only zeros are returned to GAMS.

(*default = 1*)

    0 Do not solve the fixed problem

    1 Solve the fixed problem and return duals

startalg (*integer*)

Selects the algorithm to use for the initial relaxation of a MIP.

(*default = 0*)

    0 Automatic

    1 Primal simplex

    2 Dual simplex

    3 Network simplex

    4 Barrier

    5 Sifting

    6 Concurrent

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*)

subalg (*integer*)

Strategy for solving linear sub-problems at each node.

(*default = 0*)

    0 Automatic

    1 Primal simplex

    2 Dual simplex

    3 Network optimizer followed by dual simplex

    4 Barrier with crossover

    5 Sifting

submipnodelim (*integer*)

Controls the number of nodes explored in an RINS subMIP. See option rinsheur.

(*default = 500*)

symmetry (*integer*)

Determines whether symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model.

(*default = -1*)

  -1 Automatic

    0 Turn off symmetry breaking

    1 Moderate level of symmetry breaking

    2 Aggressive level of symmetry breaking

    3 Very aggressive level of symmetry breaking

    4 Highly aggressive level of symmetry breaking

    5 Extremely aggressive level of symmetry breaking

threads (*integer*)

Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks. Cplex does not understand negative values for the `threads` parameter. GAMS/Cplex will translate this is a non-negative number by applying the following formula: $\max(1, \text{number of cores} - |\texttt{threads}|)$

(*default = GAMS Threads*)

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*)

Sets an absolute upper limit on the size (in megabytes) of the branch and cut tree. If this limit is exceeded, Cplex terminates optimization.

(*default = 1e+075*)

tuning (*string*)

Invokes the Cplex parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Cplex option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in LP, MPS or SAV format. Cplex will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Cplex option file name without considering the problem provided by GAMS (use option writesav to create a SAV file of the problem provided by GAMS and include this name in the list of problems). The result of such a run is the updated GAMS/Cplex option file with a tuned set of parameters. The solver and model status returned to GAMS will be `NORMAL COMPLETION` and `NO SOLUTION`. Tuning is incompatible with the BCH facility and other advanced features of GAMS/Cplex.

tuningdisplay (*integer*)

Specifies the level of information reported by the tuning tool as it works.

(*default = 1*)

    0 Turn off display

    1 Display standard minimal reporting

    2 Display standard report plus parameter settings being tried

    3 Display exhaustive report and log

tuningmeasure (*integer*)

Controls the measure for evaluating progress when a suite of models is being tuned. Choices are mean average and minmax of time to compare different parameter sets over a suite of models

(*default = 1*)

    1 mean average

    2 minmax

tuningrepeat (*integer*)

Specifies the number of times tuning is to be repeated on perturbed versions of a given problem. The problem is perturbed automatically by Cplex permuting its rows and columns. This repetition is helpful when only one problem is being tuned, as repeated perturbation and re-tuning may lead to more robust tuning results. This parameter applies to only one problem in a tuning session.

(*default = 1*)

tuningtilim (*real*)

Sets a time limit per model and per test set (that is, suite of models).

As an example, suppose that you want to spend an overall amount of time tuning the parameter settings for a given model, say, 2000 seconds. Also suppose that you want Cplex to make multiple attempts within that overall time limit to tune the parameter settings for your model. Suppose further that you want to set a time limit on each of those attempts, say, 200 seconds per attempt. In this case you need to specify an overall time limit of 2000 using GAMS option `reslim` or Cplex option tilim and `tuningtilim` to 200.

(*default = 0.2*GAMS ResLim*)

userincbcall (*string*)

The GAMS command line (minus the GAMS executable name) to call the incumbent checking routine. The incumbent is rejected if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.

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 sub-problems 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

workdir (*string*)

The name of an existing directory into which Cplex may store temporary working files. Used for MIP node files and by out-of-core Barrier.

(*default = current or project directory*)

workmem (*real*)

Upper limit on the amount of memory, in megabytes, that Cplex is permitted to use for working files. See parameter workdir.

(*default = 128*)

writebas (*string*)

Write a basis file.

writeflt (*string*)

Write the diversity filter to a Cplex FLT file.

writelp (*string*)

Write a file in Cplex LP format.

writemps (*string*)

Write an MPS problem file.

writemst (*string*)

Write a Cplex mst (containing the mip start) file.

writeord (*string*)

Write a Cplex ord (containing priority and branch direction information) file.

writeparam (*string*)

Write a Cplex parameter (containing all modified Cplex options) file.

writepre (*string*)

Write a Cplex LP, MPS, or SAV file of the presolved problem. The file extension determines the problem format. For example, `writepre presolved.lp` creates a file `presolved.lp` in Cplex LP format.

writesav (*string*)

Write a binary problem file.

zerohalfcuts (*integer*)

Decides whether or not to generate zero-half cuts for the problem. The value 0, the default, specifies that the attempt to generate zero-half cuts should continue only if it seems to be helping. If the dual bound of your model does not make sufficient progress, consider setting this parameter to 2 to generate zero-half cuts more aggressively.

(default = 0)

-1 Off

0 Automatic

1 Generate zero-half cuts moderately

2 Generate zero-half cuts aggressively