USER'S GUIDE FOR TOMLAB /CPLEX v12.1 [1]

Kenneth Holmström[2], Anders O. Göran[3] and Marcus M. Edvall[4]

August 14, 2009

[1] More information available at the TOMLAB home page: http://tomopt.com E-mail: **tomlab@tomopt.com**.

[2] Professor in Optimization, Mälardalen University, Department of Mathematics and Physics, P.O. Box 883, SE-721 23 Västerås, Sweden, kenneth.holmstrom@mdh.se.

[3] Tomlab Optimization AB, Västerås Technology Park, Trefasgatan 4, SE-721 30 Västerås, Sweden, anders@tomopt.com.

[4] Tomlab Optimization Inc., 1260 SE Bishop Blvd Ste E, Pullman, WA, USA, medvall@tomopt.com.

Contents

1 Introduction

1.1 Overview

Welcome to the TOMLAB /CPLEX User's Guide. TOMLAB /CPLEX includes the ILOG CPLEX 12.1 (hereafter commonly referred to as CPLEX) solver and Matlab interfaces. The software allows for execution on any number of shared memory cores or cpus on a computer.

The interface between ILOG CPLEX, Matlab and TOMLAB consists of two layers. The first layer gives direct access from Matlab to CPLEX, via calling one Matlab function that calls a pre-compiled MEX file (DLL under Windows, shared library in UNIX) that defines and solves the problem in CPLEX . The second layer is a Matlab function that takes the input in the TOMLAB format, and calls the first layer function. On return the function creates the output in the TOMLAB format.

CPLEX has a whole set of callback routines. There is one predefined Matlab routine for each callback. The user is in control of which ones to use, and should add his own code in Matlab for each callback.

Conflict refining, SA, warm start and solution pool control are supported by in the package.

1.2 Contents of this Manual

- Read carefully Section 2 on how to install TOMLAB /CPLEX.
- Section 3 gives the basic information needed to run the Matlab interface.
- The more advanced feature, using callbacks, is described in Section 4.
- Some Matlab test routines are included, described in Section 5 (non-TOMLAB format) and Section 6 (TOMLAB format). All Matlab routines are described in Appendix A.
- Section E describes a special interface for network problems.
- Section F defines the features included in IIS (infeasibility analysis) and SA (sensitivity analysis) as well as the warm start functionality.

1.3 Prerequisites

In this manual we assume that the user is familiar with CPLEX, the CPLEX Reference Manual, TOMLAB and the Matlab language.

2 Installing TOMLAB /CPLEX

2.1 Windows Systems

TOMLAB /CPLEX is installed by the general TOMLAB exe installer and can be installed with or without the TOMLAB Base Module. In either case, the folders *tomlab\cplex* and possibly also a *tomlab\shared* are created.

The *tomlab\shared* folder contains the *cplex*.dll* and must be installed on machine without an existing ILOG CPLEX installation. The installer automatically adds the location of this folder to the Windows PATH variable.

If installing TOMLAB /CPLEX together with other TOMLAB packages, the *tomlab\startup.m* file will automatically detect TOMLAB /CPLEX and set the MATLAB path accordingly.

You may also set the TOMLAB and TOMLAB /CPLEX paths permanently in the Matlab system. To find out which paths are used, run the startup commands as described above, then use the `path` command to see what paths TOMLAB created and set these permanently on your system. See the Matlab documentation on how to set Matlab paths.

2.2 Unix/Linux Systems

TOMLAB /CPLEX is installed together with the rest of TOMLAB when extracting the *tomlab-<arch>-setup.tar.gz* file. If ILOG CPLEX is not already installed on the computer, the user must set/modify the `LD_LIBRARY_PATH` environment variable in order for the runtime linking to work as intended. Assuming that TOMLAB is extracted to $HOME/tomlab/shared, do:

```
#
# csh/tcsh shells:
#
setenv LD_LIBRARY_PATH $HOME/tomlab/shared:$LD_LIBRARY_PATH

#
# bash and compatible shells:
#
export LD_LIBRARY_PATH=$HOME/tomlab/shared:$LD_LIBRARY_PATH
```

To use TOMLAB /CPLEX together with the TOMLAB Base Module, execute the *tomlab/startup* script, which automatically recognizes the presence of the cplex subdirectory.

To use TOMLAB /CPLEX by itself, execute only the *tomlab/cplex/startup* script.

2.3 Troubleshooting

Error messages like the following:

```
>> cplexmex
Unable to load mex file: d:\program files\tomlab\cplex\cplexmex.dll.
The specified module could not be found.

??? Invalid MEX-file
```

or, on Unix systems:

```
>> cplexmex
Unable to load mex file: /home/user/tomlab/cplex/cplexmex.mexglx.
libcplex91.so: cannot open shared object file: No such file or
directory ??? Invalid MEX-file
```

indicate a problem with the PATH variable on Windows systems, or equivalently the $LD_LIBRARY_PATH variable in Unix/Linux.

You can check from within Matlab that the path has been set correctly, by executing

```
>> getenv('PATH')              % Windows

>> getenv('LD_LIBRARY_PATH')   % Unix/Linux
```

The location of the *tomlab/shared* directory must be included in the result, OR, the location of the cplex*.dll (libcplex*.so) file on systems where ILOG CPLEX is already installed.

3 Using the Matlab Interface

The two main routines in the two-layer design of the interface are shown in Table 1. Page and section references are given to detailed descriptions on how to use the routines. Users not using the TOMLAB *Prob* format can skip reading about the routine *cplexTL*. A normal user, not using callbacks, only has to read about how to call the level 1 interface routine *cplex.m*.

Table 1: The interface routines.

Function	Description	Section	Page
cplex	The layer one Matlab interface routine, calls the MEX-file interface *cplexmex.dll*	A.1	12
cplexTL	The layer two TOMLAB interface routine that calls *cplex.m*. Converts the input *Prob* format before calling *cplex.m* and converts back to the output *Result* structure.	A.2	22

The CPLEX control parameters (Section G, 75 in this manual), are all possible to set from Matlab.

They could be set as input to the interface routine *cplex*, but also in the callback routines. The user sets fields in a structure called *cpxControl*, where the subfield names are the same as the names of the control variables. The following example shows how to set the values for one integer variable `ITLIM`, one double variable `EPOPT`, and one character variable valued variable `WORKDIR`. TOMLAB /CPLEX does not use the prefix `CPX_PARAM_` in the Matlab structures.

```
cpxControl.ITLIM         = 50;      % Setting maximal number of simplex iterations
cpxControl.EPOPT         = 1E-5;    % Changing reduced cost tolerance
cpxControl.WORKDIR       = '.';     % New work directory ( '.' for current directory)
```

Character valued variables should have ≤ 64 characters.

4 Callbacks in Matlab

Fifteen of the CPLEX callbacks are defined as Matlab m-files. A logical vector defines the callbacks to be used in CPLEX. This vector is named *callback* and is one of the input variables to the level 1 interface routine *cplex.m* (Section A.1). If the i^{th} entry of the logical vector callback is set, the corresponding callback is defined.

The callback calls the *m*-file specified in Table 2. The user can edit this *m*-file directly, or make a new copy. It is important that a new copy is placed in a directory that is searched before the *cplex* directory when Matlab goes through the Matlab path.

Index	m-file	Called at: **Section**	**Page**
1	*cpxcb_BARRIER.m*	D.1	55
2	*cpxcb_DISJCUT.m*	D.2	55
3	*cpxcb_DUAL.m*	D.3	56
4	*cpxcb_DUALCROSS.m*	D.4	57
5	*cpxcb_FLOWMIR.m*	D.5	58
6	*cpxcb_FRACCUT.m*	D.6	58
7	*cpxcb_MIP.m*	D.7	59
8	*cpxcb_MIPPROBE.m*	D.8	60
9	*cpxcb_PRESOLVE.m*	D.9	61
10	*cpxcb_PRIM.m*	D.10	62
11	*cpxcb_PRIMCROSS.m*	D.11	62
12	*cpxcb_QPBARRIER.m*	D.12	63
13	*cpxcb_QPSIMPLEX.m*	D.13	64
14	*cpxcb_INCUMBENT.m*	D.14	64
1	*cpxcb_NET.m*	D.16	65

Table 2: Matlab Callback routines

5 Test Routines in Non-Tomlab Format

A set of test routines have been defined illustrating the use of the *cplex* main routine. The test routines and utilities are shown in Table 3.

It is easy to call the test routines, e.g.

```
x = cpxtest1;
x = cpxtest2;
x = cpxtest3;
```

will call the three routines solving GAP problems. The *cpxaircrew* test problem has no input parameters, just call:

```
cpxaircrew;
```

The knapsack test routine runs three test examples. It is possible to change the cut strategy (second input parameter). To run the second test example, using aggressive cuts, the call is

```
cpxKnaps(2,2);
```

The first parameter selects the test problem. Calling without any parameters

```
cpxKnaps
```

is the same as the call

```
cpxKnaps(1,0);
```

Table 3: Test routines and utilities in non-Tomlab format.

Function	**Description**	**Section**	**Page**
abc2gap	Utility to convert a Generalized Assignment Problem (GAP) to standard form for CPLEX.	B.2	40
cpxbiptest	Test of three large binary integer linear problems.	C.2	42
cpxiptest	Test of three large integer linear problems.	C.3	43
cpxKnaps	Test of knapsack problems.	C.6	45
xptest1	Test of a Generalized Assignment Problem (GAP).	C.10	49
cpxTest2	Test of the same GAP problem as *cpxTest1*, but using sos1 variables.	C.11	50
cpxTest3	Test of a Generalized Assignment Problem (GAP).	C.12	51

6 Test Routines in TOMLAB Format

A set of test routines have been defined illustrating the combined use of TOMLAB and CPLEX. The test routines are shown in Table 4. The knapsack test routine *cpxKnapsTL* is similar to *cpxKnaps* discussed in the previous subsection. It runs three knapsack test examples. It is possible to change the cut strategy. The problems are setup using the TOMLAB Format.

Table 4: Test routines and utilities in TOMLAB format.

Function	**Description**	**Section**	**Page**
cpxtomtest1	Tests of problems predefined in the TOMLAB IF format. LP, QP and MIP problems are solved calling the driver routine *tomRun*.	C.4	44
cpxtomtest2	Tests of a simple MIP problem defined in the TOMLAB (TQ) format. The problem is solved as an LP and MIP problem, with or without slacks defined. *tomRun*.	C.5	44
cpxKnapsTL	The same tests as in *cpxKnaps*, but the TOMLAB problem definition format and is used.	C.7	46

A The Matlab Interface Routines - Main Routines

A.1 cplex

Purpose

CPLEX solves mixed-integer linear and quadratic programming (MILP, MIQP) and linear and quadratic programming (LP, QP) interface. For users with a full license for the optimizer, the solver also handles problems with quadratic constraints (MIQQ). CPLEX solves problems of the form

$$
\begin{array}{cccccl}
\min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x & & & & \\
s/t & x_L & \leq & x & \leq & x_U \\
 & b_L & \leq & Ax & \leq & b_U \\
 & & & x^T Q^{(i)} x + a^{(i)T} x & \leq & r_U^{(i)}, \quad i = 1,\ldots,n_{qc} \\
 & & & x_i \ \text{integer} & & i \in I
\end{array}
$$

where $c, x, x_L, x_U, a^{(i)} \in \mathbb{R}^n$, $F, Q^{(i)} \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m\times n}$ and $b_L, b_U \in \mathbb{R}^m$. $r_U^{(i)}$ is a scalar. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

If F is empty and no quadratic constraints are given, an LP or MILP problem is solved.

An additional set of logical constraints can also be defined. See the help for input parameter *logcon*.

Calling Syntax

[x, slack, v, rc, f_k, ninf, sinf, Inform, basis, lpiter, glnodes, confstat, iconfstat, sa] = cplex(c, A, x_L, x_U, b_L, b_U, cpxControl, callback, PriLev, Prob, IntVars, PI, SC, SI, sos1, sos2, F, logfile, savefile, savemode, qc, confgrps, conflictFile, saRequest, basis, xIP, logcon);

Description of Inputs

The following inputs are used:

c	Linear objective function cost coefficients, vector $n \times 1$.
A	Linear constraint matrix for linear constraints, dense or sparse matrix $m \times n$.
x_L	Lower bounds on design parameters x. If empty assumed to be zero.
x_U	Upper bounds on design parameters x.
b_L	Lower bounds on the linear constraints.

The following parameters are optional:

b_U	Upper bounds on the linear constraints. If empty, then *b_U*=*b_L* is assumed, i.e. equality constraints.
cpxControl	Structure, where the fields are set to the CPLEX control parameters that the user wants to specify values for.

The following inputs are used:, continued

callback $0-1$ vector defining which callbacks to use in CPLEX. If the i^{th} entry of the logical vector *callback* is set, the corresponding callback is defined. The callback calls the m-file specified in Table 7 below. The user may edit this file, or make a new copy, which is put in a directory that is searched before the *cplex* directory in the Matlab path.

PriLev Printing level in the *cplex* *m*-file and the CPLEX C-interface.
= 0 Silent
= 1 Summary information
= 2 More detailed information

Prob A structure. If *cplex.m* is called through *cplexTL.m*, for example when the user has used the TOMLAB driver routine *tomRun*, then *Prob* is the standard TOMLAB problem structure.

Otherwise the user optionally may set: $Prob.P = ProblemNumber;$, where ProblemNumber is some integer. If any callback is defined then problem arrays are set as fields in *Prob*, and the *Prob* structure is always passed to the callback routines as the last parameter. The defined fields are *Prob.c*, *Prob.x_L*, *Prob.x_U*, *Prob.A*, *Prob.b_L*, *Prob.b_U* and *Prob.QP.F*. If the input structure is empty ([ ]), then $Prob.P = 1$ is set.

IntVars Defines which variables are integers, of the general type I or binary B. Variable indices should be in the range [1,...,n]. If $IntVars$ is a logical vector then all variables x_i where $IntVars(i) > 0$ are defined to be integers. If $IntVars$ is determined to be a vector of indices then $x(IntVars)$ are defined as integers. If the input is empty ([ ]), then no integers of type I or B are defined. The interface routine *cplex* checks which of the integer variables have lower bound $x_L = 0$ and upper bound $x_U = 1$, i.e. are binary 0/1 variables.

PI Integer variables of type *Partially Integer* (PI), i.e. takes an integer value up to a specified limit, and any real value above that limit. PI must be a structure array where:
$PI.var$ is a vector of variable indices in the range $[1, ..., n]$.
$PI.lim$ is a vector of limit values for each of the variables specified in PI.var, i.e. for variable i, that is the PI variable with index j in $PI.var$, then $x(i)$ takes integer values in $[x_L(i), PI.lim(j)]$ and values in $[PI.lim(j), x_U(i)]$.

SC A vector with indices for the integer variables of type *Semi-continuous* (SC), i.e. that takes either the value 0 or a real value in the range $[x_L(i), x_U(i)]$, assuming for some j, that $i = SC(j)$, where i is an variable number in the range $[1, ..., n]$.

SI A vector with indices for the integer variables of type *Semi-integer* (SI), i.e. that takes either the value 0 or an integer value in the range $[x_L(i), x_U(i)]$, assuming for some j, that $i = SIe(j)$, where i is an variable number in the range $[1, ..., n]$.

The following inputs are used:, continued

sos1 A structure defining the *Special Ordered Sets of Type One* (sos1). Assume there are k sets of type sos1, then $sos1(k).var$ is a vector of indices for variables of type sos1 in set k. $sos1(k).row$ is the row number for the reference row identifying the ordering information for the sos1 set, i.e. A(sos1(k).row,sos1(k).var) identifies this information. As ordering information, also the objective function coefficients c could be used. Then as row number, 0 is instead given in $sos1(k).row$.

sos2 A structure defining the *Special Ordered Sets of Type Two* (sos2). Specified in the same way as sos1 sets; see $sos1$ input variable description.

F Quadratic coefficient matrix. Dense or sparse Matlab matrix, size $n \times n$. The matrix is always converted to Matlab sparse format before ILOG CPLEX is invoked on the problem.

logfile Name of file to write the CPLEX log information to. If empty, no log is written.

savefile Name of a file to save the CPLEX problem object. This is useful for sending problems to ILOG for analysis. The format of the file is controlled by the following parameters, *savemode*. If empty, no file is written.

savemode The format of the file given in *savefile* is possible to choose by setting *savemode* to one of the following values:

1	SAV	Binary SAV format
2	MPS	MPS format (ASCII)
3	LP	CPLEX LP format (ASCII)
4	RMP	MPS file with generic names
5	REW	MPS file with generic names
6	RLP	LP file with generic names

Modes 4-6 are of limited interest, since the TOMLAB interface does not provide a way to change the default row names in the first place.

qc Structure array defining quadratic constraints ("qc").

Please note that CPLEX only handles single-sided bounds on quadratic constraints. An arbitrary number of qc's is set using the Prob.QP.qc structure array:

qc(1).Q = sparse(<quadratic coefficient nxn matrix>);
qc(1).a = full (<linear coefficient nx1 vector >);
qc(1).r_U = <scalar upper bound>;

And similarly for qc(2), ... , qc(n_qc).

The following inputs are used:, continued

The standard interpretation is $x' * Q * x + c' * x <= r_U$, but it is possible to define an alternative sense $x' * Q * x + c' * x >= r_L$ by setting $qc(i).sense$ to a nonzero value and specifying a lower bound in $qc(i).r_L$.

Observe that the Q matrix must be sparse, non-empty and positive semi-definite for all qc's. The linear coefficient vector $qc(i).a$ may be omitted or set empty, in which case all zeros are assumed.

Likewise, if a bound r_U or r_L is empty or not present, it is assumed to be 0.0. Note that this is contrary to the usual TOMLAB standard, where an empty or omitted bound is assumed to be +/- Inf. The reason is that a single-sided constraint with an infinite bound would have no meaning.

confgrps Conflict groups descriptor (cpxBuildConflict can be used to generate the input). Set this if conflict refinement is desired in the case that infeasibility is detected by CPLEX.
A conflict group consists of lists of indices describing which of the following entities are part of a group:

confgrps(i).lowercol Column (variable) lower bounds
confgrps(i).uppercol Column (variable) upper bounds
confgrps(i).linear Linear rows
confgrps(i).quad Quadratic constraints
confgrps(i).sos Special ordered sets
confgrps(i).indicator Indicator constraints

Additionally, the group's priority value may be assigned in
confgrps(i).priority

Please refer to the TOMLAB /CPLEX User's Guide for an example of Conflict Refinement.

conflictFile Name of a file to write the conflict refinement to. No file is written if this input parameter is empty or if no conflict refinement is done.

saRequest Structure telling whether and how you want CPLEX to perform a sensitivity analysis (SA). You can complete an SA on the objective function, right hand side vector, lower and upper bounds. The saRequest structure contains four sub structures:

.obj, .rhs, .xl, .xu

Each one of these contain the field:

.index

.index contain indices to variables or constraints of which to return possible value ranges.

The following inputs are used:, continued

The .index array has to be sorted, ascending.

To get an SA of objective function on the four variables 120 to 123 (included) and variable 19, the saRequest structure would look like this:

saRequest.obj.index = [19 120 121 122 123];

The result is returned through the output parameter 'sa'.

basis Vector with CPLEX starting basis. If re-solving a similar problem several times, this can be set to the 'basis' output argument of an earlier call to cplex.m. The length of this vector must be equal to the sum of the number of rows (m) and columns (n).

The first m elements contain row basis information, with the following possible values for non-ranged rows:

0 associated slack/surplus/artificial variable nonbasic at value 0.0
1 associated slack/surplus/artificial variable basic

and for ranged rows (both upper and lower bounded)

0 associated slack/surplus/artificial variable nonbasic at its lower bound
1 associated slack/surplus/artificial variable basic
2 associated slack/surplus/artificial variable nonbasic at its upper bound

The last n elements, i.e. basis(m+1:m+n) contain column basis information:

0 variable at lower bound
1 variable is basic
2 variable at upper bound
3 variable free and nonbasic

xIP Vector with MIP starting solution, if known. Missing values may be set to NaN. Length should be equal to number of columns in problem.

logcon Structure defining logical (or "indicator") constraints. This is a special type of linear constraint which is included in a mixed integer problem only if an associated binary variable is equal to 1.

Note that when associating a variable with a logical constraint, the variable in question will be forced to become a binary variable; even if it was a continuous or integer variable with bounds other than 0-1.

Each element logcon(i) describes one logical constraint:

The following inputs are used:, continued

$y- > row' * x <= rhs$ (also == and >= possible)

The following three fields (row,var,rhs) are mandatory:

The following fields are optional:

logcon(i).row: A dense or sparse row vector with the same length as the number of variables in the problem.

logcon(i).var: The index of the variable y which should control whether the constraint is "active" or not. Must be less than or equal to the number of variables in the problem.

logcon(i).rhs: The scalar value of the right hand side of the i:th logical constraint.

The following fields are optional in the description of a logical constraint:

logcon(i).sense Defines the sense of the i:th logical constraint:

0 or 'lt' : implies row*x <= rhs
1 or 'eq' : implies row*x == rhs
2 or 'gt' : implies row*x >= rhs

logcon(i).comp: Complement flag. The default value 0 (empty field or left out entirely) implies that the logical constraint is active when the associated variable is equal to 1. If setting the comp field to a nonzero value, the binary variable is complemented and the constraint will become active when the variable is zero.

logcon(i).name: A string containing a name for the i:th logical constraint. This is only used if a save file is written.

branchprio A nonnegative vector of length n. A priority order assigns a branching priority to some or all of the integer variables in a model. CPLEX performs branches on variables with a higher assigned priority number before variables with a lower priority; variables not assigned an explicit priority value by the user are treated as having a priority value of 0.

branchdir A vector with -1, 0, 1 entries of length n. -1 forces branching towards the lower end of the integer, while 1 forces branching to the higher.

cpxSettings Structure with flags controlling various CPLEX features. Currently the following fields are defined:

tune Flag to control the CPLEX Parameter Tuning feature. The following values are recognized:

0 - Disable parameter tuning (default).

The following inputs are used:, continued

1 - Enable parameter tuning and solve problem using the parameter set found by CPLEX.
2 - Enable parameter tuning and return without solving the problem. The only meaningful output data will be the cpxControl output.

During parameter tuning, the settings in cpxControl are frozen and are not changed by CPLEX. After parameter tuning has run, the complete set of modified CPLEX parameters are returned in the cpxControl output argument.

presolve Flag controlling the CPLEX Presolve feature. The following values are recognized:

0 - No special treatment of presolve (default).
1 - Invoke CPLEX Presolve on problem before optimization begins and create information about the changes made.
2 - As 1, but also returns the presolved problem, including linear constraints, objective and bounds.

Description of Outputs

The following fields are used:

x	Solution vector x with decision variable values ($n \times 1$ vector).
slack	Slack variables ($m \times 1$ vector).
v	Lagrangian multipliers (dual solution vector) ($m \times 1$ vector).
rc	Reduced costs. Lagrangian multipliers for simple bounds on x.
f_k	Objective function value at optimum.
ninf	Number of infeasibilities.
sinf	Sum of infeasibilities.
Inform	See section A.3.
basis	Basis status of constraints and variables, ($(m+n) \times 1$ vector). See inputs for more information.
lpiter	Number of simplex iterations.
glnodes	Number of nodes visited.
confstat	Structure with extended conflict status information. This output is a replica of the Prob.CPLEX.confgrps input argument with the added fields 'status' and 'istat'. confstat(k).status gives a text description of the status of conflict group k; the corresponding istat field is the numeric value also available in iconfstat(k).
iconfstat	Conflict status information. For an infeasible problem where at least one conflict group have been supplied in the confgrps input argument, this output argument contains the status of each conflict group, in the same order as given in the confgrps input. The following values are possible: -1 Excluded 0 Possible member 1 Possible LB 2 Possible UB 3 Member 4 Upper bound 5 Lower bound If confstat is empty even though Conflict Refinement has been requested, there was a problem in the refinement process.
sa	Structure with information about the requested SA, if requested. The fields:
obj	Ranges for the variables in the objective function.
rhs	Ranges for the right hand side values.

The following fields are used:, continued

xl Ranges for the lower bound values.

xu Ranges for the upper bound values.

These fields are structures themselves. All four structures have identical field names:

status Status of the SA operation. Possible values:

1 = Successful.
0 = SA not requested.
-1 = Error: begin is greater than end.
-2 = Error: The selected range (begin...end) stretches out of available variables or constraints.
-3 = Error: No SA available.

lower The lower range.

upper The upper range.

cpxControl Structure with non-default CPLEX parameters generated during CPLEX Parameter Tuning.

presolve Structure with information about the changes CPLEX Presolve made to the problem before solving. The amount of information depends on the value of the cpxSettings.presolve flag.

For cpxSettings.presolve=0, this output is empty.

For cpxSettings.presolve=1, the presolve output contains six arrays, pcstat, prstat, ocstat, orstat, status and objoffset.

status Flag telling status of presolve results:

0: Problem was not presolved or no reductions were made
1: A presolved problem exists
2: The original problem was reduced to an empty problem

The remaining fields of presolve will contain useful information only if status==1.

pcstat Contains information about variables in the original problem. For each element pcstat(i):

>= 1: variable i corresponds to variable pcstat(i) in the presolved problem
-1: variable i is fixed to its lower bound
-2: variable i is fixed to its upper bound
-3: variable i is fixed to some other value
-4: variable i is aggregated out
-5: variable i is deleted or merged for some other reason

The following fields are used:, continued

prstat Contains information about constraints (rows) in the original problem. For each element prstat(i):

>= 1: row i corresponds to row prstat(i) in the presolved problem
-1: row i is redundant
-2: row i is used for aggregation
-3: row i is deleted for some other reason

ocstat Contains information about variables in the presolved problem: For each element ocstat(i):

>= 1: variable i in the presolved problem corresponds to variable ocstat(i) in the original problem.
-1: variable i corresponds to a linear combination of some variables in the original problem

orstat Contains information about constraints (rows) in the presolved problem. For each element orstat(i):

>= 1: row i in the presolved problem corresponds to row orstat(i) in the original problem
-1: row i is created by, for example, merging two rows in the original problem

Description

The interface routine *cplex* calls CPLEX to solve LP, QP, MILP, MIQP and MIQQ problems. The matrices A and F are transformed in *cplex.m* to the CPLEX sparse matrix format.

Error checking is made on the lengths of the vectors and matrices.

Table 7: Callback functions.

Index	m-file	Description
(1)	cpxcb_PRIM	From primal simplex
(2)	cpxcb_DUAL	From dual simplex
(3)	cpxcb_PRIMCROSS	From primal crossover
(4)	cpxcb_DUALCROSS	From dual crossover
(5)	cpxcb_BARRIER	From barrier
(6)	cpxcb_PRESOLVE	From presolve
(7)	cpxcb_MIP	From mipopt
(8)	cpxcb_MIPPROBE	From probing or clique merging
(9)	cpxcb_FRACCUT	From gomory fractional cuts
(10)	cpxcb_DISJCUT	From disjunctive cuts
(11)	cpxcb_FLOWMIR	From mixed Integer Rounding cuts
(12)	cpxcb_QPBARRIER	From quadratic Barrier
(13)	cpxcb_QPSIMPLEX	From quadratic Simplex
(14)	cpxcb_INCUMBENT	From MIP incumbent

A.2 cplexTL

Purpose

The TOMLAB /CPLEX MILP, MIQP, LP and QP solver. It solves linear programming (LP), quadratic programming (QP) , mixed integer linear programming (MILP) and mixed integer quadratic programming problems (MIQP). The solver also handles problems with quadratic constraints (MIQQ). *cplexTL* solves problems of the form

$$
\begin{array}{cccccl}
\min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x & & & & \\
s/t & x_L & \leq & x & \leq & x_U \\
 & b_L & \leq & Ax & \leq & b_U \\
 & & & x^T Q^{(i)} x + a^{(i)T} x & \leq & r_U^{(i)}, \quad i = 1, \ldots, n_{qc} \\
 & & & x_i \text{ integer} & & i \in I
\end{array}
$$

where $c, x, x_L, x_U, a^{(i)} \in \mathbb{R}^n$, $F, Q^{(i)} \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m\times n}$ and $b_L, b_U \in \mathbb{R}^m$. $r_U^{(i)}$ is a scalar. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

An additional set of logical constraints can also be defined. See the help for input parameter *CPLEX.logcon*.

Calling Syntax

Prob = ProbCheck(Prob, 'cplex');
Result = cplexTL(Prob); or
Result = tomRun('cplex', Prob, 1);

Description of Inputs

Problem description structure. The following fields are used:

QP.c	Linear objective function cost coefficients, vector $n \times 1$.
QP.F	Square $n \times n$ dense or sparse matrix. Empty if non-quadratic problem.
A	Linear constraint matrix for linear constraints, dense or sparse $m \times n$ matrix.
b_L	Lower bounds on the linear constraints.
b_U	Upper bounds on the linear constraints.
x_L	Lower bounds on design parameters x. If empty assumed to be $-Inf$.
x_U	Upper bounds on design parameters x. If empty assumed to be Inf.
QP.qc	Structure array defining quadratic constraints ("qc").

Please note that CPLEX only handles single-sided bounds on qc's. An arbitrary number of qc's is set using the Prob.QP.qc structure array:

qc(1).Q = sparse(<quadratic coefficient nxn matrix>);
qc(1).a = full (<linear coefficient nx1 vector >);
qc(1).r_U = <scalar upper bound>;

And similarly for qc(2), ... , qc(n_qc).

Problem description structure. The following fields are used:, continued

The standard interpretation is $x' * Q * x + c' * x <= r_U$, but it is possible to define an alternative sense $x' * Q * x + c' * x >= r_L$ by setting $qc(i).sense$ to a nonzero value and specifying a lower bound in $qc(i).r_L$.

Observe that the Q matrix must be sparse, non-empty and positive semi-definite for all qc's. The linear coefficient vector $qc(i).a$ may be omitted or set empty, in which case all zeros are assumed.

Likewise, if a bound r_U or r_L is empty or not present, it is assumed to be 0.0. Note that this is contrary to the usual TOMLAB standard, where an empty or omitted bound is assumed to be +/- Inf. The reason is that a single-sided constraint with an infinite bound would have no meaning.

PriLevOpt Printing level in *cplex.m* file and the CPLEX C-interface.
= 0 Silent
= 1 Warnings and Errors
= 2 Summary information
= 3 More detailed information
> 10 Pause statements, and maximal printing (debug mode)

optParam Structure with optimization parameters. The following fields are used:

MaxIter Limit of iterations. If a value is given here, it is set as *cpxControl.ITLIM*. Note that a value given directly in *Prob.MIP.cpxControl.ITLIM* takes precedence.

MIP Structure holding information about mixed integer optimization. Also found here is the *cpxControl* structure in which CPLEX parameter settings can be made. The fields used are:

cpxControl Structure, where fields are set to the CPLEX control parameters that the user wants to specify values for. Please refer to Section G for more information on how to set the fields.

IntVars Defines which variables are integers, of the general type I or binary B. Variable indices should be in the range [1,...,n]. If $IntVars$ is a logical vector then all variables i where $IntVars(i) > 0$ are defined to be integers. If $IntVars$ is determined to be a vector of indices then $x(IntVars)$ are defined as integers. If the input is empty ([ ]), then no integers of type I or B are defined. The interface routine *cplex.m* checks which of the integer variables have lower bound $x_L = 0$ and upper bound $x_U = 1$, i.e. are binary 0/1 variables.

PI Integer variables of type *Partially Integer* (PI), i.e. takes an integer value up to a specified limit, and any real value above that limit. PI must be a structure array where:
$PI.var$ is a vector of variable indices in the range $[1, ..., n]$.
$PI.lim$ is a vector of limit values for each of the variables specified in PI.var, i.e. for variable i, that is the PI variable with index j in $PI.var$, then $x(i)$ takes integer values in $[x_L(i), PI.lim(j)]$ and continuous values in $[PI.lim(j), x_U(i)]$.

Problem description structure. The following fields are used:, continued

SC	A vector with indices for the integer variables of type *Semi-continuous* (SC), i.e. that takes either the value 0 or a real value in the range $[x_L(i), x_U(i)]$, assuming for some j, that $i = SC(j)$, where i is an variable number in the range $[1, ..., n]$.
SI	A vector with indices for the integer variables of type *Semi-integer* (SI), i.e. that takes either the value 0 or an integer value in the range $[x_L(i), x_U(i)]$, assuming for some j, that $i = SI(j)$, where i is an variable number in the range $[1, ..., n]$.
sos1	A structure defining the *Special Ordered Sets of Type One* (sos1). Assume there are k sets of type sos1, then $sos1(k).var$ is a vector of indices for variables of type sos1 in set k. $sos1(k).row$ is the row number for the reference row identifying the ordering information for the sos1 set, i.e. A(sos1(k).row,sos1(k).var) identifies this information. As ordering information, also the objective function coefficients c could be used. Then as row number, 0 is instead given in $sos1(k).row$.
sos2	A structure defining the *Special Ordered Sets of Type Two* (sos2). Specified exactly as sos1 sets, see $MIP.sos1$ input variable description.
basis	Basis for warm start of solution process. See Section A.1 and F.4 for more information.
xIP	Vector with MIP starting solution, if known. NaN can be used to indicate missing values. Length should be equal to number of columns in problem. Values of continuous variables are ignored.
callback	Logical vector defining which callbacks to use in CPLEX. If the i^{th} entry of the logical vector *callback* is set, the corresponding callback is defined. The callback calls the m-file specified in Table 10 below. The user may edit this file, or make a new copy, which is put in a directory that is searched before the *cplex* directory in the Matlab path.
CPLEX	Structure with solver specific parameters for logging and saving problems. The following fields are used:
LogFile	Name of file to write the CPLEX log information to. If empty, no log is written.
SaveFile	Name of a file to save the CPLEX problem object. This is useful for sending problems to ILOG for analysis. The format of the file is controlled by the *Prob.CPLEX.SaveMode*. If empty, no file is written.
SaveMode	The format of the file given in *SaveFile* is possible to choose by setting *SaveMode* to one of the following values:

Problem description structure. The following fields are used:, continued

1	SAV	Binary SAV format
2	MPS	MPS format (ASCII)
3	LP	CPLEX LP format (ASCII)
4	RMP	MPS file with generic names
5	REW	MPS file with generic names
6	RLP	LP file with generic names

Modes 4-6 are of limited interest, since the TOMLAB interface does not provide a way to change the default row names.

confgrps Conflict groups descriptor (cpxBuildConflict can be used to generate the input). Set this if conflict refinement is desired in the case that infeasibility is detected by CPLEX.
A conflict group consists of lists of indices describing which of the following entities are part of a group:

confgrps(i).lowercol Column (variable) lower bounds
confgrps(i).uppercol Column (variable) upper bounds
confgrps(i).linear Linear rows
confgrps(i).quad Quadratic constraints
confgrps(i).sos Special ordered sets
confgrps(i).indicator Indicator constraints

Additionally, the group's priority value may be assigned in
confgrps(i).priority

Please refer to the TOMLAB /CPLEX User's Guide for an example of Conflict Refinement.

conflictFile Name of a file to write the conflict refinement to. No file is written if this input parameter is empty or if no conflict refinement is done.

sa Structure telling whether and how you want CPLEX to perform a sensitivity analysis (SA). You can complete an SA on the objective function, right hand side vector, lower and upper bounds. The saRequest structure contains four sub structures:

.obj, .rhs, .xl, .xu

Each one of these contain the field:

.index

.index contain indices to variables or constraints of which to return possible value ranges.

The .index array has to be sorted, ascending.

Problem description structure. The following fields are used:, continued

To get an SA of objective function on the four variables 120 to 123 (included) and variable 19, the saRequest structure would look like this:

saRequest.obj.index = [19 120 121 122 123];

The result is returned through the output parameter 'sa'.

logcon Structure defining logical (or "indicator") constraints. This is a special type of linear constraint which is included in a mixed integer problem only if an associated binary variable is equal to 1.

Note that when associating a variable with a logical constraint, the variable in question will be forced to become a binary variable; even if it was a continuous or integer variable with bounds other than 0-1.

Each element logcon(i) describes one logical constraint:

$y- > row' * x <= rhs$ (also == and >= possible)

The following three fields (row,var,rhs) are mandatory:

The following fields are optional:

logcon(i).row: A dense or sparse row vector with the same length as the number of variables in the problem.

logcon(i).var: The index of the variable y which should control whether the constraint is "active" or not. Must be less than or equal to the number of variables in the problem.

logcon(i).rhs: The scalar value of the right hand side of the i:th logical constraint.

The following fields are optional in the description of a logical constraint:

logcon(i).sense Defines the sense of the i:th logical constraint:

0 or 'lt' : implies row*x <= rhs
1 or 'eq' : implies row*x == rhs
2 or 'gt' : implies row*x >= rhs

logcon(i).comp: Complement flag. The default value 0 (empty field or left out entirely) implies that the logical constraint is active when the associated variable is equal to 1. If setting the comp field to a nonzero value, the binary variable is complemented and the constraint will become active when the variable is zero.

Problem description structure. The following fields are used:, continued

	logcon(i).name: A string containing a name for the i:th logical constraint. This is only used if a save file is written.
BranchPrio	A nonnegative vector of length n. A priority order assigns a branching priority to some or all of the integer variables in a model. CPLEX performs branches on variables with a higher assigned priority number before variables with a lower priority; variables not assigned an explicit priority value by the user are treated as having a priority value of 0.
BranchDir	A vector with -1, 0, 1 entries of length n. -1 forces branching towards the lower end of the integer, while 1 forces branching to the higher.
Tune	Flag controlling the CPLEX Parameter Tuning feature. The following values are recognized: 0 - Disable Parameter Tuning (default). 1 - Enable Parameter Tuning and solve problem after tuning. 2 - Enable Parameter Tuning and return after tuning. No solution will be generated. The non-default CPLEX parameters found during Parameter Tuning is returned in Result.MIP.cpxControl together with any settings given in Prob.MIP.cpxControl. The settings given as input are considered as constants during the tuning process.
Presolve	Flag controlling the CPLEX Presolve feature. The following values are recognized: 0 - No special treatment of presolve (default). 1 - Invoke CPLEX Presolve on problem before optimization begins and create information about the changes made. 2 - As 1, but also returns the presolved problem, including linear constraints, objective and bounds. The results of the presolve phase (when Presolve¿0) are returned in Result.CPLEX.Presolve.

Description of Outputs

Result structure. The following fields are used:

Iter	Number of iterations, or nodes visited.
ExitFlag	0: OK. 1: Maximal number of iterations reached. 2: Unbounded feasible region. 4: No feasible point found. 5: Error of some kind.
ExitText	Number of iterations, or nodes visited.

Result. The following fields are used:, continued

Inform	Result of CPLEX run. See section A.3 for details on the ExitText and possible Inform values.
x_0	Initial starting point not known, set as empty.
QP.B	Optimal active set, basis vector, in TOMLAB QP standard. $B(i) = 1$: Include variable $x(i)$ is in basic set. $B(i) = 0$: Variable $x(i)$ is set on its lower bound. $B(i) = -1$: Variable $x(i)$ is set on its upper bound.
f_k	Function value at optimum, $f(x_k)$.
g_k	Gradient value at optimum, c or $c + F * x$.
x_k	Optimal solution vector x_k.
v_k	Lagrangian multipliers (for bounds and dual solution vector). Set as $v_k = [rc; v]$, where rc is the n-vector of reduced costs and v holds the m dual variables.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
Solver	Solver used - CPLEX .
SolverAlgorithm	Solver algorithm used.
FuncEv	Number of function evaluations. Set to *Iter*.
GradEv	Number of gradient evaluations. Set to *Iter*.
ConstrEv	Number of constraint evaluations. Set to *Iter*.
Prob	Problem structure used.
MIP.ninf	Number of infeasibilities.
MIP.sinf	Sum of infeasibilities.
MIP.slack	Slack variables ($m \times 1$ vector).
MIP.lpiter	Number of LP iterations.
MIP.glnodes	Number of nodes visited.
MIP.basis	Basis status of constraints and variables ($(m + n) \times 1$ vector) in the CPLEX format, fields xState and bState has the same information in the TOMLAB format. See Section A.1 and F.4 for more information.
MIP.cpxControl	Structure with non-default CPLEX parameters generated during CPLEX Parameter Tuning.
CPLEX.sa	Structure with information about the requested SA, if requested. The fields:
obj	Ranges for the variables in the objective function.
rhs	Ranges for the right hand side values.
xl	Ranges for the lower bound values.

Result. The following fields are used:, continued

xu Ranges for the upper bound values.

These fields are structures themselves. All four structures have identical field names:

status Status of the SA operation. Possible values:

1 = Successful.
0 = SA not requested.
-1 = Error: begin is greater than end.
-2 = Error: The selected range (begin...end) stretches out of available variables or constraints.
-3 = Error: No SA available.

lower The lower range.

upper The upper range.

CPLEX.confstat Structure with extended conflict status information. This output is a replica of the Prob.CPLEX.confgrps input argument with the added fields 'status' and 'istat'. confstat(k).status gives a text description of the status of conflict group k; the corresponding istat field is the numeric value also available in iconfstat(k).

CPLEX.iconfstat Conflict status information. For an infeasible problem where at least one conflict group have been supplied in the Prob.CPLEX.confgrps input argument, this output argument contains the status of each conflict group, in the same order as given in the confgrps input.

The following values are possible:

-1 Excluded
0 Possible member
1 Possible LB
2 Possible UB
3 Member
4 Upper bound
5 Lower bound

If confstat is empty even though Conflict Refinement has been requested, there was a problem in the refinement process.

CPLEX.Presolve Structure with information about the changes CPLEX Presolve made to the problem before solving. The amount of information depends on the value of the Prob.CPLEX.Presolve flag.

For Prob.CPLEX.Presolve=0, this output is empty.

Result. The following fields are used:, continued

For Prob.CPLEX.Presolve=1, the presolve output contains six arrays, pcstat, prstat, ocstat, orstat, status and objoffset.

status Flag telling status of presolve results:

0: Problem was not presolved or no reductions were made
1: A presolved problem exists
2: The original problem was reduced to an empty problem

The remaining fields of Result.CPLEX.Presolve will contain useful information only if status==1.

pcstat Contains information about variables in the original problem. For each element pcstat(i):

>= 1: variable i corresponds to variable pcstat(i) in the presolved problem
-1: variable i is fixed to its lower bound
-2: variable i is fixed to its upper bound
-3: variable i is fixed to some other value
-4: variable i is aggregated out
-5: variable i is deleted or merged for some other reason

prstat Contains information about constraints (rows) in the original problem. For each element prstat(i):

>= 1: row i corresponds to row prstat(i) in the presolved problem
-1: row i is redundant
-2: row i is used for aggregation
-3: row i is deleted for some other reason

ocstat Contains information about variables in the presolved problem: For each element ocstat(i):

>= 1: variable i in the presolved problem corresponds to variable ocstat(i) in the original problem.
-1: variable i corresponds to a linear combination of some variables in the original problem

orstat Contains information about constraints (rows) in the presolved problem. For each element orstat(i):

>= 1: row i in the presolved problem corresponds to row orstat(i) in the original problem
-1: row i is created by, for example, merging two rows in the original problem

Global Parameters Used

cpxCBInfo
cpxRetVec

Description
The TOMLAB CPLEX MILP, MIQP, QP and LP interface calls the interface routine *cplex.m*. Values $> 10^{10}$ and Inf values are set to 10^{10}, and the opposite for negative numbers. An empty objective coefficient c-vector is set to the zero-vector.

Examples
See *mip_prob*

M-files Used
cplex.m

See Also
mipSolve

Table 10: Callback functions.

Index	m-file	Description
(1)	cpxcb_PRIM	From primal simplex
(2)	cpxcb_DUAL	From dual simplex
(3)	cpxcb_PRIMCROSS	From primal crossover
(4)	cpxcb_DUALCROSS	From dual crossover
(5)	cpxcb_BARRIER	From barrier
(6)	cpxcb_PRESOLVE	From presolve
(7)	cpxcb_MIP	From mipopt
(8)	cpxcb_MIPPROBE	From probing or clique merging
(9)	cpxcb_FRACCUT	From gomory fractional cuts
(10)	cpxcb_DISJCUT	From disjunctive cuts
(11)	cpxcb_FLOWMIR	From mixed Integer Rounding cuts
(12)	cpxcb_QPBARRIER	From quadratic Barrier
(13)	cpxcb_QPSIMPLEX	From quadratic Simplex
(14)	cpxcb_INCUMBENT	From MIP incumbent

A.3 cplexStatus

Purpose

cplexStatus analyzes the CPLEX output Inform code and returns the CPLEX solution status message in ExitText and the TOMLAB exit flag in ExitFlag.

Calling Syntax

$[ExitText, ExitFlag] = cplexStatus(Inform)$

Description of Inputs

The following inputs are used:

Inform Result of CPLEX run. (S=Simplex, B=Barrier, MIP=Mixed-Integer)

1	(S,B) Optimal solution is available
2	(S,B) Model has an unbounded ray
3	(S,B) Model has been proved infeasible
4	(S,B) Model has been proved either infeasible or unbounded
5	(S,B) Optimal solution is available, but with infeasibilities after unscaling
6	(S,B) Solution is available, but not proved optimal, due to numeric difficulties
10	(S,B) Stopped due to limit on number of iterations
11	(S,B) Stopped due to a time limit
12	(S,B) Stopped due to an objective limit
13	(S,B) Stopped due to a request from the user
14	(S,B) Feasible relaxed sum found (FEASOPTMODE)
15	(S,B) Optimal relaxed sum found (FEASOPTMODE)
16	(S,B) Feasible relaxed infeasibility found (FEASOPTMODE)
17	(S,B) Optimal relaxed infeasibility found (FEASOPTMODE)
18	(S,B) Feasible relaxed quad sum found (FEASOPTMODE)
19	(S,B) Optimal relaxed quad sum found (FEASOPTMODE)
20	(B) Model has an unbounded optimal face
21	(B) Stopped due to a limit on the primal objective
22	(B) Stopped due to a limit on the dual objective
30	The model appears to be feasible; no conflict is available
31	The conflict refiner found a minimal conflict
32	A conflict is available, but it is not minimal
33	The conflict refiner terminated because of a time limit
34	The conflict refiner terminated because of an iteration limit
35	The conflict refiner terminated because of a node limit
36	The conflict refiner terminated because of an objective limit
37	The conflict refiner terminated because of a memory limit
38	The conflict refiner terminated because a user terminated the application

The following inputs are used:, continued

101	Optimal integer solution found
102	Optimal sol. within epgap or epagap tolerance found
103	Solution is integer infeasible
104	The limit on mixed integer solutions has been reached
105	Node limit exceeded, integer solution exists
106	Node limit exceeded, no integer solution
107	Time limit exceeded, integer solution exists
108	Time limit exceeded, no integer solution
109	Terminated because of an error, but integer solution exists
110	Terminated because of an error, no integer solution
111	Limit on tree memory has been reached, but an integer solution exists
112	Limit on tree memory has been reached; no integer solution
113	Stopped, but an integer solution exists
114	Stopped; no integer solution
115	Problem is optimal with unscaled infeasibilities
116	Out of memory, no tree available, integer solution exists
117	Out of memory, no tree available, no integer solution
118	Model has an unbounded ray
119	Model has been proved either infeasible or unbounded

The following inputs are used:, continued

120	(MIP) Feasible relaxed sum found (FEASOPTMODE)
121	(MIP) Optimal relaxed sum found (FEASOPTMODE)
122	(MIP) Feasible relaxed infeasibility found (FEASOPTMODE)
123	(MIP) Optimal relaxed infeasibility found (FEASOPTMODE)
124	(MIP) Feasible relaxed quad sum found (FEASOPTMODE)
125	(MIP) Optimal relaxed quad sum found (FEASOPTMODE)
126	(MIP) Relaxation aborted due to limit (FEASOPTMODE)
127	(MIP) Feasible solution found (FEASOPTMODE)
1001	Insufficient memory available
1014	CPLEX parameter is too small
1015	CPLEX parameter is too big
1100	Lower and upper bounds contradictory
1101	The loaded problem contains blatant infeasibilities or unboundedness
1106	The user halted preprocessing by means of a callback
1117	The loaded problem contains blatant infeasibilities
1118	The loaded problem contains blatant unboundedness
1123	Time limit exceeded during presolve.
1225	Numeric entry is not a double precision number (NAN)
1233	Data checking detected a number too large
1256	CPLEX cannot factor a singular basis
1261	No basic solution exists (use crossover)
1262	No basis exists (use crossover)
1719	No conflict is available
1805	MIP dynamic search incompatible with control callbacks.
3413	Tree memory limit exceeded
5002	Non-positive semidefinite matrix in quadratic problem
5012	Non-symmetric matrix in quadratic problem
32201	A licensing error has occurred
32024	Licensing problem: Optimization algorithm not licensed

Description of Outputs

The following fields are used:

ExitText	Text interpretation of CPLEX result.
ExitFlag	TOMLAB standard exit flag.

A.4 cpxRetVec

Purpose

cpxRetVec is a global variable that CPLEX can write more detailed solution information to. For all fields, the default value is NaN and appears whenever the element in question is not available/not applicable for the problem type.

Note about integer and double quality values:
Some quality values are present in both the integer and double lists. This is because these quality identifiers have a meaning both as double and integer qualities. Example: The double interpretation is normally the largest (absolute) value of the variables, while the integer interpretation is the first index where that value occurs.

Calling Syntax

global cpxRetVec
% Call cplex by tomRun or directly

CPLEX functions or parameter names in cpxRetVec

The following outputs are created:

Index Result of CPLEX run. (S=Simplex, B=Barrier, MIP=Mixed-Integer)

20	(S,B) Solver method (1 = Primal, 2 = Dual, 4 = Barrier)
1	Solution objective value
2	(MIP) The currently best known bound on the optimal solution value of a MIP problem. When a problem has been solved to optimality, this value matches the optimal solution value. Otherwise, this value is computed for a minimization (maximization) problem as the minimum (maximum) objective function value of all remaining unexplored nodes.
3	(MIP) The MIP cutoff value being used during mixed integer optimization. The cutoff is updated with the objective function value, each time an integer solution is found during branch and cut.
4	(MIP) The node number of the best known integer solution.
7	(MIP) The cumulative number of simplex iterations used to solve a mixed integer problem.
8	(MIP) The number of nodes used to solve a mixed integer problem.
9	(MIP) The number of unexplored nodes left in the branch and cut tree.

The following outputs are created:, continued

5	(S) The total number of simplex iterations to solve an LP problem, or the number of crossover iterations in the case that the barrier optimizer is used.
10	(S,MIP) The number of dual super-basic variables in the current solution.
15	(S,MIP) The number of primal super-basic variables in the current solution.
6	(B) The total number of Barrier iterations to solve an LP problem.
16	(B) The number of dual exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.
17	(B) The number of dual push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.
18	(B) The number of primal exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.
19	(B) The number of primal push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.
12	(S) The number of Phase I iterations to solve a problem using the primal or dual simplex method.

Double-type quality values:

21	The maximum primal infeasibility or, equivalently, the maximum bound violation including slacks for the unscaled problem.
22	The maximum primal infeasibility or, equivalently, the maximum bound violation including slacks for the scaled problem.
23	The sum of primal infeasibilities or, equivalently, the sum of bound violations for the unscaled problem.
24	The sum of primal infeasibilities or, equivalently, the sum of bound violations for the scaled problem.
25	(S,B) The maximum of dual infeasibility or, equivalently, the maximum reduced-cost infeasibility for the unscaled problem.
26	(S,B) The maximum of dual infeasibility or, equivalently, the maximum reduced-cost infeasibility for the scaled problem.
27	(S,B) The sum of dual infeasibilities or, equivalently, the sum of reduced-cost bound violations for the unscaled problem .
28	(S,B) The sum of dual infeasibilities or, equivalently, the sum of reduced-cost bound violations for the scaled problem .
29	(MIP) The maximum of integer infeasibility for the unscaled problem.
30	(MIP) The sum of integer infeasibilities for the unscaled problem.

The following outputs are created:, continued

31	The maximum of the vector $\lvert Ax - b\rvert$ for the unscaled problem.
32	The maximum of the vector $\lvert Ax - b\rvert$ for the scaled problem.
33	The sum of the elements of vector $\lvert Ax - b\rvert$ for the unscaled problem.
34	The sum of the elements of vector $\lvert Ax - b\rvert$ for the unscaled problem.
35	(S,B) The maximum dual residual value. For a simplex solution, this is the maximum of the vector —c-B'pi—, and for a barrier solution, it is the maximum of the vector —A'pi+rc-c— for the unscaled problem.
36	(S,B) The maximum dual residual value for the scaled problem.
37	(S,B) The sum of the absolute values of the dual residual vector for the unscaled problem.
38	(S,B) The sum of the absolute values of the dual residual vector for the scaled problem.
39	(B) The maximum violation of the complementary slackness conditions for the unscaled problem.
41	(B) The sum of the violations of the complementary slackness conditions for the unscaled problem.
43	The maximum absolute value in the primal solution vector for the unscaled problem.
44	The maximum absolute value in the primal solution vector for the scaled problem.
45	(S,B) The maximum absolute value in the dual solution vector for the unscaled problem.
46	(S,B) The maximum absolute value in the dual solution vector for the scaled problem.
47	The maximum absolute slack value for the unscaled problem.
48	The maximum absolute slack value for the scaled problem.
49	(S,B) The maximum absolute reduced cost value for the unscaled problem.
50	(S,B) The maximum absolute reduced cost value for the scaled problem.
51	The sum of the absolute values in the primal solution vector for the unscaled problem.
52	The sum of the absolute values in the primal solution vector for the scaled problem.
53	(S,B) The sum of the absolute values in the dual solution vector for the unscaled problem.
54	(S,B) The sum of the absolute values in the dual solution vector for the scaled problem.
55	The sum of the absolute slack values for the unscaled problem.
56	The sum of the absolute slack values for the scaled problem.
57	(S,B) The sum of the absolute reduced cost values for the unscaled problem.
58	(S,B) The sum of the absolute reduced cost values for the unscaled problem.
59	(S) The estimated condition number of the scaled basis matrix.
60	(B) The objective value gap between the primal and dual objective value solution.
61	(B) The objective value relative to the dual barrier solution.
62	(B) The objective value relative to the primal barrier solution.

Integer-type quality values:

The following outputs are created:, continued

63	The lowest index of a column or row where the maximum primal infeasibility occurs for the unscaled problem.
64	The lowest index of a column or row where the maximum primal infeasibility occurs for the scaled problem.
65	(S,B) The lowest index where the maximum dual infeasibility occurs for the unscaled problem.
66	(S,B) the lowest index where the maximum dual infeasibility occurs for the scaled problem.
67	(MIP) The lowest index where the maximum integer infeasibility occurs for the unscaled problem.
68	(MIP) The lowest index where the maximum primal residual occurs for the unscaled problem.
69	(MIP) The lowest index where the maximum primal residual occurs for the scaled problem.
70	(S,B) The lowest index where the maximum dual residual occurs for the unscaled problem .
71	(S,B) The lowest index where the maximum dual residual occurs for the scaled problem .
72	(B) The lowest index of a row or column with the largest violation of the complementary slackness conditions.
73	The lowest index where the maximum x value occurs for the unscaled problem.
74	The lowest index where the maximum x value occurs for the scaled problem.
75	(S,B) The lowest index where the maximum pi value occurs for the unscaled problem.
76	(S,B) The lowest index where the maximum pi value occurs for the scaled problem.
77	The lowest index where the maximum slack value occurs for the unscaled problem.
78	The lowest index where the maximum slack value occurs for the scaled problem.
79	(S,B) The lowest index where the maximum reduced cost value occurs for the unscaled problem.
80	(S,B) The lowest index where the maximum reduced cost value occurs for the scaled problem.
81	(MIP) The relative objective gap for a MIP optimization.

B The Matlab Interface Routines - Utility Routines

B.1 cpx2mat

Purpose

cpx2mat reads an (X)MPS file and more. The file is converted to matrices and vectors made available in MATLAB. MPS and extended MPS for LP, MILP, QP and MIQP are the supported file types, however it is possible to supply a wide range of file types.

Calling Syntax

$[F, c, A, b\_L, b\_U, x\_L, x\_U, IntVars]$ = cpx2mat(Name,PriLev);

Description of Input

Name	Name of the MPS file with extension. cpx2mat can recognize many different file extensions, e.g.: .mps, .lp, .mat, .qps.
PriLev	Print level of cpx2mat. Set to 0 to have it silent, 1 to print warnings, and 2 to print debug information.

Description of Output

F	The quadratic term matrix. Empty for non-QP problems.
c	The linear term vector.
A	The constraint matrix.
b_L	The lower bounds of the constraints.
b_U	The upper bounds of the constraints.
x_L	The lower box bounds of x.
x_U	The upper box bounds of x.
IntVars	Logical vector describing what variables that are integer or binary variables. Empty if the problem is not a mixed integer problem.

B.2 abc2gap

Purpose

Converting a general assignment problem (GAP) to a standard form suitable for a MIP solver.

The GAP problem is formulated as

$$\begin{array}{ll} \min\limits_{x_{ij}} & f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} * x_{ij} \\ \\ s/t & \sum_{j=1}^{n} x_{ij} = 1 \quad , i = 1, ..., m \\ & \sum_{i=1}^{m} a_{ij} * x_{ij} \leq b_j \quad , j = 1, .., n \\ & x \in B^{m \times n}, B = \{0, 1\}. \end{array}$$

Calling Syntax

[c, x_L, x_U, b_L, b_U, a, sos1] = abc2gap(A, b, C, SOS1);

Description of Input

A	A $m \times n$ constraint matrix for GAP constraints.
b	A $m \times 1$ right hand side vector.
C	A $m \times n$ cost matrix for GAP constraints.
SOS1	Logical variable, default false. If true, generate output for sos1 handling with CPLEX. Otherwise generate output giving an equivalent formulation with standard integer variables.

Description of Output

c	Linear objective function cost coefficients, vector $m * n \times 1$.
x_L	Lower bounds on design parameters x.
x_U	Upper bounds on design parameters x.
b_L	Lower bounds on the $m + n$ linear constraints.
b_U	Upper bounds on the linear constraints.
a	Sparse $m + n \times m * n$ matrix for linear constraints.
sos1	If input variable $SOS1$ is true, structure with sos1 variable information in the form suitable for the Matlab CPLEX interface routine *cplex.m*, otherwise empty.

Description

Converting a general assignment problem (GAP) to standard form suitable for a mixed-integer programming solver.

Either binary or sos1 variables are used.

C The Matlab Interface Routines - Test Routines

C.1 cpxaircrew

Purpose
Test of an air-crew schedule generation problem.

Calling Syntax
cpxaircrew

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description
Test of an air-crew schedule generation problem. Based on D.M.Ryan, Airline Industry, Encyclopedia of Operations Research and Management Science. Two subfunctions are used (defined at the end of the *cpxaircrew.m* file): The function *generateToDs* create ToDs, i.e. Tours of Duty. The function *sectordata* generates some test data.

M-files Used
abc2gap.m, *cplex.m*

C.2 cpxbiptest

Purpose

Test of TOMLAB /CPLEX level 1 interface solving three larger binary integer linear optimization problems calling the CPLEX solver.

Calling Syntax

function cpxbiptest(Cut, PreSolve, cpxControl)

Description of Input

Cut	Value of the cut strategy control parameter, default $Cut = -1$. $Cut = -1$, auto select of $Cut = 1$ or $Cut = 2$. $Cut = 0$, no cuts. $Cut = 1$, conservative cut strategy. $Cut = 2$, aggressive cut strategy The cut strategy choice is implemented by setting the following parameters (omitting prefix "`CPX_PARAM_`"): `CLIQUES, COVERS, DISCJUTS, FLOWCOVERS, FLOWPATHS, FRACCUTS, GUBCOVERS, IMPLED, MIRCUTS` in the *cpxControl* structure.
PreSolve	Value of the PRESOLVE control parameter, default $PreSolve = 1$. $PreSolve = 0$: no presolve. $PreSolve = 1$, do presolve.
cpxControl	The initial CPLEX parameter structure. Here the user may set additional control parameters. Default empty.

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description

Test of three larger binary integer linear optimization problems calling the CPLEX solver. The test problem 1 and 2 have 1956 variables, 23 equalities and four inequalities with both lower and upper bounds set.

Test problem 1, in $bilp1.mat$, is randomly generated. It has several minima with optimal zero value. CPLEX runs faster if avoiding the use of a cut strategy, and skipping presolve. Test problem 2, in $bilp2.mat$, has a unique minimum. Runs faster if avoiding the use of presolve.

Test problem 3, in $bilp1211.mat$, has 1656 variables, 23 equalities and four inequalities with lower and upper bounds set. Runs very slow without the use of cuts. A call $cpxbiptest(0, 0)$ gives the fastest execution for the first two problems, but will be extremely slow for the third problem.

Timings are made with the Matlab functions *tic* and *toc*.

M-files Used

cplex.m, *cpxPrint.m*

C.3 cpxiptest

Purpose

Test of the TOMLAB /CPLEX level 1 interface solving three larger integer linear optimization problems calling the CPLEX solver.

Calling Syntax

function cpxiptest(Cut, PreSolve, cpxControl)

Description of Input

Cut	Value of the cut strategy parameters, default $Cut = -1$. $Cut = -1$, auto select of $Cut = 1$ or $Cut = 2$. $Cut = 0$, no cuts. $Cut = 1$, conservative cut strategy. $Cut = 2$, aggressive cut strategy See *cpxbiptest*, page 42.
PreSolve	Value of the PRESOLVE control parameter, default $PreSolve = 1$. $PreSolve = 0$, no presolve. $PreSolve = 1$, do presolve.
cpxControl	The initial cpxControl structure. Here the user may set additional control parameter. Default empty.

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description

Test of three larger integer linear optimization problems calling the CPLEX solver. The test problems have 61 variables and 138 linear inequalities. 32 of the 138 inequalities are just zero rows in the matrix A. The three problems are stored in $ilp061.mat$, $ilp062.mat$ and $ilp063.mat$.

Code is included to remove the 32 zero rows, and compute better upper bounds using the positivity of the matrix elements, right hand side and the variables. But this does not influence the timing much, the CPLEX presolve will do all these problem changes.

Timings are made with the Matlab functions *tic* and *toc*.

M-files Used

cplex, *xprinti*, *cpxPrint*

C.4 cpxtomtest1

Purpose
Test of using TOMLAB to call CPLEX for problems defined in the TOMLAB IF format.

Calling Syntax
cpxtomtest1

Description
Test of using TOMLAB to call CPLEX for problems defined in the TOMLAB IF format. The examples show the solution of LP, QP and MILP problems.

M-files Used
tomRun.

See Also
cplexTL.

C.5 cpxtomtest2

Purpose
Test of using TOMLAB to call CPLEX for problems defined in the TOMLAB TQ format.

Calling Syntax
cpxtomtest2

Description
Test of using TOMLAB to call CPLEX for problems defined in the TOMLAB TQ format. The routine *mipAssign* is used to define the problem. A simple problem is solved with CPLEX both as an LP problem and as a MILP problem. The problem is solved both with and without explicitly defining the slack variables.

M-files Used
mipAssign, *tomRun* and *PrintResult*.

See Also
cplexTL and *cplex*.

C.6 cpxKnaps

Purpose

CPLEX Matlab Level 1 interface Knapsack test routine

Calling Syntax

cpxKnaps(P, Cut)

Description of Input

P	Problem number 1-3. Default 1.
Cut	Cut strategy. 0 = no cuts, 1 = cuts, 2 = aggressive cuts. Default 0.

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description

The CPLEX Matlab level 1 interface knapsack test routine runs three different test problems. It is possible to change cut strategy and use heuristics defined in callbacks.

Currently defined knapsack problems:

Problem	Name	Knapsacks	Variables
1	Weingartner 1	2	28
2	Hansen, Plateau 1	4	28
3	PB 4	2	29

M-files Used

cplex.m

C.7 cpxKnapsTL

Purpose

CPLEX Matlab Level 2 interface Knapsack test routine

Calling Syntax

cpxKnapsTL(P, Cut)

Description of Input

P	Problem number 1-3. Default 1.
Cut	Cut strategy. 0 = no cuts, 1 = cuts, 2 = aggressive cuts. Default 0.

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description

The CPLEX Matlab level 2 interface knapsack test routine runs three different test problems. It is possible to change cut strategy.

Currently defined knapsack problems:

Problem	Name	Knapsacks	Variables
1	Weingartner 1	2	28
2	Hansen, Plateau 1	4	28
3	PB 4	2	29

M-files Used

cplex.m

C.8 cpxSolutionPool

Purpose
Test of TOMLAB /CPLEX solution pool capabilities.

Calling Syntax
[x,f] = cpxSolutionPool;

Description
Exemplifies the use of the solution pool features in TOMLAB /CPLEX and the parameters associated with this, SOLNPOOLCAPACITY, SOLNPOOLGAP, SOLNPOOLINTENSITY and SOLNPOOLREPLACE.

C.9 cpxSolverTuning

Purpose
Test of TOMLAB /CPLEX solver tuning capabilities.

Calling Syntax
[cpxControl1, cpxControl2] = cpxSolverTuning;

Description
Exemplifies the use of the solver tuning features in TOMLAB /CPLEX and the parameters associated with this, TUNINGDISPLAY, TUNINGREPEAT, TUNINGTILIM and TUNINGMEASURE.

C.10 cpxTest1

Purpose
Test routine 1, calls CPLEX Matlab level 1 interface to solve a GAP problem.

Calling Syntax
x = cpxTest1

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description
Running a generalized assignment problem (GAP) from Wolsey [1, 9.8.16, pp165]. In this test the linear sos1 constraints are defined explicitly.

Given the matrices A (constraints) and C (costs), *cxpTest1* is using the utility *abc2gap* to reformulate the problem into the standard form suitable for CPLEX.

The number of iterations are increased, no presolve is used, and an aggressive cut strategy.

M-files Used
abc2gap.m, *cplex.m*

C.11 cpxTest2

Purpose
Test routine 2, calls CPLEX Matlab level 1 interface to solve a GAP problem.

Calling Syntax
x = cpxTest2

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description
Running a generalized assignment problem (GAP) from Wolsey [1, 9.8.16, pp165]. In this test sos1 variables are used.

Given the matrices A (constraints) and C (costs), *cpxTest2* is using the utility *abc2gap* to reformulate the problem into the standard form suitable for CPLEX.

The number of iterations are increased, no presolve is used, and an aggressive cut strategy is applied.

M-files Used
abc2gap.m, *cplex.m*

See Also
cpxTest3.m

C.12 cpxTest3

Purpose

Test routine 3, calls CPLEX Matlab level 1 interface to solve a GAP problem.

Calling Syntax

x = cpxTest3

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description

Running a generalized assignment problem (GAP) from Wolsey [1, 9.6, pp159]. In this test the linear sos1 constraints are defined explicitly.

Given the matrices A (constraints) and C (costs), *cpxTest1* is using the utility *abc2gap* to reformulate the problem into the standard form suitable for CPLEX.

The number of iterations are increased, no presolve is used, and no cut strategy is used.

M-files Used

abc2gap.m, *cplex.m*

See Also

cpxTest2

C.13 cpxTestQP1

Purpose
Simple test of calling CPLEX Matlab level 1 interface to solve a QP problem.

Calling Syntax
x = cpxTestQP1(MIP,DEFPARAM)

Description of Input

MIP	If $MIP = 1$, run as a MIQP problem, trying to make the third variable integer valued. Otherwise run as a pure QP problem. Default $MIP = 0$.
DEFPARAM	If 1, use default CPLEX parameters for fastest execution. If 0, disable cuts and presolve for slower execution.

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description
Simple test of calling CPLEX Matlab level 1 interface to solve a QP or MIQP problem. The problem is

$$
\begin{array}{ll}
\min\limits_{x} & f(x) = x_1^2 + x_2^2 + x_3^2 \\
 & \\
\text{subject to} & \begin{array}{lcl} x_1 + 2x_2 - x_3 & = & 4 \\ x_1 - x_2 + x_3 & = & -2 \end{array} \\
 & \\
 & -10 \leq x_i \leq 10, \quad i = 1, 2, 3 \\
 & x_3 \text{ integer if } MIP \neq 0
\end{array}
$$

M-files Used
cplex.m

C.14 cpxTestQP2

Purpose

Simple test of calling CPLEX Matlab level 1 interface to solve a mixed integer quadratic (MIQP) problem.

Calling Syntax

x = cpxTestQP2(MIP)

Description of Input

MIP	If $MIP = 1$ (default), run as a MIQP problem, trying to make the first variable integer valued, otherwise run as a pure QP problem.
DEFPARAM	If 1, use default parameters, presolve, cuts, dual simplex for fastest execution. If 0, do not use presolve or cuts. Choose Primal Simplex for slower execution. Default: 0

Global Parameters Used

MAX_x	Maximal number of x elements printed in output statements. Default 20.
MAX_c	Maximal number of constraint elements printed in output statements. Default 20.

Description

Simple test of MIQP problem running CPLEX . The problem is defined as

$$\begin{array}{ll} \min\limits_{x} & f(x) = 2x_1^2 - 2x_1x_2 + 2x_2^2 - 6x_1 \\ & \\ s/t & \begin{array}{rcccl} 0 & \leq & x_1, x_2 & \leq & \infty \\ & & x_1 + x_2 & \leq & 1.9 \end{array} \\ & \\ & x_1 \text{ integer if } MIP \neq 0. \end{array}$$

M-files Used

cplex.m

C.15 cpxTestConflict

Purpose
Demonstration of the TOMLAB /CPLEX Conflict Refinement feature.

Calling Syntax
x = cpxTestConflict()

Description
Define the linear sos1 constraints explicitly. Modify a bound to produce an infeasibility and invoke CPLEX again with Conflict Refinement enabled.

D The Matlab Interface Routines - Callback Routines

D.1 cpxcb_BARRIER

CPLEX Barrier callback.

Called from TOMLAB /CPLEX when solving linear problems using the barrier algorithm.

This callback is enabled by setting *callback*(5) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(5) = 1 if using tomRun('cplex',...).

cpxcb_BARRIER is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
  i  cpxCBInfo(i)    - Value
 ---------------------------------------------------------------
  1  PRIMAL_OBJ      - primal objective value
  2  DUAL_OBJ        - dual objective value
  3  PRIMAL_INFMEAS  - measure of primal infeasibility
  4  DUAL_INFMEAS    - measure of dual infeasibility
  5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
  6  DUAL_FEAS       - 1 if dual feasible, 0 if not
  7  ITCOUNT         - iteration count
  8  CROSSOVER_PPUSH - primal push crossover itn. count
  9  CROSSOVER_PEXCH - primal exchange crossover itn. count
 10  CROSSOVER_DPUSH - dual push crossover itn. count
 11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_BARRIER, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.2 cpxcb_DISJCUT

CPLEX Disjunctive cut callback.

Called from TOMLAB /CPLEX during disjunctive cuts processing.

This callback is enabled by setting *callback*(10) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(10) = 1 if using tomRun('cplex',...).

cpxcb_DISJCUT is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo variable:

```
  i  cpxCBInfo(i)    - Value
```

```
-------------------------------------------------------------
 1 BEST_INTEGER     - obj. value of best integer solution
 2 BEST_REMAINING   - obj. value of best remaining node
 3 NODE_COUNT       - total number of nodes solved
 4 NODES_LEFT       - number of remaining nodes
 5 MIP_ITERATIONS   - total number of MIP iterations
 6 MIP_FEAS         - returns 1 if feasible solution exists; otherwise, 0
 7 CUTOFF           - updated cutoff value
 8 CLIQUE_COUNT     - number of clique cuts added
 9 COVER_COUNT      - number of cover cuts added
10 DISJCUT_COUNT    - number of disjunctive cuts added
11 FLOWCOVER_COUNT  - number of flow cover cuts added
12 FLOWPATH_COUNT   - number of flow path cuts added
13 FRACCUT_COUNT    - number of Gomory fractional cuts added
14 GUBCOVER_COUNT   - number of GUB cover cuts added
15 IMPLBD_COUNT     - number of implied bound cuts added
16 MIRCUT_COUNT     - number of mixed integer rounding cuts added
17 PROBE_PHASE      - current phase of probing (0-3)
18 PROBE_PROGRESS   - fraction of probing phase completed (0.0-1.0)
19 FRACCUT_PROGRESS - fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
20 DISJCUT_PROGRESS - fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
21 FLOWMIR_PROGRESS - fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
22 MY_THREAD_NUM    - identifier of the parallel thread making this call (always 0)
23 USER_THREADS     - total number of parallel threads currently running (always 1)
```

By returning a nonzero value from cpxcb_DISJCUT, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.3 cpxcb_DUAL

CPLEX Dual simplex callback.

Called from TOMLAB /CPLEX when solving linear problems using the dual simplex algorithm.

This callback is enabled by setting *callback*(2) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(2) = 1 if using tomRun('cplex',...).

cpxcb_DUAL is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
i  cpxCBInfo(i)    - Value
-------------------------------------------------------------
1  PRIMAL_OBJ      - primal objective value
2  DUAL_OBJ        - dual objective value
3  PRIMAL_INFMEAS  - measure of primal infeasibility
```

```
 4  DUAL_INFMEAS    - measure of dual infeasibility
 5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
 6  DUAL_FEAS       - 1 if dual feasible, 0 if not
 7  ITCOUNT         - iteration count
 8  CROSSOVER_PPUSH - primal push crossover itn. count
 9  CROSSOVER_PEXCH - primal exchange crossover itn. count
10  CROSSOVER_DPUSH - dual push crossover itn. count
11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_DUAL, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.4 cpxcb_DUALCROSS

CPLEX Dual crossover callback.

Called from TOMLAB /CPLEX during the dual crossover algorithm.

This callback is enabled by setting *callback*(4) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(4) = 1 if using tomRun('cplex',...).

cpxcb_DUALCROSS is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
 i  cpxCBInfo(i)    - Value
-------------------------------------------------------------
 1  PRIMAL_OBJ      - primal objective value
 2  DUAL_OBJ        - dual objective value
 3  PRIMAL_INFMEAS  - measure of primal infeasibility
 4  DUAL_INFMEAS    - measure of dual infeasibility
 5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
 6  DUAL_FEAS       - 1 if dual feasible, 0 if not
 7  ITCOUNT         - iteration count
 8  CROSSOVER_PPUSH - primal push crossover itn. count
 9  CROSSOVER_PEXCH - primal exchange crossover itn. count
10  CROSSOVER_DPUSH - dual push crossover itn. count
11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_DUALCROSS, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.5 cpxcb_FLOWMIR

CPLEX Mixed integer rounding cut callback.

Called from TOMLAB /CPLEX during Mixed integer rounding cuts processing.

This callback is enabled by setting *callback*(11) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(11) = 1 if using tomRun('cplex',...).

cpxcb_FLOWMIR is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo variable:

```
  i  cpxCBInfo(i)    - Value
 -------------------------------------------------------------
  1 BEST_INTEGER     - obj. value of best integer solution
  2 BEST_REMAINING   - obj. value of best remaining node
  3 NODE_COUNT       - total number of nodes solved
  4 NODES_LEFT       - number of remaining nodes
  5 MIP_ITERATIONS   - total number of MIP iterations
  6 MIP_FEAS         - returns 1 if feasible solution exists; otherwise, 0
  7 CUTOFF           - updated cutoff value
  8 CLIQUE_COUNT     - number of clique cuts added
  9 COVER_COUNT      - number of cover cuts added
 10 DISJCUT_COUNT    - number of disjunctive cuts added
 11 FLOWCOVER_COUNT  - number of flow cover cuts added
 12 FLOWPATH_COUNT   - number of flow path cuts added
 13 FRACCUT_COUNT    - number of Gomory fractional cuts added
 14 GUBCOVER_COUNT   - number of GUB cover cuts added
 15 IMPLBD_COUNT     - number of implied bound cuts added
 16 MIRCUT_COUNT     - number of mixed integer rounding cuts added
 17 PROBE_PHASE      - current phase of probing (0-3)
 18 PROBE_PROGRESS   - fraction of probing phase completed (0.0-1.0)
 19 FRACCUT_PROGRESS - fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
 20 DISJCUT_PROGRESS - fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
 21 FLOWMIR_PROGRESS - fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
 22 MY_THREAD_NUM    - identifier of the parallel thread making this call (always 0)
 23 USER_THREADS     - total number of parallel threads currently running (always 1)
```

By returning a nonzero value from cpxcb_FLOWMIR, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.6 cpxcb_FRACCUT

CPLEX Gomory fractional cut callback.

Called from TOMLAB /CPLEX during Gomory fractional cuts processing

This callback is enabled by setting *callback*(9) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(9) = 1 if using tomRun('cplex',...).

cpxcb_FRACCUT is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo variable:

```
 i  cpxCBInfo(i)    - Value
-------------------------------------------------------------
 1 BEST_INTEGER     - obj. value of best integer solution
 2 BEST_REMAINING   - obj. value of best remaining node
 3 NODE_COUNT       - total number of nodes solved
 4 NODES_LEFT       - number of remaining nodes
 5 MIP_ITERATIONS   - total number of MIP iterations
 6 MIP_FEAS         - returns 1 if feasible solution exists; otherwise, 0
 7 CUTOFF           - updated cutoff value
 8 CLIQUE_COUNT     - number of clique cuts added
 9 COVER_COUNT      - number of cover cuts added
10 DISJCUT_COUNT    - number of disjunctive cuts added
11 FLOWCOVER_COUNT  - number of flow cover cuts added
12 FLOWPATH_COUNT   - number of flow path cuts added
13 FRACCUT_COUNT    - number of Gomory fractional cuts added
14 GUBCOVER_COUNT   - number of GUB cover cuts added
15 IMPLBD_COUNT     - number of implied bound cuts added
16 MIRCUT_COUNT     - number of mixed integer rounding cuts added
17 PROBE_PHASE      - current phase of probing (0-3)
18 PROBE_PROGRESS   - fraction of probing phase completed (0.0-1.0)
19 FRACCUT_PROGRESS - fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
20 DISJCUT_PROGRESS - fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
21 FLOWMIR_PROGRESS - fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
22 MY_THREAD_NUM    - identifier of the parallel thread making this call (always 0)
23 USER_THREADS     - total number of parallel threads currently running (always 1)
```

By returning a nonzero value from cpxcb_FRACCUT, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.7 cpxcb_MIP

CPLEX MIP callback.

Called from TOMLAB /CPLEX during mixed integer optimization.

This callback is enabled by setting *callback*(7) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(7) = 1 if using tomRun('cplex',...).

cpxcb_MIP is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo variable:

```
 i  cpxCBInfo(i)    - Value
-------------------------------------------------------------
 1 BEST_INTEGER     - obj. value of best integer solution
 2 BEST_REMAINING   - obj. value of best remaining node
 3 NODE_COUNT       - total number of nodes solved
 4 NODES_LEFT       - number of remaining nodes
 5 MIP_ITERATIONS   - total number of MIP iterations
 6 MIP_FEAS         - returns 1 if feasible solution exists; otherwise, 0
 7 CUTOFF           - updated cutoff value
 8 CLIQUE_COUNT     - number of clique cuts added
 9 COVER_COUNT      - number of cover cuts added
10 DISJCUT_COUNT    - number of disjunctive cuts added
11 FLOWCOVER_COUNT  - number of flow cover cuts added
12 FLOWPATH_COUNT   - number of flow path cuts added
13 FRACCUT_COUNT    - number of Gomory fractional cuts added
14 GUBCOVER_COUNT   - number of GUB cover cuts added
15 IMPLBD_COUNT     - number of implied bound cuts added
16 MIRCUT_COUNT     - number of mixed integer rounding cuts added
17 PROBE_PHASE      - current phase of probing (0-3)
18 PROBE_PROGRESS   - fraction of probing phase completed (0.0-1.0)
19 FRACCUT_PROGRESS - fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
20 DISJCUT_PROGRESS - fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
21 FLOWMIR_PROGRESS - fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
22 MY_THREAD_NUM    - identifier of the parallel thread making this call (always 0)
23 USER_THREADS     - total number of parallel threads currently running (always 1)
```

By returning a nonzero value from cpxcb_MIP, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.8 cpxcb_MIPPROBE

CPLEX MIP Probe and Clique Merging callback.

Called from TOMLAB /CPLEX during MIP Probing and Clique Merging.

This callback is enabled by setting *callback*(8) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(8) = 1 if using tomRun('cplex',...).

cpxcb_MIPPROBE is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo variable:

```
 i  cpxCBInfo(i)    - Value
-------------------------------------------------------------
 1 BEST_INTEGER     - obj. value of best integer solution
```

```
 2 BEST_REMAINING   - obj. value of best remaining node
 3 NODE_COUNT       - total number of nodes solved
 4 NODES_LEFT       - number of remaining nodes
 5 MIP_ITERATIONS   - total number of MIP iterations
 6 MIP_FEAS         - returns 1 if feasible solution exists; otherwise, 0
 7 CUTOFF           - updated cutoff value
 8 CLIQUE_COUNT     - number of clique cuts added
 9 COVER_COUNT      - number of cover cuts added
10 DISJCUT_COUNT    - number of disjunctive cuts added
11 FLOWCOVER_COUNT  - number of flow cover cuts added
12 FLOWPATH_COUNT   - number of flow path cuts added
13 FRACCUT_COUNT    - number of Gomory fractional cuts added
14 GUBCOVER_COUNT   - number of GUB cover cuts added
15 IMPLBD_COUNT     - number of implied bound cuts added
16 MIRCUT_COUNT     - number of mixed integer rounding cuts added
17 PROBE_PHASE      - current phase of probing (0-3)
18 PROBE_PROGRESS   - fraction of probing phase completed (0.0-1.0)
19 FRACCUT_PROGRESS - fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
20 DISJCUT_PROGRESS - fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
21 FLOWMIR_PROGRESS - fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
22 MY_THREAD_NUM    - identifier of the parallel thread making this call (always 0)
23 USER_THREADS     - total number of parallel threads currently running (always 1)
```

By returning a nonzero value from cpxcb_MIPPROBE, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.9 cpxcb_PRESOLVE

CPLEX Presolve callback.

Called at regular intervals from TOMLAB /CPLEX during presolve.

This callback is enabled by setting *callback*(6) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(6) = 1 if using tomRun('cplex',...).

cpxcb_PRESOLVE is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo variable:

```
i  cpxCBInfo(i)     - Value
--------------------------------------------------------------
1  PRESOLVE_ROWSGONE - number of rows eliminated
2  PRESOLVE_COLSGONE - number of columns eliminated
3  PRESOLVE_AGGSUBST - number of aggregator substitutions
4  PRESOLVE_COEFFS   - number of modified coefficients
```

By returning a nonzero value from cpxcb_PRESOLVE, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.10 cpxcb_PRIM

CPLEX Primal simplex callback

Called from TOMLAB /CPLEX when solving linear problems using the primal simplex algorithm.

This callback is enabled by setting $callback(1) = 1$ in the call to *cplex.m*, or $Prob.MIP.callback(1) = 1$ if using tomRun('cplex',...).

cpxcb_PRIM is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
  i  cpxCBInfo(i)    - Value
 ------------------------------------------------------------
  1  PRIMAL_OBJ      - primal objective value
  2  DUAL_OBJ        - dual objective value
  3  PRIMAL_INFMEAS  - measure of primal infeasibility
  4  DUAL_INFMEAS    - measure of dual infeasibility
  5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
  6  DUAL_FEAS       - 1 if dual feasible, 0 if not
  7  ITCOUNT         - iteration count
  8  CROSSOVER_PPUSH - primal push crossover itn. count
  9  CROSSOVER_PEXCH - primal exchange crossover itn. count
 10  CROSSOVER_DPUSH - dual push crossover itn. count
 11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_PRIM, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.11 cpxcb_PRIMCROSS

CPLEX Primal crossover callback

Called from TOMLAB /CPLEX during the primal crossover algorithm.

This callback is enabled by setting $callback(3) = 1$ in the call to *cplex.m*, or $Prob.MIP.callback(3) = 1$ if using tomRun('cplex',...).

cpxcb_PRIMCROSS is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
  i  cpxCBInfo(i)    - Value
 ------------------------------------------------------------
  1  PRIMAL_OBJ      - primal objective value
  2  DUAL_OBJ        - dual objective value
  3  PRIMAL_INFMEAS  - measure of primal infeasibility
  4  DUAL_INFMEAS    - measure of dual infeasibility
  5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
  6  DUAL_FEAS       - 1 if dual feasible, 0 if not
  7  ITCOUNT         - iteration count
  8  CROSSOVER_PPUSH - primal push crossover itn. count
  9  CROSSOVER_PEXCH - primal exchange crossover itn. count
 10  CROSSOVER_DPUSH - dual push crossover itn. count
 11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_PRIMCROSS, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.12 cpxcb_QPBARRIER

CPLEX Quadratic Barrier callback.

Called from TOMLAB /CPLEX when solving quadratic problems using the barrier algorithm.

This callback is enabled by setting *callback*(12) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(12) = 1 if using tomRun('cplex',...).

cpxcb_QPBARRIER is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
  i  cpxCBInfo(i)    - Value
 ------------------------------------------------------------
  1  PRIMAL_OBJ      - primal objective value
  2  DUAL_OBJ        - dual objective value
  3  PRIMAL_INFMEAS  - measure of primal infeasibility
  4  DUAL_INFMEAS    - measure of dual infeasibility
  5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
  6  DUAL_FEAS       - 1 if dual feasible, 0 if not
  7  ITCOUNT         - iteration count
  8  CROSSOVER_PPUSH - primal push crossover itn. count
  9  CROSSOVER_PEXCH - primal exchange crossover itn. count
 10  CROSSOVER_DPUSH - dual push crossover itn. count
 11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_QPBARRIER, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.13 cpxcb_QPSIMPLEX

CPLEX Quadratic Simplex callback.

Called from TOMLAB /CPLEX when solving quadratic problems using the simplex algorithm.

This callback is enabled by setting *callback*(13) = 1 in the call to *cplex.m*, or *Prob.MIP.callback*(13) = 1 if using tomRun('cplex',...).

cpxcb_QPSIMPLEX is called with one argument, the cpxCBInfo progress information vector.

Contents of cpxCBInfo vector:

```
 i  cpxCBInfo(i)    - Value
 --------------------------------------------------------------
 1  PRIMAL_OBJ      - primal objective value
 2  DUAL_OBJ        - dual objective value
 3  PRIMAL_INFMEAS  - measure of primal infeasibility
 4  DUAL_INFMEAS    - measure of dual infeasibility
 5  PRIMAL_FEAS     - 1 if primal feasible, 0 if not
 6  DUAL_FEAS       - 1 if dual feasible, 0 if not
 7  ITCOUNT         - iteration count
 8  CROSSOVER_PPUSH - primal push crossover itn. count
 9  CROSSOVER_PEXCH - primal exchange crossover itn. count
10  CROSSOVER_DPUSH - dual push crossover itn. count
11  CROSSOVER_DEXCH - dual exchange crossover itn. count
```

By returning a nonzero value from cpxcb_QPSIMPLEX, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.14 cpxcb_INCUMBENT

CPLEX MIP Incumbent callback.

Called from TOMLAB /CPLEX during mixed integer optimization when a new integer solution has been found but before this solution has replaced the current best known integer solution.

This file can be used to perform any desired analysis of the new integer solution and return a status flag to the solver deciding whether to stop or continue the optimization, and also whether to accept or discard the newly found solution.

This callback is enabled by setting callback(14)=1 in the call to *cplex.m*, or *Prob.MIP.callback*(14) = 1 if using tomRun('cplex',...).

cpxcb_INCUMBENT is called by the solver with three arguments:

- x - The new integer solution
- f - The objective value at x
- Prob - The TOMLAB problem structure

cpxcb_INCUMBENT should return one of the following scalar values:

- 0 - Continue optimization and accept new integer solution
- 1 - Continue optimization but discard new integer solution
- 2 - Stop optimization and accept new integer solution
- 3 - Stop optimization and discard new integer solution

Any other return value will be interpreted as 0.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path.

D.15 cpxcb_USERCUT

CPLEX MIP User cut callback.

The User Cut Callback is enabled by setting $callback(15) = 1$ in the call to *cplex.m*, or $Prob.MIP.callback(15) = 1$ if using tomRun('cplex',...)

This callback is called by CPLEX during MIP branch & cut for every node that has an LP optimal solution with objective value below the cutoff and is integer infeasible. CPLEX also calls the callback when comparing an integer feasible solution, including one provided by a MIP start before any nodes exist, against lazy constraints.

The callback routine can add globally valid cuts to the LP subproblem. A cut is a constraint of the following form:

$c1 * x(1) + c2 * x(2) + ... + cn * x(n) <? > rhs$

where <? > is exactly one of the relations <=, >= or = and rhs is a scalar right hand side limit.

By returning a nonzero value from cpxcb_USERCUT, the user can terminate the optimization.

If modifying this file, it is recommended to make a copy of it which is placed before the original file in the MATLAB path. See *help cpxcb_USERCUT* for more information.

D.16 cpxcb_NET

Further details to be added.

E TOMLAB /CPLEX Network Solver

The TOMLAB /CPLEX network solver is a special interface for network problems described by a set of nodes and arcs. The TOMLAB format is not applicable for these types of problem. See *cplexnet* for information on calling the solver.

A network-flow problem finds the minimal-cost flow through a network, where a network consists of a set N of nodes and a set A of arcs connecting the nodes. An arc a in the set A is an ordered pair (i, j) where i and j are nodes in the set N; node i is called the tail or the from-node and node j is called the head or the to-node of the arc a. Not all the pairs of nodes in a set N are necessarily connected by arcs in the set A. More than one arc may connect a pair of nodes; in other words, a1 = (i, j) and a2 = (i, j) may be two different arcs in A, both connecting the nodes i and j in N.

Each arc a may be associated with four values:

- x_a is the flow value, that is, the amount passing through the arc a from its tail (or from-node) to its head (or to-node). The flow values are the modeling variables of a network-flow problem. Negative values are allowed; a negative flow value indicates that there is flow from the head to the tail.
- l_a, the lower bound, determines the minimum flow allowed through the arc a. By default, the lower bound on an arc is 0 (zero).
- u_a, the upper bound, determines the maximum flow allowed through the arc a. By default, the upper bound on an arc is positive infinity.
- c_a, the objective value, determines the contribution to the objective function of one unit of flow through the arc.

Each node n is associated with one value:

- s_n is the supply value at node n.

By convention, a node with strictly positive supply value (that is, $s_n > 0$) is called a supply node or a source, and a node with strictly negative supply value (that is, $s_n < 0$) is called a demand node or a sink. A node where $s_n = 0$ is called a transshipment node. The sum of all supplies must match the sum of all demands; if not, then the network flow problem is infeasible.

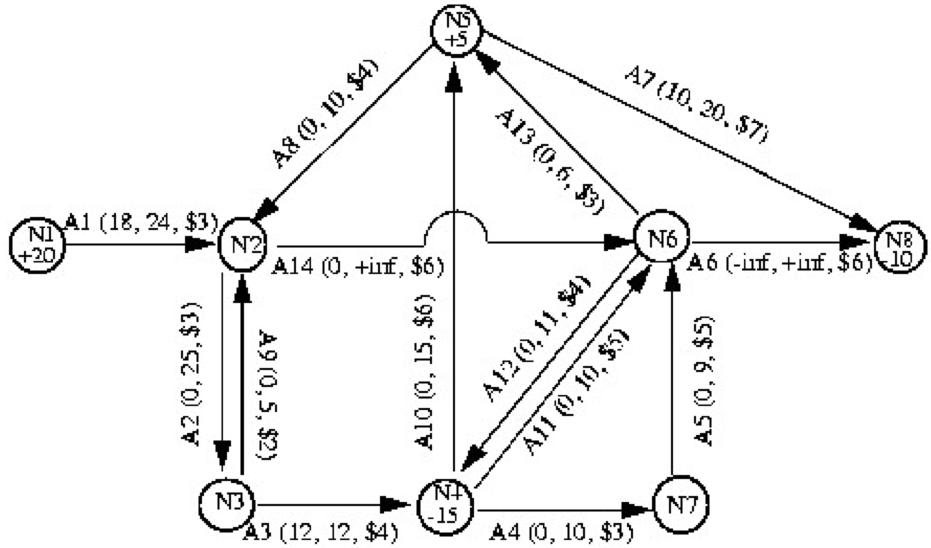
T_n is the set of arcs whose tails are node n; H_n is the set of arcs whose heads are node n. The usual form of a network problem looks like this:

$$
\begin{array}{cl}
\min\limits_{x} & \sum\limits_{a \in A} c_a x_a \\
 & \\
s/t & \sum\limits_{a \in T_a} x_a - \sum\limits_{a \in H_a} x_a = s_n \forall n \in N \\
 & l_a \quad \leq \quad x_a \quad \leq \quad u_a
\end{array}
\tag{1}
$$

A test routines that illustrates a simple problem is included in the TOMLAB distribution. Figure 1 shows the network problem solved:

The following code will call the network solver and deliver the optimal solution.

Figure 1: Network problem in cpxNetTest1.m




```
x = cpxNetTest1;
```

It is possible to call the TOMLAB /CPLEX solver using a special non-MATLAB input format. Example *cpxNetTest2* illustrates how to load and solve the problem described in *nexample.net*.

E.1 cplexnet

Purpose

The Network Interface. It solves network programming (NP) problems. Equation 1 describes the problem structure.

Calling Syntax

$[x, slack, v, rc, f\_k, Inform, Iter]$ = cplexnet(obj, ub, lb, tail, head, supply, callback, PriLev, BIG, cpxControl, logfile, savefile, savemode, netfile);

Description of Inputs

Problem inputs. The following fields are used:

obj	Objective function cost coefficients for the arcs.
ub	Upper bounds for the arcs.
lb	Lower bounds for the arcs.
tail	Indices for the tails (start).
head	Indices for the heads (end).

Problem inputs. The following fields are used:, continued

supply	The supply and demand vector for the nodes.

The following parameters are optional:

callback	Logical scalar defining if callback is used in CPLEX callback = 1 activates the callback. See TOMLAB /CPLEX User's Guide. The callback calls the m-file specified below. The user may edit this file, or make a new copy, which is put before in the Matlab path.
PriLev	Printing level in *cplex.m* file and the CPLEX C-interface. = 0 Silent = 1 Warnings and Errors = 2 Summary information = 3 More detailed information
BIG	Defines default lower and upper bounds, default 1E20. = 0 Silent = 1 Warnings and Errors = 2 Summary information = 3 More detailed information > 10 Pause statements, and maximal printing (debug mode)
cpxControl	Structure, where the fields are set to the CPLEX parameters that the user wants to specify values for. The following parameters are the only ones of general interest. Default values are recommended:
NETITLIM	Limits the number of iterations that the network optimizer performs. Default BIGINT.
NETEPOPT	Optimality tolerance for the network optimizer. The optimality tolerance specifies the amount a reduced cost may violate the criterion for an optimal solution. Default 1e-6. Valid values from 1e-11 to 1e-1.
NETEPRHS	Feasibility tolerance for the network optimizer. The feasibility tolerance specifies the degree to which a problem's flow value may violate its 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. If you encounter reports of infeasibility in the optimization, a small adjustment in the feasibility tolerance may improve performance. Default 1e-6. Valid values from 1e-11 to 1e-1.
NETPPRIIND	Pricing algorithm for the network optimizer. On the rare occasions when the network optimizer seems to take too long to find a solution, you may want to change the pricing algorithm to try to speed up computation. All the choices use variations of partial reduced-cost pricing. NETPPRIIND = 0: automatic, default (same as 3) NETPPRIIND = 1: Partial pricing. NETPPRIIND = 2: Multiple partial pricing. NETPPRIIND = 3: Multiple partial pricing with sorting.

Problem inputs. The following fields are used:, continued

NETFIND	The CPLEX network extractor searches an LP constraint matrix for a submatrix with the following characteristics: - the coefficients of the submatrix are all 0 (zero), 1 (one), or -1 (minus one); - each variable appears in at most two rows with at most one coefficient of +1 and at most one coefficient of -1. CPLEX can perform different levels of extraction. The level it performs depends on the NETFIND parameter. NETFIND = 1: CPLEX extracts only the obvious network; it uses no scaling; it scans rows in their natural order; it stops extraction as soon as no more rows can be added to the network found so far. NETFIND = 2: Default. CPLEX also uses reflection scaling (that is, it multiplies rows by -1) in an attempt to extract a larger network. NETFIND = 3: CPLEX uses general scaling, rescaling both rows and columns, in an attempt to extract a larger network. In terms of total solution time expended, it may or may not be advantageous to extract the largest possible network. Characteristics of your problem will determine the tradeoff between network size and the number of simplex iterations required to finish solving the model after solving the embedded network. Even if your problem does not conform precisely to network conventions, the network optimizer may still be advantageous to use. When it is possible to transform the original statement of a linear program into network conventions by these algebraic operations: - changing the signs of coefficients. - multiplying constraints by constants. - rescaling columns. - adding or eliminating redundant relations. then CPLEX will carry out such transformations automatically if you set the NETFIND parameter appropriately.
PREPASS	If your LP problem includes network structures, there is a possibility that CPLEX preprocessing may eliminate those structures from your model. For that reason, you should consider turning off preprocessing before you invoke the network optimizer on a problem. PREPASS = -1: Default. Determined automatically. PREPASS = 0: Do not use Presolve.

logfile	Name of file to write the CPLEX log information to. If empty, no log is written.
savefile	Name of a file to save the CPLEX problem object. This is useful for sending problems to ILOG for analysis. The format of the file is controlled by the *savemode*. If empty, no file is written.
savemode	The format of the file given in *savefile* is possible to choose by setting *savemode* to one of the following values:

Problem inputs. The following fields are used:, continued

1	SAV	Binary SAV format
2	MPS	MPS format (ASCII)
3	LP	CPLEX LP format (ASCII)
4	RMP	MPS file with generic names
5	REW	MPS file with generic names
6	RLP	LP file with generic names

Modes 4-6 are of limited interest, since the TOMLAB interface does not provide a way to change the default row names.

netfile File for input.

Description of Outputs

Result structure. The following fields are used:

x	Solution vector x with decision variable values ($n \times 1$ vector).
slack	Slack variables ($m \times 1$ vector).
v	Lagrangian multipliers (dual solution vector) ($m \times 1$ vector).
rc	Reduced costs. Lagrangian multipliers for simple bounds on x.
f_k	Objective function value $f(x) = c^T * x$ at optimum.
Inform	Result of CPLEX run. See the m-file help.
Iter	Number of iterations.

F Conflict refiner, IIS, SA and Warm Start

It is possible to perform infeasibility and sensitivity analysis with TOMLAB /CPLEX. The inputs and outputs are described in detail in Section A.1 and A.2.

F.1 Conflict refiner

A conflict is a set of mutually contradictory constraints and bounds within a model. Given an infeasible model, TOMLAB /CPLEX can identify conflicting constraints and bounds within it. TOMLAB /CPLEX refines an infeasible model by examining elements that can be removed from the conflict to arrive at a minimal conflict. A conflict smaller than the full model may make it easier for the user to analyze the source of infeasibilities in the original model.

If the model happens to contain multiple independent causes of infeasibility, it may be necessary for the user to repair one cause and then repeat the process with a further refinement.

A file included in the TOMLAB distribution to enable easy use of the feature.

F.1.1 cpxBuildConflict

Purpose
cpxBuildConflict provides a shortcut for generating conflict refinement groups, for use with the Conflict Refinement feature of TOMLAB /CPLEX.

Calling Syntax
(1) function confgrps = cpxBuildConflict(Prob,mode)
OR
(2) function confgrps = cpxBuildConflict(n,m_lin,m_quad,m_sos,m_ind,'mode')

Description of Inputs

The following inputs are used:

Inputs for (1): function confgrps = cpxBuildConflict(Prob,mode

Prob TOMLAB problem structure, describing a LP/QP/MILP/MIQP/MIQQ problem.

mode String indicating which type of conflict group set is desired.
A 'full' conflict group set will consist of one group for each individual variable (upper+lower bound), linear, quadratic, sos and indicator constraint in the problem. This will be very large group set.
A 'minimal' set consists of at the most 6 groups: one each for all variable lower+upper bounds, linear, sos, indicator, quad constraints.

Inputs for (2): function confgrps = cpxBuildConflict(n,m_lin,m_quad,m_sos,m_ind,'mode')

The following inputs are used:, continued

n	Number of variables
m_lin	Number of linear constraints
m_quad	Number of quadratic constraints
m_sos	Number of SOS constraints
m_ind	Number of indicator constraints
mode	Mode indicator as described above

Description

The confgrps is used as an input to *cplex.m* or *cplexTL.m*.

F.2 IIS

IIS is obsolete in the latest version of TOMLAB /CPLEX.

If TOMLAB /CPLEX reports that your problem is infeasible, then you can invoke the TOMLAB /CPLEX infeasibility finder to help you analyze the source of the infeasibility. This diagnostic tool computes a set of infeasible constraints and column bounds that would be feasible if one of them (a constraint or variable) were removed. Such a set is known as an irreducibly inconsistent set (IIS).

To work, the infeasibility finder must have a problem that satisfies two conditions:

- the problem has been optimized by the primal or dual simplex optimizer or by the barrier optimizer with crossover, and
- the optimizer has terminated with a declaration of infeasibility.

Correcting Multiple Infeasibilities

The infeasibility finder will find only one irreducibly inconsistent set (IIS), though a given problem may contain many independent IISs. Consequently, even after you detect and correct one such IIS in your problem, it may still remain infeasible. In such a case, you need to run the infeasibility finder more than once to detect those multiple causes of infeasibility in your problem.

Interpreting IIS Output

The size of the IIS reported by TOMLAB /CPLEX depends on many factors in the model. If an IIS contains hundreds of rows and columns, you may find it hard to determine the cause of the infeasibility. Fortunately, there are tactics to help you interpret IIS output:

- Consider selecting an alternative IIS algorithm. The default algorithm emphasizes computation speed, and it may give rise to a relatively large IIS. See parameter IISIND.
- If the problem contains equality constraints, examine the cumulative constraint consisting of the sum of the equality rows.
- Try preprocessing with the TOMLAB /CPLEX presolver and aggregator. The presolver may even detect infeasibility by itself. If not, running the infeasibility finder on the presolved problem may help by reducing

the problem size and removing extraneous constraints that do not directly cause the infeasibility but still appear in the IIS. Similarly, running the infeasibility finder on an aggregated problem may help because the aggregator performs substitutions that may remove extraneous variables that clutter the IIS output. More generally, if you perform substitutions, you may simplify the output so that it can be interpreted more easily.

- Other simplifications of the constraints in the IIS, such as combining variables, multiplying constraints by constants, and rearranging sums, may make it easier to interpret the IIS.

F.3 SA

The availability of a basis for an LP allows you to perform sensitivity analysis for your model, if it is an LP. Such analysis tells you by how much you can modify your model without affecting the solution you found. The modifications supported by the sensitivity analysis function include bound changes, changes of the right hand side vector and changes of the objective function.

F.4 Warm Start

When solving a large number of small and similar LP problems with the same size it is recommended to use TOMLAB /CPLEX in a slightly different manner to avoid unnecessary overhead and preserve memory.

This objective is achieved by calling *cplexmex* directly as done internally in *cplex*.

A call to *cplexmex* will return a basis, which can be used to efficiently warm start the solution process of a modified problem. The following code exemplifies the process. In general it is recommended to use the TOMLAB format as well and compare solutions to make sure that the problem is correctly entered.

```
% See cplex.m to backtrack the inputs.
%
Prob = lpAssign(...);
PriLev = 0;
basis = [];

[x, slack, v, rc, f_k, ninf, sinf, Inform, basis] = ...
cplexmex(Prob.QP.c, sparse([]), sparse(Prob.A), zeros(12,1) , ...
Prob.x_L, Prob.x_U, Prob.b_L, Prob.b_U, 1e20, 1, PriLev, Prob, ...
               zeros(Prob.N,1), [], [], [], [], [], [], [], ...
               [], [], [], [], [], [], [], basis, [], []);

% Change the problem and input the basis returned above

Prob.x_L(1) = 2;

[x, slack, v, rc, f_k, ninf, sinf, Inform, basis] = ...
cplexmex(Prob.QP.c, sparse([]), sparse(Prob.A), zeros(12,1) , ...
Prob.x_L, Prob.x_U, Prob.b_L, Prob.b_U, 1e20, 1, PriLev, Prob, ...
               zeros(Prob.N,1), [], [], [], [], [], [], [], ...
               [], [], [], [], [], [], [], basis, [], []);
```

F.5 Solution Pool

The solution pool is used for storing multiple solutions to a mixed integer programming problem (MILP, MIQP and MIQQ). Typically the feature is used for obtaining multiple solutions to help facilitate a selection based on post-processing criteria.

The parameters of interest are listed in Section G and start with *SOLNPOOL** (also *POPULATELIM* is relevant). To enable the collection the following code could be used:

```
% Store up to 20 solutions.
Prob.MIP.cpxControl.SOLNPOOLCAPACITY = 20;
% Use very aggressive collection
Prob.MIP.cpxControl.SOLNPOOLINTENSITY = 4;
% Build diverse set
Prob.MIP.cpxControl.SOLNPOOLREPLACE = 3;
```

The effect of SOLNPOOLINTENSITY is to increase the amount of effort spent setting up the branch and cut tree to prepare for the solution generation.

The details about the settings are as follows:

- Its default value, 0 (zero), lets CPLEX choose which intensity to apply.
- For value 1 (one), 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.
- 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.
- 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 (one) 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.
- 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. In this case, remember to set the populate limit parameter (POPULATELIM) to a value appropriate for your model; otherwise, the populate procedure will stop prematurely because of this stopping criterion instead of enumerating all solutions.

The solutions are stores in *Result.x_k* column-wise, with corresponding objective functions in *Result.f_k*.

G CPLEX Parameters Interface

G.1 Setting CPLEX Parameters in Matlab

The behavior of the CPLEX solver is controlled by means of a large number of *parameters*. It is possible to set all of these parameters from Matlab.

If using the *cplexTL* interface for solving problems defined in a TOMLAB *Prob* structure, the field *Prob.MIP.cpxControl* is used to set values for parameters. *The user needs to set only those parameters that he/she wants to change.*

The non-TOMLAB format *cplex.m* interface has a corresponding input parameter, *cpxControl*.

When setting parameter values in the *cpxControl* structure, this prefix should be omitted. For example, to set the iterations for the dual simplex optimizer do:

```
>> cpxControl.ITLIM = 1000;
>> cpxControl.LPMETHOD = 2;
```

The complete list of CPLEX parameters are given in Table 17 on pages 76–104.

G.2 The CPLEX Parameter Table

Table 17: CPLEX Parameters Overview

TOMLAB parameter	Value
`cpxControl.ADVIND`	0 Off: do not use advanced start information 1 On: CPLEX will use an advanced basis supplied by the user 2 On: CPLEX will crush an advanced basis or starting vector supplied by the user Default: 0
Description: An indicator which, if set to 1 or 2, uses advanced starting information when optimization is initiated. Setting 2 may be effective for MIPs in which the percentage of integer constraints is low. It may also reduce the solution time of fixed MIPs.	
`cpxControl.AGGCUTLIM`	Any non-negative integer. Default: 3
Description: Constraint aggregation limit for cut generation. Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts	
`cpxControl.AGGFILL`	Any non-negative integer Default: 10
Description: Preprocessing aggregator fill. Limits variable substitutions by the aggregator. If the net result of a single substitution is more nonzeros than this value, the substitution is not made.	
`cpxControl.AGGIND`	-1 Automatic (1 for LP, infinite for MIP) 0 Do not use any aggregator Any positive integer Default: -1
Description: Preprocessing aggregator application limit. Invokes the aggregator to use substitution where possible to reduce the number of rows and columns before the problem is solved. If set to a positive value, the aggregator is applied the specified number of times or until no more reductions are possible.	
`cpxControl.BARALG`	0 Default setting 1 Infeasibility-estimate start 2 Infeasibility-constant start 3 Standard barrier Default: 0
Description: Barrier algorithm. The default setting 0 uses the "infeasibility - estimate start" algorithm (setting 1) when solving sub-problems in a Mixed Integer Programming problem, and the standard barrier algorithm (setting 3) in other cases. The standard barrier algorithm is almost always fastest. However, on problems that are primal or dual infeasible (common for Mixed Integer sub-problems), the standard algorithm may not work as well as the alternatives. The two alternative algorithms (settings 1 and 2) may eliminate numerical difficulties related to infeasibility, but are generally slower.	

TOMLAB parameter	Value
`cpxControl.BARCOLNZ`	0 Dynamically calculated or, any positive integer Default: 0
Description: Barrier column nonzeros. Used in the recognition of dense columns. If columns in the presolved and aggregated problem exist with more entries than this value, such columns are considered dense and are treated specially by the CPLEX Barrier Optimizer to reduce their effect. If the problem contains fewer than 400 rows, dense column handling is NOT initiated.	
`cpxControl.BARCROSSALG`	-1 No crossover 0 Automatic 1 Primal crossover 2 Dual crossover Default: 0
Description: Barrier crossover method. Determines which, if any, crossover method is performed at the end of a Barrier optimization.	
`cpxControl.BARDISPLAY`	0 No progress information 1 Normal setup and iteration information 2 Diagnostic information Default: 1
Description: Barrier display information. Determines the level of barrier information to be displayed.	
`cpxControl.BAREPCOMP`	Any positive number $\geq 10^{-12}$ Default: 10^{-8}
Description: Convergence tolerance for LP and QP problems. For problems with quadratic constraints (QCP), see BARQCPEPCOMP. Sets the tolerance on complementarity for convergence. The barrier algorithm terminates with an optimal solution if the relative complementarity is smaller than this value. Changing this tolerance to a smaller value may result in greater numerical precision of the solution, but also increases the chance of a convergence failure in the algorithm and consequently may result in no solution at all. Therefore, caution is advised in deviating from the default setting.	
`cpxControl.BARGROWTH`	1.0 or greater. Default: 10^{8}
Description: Barrier growth. Used to detect unbounded optimal faces. At higher values, the barrier algorithm is less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem has an unbounded face.	
`cpxControl.BARITLIM`	0 No Barrier iterations or, any positive integer Default: Large (varies by computer)

TOMLAB parameter	Value
Description: Barrier iteration limit. Sets the number of Barrier iterations before termination. When set to 0, no Barrier iterations occur, but problem "setup" occurs and information about the setup is displayed (such as Cholesky factorization information).	
`cpxControl.BARMAXCOR`	-1 Automatically determined 0 None or, any positive integer Default: -1
Description: Barrier maximum correction limit. Sets the maximum number of centering corrections done on each iteration. An explicit value greater than 0 may improve the numerical performance of the algorithm at the expense of computation time.	
`cpxControl.BAROBJRNG`	Any positive number Default: 10^{20}
Description: Barrier objective range. Sets the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.	
`cpxControl.BARORDER`	0 Automatic 1 Approximate minimum degree (AMD) 2 Approximate minimum fill (AMF) 3 Nested dissection (ND) Default: 0 values
Description: Barrier ordering algorithm. Sets the algorithm to be used to permute the rows of the constraint matrix in order to reduce fill in the Cholesky factor.	
`cpxControl.BARQCPEPCOMP`	Any positive number $\geq 10^{-12}$ Default: 10^{-6}
Description: Convergence tolerance for QCP problems. That is, for quadratically constrained problems. For LPs and for QPs (that is, when all the constraints are linear) see `BAREPCOMP`. Sets the tolerance on complementarity for convergence. The barrier algorithm terminates with an optimal solution if the relative complementarity is smaller than this value. Changing this tolerance to a smaller value may result in greater numerical precision of the solution, but also increases the chance of a convergence failure in the algorithm and consequently may result in no solution at all. Therefore, caution is advised in deviating from the default setting.	
`cpxControl.BARSTARTALG`	1 Dual is 0 2 Estimate dual 3 Average of primal estimate, dual 0 4 Average of primal estimate, estimate dual Default: 1
Description: Barrier starting point algorithm. Sets the algorithm to be used to compute the initial starting point for the barrier optimizer.	

TOMLAB parameter	Value
cpxControl.BBINTERVAL	0 Best estimate node always selected or, any positive integer Default: 7
Description: MIP strategy bbinterval. When using nodeselect 2, the bbinterval is the interval at which the best bound node, instead of the best estimate node, is selected from the tree. A bbinterval of 0 means to never select the best bound node. A bbinterval of 1 means to always select the best bound node, and is thus equivalent to nodeselect 1. Higher values of bbinterval mean that the best bound node will be selected less frequently; experience has shown it to be beneficial to occasionally select the best bound node, and therefore the default bbinterval is 7.	
cpxControl.BNDSTRENIND	-1 Automatically determined 0 Do not apply bound strengthening 1 Apply bound strengthening Default: -1
Description: Bound strengthening indicator. Used when solving mixed integer programs. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during branch & cut. This reduction is usually beneficial, but occasionally, due to its iterative nature, takes a long time.	
cpxControl.BRDIR	-1 Down branch selected first 0 Automatically determined 1 Up branch selected first Default: 0
Description: MIP branching direction. Used to decide which branch, the up or the down branch, should be taken first at each node.	
cpxControl.BTTOL	Any number from 0.0 to 1.0 Default: 0.9999
Description: Backtracking tolerance. Controls how often backtracking is done during the branching process. The decision when to backtrack depends on three values that change during the course of the optimization: - the objective function value of the best integer feasible solution ("incumbent") - the best remaining objective function value of any unexplored node ("best node") - the objective function value of the most recently solved node ("current objective"). If a cutoff tolerance (see `CUTUP` and `CUTLO`) has been set by the user then that value is used as the incumbent until an integer feasible solution is found. The "target gap" is defined to be the absolute value of the difference between the incumbent and the best node, multiplied by this backtracking parameter. CPLEX does not backtrack until the absolute value of the difference between the current objective and the best node is at least as large as the target gap. Low values of this backtracking parameter thus tend to increase the amount of backtracking, which makes the search process more of a pure best-bound search. Higher parameter values tend to decrease backtracking, making the search more of a pure depth-first search. The backtracking value has effect only after an integer feasible solution is found or when a cutoff has been specified. Note that this backtracking value merely permits backtracking but does not force it; CPLEX may choose to continue searching a limb of the tree if it seems a promising candidate for finding an integer feasible solution.	

TOMLAB parameter	Value
cpxControl.CLIQUES	-1 Do not generate clique cuts 0 Automatically determined 1 Generate clique cuts moderately 2 Generate clique cuts aggressively Default: 0
Description: MIP cliques indicator. Determines whether or not clique cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate cliques should continue only if it seems to be helping.	
cpxControl.CLOCKTYPE	1 CPU time 2 Wall clock time (total physical time elapsed) Default: 1
Description: Computation time reporting. Determines how computation times are measured.	
cpxControl.COEREDIND	0 Do not use coefficient reduction 1 Reduce only to integral coefficients 2 Reduce all potential coefficients Default: 2
Description: Coefficient reduction setting. Determines how coefficient reduction is used. Coefficient reduction improves the objective value of the initial (and subsequent) LP relaxations solved during branch & cut by reducing the number of non-integral vertices.	
cpxControl.COLREADLIM	Any integer from 0 to 268 435 450 Default: Varies by computer.
Description: Variable (column) read limit. Sets the number of variables that can be read.	
cpxControl.COVERS	-1 Do not generate cover cuts 0 Automatically determined 1 Generate cover cuts moderately 2 Generate cover cuts aggressively 3 Generate cover cuts very aggressively Default: 0
Description: MIP covers indicator. Determines whether or not cover cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate covers should continue only if it seems to be helping.	

TOMLAB parameter	Value
cpxControl.CRAIND	**LP Primal:** 0 Ignore objective coefficients during crash -1 or 1 Alternate ways of using objective coefficients **LP Dual:** 1 Default starting basis 0 or -1 Aggressive starting basis **QP Primal:** -1 Slack basis 0 Ignore Q terms and use LP solver for crash 1 Ignore objective and use LP solver for crash **QP Dual:** -1 Slack basis 0 or 1 Use Q terms for crash
Description: Simplex crash ordering. Determines how CPLEX orders variables relative to the objective function when selecting an initial basis.	
cpxControl.CUTLO	Any number Default: -10^{75}
Description: Lower cutoff. When the problem is a maximization problem, the LOWERCUTOFF parameter is used to cut off any nodes that have an objective value below the lower cutoff value. On a continued mixed integer optimization, the larger of these values and the updated cutoff found during optimization are used during the next mixed integer optimization. A too-restrictive value for the LOWERCUTOFF parameter may result in no integer solutions being found.	
cpxControl.CUTPASS	-1 None 0 Automatically determined Positive values give number of passes to perform Default: 0
Description: Number of cutting plane passes. Sets the upper limit on the number of passes CPLEX performs when generating cutting planes on a MIP model.	
cpxControl.CUTSFACTOR	Any non-negative number. Default: 4.0
Description: Row multiplier factor for cuts. 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 number of rows. If the problem is presolved, the original number of rows is that from the presolved problem. A CUTSFACTOR of 1.0 or less means that no cuts will be generated. Because cuts can be added and removed during the course of optimization, CUTSFACTOR may not correspond directly to the number of cuts seen during the node log or in the summary table at the end of optimization.	

TOMLAB parameter	Value
`cpxControl.CUTUP`	Any number. Default: 10^{75}.
Description: Upper cutoff. Cuts off any nodes that have an objective value above the upper cutoff value, when the problem is a minimization problem. When a mixed integer optimization problem is continued, the smaller of these values and the updated cutoff found during optimization are used during the next mixed integer optimization. A too-restrictive value for the `UPPERCUTOFF` parameter may result in no integer solutions being found.	
`cpxControl.DATACHECK`	0 Off (do not check) 1 On (check) Default: 0
Description: Data consistency checking indicator. When set to 1 (On), extensive checking is performed on data in the array arguments, such as checking that indices are within range, that there are no duplicate entries and that values are valid for the type of data or are valid numbers. This is useful for debugging applications.	
`cpxControl.DEPIND`	0 Off (do not use dependency checker) 1 On (use dependency checker) Default: 0
Description: Dependency indicator. Determines whether to activate the "dependency checker". If on, the dependency checker searches for dependent rows during preprocessing. If off, dependent rows are not identified.	
`cpxControl.DISJCUTS`	-1 Do not generate disjunctive cuts 0 Automatically determined 1 Generate disjunctive cuts moderately 2 Generate disjunctive cuts aggressively 3 Generate disjunctive cuts very aggressively Default: 0
Description: MIP disjunctive cuts indicator. Determines whether or not disjunctive cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate disjunctive cuts should continue only if it seems to be helping.	
`cpxControl.DIVETYPE`	0 automatic 1 traditional dive 2 probing dive 3 guided dive Default: 0

TOMLAB parameter	Value
Description: MIP dive strategy. 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 lets CPLEX choose when to perform a probing dive, 1 directs CPLEX never to perform probing dives, 2 always to probe, 3 spend more time exploring potential solutions that are similar to the current incumbent. Setting 2, always to probe, is helpful for finding integer solutions.	
`cpxControl.DPRIIND`	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 Default: 0
Description: Dual simplex pricing algorithm. The default pricing (0) usually provides the fastest solution time, but many problems benefit from alternate settings.	
`cpxControl.EACHCUTLIM`	0 No cuts N+ Limit each type of cut Default: 2.1e9
Description: Type of cut limit. Sets a limit for each type of cut. This parameter allows you to set a uniform limit on the number of cuts of a each type that CPLEX generates. By default, the limit is the largest integer supported by a given platform; 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. This parameter does not influence the number of Gomory cuts.	
`cpxControl.EPAGAP`	Any non-negative number. Default: 10^{-6}.
Description: Absolute mipgap tolerance. Sets an absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When this difference falls below the value of the `ABSMIPGAP` parameter, the mixed integer optimization is stopped.	
`cpxControl.EPGAP`	Any number from 0.0 to 1.0 Default: 10^{-4}

TOMLAB parameter	Value
Description: Relative mipgap tolerance. Sets a relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value $$\frac{\lvert bestnode - bestinteger \rvert}{10^{-10} + \lvert bestinteger \rvert}$$ falls below the value of the `MIPGAP` parameter, the mixed integer optimization is stopped. For example, to instruct CPLEX to stop as soon as it has found a feasible integer solution proved to be within five percent of optimal, set the relative mipgap tolerance to 0.05.	
`cpxControl.EPINT`	Any number from 10^{-9} to 0.5. Default: 10^{-5}
Description: Integrality tolerance. Specifies the amount by which an integer variable can be different from an integer and still be considered feasible.	
`cpxControl.EPMRK`	Any number from 0.0001 to 0.99999 Default: 0.01
Description: Markowitz tolerance. Influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.	
`cpxControl.EPOPT`	Any number from 10^{-9} to 10^{-1} Default: 10^{-6}
Description: Optimality tolerance. Influences the reduced-cost tolerance for optimality. This parameter governs how closely CPLEX must approach the theoretically optimal solution.	
`cpxControl.EPPER`	Any positive number $\geq 10^{-8}$ Default: 10^{-6}
Description: Perturbation constant. Sets the amount by which CPLEX perturbs the upper and lower bounds on the variables when a problem is perturbed. This parameter can be set to a smaller value if the default value creates too large a change in the problem.	
`cpxControl.EPRELAX`	Any positive number Default: 10^{-6}
Description: FeasOpt tolerance. Sets epsilon used to measure relaxation in FeasOpt.	
`cpxControl.EPRHS`	Any number from 10^{-9} to 10^{-1} Default: 10^{-6}

TOMLAB parameter	Value
Description: Feasibility tolerance. The feasibility tolerance specifies the degree to which a problem's basic variables may violate their bounds. FEASIBILITY 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. If you encounter reports of infeasibility during Phase II of the optimization, a small adjustment in the feasibility tolerance may improve performance.	
`cpxControl.FEASOPTMODE`	0 Minimize the sum of all required relaxations in first phase only 1 Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations 2 Minimize the number of constraints and bounds requiring relaxation in first phase only 3 Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations 4 Minimize the sum of squares of required relaxations in first phase only 5 Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations Default: 0
Description: FeasOpt settings. FeasOpt works in 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 among those that require only as much relaxation as it found necessary in the first phase.	
`cpxControl.FLOWCOVERS`	-1 Do not generate flow cover cuts 0 Automatically determined 1 Generate flow cover cuts moderately 2 Generate flow cover cuts aggressively Default: 0
Description: MIP flow cover cuts indicator. Determines whether or not to generate flow cover cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate flow cover cuts should continue only if it seems to be helping.	

TOMLAB parameter	Value
cpxControl.FLOWPATHS	-1 Do not generate flow path cuts 0 Automatically determined 1 Generate flow path cuts moderately 2 Generate flow path cuts aggressively Default: 0
Description: MIP flow path cut indicator. Determines whether or not flow path cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate flow path cuts should continue only if it seems to be helping.	
cpxControl.FPHEUR	-1 Do not apply the feasibility pump heuristic 0 Automatic: let CPLEX choose 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 Default: 0
Description: Feasibility pump switch. Turns on or off the feasibility pump heuristic. At the default setting 0 (zero), CPLEX automatically chooses whether or not to apply the feasibility pump heuristic on the basis of characteristics of the model. To turn off the feasibility pump heuristic, set the parameter to -1 (minus one). If the parameter is set to 1 (one), the feasibility pump tries to find a feasible solution without taking the objective function into account. If the parameter is set to 2, the heuristic usually finds solutions of better objective value, but is more likely to fail to find a feasible solution.	
cpxControl.FRACCAND	Any positive integer. Default: 200
Description: Candidate limit for generating Gomory fractional cuts. Limits the number of candidate variables for generating Gomory fractional cuts.	
cpxControl.FRACCUTS	-1 Do not generate Gomory fractional cuts 0 Automatically determined 1 Generate Gomory fractional cuts moderately 2 Generate Gomory fractional cuts aggressively Default: 0
Description: MIP Gomory fractional cuts indicator. Determines whether or not Gomory fractional cuts should be generated for the problem. Setting the value to 0, the default, indicates that the attempt to generate Gomory fractional cuts should continue only if it seems to be helping.	

TOMLAB parameter	Value
`cpxControl.FRACPASS`	0 Automatic or, any positive integer Default: 0
Description: Pass limit for generating Gomory fractional cuts. Limits the number of passes for generating Gomory fractional cuts. At the default setting of 0, CPLEX decides. The parameter is ignored if the Gomory fractional cut parameter, `FRACCUTS`, is set to a nonzero value.	
`cpxControl.GUBCOVERS`	-1 Do not generate GUB cuts 0 Automatically determined 1 Generate GUB cuts moderately 2 Generate GUB cuts aggressively Default: 0
Description: MIP GUB cuts indicator. Determines whether or not to generate GUB cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.	
`cpxControl.HEURFREQ`	-1 None 0 Automatic or, any positive integer Default: 0
Description: MIP heuristic frequency. Determines how often to apply the periodic heuristic. Setting the value to -1 turns off the periodic heuristic. Setting the value to 0, the default, applies the periodic heuristic at an interval chosen automatically. Setting the value to a positive number applies the heuristic at the requested node interval. For example, setting `HEURISTICFREQ` to 20 dictates that the heuristic be called at node 0, 20, 40, 60, etc.	
`cpxControl.IMPLBD`	-1 Do not generate implied bound cuts 0 Automatically determined 1 Generate implied bound cuts moderately 2 Generate implied bound cuts aggressively Default: 0
Description: MIP implied bound cuts indicator. Determines whether or not to generate implied bound cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate implied bound cuts should continue only if it seems to be helping.	
`cpxControl.INTSOLLIM`	Any positive integer Default: Large (varies by computer)
Description: MIP solution limit. Set the number of MIP solutions to be found before stopping.	
`cpxControl.ITLIM`	Any non-negative integer. Default: Large (varies by computer)

TOMLAB parameter	Value
Description: Simplex maximum iteration limit. Sets the maximum number of iterations to be performed before the algorithm terminates without reaching optimality.	
cpxControl.LBHEUR	0 Off 1 On Default: 0
Description: Local branching heuristic. 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 false; that is, it is off by default. 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.	
cpxControl.LPMETHOD	0 Automatic 1 Primal Simplex 2 Dual Simplex 3 Network Simplex 4 Barrier 5 Sifting 6 Concurrent Dual, Barrier and Primal Default: 0
Description: Method for linear optimization. Determines which algorithm is used. Currently, the behavior of the Automatic setting is that CPLEX almost always invokes the dual simplex method. The one exception is when solving the relaxation of an MILP model when multiple threads have been requested. In this case, the Automatic setting will use the concurrent optimization method. The Automatic setting may be expanded in the future so that CPLEX chooses the method based on additional problem characteristics.	
cpxControl.MEMORYEMPHASIS	0 Off: Do not emphasize conservation of memory 1 On: Emphasize conservation of memory Default: Off
Description: Memory setting. Some information (that require a basis) may be unavailable when using this parameter.	
cpxControl.MIPDISPLAY	0 No display 1 Display integer feasible solutions 2 Display nodes under MIPINTERVAL 3 Same as 2 with information on node cuts 4 Same as 3 with LP subproblem information at root 5 Same as 4 with LP subproblem information at nodes Default: 2

TOMLAB parameter	Value
Description: MIP node log display information. Determines what CPLEX reports to the screen during mixed integer optimization. The amount of information displayed increases with increasing values of this parameter. A setting of 0 causes no node log to be displayed until the optimal solution is found. A setting of 1 displays an entry for each integer feasible solution found. Each entry contains the objective function value, the node count, the number of unexplored nodes in the tree, and the current optimality gap. A setting of 2 also generates an entry for every nth node (where n is the setting of the MIP INTERVAL parameter). A setting of 3 additionally generates an entry for every nth node giving the number of cuts added to the problem for the previous INTERVAL nodes. A setting of 4 additionally generates entries for the LP root relaxation according to the SIMDISPLAY setting. A setting of 5 additionally generates entries for the LP subproblems, also according to the SIMDISPLAY setting.	
`cpxControl.MIPEMPHASIS`	0 `[BALANCED]` Balance optimality and feasibility 1 `[FEASIBILITY]` Emphasize feasibility over optimality 2 `[OPTIMALITY]` Emphasize optimality over feasibility 3 `[BESTBOUND]` Emphasize moving best bound 4 `[HIDDENFEAS]` Emphasize hidden feasibility Default: 0
Description: MIP emphasis indicator. With the default setting of BALANCED, CPLEX works toward a rapid proof of an optimal solution, but balances that with effort toward finding high quality feasible solutions early in the optimization. When set to FEASIBILITY, CPLEX frequently will generate more feasible solutions as it optimizes the problem, at some sacrifice in the speed to the proof of optimality. When set to OPTIMALITY, less effort may be applied to finding feasible solutions early. With the setting BESTBOUND, even greater emphasis is placed on proving optimality through moving the best bound value, so that the detection of feasible solutions along the way becomes almost incidental. When set to HIDDENFEAS, the MIP optimizer works hard to find high quality feasible solutions that are otherwise very difficult to find, so consider this setting when the FEASIBILITY emphasis has difficulty finding solutions of acceptable quality.	
`cpxControl.MIPINTERVAL`	Any positive integer Default: 100
Description: MIP node log interval Controls the frequency of node logging when `MIPDISPLAY` is set higher than 1.	
`cpxControl.MIPORDIND`	0 Off (do not use order information) 1 On (use order information if it exists) Default: 1
Description: MIP priority order indicator. When set to on, uses the priority order (if it exists) for the next mixed integer optimization.	

TOMLAB parameter	Value
cpxControl.MIPORDTYPE	0 Do not generate a priority order 1 Use decreasing cost 2 Use increasing bound range 3 Use increasing cost per coefficient count Default: 0
Description: MIP priority order generation. Used to select the type of generic priority order to generate when no priority order is present.	
cpxControl.MIPSEARCH	0 Automatic: let CPLEX choose. 1 Apply traditional branch and cut strategy; disable dynamic search 2 Apply dynamic search Default: 0
Description: MIP dynamic search switch. Sets the search strategy for a mixed integer program (MIP). By default, CPLEX chooses whether to apply dynamic search or conventional branch and cut based on characteristics of the model and presence (or absence) of callbacks. To benefit from dynamic search, a MIP must not include query callbacks.	
cpxControl.MIQCPSTRAT	0 Automatic: let CPLEX choose. 1 Solve a QCP node relaxation at each node 2 Solve an LP node relaxation at each node Default: 0
Description: MIQQ strategy switch. Sets the strategy that CPLEX uses to solve a quadratically constrained mixed integer program (MIQQ). When you set this parameter to the value 1 (one), you tell CPLEX to solve a QCP node relaxation of the model at each node. When you set this parameter to the value 2, you tell CPLEX to attempt to solve only an LP node relaxation of the model at each node.	
cpxControl.MIRCUTS	-1 Do not generate MIR cuts 0 Automatically determined 1 Generate MIR cuts moderately 2 Generate MIR cuts aggressively Default: 0
Description: MIP MIR (mixed integer rounding) cut indicator. Determines whether or not to generate MIR cuts for the problem. Setting the value to 0, the default, indicates that the attempt to generate MIR cuts should continue only if it seems to be helping.	
cpxControl.MPSLONGNUM	0: Write file in standard MPS format 1 Write with full precision (up to 15 digits) Default: 1

TOMLAB parameter	Value
Description: Degree of precision displayed in output files (MPS). 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.	
`cpxControl.NETDISPLAY`	0 No display 1 Display true objective values 2 Display penalized objective values Default: 2
Description: Network logging display indicator. Settings 1 and 2 differ only during Phase I. Setting 2 shows monotonic values, whereas 1 usually does not.	
`cpxControl.NETEPOPT`	Any number from 10^{-11} to 10^{-1} Default: 10^{-6}
Description: Optimality tolerance for the network optimizer. The optimality tolerance specifies the amount a reduced cost may violate the criterion for an optimal solution.	
`cpxControl.NETEPRHS`	Any number from 10^{-11} to 10^{-1} Default: 10^{-6}
Description: Feasibility tolerance for the network optimizer. The feasibility tolerance specifies the degree to which a problem's flow value may violate its 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. If you encounter reports of infeasibility during Phase II of the optimization, a small adjustment in the feasibility tolerance may improve performance.	
`cpxControl.NETFIND`	1 Extract pure network only 2 Try reflection scaling 3 Try general scaling Default: 2 values
Description: Simplex network extraction level. Establishes the level of network extraction for network simplex optimizations. The default value is suitable for recognizing commonly used modeling approaches when representing a network problem within an LP formulation.	
`cpxControl.NETITLIM`	Any non-negative integer Default: Large (varies by computer)
Description: Network Simplex iteration limit. Sets the maximum number of iterations to be performed before the algorithm terminates without reaching optimality.	

TOMLAB parameter	Value
`cpxControl.NETPPRIIND`	0 Automatic 1 Partial pricing 2 Multiple partial pricing 3 Multiple partial pricing with sorting Default: 0
Description: Network Simplex pricing algorithm. The default (0) shows best performance for most problems, and currently is equivalent to 3.	
`cpxControl.NODEFILEIND`	0 No node file 1 Node file in memory and compressed 2 Node file on disk 3 Node file on disk and compressed Default: 1
Description: Node storage file indicator. Used when working memory, WORKMEM, has been exceeded by the size of the tree. If the node file parameter is set to zero 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, in compressed and uncompressed form respectively, into a directory named by the WORKDIR parameter, and CPLEX actively manages which nodes remain in memory for processing. The use of node files is described in more detail in the CPLEX User's Manual.	
`cpxControl.NODELIM`	Any non-negative integer Default: Large (varies by computer)
Description: MIP node limit. Sets the maximum number of nodes solved before the algorithm terminates, without reaching optimality. When this parameter is set to 0 (zero), CPLEX completes processing at the root; that is, cuts are created and heuristics are applied at the root, but no nodes are created. When the parameter is set to 1 (one), CPLEX branches at the root; that is, nodes are created but not solved.	
`cpxControl.NODESEL`	0 Depth-first search 1 Best-bound search 2 Best-estimate search 3 Alternative best-estimate search Default: 1
Description: MIP node selection strategy. Used to set the rule for selecting the next node to process when backtracking. The depth-first search strategy chooses the most recently created node. The best-bound strategy chooses the node with the best objective function for the associated LP relaxation. The best-estimate strategy selects the node with the best estimate of the integer objective value that would be obtained from a node once all integer infeasibilities are removed. An alternative best-estimate search is also available.	

TOMLAB parameter	Value
cpxControl.NUMERICALEMPHASIS	0 Off: Do not emphasize extreme caution in computation 1 On: Emphasize extreme caution in computation Default: Off
Description: Numerical emphasis.	
cpxControl.NZREADLIM	Any integer from 0 to 268 435 450 Default: 500
Description: Nonzero element read limit. Sets the number of nonzeros that can be read.	
cpxControl.OBJDIF	Any number Default: -10^{75}
Description: Absolute objective difference cutoff. Used to update the cutoff each time a mixed integer solution is found. This absolute value is subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the mixed integer optimization to ignore integer solutions that are not at least this amount better than the one found so far. The OBJDIFFERENCE parameter can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this parameter 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 parameter can result in some integer solutions that are worse than or the same as those previously generated, but does not necessarily result in the generation of all possible integer solutions.	
cpxControl.OBJLLIM	Any number Default: -10^{75}
Description: Lower objective value limit. Setting a lower objective function limit causes CPLEX to halt the optimization process once the minimum objective function value limit has been exceeded. This limit applies only during Phase II of the simplex method.	
cpxControl.OBJULIM	Any number Default: 10^{75}
Description: Upper objective value limit. Setting an upper objective function limit causes CPLEX to halt the optimization process once the maximum objective function value limit has been exceeded. This limit applies only during Phase II of the simplex method.	
cpxControl.PARALLELMODE	-1 Enable opportunistic parallel search mode 0 Automatic: let CPLEX decide whether to invoke deterministic or opportunistic search, depending on the threads parameter 1 Automatic: Enable deterministic parallel search mode Default: 0

TOMLAB parameter	Value
Description: Parallel mode switch. See Section G.3.	
cpxControl.PERIND	0 Off 1 On Default: 0
Description: Simplex perturbation indicator. Setting this parameter to 1 causes all problems to be automatically perturbed as optimization begins. A setting of 0 allows CPLEX to determine dynamically, during solution, whether progress is slow enough to merit a perturbation. The situations in which a setting of 1 helps are rare and restricted to problems that exhibit extreme degeneracy.	
cpxControl.PERLIM	0 Determined automatically or, any positive integer Default: 0
Description: Simplex perturbation limit. Sets the number of stalled iterations before perturbation is performed.	
cpxControl.POLISHTIME	Any positive number in seconds Default: 0
Description: Polishing best solution. Regulates the amount of time spent on polishing the best solution found. During solution polishing, CPLEX applies its effort to improve the best feasible solution. Polishing can yield better solutions in some situations. The default value of the polishing time parameter is 0 (zero); that is, spend no time polishing.	
cpxControl.POPULATELIM	Any nonnegative integer Default: 20
Description: Limit on number of solutions generated for the solution pool. 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, consistent with the relative and absolute pool gap parameters, and has not been rejected by the incumbent callback (if any exists), whether or not it improves the objective of the model.	
cpxControl.PPRIIND	-1 Reduced-cost pricing 0 Hybrid reduced-cost & devex pricing 1 Devex pricing 2 Steepest-edge pricing 3 Steepest-edge pricing with slack initial norms 4 Full pricing Default: 0
Description: Primal Simplex pricing algorithm. The default pricing (0) usually provides the fastest solution time, but many problems benefit from alternative settings	

TOMLAB parameter	Value
`cpxControl.PREDUAL`	-1 Off 0 Automatic 1 On Default: 0
Description: Presolve dual setting. Determines whether CPLEX Presolve should pass the primal or dual linear programming problem to the linear programming optimization algorithm. By default, CPLEX chooses automatically. If the DUAL indicator is set to 1, the CPLEX presolve algorithm is applied to the primal problem, but the resulting dual linear program is passed to the optimizer. This is a useful technique for problems with more constraints than variables.	
`cpxControl.PREIND`	0 Off (do not use presolve) 1 On (use presolve) Default: 1
Description: Presolve indicator. When set to 1, invokes the CPLEX Presolve to simplify and reduce problems.	
`cpxControl.PRELINEAR`	0 Only linear reductions 1 Full reductions Default: 1
Description: Linear reduction indicator. 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. This guarantees, for example, that users can add their own custom cuts to the presolved model.	
`cpxControl.PREPASS`	-1 Determined automatically 0 Do not use Presolve or, any positive integer Default: -1
Description: Limit on the number of Presolve passes made. When set to a nonzero value, invokes the CPLEX Presolve to simplify and reduce problems. When set to a positive value, the Presolve is applied the specified number of times, or until no more reductions are possible. At the default value of -1, Presolve should continue only if it seems to be helping.	
`cpxControl.PRESLVND`	-1 No node presolve 0 Automatic 1 Force node presolve 2 Probes all integer-infeasible variables at each node to find those that can be fixed Default: 0
Description: Node presolve selector. 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 at determining whether to apply node presolve, although runtimes can be reduced for some models by turning node presolve off.	

TOMLAB parameter	Value
`cpxControl.PRICELIM`	0 Determined automatically or, any positive integer Default: 0 values
Description: Simplex pricing candidate list size. Sets the maximum number of variables kept in the pricing candidate list.	
`cpxControl.PROBE`	-1 No probing 0 Automatic 1-3 Probing level Default: 0
Description: MIP probe. Determines the amount of probing on variables to be performed before MIP branching. Higher settings perform more probing. Probing can be very powerful but very time consuming at the start. Setting the parameter to values above the default of 0 (automatic) can result in dramatic reductions or dramatic increases in solution time, depending on the model.	
`cpxControl.PROBETIME`	Any positive number in seconds Default: 1e75
Description: Amount of time spent on probing.	
`cpxControl.QPMAKEPSDIND`	0 Off 1 On Default: On
Description: Indefinite MIQP indicator. 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.	
`cpxControl.QPMETHOD`	0 Automatic 1 Primal Simplex 2 Dual Simplex 3 Network Simplex 4 Barrier Default: 0
Description: Method for continuous quadratic optimization. Determines algorithm. Currently, the behavior of the Automatic setting is that CPLEX invokes the barrier method for continuous QP models, and the dual simplex method for root relaxations of MIQP models. The Automatic setting may be expanded in the future so that CPLEX chooses the method based on additional problem characteristics.	

TOMLAB parameter	Value
`cpxControl.QPNZREADLIM`	Any integer from 0 to 268 435 450 Default: 500
Description: QP Q matrix nonzero read limit. Sets the number of Q matrix nonzeros that can be read.	
`cpxControl.REDUCE`	0 No primal and dual reductions 1 Only primal reductions 2 Only dual reductions 3 Both primal and dual reductions Default: 3 values
Description: Primal and dual reduction type. Determines whether primal reductions, dual reductions, or both, are performed during preprocessing.	
`cpxControl.REINV`	0 Determined automatically or, any integer from 1 to 10 000 Default: 0
Description: Simplex refactorization frequency. Sets the number of iterations between refactorizations of the basis matrix.	
`cpxControl.RELAXPREIND`	-1 Determined automatically 0 Off (do not use presolve on initial relaxation) 1 On (use presolve on initial relaxation) Default: -1
Description: Relaxed LP presolve indicator. Determines whether LP presolve is applied to the root relaxation in a mixed integer program. Sometimes additional reductions can be made beyond any MIP presolve reductions that were already done.	
`cpxControl.RELOBJDIF`	Any integer from 0.0 to 1.0
Description: Relative objective difference cutoff. Used to update the cutoff each time a mixed integer solution is found. The value is multiplied by the absolute value of the integer objective and subtracted from (added to) the newly found integer objective when minimizing (maximizing). This forces the mixed integer optimization to ignore integer solutions that are not at least this amount better than the one found so far. The relative objective difference parameter can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this parameter at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. If both RELOBJDIFFERENCE and OBJDIFFERENCE are nonzero, the value of OBJDIFFERENCE is used.	
`cpxControl.REPAIRTRIES`	Any positive integer Default: 1
Description: Repair tries. If a user provides a MIP start (full or partial) that cannot be extended into a feasible solution, CPLEX will try to repair it.	

TOMLAB parameter	Value
cpxControl.REPEATPRESOLVE	-1 Automatic: Let CPLEX choose whether to re-apply presolve 0 Turn off repeat presolve 1 Repeat presolve without cuts 2 Repeat presolve with cuts 3 Repeat presolve with cuts and allow new root cuts Default: -1
Description: MIP presolve setting. How to re-apply the MIP presolve techniques of the preprocessor to a MIP model at the root after preprocessing has otherwise finished (that is, after cut generation at the root).	
cpxControl.RINSHEUR	-1 None 0 Automatic (default) or, any positive integer Default: 0
Description: Relaxation induced neighborhood search heuristic determines how often to apply the relaxation induced neighborhood search heuristic (RINS heuristic). Setting the value to -1 turns off the RINS heuristic. Setting the value to 0, the default, applies the RINS heuristic at an interval chosen automatically by CPLEX. Setting the value to a positive number applies the RINS heuristic at the requested node interval. For example, setting RINSHEUR to 20 dictates that the RINS heuristic be called at node 0, 20, 40, 60, etc.	
cpxControl.ROWREADLIM	Any integer from 0 to 268 435 450 Default: Varies by computer
Description:	
cpxControl.SCAIND	-1 No scaling 0 Equilibrium scaling method 1 More aggressive scaling Default: 0
Description: Scale parameter. Sets the method to be used for scaling the problem matrix.	
cpxControl.SCRIND	0 Off 1 On Default: On
Description: Messages to screen indicator. Indicates whether or not results messages are displayed on screen.	

TOMLAB parameter	Value
cpxControl.SIFTALG	0 Automatic 1 Primal Simplex 2 Dual Simplex 3 Network Simplex 4 Barrier Default: 0
Description: Sifting subproblem algorithm. Sets the algorithm to be used for solving sifting subproblems.	
cpxControl.SIFTDISPLAY	0 No display 1 Display major iterations 2 Display LP subproblem information within each sifting iteration Default: 1
Description: Sifting display information. Determines the amount of sifting progress information to be displayed.	
cpxControl.SIFTITLIM	Any nonnegative integer Default: BIGINT
Description: Upper limit on sifting iterations. Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.	
cpxControl.SIMDISPLAY	0 No iteration messages until solution 1 Iteration info after each refactorization 2 Iteration info for each iteration Default: 1
Description: Simplex iteration display information. Determines how often CPLEX reports during simplex optimization.	
cpxControl.SINGLIM	Any positive integer Default: 10
Description: Simplex singularity repair limit. Restricts the number of times CPLEX attempts to repair the basis when singularities are encountered. Once this limit is exceeded, CPLEX replaces the current basis with the best factorizable basis that has been found.	
cpxControl.SOLNPOOLAGAP	Any nonnegative real number. Default: 1.0e+75
Description: Absolute gap for solution pool. 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.	

TOMLAB parameter	Value
cpxControl.SOLNPOOLCAPACITY	Any nonnegative integer; 0 (zero) turns off all features of the solution pool. Default: 0
Description: Limit on the number of solutions kept in the solution pool. 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). If the solution pool replacement parameter is set to its default value then the value that the user sets for the solution pool capacity parameter will be increased automatically in cases where the pool is extended.	
cpxControl.SOLNPOOLGAP	Any nonnegative real number. Default: 1.0e+75
Description: Relative gap for the solution pool. 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.	
cpxControl.SOLNPOOLINTENSITY	0 Automatic: let CPLEX choose 1 Mild: generate few solutions quickly 2 Moderate: generate a larger number of solutions 3 Aggressive: generate many solutions and expect performance penalty 4 Very aggressive: enumerate all practical solutions Default: 0
Description: 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 (one) 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.	
cpxControl.SOLNPOOLREPLACE	0 Extend the pool if necessary to accommodate more solutions 1 Replace the first solution by the most recent solution; first in, first out 2 Replace the solution which has the worst objective 3 Replace solutions in order to build a set of diverse solutions Default: 0
Description: Solution pool replacement strategy. Designates the strategy for replacing a solution in the solution pool when the solution pool has reached its capacity.	

TOMLAB parameter	Value
cpxControl.STARTALG	0 Automatic 1 Primal Simplex 2 Dual Simplex 3 Network Simplex 4 Barrier 5 Sifting 6 Concurrent Dual, Barrier and Primal Default: 0
Description: MIP starting LP algorithm. Determines which LP algorithm should be used to solve the initial relaxation of the MIP.	
cpxControl.STRONGCANDLIM	Any positive number Default: 10
Description: MIP candidate list Controls the length of the candidate list when using the ”strong branching” variable selection setting (VARSEL 3).	
cpxControl.STRONGITLIM	Any positive number Default: 0
Description: MIP simplex iterations Controls the number of simplex iterations performed on each variable in the candidate list when using the ”strong branching” variable selection setting (VARSEL 3). The default setting 0 chooses the iteration limit automatically.	
cpxControl.SUBALG	1 Primal Simplex 2 Dual Simplex 3 Network Simplex 4 Barrier 5 Sifting Default: 2
Description: MIP subproblem LP algorithm. Sets the algorithm to be used on MIP subproblems.	
cpxControl.SUBMIPNODELIM	Any positive integer. Default: 500
Description: MIP subnode limit. Restricts the number of nodes searched, during application of the relaxation induced neighborhood search (RINS) heuristic.	

TOMLAB parameter	Value
`cpxControl.SYMMETRY`	-1 Determines automatically 0 Off 1 Moderate 2 Aggressive 3 Very aggressive 4 Highly aggressive 5 Extremely aggressive ++ Default: -1
Description: Symmetry breaking cuts. Determines whether symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model.	
`cpxControl.THREADS`	Minimum: 1 Maximum: determined by computer Default: 0 (Determined by CPLEX)
Description: Global default thread count. Determines the default number of parallel processes (threads) that will be invoked by any CPLEX parallel optimizer. This provides a convenient way to control parallelism with a single parameter setting.	
`cpxControl.TILIM`	Any non-negative number Default: 10^{75}
Description: Global time limit. Sets the maximum time, in seconds, for computations before termination, as measured according to the setting of the CLOCKTYPE parameter. The time limit applies to primal simplex, dual simplex, barrier, and mixed integer optimizations, as well as infeasibility finder computations. (Network simplex and barrier crossover operations are exceptions; these processes do not terminate if the time limit is exceeded.) The time limit includes preprocessing time. For 'hybrid' optimizations (such as network optimization followed by dual or primal simplex, barrier optimization followed by crossover), the cumulative time applies.	
`cpxControl.TRELIM`	Any non-negative number Default: 10^{75}
Description: Tree memory limit. Sets an absolute upper limit on the size (in megabytes) of the branch & cut tree. If this limit is exceeded, CPLEX terminates optimization.	
`cpxControl.TUNINGDISPLAY`	0 Turn off display 1 Display standard, minimal reporting 2 Display standard report plus parameter settings being tried 3 Display exhaustive report and log Default: 1
Description: Tuning information display. Specifies the level of information reported by the tuning tool as it works.	

TOMLAB parameter	Value
cpxControl.TUNINGMEASURE	1 Mean average of time to compare different parameter sets 2 Minmax approach to compare the time of different parameter sets Default: 1
Description: Tuning measure. Controls the measure for evaluating progress when a model is being tuned.	
cpxControl.TUNINGREPEAT	Any non-negative number Default: 1
Description: Tuning repeater. 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.	
cpxControl.TUNINGTILIM	Any non-negative number Default: 10000
Description: Tuning time limit. Sets a time limit per model.	
cpxControl.VARSEL	-1 Branch on variable with minimum infeasibility 0 Branch variable automatically selected 1 Branch on variable with maximum infeasibility 2 Branch based on pseudo costs 3 Strong branching 4 Branch based on pseudo reduced costs Default: 0
Description: MIP variable selection strategy. Used to set the rule for selecting the branching variable at the node which has been selected for branching. The maximum infeasibility rule chooses the variable with the largest fractional value; the minimum infeasibility rule chooses the variable with the smallest fractional value. The minimum infeasibility rule (-1) may lead more quickly to a first integer feasible solution, but is usually slower overall to reach the optimal integer solution. The maximum infeasibility rule (1) forces larger changes earlier in the tree, which tend to produce faster overall times to reach the optimal integer solution. Pseudo cost (2) variable selection is derived from pseudo-shadow prices. Strong branching (3) causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is the most promising. This strategy can be effective on large, difficult MIP problems. Pseudo reduced costs (4) are a computationally less-intensive form of pseudo costs. The default value (0) allows CPLEX to select the best rule based on the problem and its progress.	
cpxControl.WORKDIR	Default: "."
Description: Directory for working files. Specifies the name of an existing directory into which CPLEX may store temporary working files, such as for MIP node files or for out-of-core Barrier.	

TOMLAB parameter	Value
cpxControl.WORKMEM	Any positive number, in megabytes Default: 128.0
Description: Memory available for working storage. Specifies an upper limit on the amount of central memory, in megabytes, that CPLEX is permitted to use for working files (see WORKDIR).	
cpxControl.ZEROHALFCUTS	-1 Do not generate zero-half cuts 0 Automatic: let CPLEX choose 1 Generate zero-half cuts moderately 2 Generate zero-half cuts aggressively Default: 0
Description: MIP zero-half cuts switch. Decides whether or not to generate zero-half cuts for the problem. The value 0 (zero), the default, specifies that the attempt to generate zero-half cuts should continue only if it seems to be helping. If you find that too much time is spent generating zero-half cuts for your model, consider setting this parameter to -1 (minus one) to turn off zero-half cuts. 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.	

G.3 Parallel mode

Sets the parallel optimization mode (PARALLELMODE in Table 17). 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. In multithreaded applications, the opportunistic setting entails less synchronization between threads and consequently may provide better performance.
By default, 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. This condition is referred to as the deterministic mode.
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 and 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.

Deterministic and Sequential Optimization
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.

Callbacks and MIP Optimization

If callbacks other than informational callbacks are used for solving a MIP, the order in which the callbacks are called cannot be guaranteed to remain deterministic, not even in deterministic mode. Thus, to make sure of deterministic runs when the parallel mode parameter is at its default setting, CPLEX will revert to sequential solving of the MIP in the presence of query callbacks, diagnostic callbacks, or control callbacks.

Consequently, if your application invokes query, diagnostic, or control callbacks, and you still prefer deterministic search, you can choose value 1 (one), overriding the automatic setting and turning on deterministic search. It is then your responsibility to make sure that your callbacks do not perform operations that could lead to opportunistic behavior and are implemented in a thread-safe way. To meet these conditions, your application must not store and must not update any information in the callbacks.

Determinism vs Opportunism in Development and Deployment

This parameter also allows you to turn off this default setting by choosing value -1 (minus one). Cases where you might wish to turn off deterministic search include situations where you want to take advantage of possibly faster performance of opportunistic parallel MIP optimization in multiple threads after you have confirmed that deterministic parallel MIP optimization produced the results you expected. For example, you may want to develop your application in deterministic mode, taking advantage of the invariance and repeatability of the solution path and results until you verify that your model and application behave as expected; then you might want to turn off default deterministic mode and run your application in opportunistic mode to see whether your particular model and application benefit from possible performance improvements; if so, you might choose to deploy your application in opportunistic mode.

Interaction with Threads Parameter

Settings of this parallel mode parameter interact with settings of the thread parameter (THREADS).

The default (automatic) setting of the parallel mode parameter allows CPLEX to choose between deterministic and opportunistic mode depending on the threads parameter. If the threads parameter is set to its automatic setting (the default), CPLEX chooses deterministic mode.

Otherwise, if the threads parameter is set to a value greater than one, CPLEX chooses opportunistic mode.

References

[1] Laurence A. Wolsey. *Integer Programming.* John Wiley and Sons, New York, 1998.