

COIN-OR

Stefan Vigerske, Humboldt University Berlin, Germany

Contents

1	Introduction	2
2	CoinBonmin and CoinBonminD	2
2.1	Model requirements	3
2.2	Usage of CoinBonmin	3
2.3	Specification of CoinBonmin Options	4
2.4	Description of CoinBonmin Options	4
3	CoinCbc and CoinCbcD	17
3.1	Model requirements	17
3.2	Usage of CoinCbc	17
3.3	Summary of CoinCBC Options	18
3.4	Detailed Descriptions of CoinCBC Options	20
4	CoinCouenne	33
4.1	Model requirements	34
4.2	Usage of CoinCouenne	34
4.3	Specification of CoinCouenne Options	34
4.4	Description of CoinCouenne Options	34
5	CoinGlpk	42
5.1	Model requirements	43
5.2	Usage of CoinGlpk	43
5.3	Summary of CoinGlpk Options	43
5.4	Detailed Descriptions of CoinGlpk Options	44
6	CoinIpopt and CoinIpoptD	47
6.1	Model requirements	47
6.2	Usage of CoinIpopt	47
6.3	The linear solver in Ipopt	47
6.4	Output	48
6.5	Specification of CoinIpopt Options	50
6.6	Detailed Description of CoinIpopt Options	50
7	CoinOS	74
7.1	Model requirements	74
7.2	Usage of CoinOS	74
7.3	Detailed Descriptions of CoinOS Options	75
8	CoinScip	75
8.1	Model requirements	76
8.2	Usage of CoinScip	76
8.3	Specification of CoinScip Options	76

8.4	Description of CoinScip options	76
9	CoinCplex, CoinGurobi, CoinMosek, CoinXpress	79
9.1	Model requirements	79
9.2	Usage of these links	79

1 Introduction

COIN-OR (**CO**mputational **IN**frastructure for **O**perations **R**esearch, <http://www.coin-or.org>) is an initiative to spur the development of open-source software for the operations research community. One of the projects hosted at COIN-OR is the GAMSlinks project (<https://projects.coin-or.org/GAMSlinks>). It is dedicated to the development of interfaces between GAMS and open source solvers. Some of these links and solvers have also found their way into the regular GAMS distribution. They are currently available for Linux (32 and 64 bit), Windows (32 and 64 bit), Sun Solaris (Intel 64 bit), and Darwin (Intel 32 and 64 bit) systems. With the availability of source code for the GAMSlinks the user is not limited to the out of the box solvers that come with a regular GAMS distribution, but can extend and build these interfaces by themselves.

Available solvers and tools include:

- CoinBonmin 1.1: Basic Open-source Nonlinear Mixed Integer programming
(model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- CoinCbc 2.3: COIN-OR Branch and Cut
(model types: LP, MIP, RMIP)
- CoinCouenne 0.2: Convex Over and Under Envelopes for Nonlinear Estimation
(model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- CoinGlpk 4.39: Gnu Linear Programming Kit
(model types: LP, MIP, RMIP)
- CoinIpopt 3.7: Interior Point Optimizer
(model types: LP, RMIP, DNLP, NLP, RMINLP, QCP, RMIQCP)
- CoinOS 2.0: Optimization Services
(model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- CoinScip 1.2.0: Solving Constrained Integer Programs
(model types: LP, MIP, RMIP)

For more information see the COIN-OR/GAMSlinks web site at <https://projects.coin-or.org/GAMSlinks>.

2 CoinBonmin and CoinBonminD

GAMS/CoinBonmin brings the open source MINLP solver Bonmin to the broad audience of GAMS users.

Bonmin (**B**asic **O**pen-source **N**onlinear **M**ixed **I**nteger programming) is an open-source solver for convex mixed-integer nonlinear programming (MINLPs). The code is developed in a joined project of IBM and the Carnegie Mellon University. The COIN-OR project leader for Bonmin is Pierre Bonami.

Bonmin implements five different algorithms:

- B-BB (**default**): a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on integer variables; this algorithm is similar to the one implemented in the solver SBB
- B-OA: an outer-approximation based decomposition algorithm based on iterating solving and improving of a MIP relaxation and solving NLP subproblems; this algorithm is similar to the one implemented in the solvers DICOPT and AlphaECP

- B-QG: an outer-approximation based branch-and-cut algorithm (by Queseda and Grossmann) based on solving a continuous linear program at each node of the search tree, improving the linear program by outer approximation, and branching on integer variables
- B-Hyb: a branch-and-bound algorithm which is a hybrid of B-BB and B-QG and is based on solving either a continuous nonlinear or a continuous linear program at each node of the search tree, improving the linear program by outer approximation, and branching on integer variables
- B-Ecp: an ECP cuts based branch-and-cut algorithm a la FilMINT

The algorithms are exact when the problem is **convex**, otherwise they are heuristics.

For convex MINLPs, experiments on a reasonably large test set of problems have shown that B-Hyb is the algorithm of choice (it solved most of the problems in 3 hours of computing time). Nevertheless, there are cases where B-OA is much faster than B-Hyb and others where B-BB is interesting. B-QG corresponds mainly to a specific parameter setting of B-Hyb where some features are disabled. B-Ecp has recently been added to Bonmin and is still experimental. For **nonconvex** MINLPs, it is strongly recommend to use B-BB, even though it is a heuristic for such problems too.

For more information we refer to

- the Bonmin web site <https://projects.coin-or.org/Bonmin> and
- the paper P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuejols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N.Sawaya and A. Wächter, An Algorithmic Framework for Convex Mixed Integer Nonlinear Programs, *Discrete Optimization* 5, 186–204, 2008, and *IBM Research Report RC23771*, Oct. 2005.

Most of the Bonmin documentation in the section is taken from the Bonmin manual available on the Bonmin web site.

Gams/BonminD is an experimental in-core communication link. It offers in-core communication between GAMS and the solver, making potentially large model scratch files unnecessary. This links supports all features of the traditional link except for the GAMS branch-cut-heuristic facility (BCH).

2.1 Model requirements

Bonmin can handle mixed-integer nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. The Bonmin link in GAMS supports continuous, binary, and integer variables, special ordered sets, branching priorities, but no semi-continuous or semi-integer variables (see chapter 17.1 of the GAMS User's Guide).

If GAMS/CoinBonmin is called for a model with only continuous variables, the interface directly calls Ipopt.

2.2 Usage of CoinBonmin

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

```
Option MINLP = CoinBonmin;      { or LP, RMIP, MIP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP }
```

The above statement should appear before the Solve statement. If CoinBonmin was specified as the default solver during GAMS installation, the above statement is not necessary.

GAMS/CoinBonmin supports the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. The GAMS BCH facility automates all major steps necessary to define, execute, and control the use of user defined routines within the framework of general purpose MIP and MINLP codes. Currently supported are user defined cut generators and heuristics, where cut generator cannot be used in Bonmins pure B&B algorithm (B-BB). Please see the BCH documentation at <http://www.gams.com/docs/bch.htm> for further information.

2.3 Specification of CoinBonmin Options

A Bonmin option file contains both Ipopt and Bonmin options, for clarity all Bonmin options should be preceded with the prefix “bonmin.”, except those corresponding to the BCH facility. The scheme to name option files is the same as for all other GAMS solvers. Specifying `optfile=1` let Gams/CoinBonmin read `coinbonmin.opt`, `optfile=2` corresponds to `coinbonmin.op2`, and so on. The format of the option file is the same as for Ipopt (see Section 6.5).

The most important option in Bonmin is the choice of the solution algorithm. This can be set by using the option named `bonmin.algorithm` which can be set to B-BB, B-OA, B-QG, B-Hyb, or B-Ecp (its default value is B-BB). Depending on the value of this option, certain other options may be available or not.

In the context of outer approximation decomposition (B-OA), several options are available for configuring the MIP subsolver CBC. By setting the option `bonmin.milp_subsolver` to `Cbc_Par`, a version of CBC is chosen that can be parameterized by the user. The options that can be set are the node-selection strategy, the number of strong-branching candidates, the number of branches before pseudo costs are to be trusted, and the frequency of some cut generators. Their name coincide with Bonmins options, except that the “bonmin.” prefix is replaced with “milp_sub.”.

An example of a `bonmin.opt` file is the following:

```
bonmin.algorithm      B-Hyb
bonmin.oa_log_level  4
bonmin.milp_subsolver Cbc_Par
milp_sub.cover_cuts  0
print_level          6
userheurcall         "bchheur.gms reslim 10"
```

This sets the algorithm to be used to the hybrid algorithm, the level of outer approximation related output to 4, the MIP subsolver for outer approximation to a parameterized version of CBC, switches off cover cutting planes for the MIP subsolver, sets the print level for Ipopt to 6, and let Bonmin call a user defined heuristic specified in the model `bchheur.gms` with a timelimit of 10 seconds.

GAMS/CoinBonmin understands currently the following GAMS parameters: `reslim` (time limit), `iterlim` (iteration limit), `nodlim` (node limit), `cutoff`, `optca` (absolute gap tolerance), and `optcr` (relative gap tolerance). You can set them either on the command line, e.g. `nodlim=1000`, or inside your GAMS program, e.g. `Option nodlim=1000;`.

2.4 Description of CoinBonmin Options

The following tables gives the list of options together with their types, default values and availability in each of the four main algorithms. The column labeled ‘type’ indicates the type of the parameter (‘F’ stands for float, ‘I’ for integer, and ‘S’ for string). The column labeled ‘default’ indicates the global default value. Then for each of the four algorithm B-BB, B-OA, B-QG, and B-Hyb, ‘+’ indicates that the option is available for that particular algorithm while ‘-’ indicates that it is not. Options that are marked with * are those that can be used to configure the MIP subsolver.

Option	type	default	B-BB	B-OA	B-QG	B-Hyb
Algorithm choice						
algorithm	S	B-BB	+	+	+	+
Output						
bb_log_interval	I	100	+	-	+	+
bb_log_level	I	1	+	-	+	+
lp_log_level	I	0	-	-	+	+
milp_log_level	I	0	-	+	-	+
nlp_log_level	I	1	+	+	+	+
oa_cuts_log_level	I	0	-	+	+	+
oa_log_frequency	F	100	+	+	+	+
oa_log_level	I	1	+	+	+	+

Option	type	default	B-BB	B-OA	B-QG	B-Hyb
Branch-and-bound options						
allowable_fraction_gap	F	optcr (0.1)	+	+	+	+
allowable_gap	F	optca (0)	+	+	+	+
cutoff	F	cutoff or 10^{100}	+	+	+	+
cutoff_decr	F	10^{-5}	+	+	+	+
integer_tolerance	F	10^{-6}	+	+	+	+
iteration_limit	I	∞	+	+	+	+
nlp_failure_behavior	S	stop	+	+	+	+
node_comparison	S	best-bound	+	+	+	+
node_limit	I	nodlim or iterlim (∞)	+	+	+	+
num_cut_passes	I	1	-	+	+	+
num_cut_passes_at_root	I	20	-	+	+	+
number_before_trust	I	8	+	+	+	+
number_strong_branch	I	20	+	+	+	+
solution_limit	I	∞	+	+	+	+
time_limit	F	reslim (1000)	+	+	+	+
tree_search_strategy	S	probed-dive	+	+	+	+
variable_selection	S	strong-branching	+	+	+	+
Diving options						
max_backtracks_in_dive	I	5	+	+	+	+
max_dive_depth	I	∞	+	+	+	+
stop_diving_on_cutoff	S	no	+	+	+	+
ECP based strong branching						
eep_abs_tol_strong	F	10^{-6}	+	+	+	+
eep_max_rounds_strong	I	0	+	+	+	+
eep_rel_tol_strong	F	0.1	+	+	+	+
lp_strong_warmstart_method	S	Basis	+	+	+	+
ECP cuts generation						
eep_abs_tol	F	10^{-6}	-	-	-	+
eep_max_rounds	I	5	-	-	-	+
eep_propability_factor	F	1000	-	-	-	+
eep_rel_tol	F	0	-	-	-	+
filmint_eep_cuts	I	0	-	-	-	+
GAMS Branch Cut and Heuristic Facility						
usercutcall	S	none	-	+	+	+
usercutfirst	I	10	-	+	+	+
usercutfreq	I	10	-	+	+	+
usercutinterval	I	100	-	+	+	+
usercutmult	I	2	-	+	+	+
usercutnewint	I	0	-	+	+	+
usergdxin	S	bchin.gdx	+	+	+	+
usergdxname	S	bchout.gdx	+	+	+	+
usergdxnameinc	S	bchout.i.gdx	+	+	+	+
usergdxprefix	S	none	+	+	+	+
userheurcall	S	none	+	+	+	+
userheurfirst	I	10	+	+	+	+
userheurfreq	I	10	+	+	+	+
userheurinterval	I	100	+	+	+	+
userheurmult	I	2	+	+	+	+
userheurnewint	I	0	+	+	+	+
userheurobjfirst	I	0	+	+	+	+
userjobid	S	none	+	+	+	+
userkeep	I	0	+	+	+	+

Option	type	default	B-BB	B-OA	B-QG	B-Hyb
MILP cutting planes in hybrid algorithm						
2mir_cuts	I	0	-	+	-	+
Gomory_cuts	I	-5	-	+	-	+
clique_cuts	I	-5	-	+	-	+
cover_cuts	I	-5	-	+	-	+
flow_covers_cuts	I	-5	-	+	-	+
lift_and_project_cuts	I	0	-	+	-	+
mir_cuts	I	-5	-	+	-	+
reduce_and_split_cuts	I	0	-	+	-	+
NLP interface						
warm_start	S	none	+	-	-	-
NLP solution robustness						
max_consecutive_failures	I	10	+	+	+	+
max_random_point_radius	F	100000	+	-	-	-
num_iterations_suspect	I	-1	+	+	+	+
num_retry_unsolved_random_point	I	0	+	+	+	+
random_point_perturbation_interval	F	1	+	-	-	-
random_point_type	S	Jon	+	-	-	-
NLP solves in hybrid algorithm						
nlp_solve_frequency	I	10	-	-	-	+
nlp_solve_max_depth	I	10	-	-	-	+
nlp_solves_per_depth	F	10^{30}	-	-	-	+
Nonconvex problems						
max_consecutive_infeasible	I	0	+	+	+	+
num_resolve_at_infeasibles	I	0	+	-	-	-
num_resolve_at_node	I	0	+	-	-	-
num_resolve_at_root	I	0	+	-	-	-
Outer Approximation decomposition						
milp_subsolver	S	Cbc.D	-	+	-	+
oa_dec_time_limit	F	30	+	+	+	+
Outer Approximation cuts generation						
add_only_violated_oa	S	no	-	+	+	+
cut_strengthening_type	S	none	-	+	+	+
disjunctive_cut_type	S	none	-	+	+	+
oa_cuts_scope	S	global	-	+	+	+
tiny_element	F	10^{-8}	-	+	+	+
very_tiny_element	F	10^{-17}	-	+	+	+
Strong branching setup						
candidate_sort_criterion	S	best-ps-cost	+	+	+	+
maxmin_crit_have_sol	F	0.1	+	+	+	+
maxmin_crit_no_sol	F	0.7	+	+	+	+
min_number_strong_branch	I	0	+	+	+	+
number_before_trust_list	I	0	+	+	+	+
number_look_ahead	I	0	+	+	+	+
number_strong_branch_root	I	∞	+	+	+	+
setup_pseudo_frac	F	0.5	+	+	+	+
trust_strong_branching_for_pseudo_cost	S	yes	+	+	+	+

Algorithm choice

algorithm: Choice of the algorithm.

This will preset some of the options of bonmin depending on the algorithm choice. The default value for this string option is "B-BB".

Possible values:

- B-BB: simple branch-and-bound algorithm,
- B-OA: OA Decomposition algorithm,
- B-QG: Quesada and Grossmann branch-and-cut algorithm,
- B-Hyb: hybrid outer approximation based branch-and-cut,
- B-Ecp: ecp cuts based branch-and-cut a la FilmINT.

Output

bb_log_interval: Interval at which node level output is printed.

Set the interval (in terms of number of nodes) at which a log on node resolutions (consisting of lower and upper bounds) is given. The valid range for this integer option is $0 \leq \text{bb_log_interval} < +\text{inf}$ and its default value is 100.

bb_log_level: specify main branch-and-bound log level.

Set the level of output of the branch-and-bound: 0 - none, 1 - minimal, 2 - normal low, 3 - normal high. The valid range for this integer option is $0 \leq \text{bb_log_level} \leq 5$ and its default value is 1.

lp_log_level: specify LP log level.

Set the level of output of the linear programming sub-solver in B-Hyb or B-QG: 0 - none, 1 - minimal, 2 - normal low, 3 - normal high, 4 - verbose. The valid range for this integer option is $0 \leq \text{lp_log_level} \leq 4$ and its default value is 0.

milp_log_level: specify MILP subsolver log level.

Set the level of output of the MILP subsolver in OA: 0 - none, 1 - minimal, 2 - normal low, 3 - normal high. The valid range for this integer option is $0 \leq \text{milp_log_level} \leq 3$ and its default value is 0.

nlp_log_level: specify NLP solver interface log level (independent from ipopt print_level).

Set the level of output of the OsiTMINLPInterface: 0 - none, 1 - normal, 2 - verbose. The valid range for this integer option is $0 \leq \text{nlp_log_level} \leq 2$ and its default value is 1.

oa_cuts_log_level: level of log when generating OA cuts.

0: outputs nothing, 1: when a cut is generated, its violation and index of row from which it originates, 2: always output violation of the cut. 3: output generated cuts incidence vectors. The valid range for this integer option is $0 \leq \text{oa_cuts_log_level} < +\text{inf}$ and its default value is 0.

oa_log_frequency: display an update on lower and upper bounds in OA every n seconds

The valid range for this real option is $0 < \text{oa_log_frequency} < +\text{inf}$ and its default value is 100.

oa_log_level: specify OA iterations log level.

Set the level of output of OA decomposition solver: 0 - none, 1 - normal, 2 - verbose. The valid range for this integer option is $0 \leq \text{oa_log_level} \leq 2$ and its default value is 1.

Branch-and-bound options

allowable_fraction_gap: Specify the value of relative gap under which the algorithm stops.

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of the best known solution value. The valid range for this real option is $-1 \cdot 10^{+20} \leq \text{allowable_fraction_gap} \leq 1 \cdot 10^{+20}$ and its default value is the value of the GAMS parameter optcr, which default value is 0.1.

allowable_gap: Specify the value of absolute gap under which the algorithm stops.

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this. The valid range for this real option is $-1 \cdot 10^{+20} \leq \text{allowable_gap} \leq 1 \cdot 10^{+20}$ and its default value is the value of the GAMS parameter `optca`, which default value is 0.

cutoff: Specify cutoff value.

`cutoff` should be the value of a feasible solution known by the user (if any). The algorithm will only look for solutions better than `cutoff`. The valid range for this real option is $-1 \cdot 10^{+100} \leq \text{cutoff} \leq 1 \cdot 10^{+100}$ and its default value is the value of the GAMS parameter `cutoff`, if set, otherwise it is 10^{100} .

cutoff_decr: Specify cutoff decrement.

Specify the amount by which `cutoff` is decremented below a new best upper-bound (usually a small positive value but in non-convex problems it may be a negative value). The valid range for this real option is $-1 \cdot 10^{+10} \leq \text{cutoff_decr} \leq 1 \cdot 10^{+10}$ and its default value is $1 \cdot 10^{-05}$.

integer_tolerance: Set integer tolerance.

Any number within that value of an integer is considered integer. The valid range for this real option is $0 < \text{integer_tolerance} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

iteration_limit: Set the cumulated maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.

value 0 deactivates option. The valid range for this integer option is $0 \leq \text{iteration_limit} < +\text{inf}$ and its default value is 2147483647.

nlp_failure_behavior: Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).

If set to "fathom", the algorithm will fathom the node when Ipopt fails to find a solution to the nlp at that node within the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal. The default value for this string option is "stop".

Possible values:

- stop: Stop when failure happens.
- fathom: Continue when failure happens.

node_comparison: Choose the node selection strategy.

Choose the strategy for selecting the next node to be processed. The default value for this string option is "best-bound".

Possible values:

- best-bound: choose node with the smallest bound,
- depth-first: Perform depth first search,
- breadth-first: Perform breadth first search,
- dynamic: Cbc dynamic strategy (starts with a depth first search and turn to best bound after 3 integer feasible solutions have been found).
- best-guess: choose node with smallest guessed integer solution

node_limit: Set the maximum number of nodes explored in the branch-and-bound search.

The valid range for this integer option is $0 \leq \text{node_limit} < +\text{inf}$ and its default value is the value of the GAMS parameter `nodlim`, if set, otherwise it is the value of the GAMS parameter `iterlim`, which default value is ∞ .

num_cut_passes: Set the maximum number of cut passes at regular nodes of the branch-and-cut. The valid range for this integer option is $0 \leq \text{num_cut_passes} < +\text{inf}$ and its default value is 1.

num_cut_passes_at_root: Set the maximum number of cut passes at regular nodes of the branch-and-cut. The valid range for this integer option is $0 \leq \text{num_cut_passes_at_root} < +\text{inf}$ and its default value is 20.

number_before_trust: Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching. A value of 0 disables pseudo costs. The valid range for this integer option is $0 \leq \text{number_before_trust} < +\text{inf}$ and its default value is 8.

number_strong_branch: Choose the maximum number of variables considered for strong branching. Set the number of variables on which to do strong branching. The valid range for this integer option is $0 \leq \text{number_strong_branch} < +\text{inf}$ and its default value is 20.

solution_limit: Abort after that much integer feasible solution have been found by algorithm value 0 deactivates option The valid range for this integer option is $0 \leq \text{solution_limit} < +\text{inf}$ and its default value is 2147483647.

time_limit: Set the global maximum computation time (in secs) for the algorithm. The valid range for this real option is $0 < \text{time_limit} < +\text{inf}$ and its default value is the value of the GAMS parameter reslim, which default value is 1000.

tree_search_strategy: Pick a strategy for traversing the tree All strategies can be used in conjunction with any of the node comparison functions. Options which affect dfs-dive are max-backtracks-in-dive and max-dive-depth. The dfs-dive won't work in a non-convex problem where objective does not decrease down branches. The default value for this string option is "probed-dive". Possible values:

- top-node: Always pick the top node as sorted by the node comparison function
- dive: Dive in the tree if possible, otherwise pick top node as sorted by the tree comparison function.
- probed-dive: Dive in the tree exploring two childs before continuing the dive at each level.
- dfs-dive: Dive in the tree if possible doing a depth first search. Backtrack on leaves or when a prescribed depth is attained or when estimate of best possible integer feasible solution in subtree is worst than cutoff. Once a prescribed limit of backtracks is attained pick top node as sorted by the tree comparison function
- dfs-dive-dynamic: Same as dfs-dive but once enough solution are found switch to best-bound and if too many nodes switch to depth-first.

variable_selection: Chooses variable selection strategy The default value for this string option is "strong-branching". Possible values:

- most-fractional: Choose most fractional variable
- strong-branching: Perform strong branching
- reliability-branching: Use reliability branching
- curvature-estimator: Use curvature estimation to select branching variable
- qp-strong-branching: Perform strong branching with QP approximation

- `lp-strong-branching`: Perform strong branching with LP approximation
- `nlp-strong-branching`: Perform strong branching with NLP approximation
- `osi-simple`: Osi method to do simple branching
- `osi-strong`: Osi method to do strong branching
- `random`: Method to choose branching variable randomly

Diving options

max_backtracks_in_dive: Set the number of backtracks in a dive when using dfs-dive tree searchstrategy. The valid range for this integer option is $0 \leq \text{max_backtracks_in_dive} < +\text{inf}$ and its default value is 5.

max_dive_depth: When using dfs-dive search. Maximum depth to go to from the diving board (node where the diving started). The valid range for this integer option is $0 \leq \text{max_dive_depth} < +\text{inf}$ and its default value is 2147483647.

stop_diving_on_cutoff: Flag indicating whether we stop diving based on guessed feasible objective and the current cutoff. The default value for this string option is "no". Possible values: "no" and "yes".

ECP based strong branching

ecp_abs_tol_strong: Set the absolute termination tolerance for ECP rounds in strong branching. The valid range for this real option is $0 \leq \text{ecp_abs_tol_strong} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

ecp_max_rounds_strong: Set the maximal number of rounds of ECP cuts in strong branching. The valid range for this integer option is $0 \leq \text{ecp_max_rounds_strong} < +\text{inf}$ and its default value is 0.

ecp_rel_tol_strong: Set the relative termination tolerance for ECP rounds in strong branching. The valid range for this real option is $0 \leq \text{ecp_rel_tol_strong} < +\text{inf}$ and its default value is 0.1.

lp_strong_warmstart_method: Choose method to use for warm starting lp in strong branching (Advanced stuff) The default value for this string option is "Basis". Possible values:

- `Basis`: Use optimal basis of node
- `Clone`: Clone optimal problem of node

ECP cuts generation

ecp_abs_tol: Set the absolute termination tolerance for ECP rounds. The valid range for this real option is $0 \leq \text{ecp_abs_tol} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

ecp_max_rounds: Set the maximal number of rounds of ECP cuts. The valid range for this integer option is $0 \leq \text{ecp_max_rounds} < +\text{inf}$ and its default value is 5.

ecp_propability_factor: Factor appearing in formula for skipping ECP cuts. Choosing -1 disables the skipping. The valid range for this real option is $-\text{inf} < \text{ecp_propability_factor} < +\text{inf}$ and its default value is 1000.

ecp_rel_tol: Set the relative termination tolerance for ECP rounds. The valid range for this real option is $0 \leq \text{ecp_rel_tol} < +\text{inf}$ and its default value is 0.

filmint_ecp_cuts: Specify the frequency (in terms of nodes) at which some a la filmint ecp cuts are generated. A frequency of 0 amounts to to never solve the NLP relaxation. The valid range for this integer option is $0 \leq \text{filmint_ecp_cuts} < +\text{inf}$ and its default value is 0.

GAMS Branch-Cut-Heuristic Facility

usercutcall: The GAMS command line to call the cut generator. Setting this option and having `bonmin.algorithm` left on B-BB, changes the algorithm to B-Hyb. Possible values are any string enclosed in quotes ("").

usercutfirst: Calls the cut generator for the first n nodes. The valid range for this integer option is $0 \leq \text{usercutfirst} < +\text{inf}$ and its default value is 10.

usercutfreq: Determines the frequency of the cut generator model calls. The valid range for this integer option is $0 \leq \text{usercutfreq} < +\text{inf}$ and its default value is 10.

usercutinterval: Determines the interval when to apply the multiplier for the frequency of the cut generator model calls. The valid range for this integer option is $0 \leq \text{usercutinterval} < +\text{inf}$ and its default value is 100.

usercutmult: Determines the multiplier for the frequency of the cut generator model calls. The valid range for this integer option is $0 \leq \text{usercutmult} < +\text{inf}$ and its default value is 2.

usercutnewint: Calls the cut generator if the solver found a new integer feasible solution. The valid range for this integer option is $0 \leq \text{usercutnewint} \leq 1$ and its default value is 0.

usergdxin: The name of the GDX file read back into Bonmin. The default value for this string option is "bchin.gdx".

usergdxname: The name of the GDX file exported from the solver with the solution at the node. The default value for this string option is "bchout.gdx".

usergdxnameinc: The name of the GDX file exported from the solver with the incumbent solution. The default value for this string option is "bchout.i.gdx".

usergdxprefix: Prefixes `usergdxin`, `usergdxname`, and `usergdxnameinc`. Possible value is any string.

userheurcall: The GAMS command line to call the heuristic. Possible values are any string enclosed in quotes ("").

userheurfirst: Calls the heuristic for the first n nodes.

The valid range for this integer option is $0 \leq \text{userheurfirst} < +\text{inf}$ and its default value is 10.

userheurfreq: Determines the frequency of the heuristic model calls.

The valid range for this integer option is $0 \leq \text{userheurfreq} < +\text{inf}$ and its default value is 10.

userheurinterval: Determines the interval when to apply the multiplier for the frequency of the heuristic model calls.

The valid range for this integer option is $0 \leq \text{userheurinterval} < +\text{inf}$ and its default value is 100.

userheurmuilt: Determines the multiplier for the frequency of the heuristic model calls.

The valid range for this integer option is $0 \leq \text{userheurmuilt} < +\text{inf}$ and its default value is 2.

userheurnewint: Calls the heuristic if the solver found a new integer feasible solution.

The valid range for this integer option is $0 \leq \text{userheurnewint} \leq 1$ and its default value is 0.

userheurobjfirst: Calls the heuristic if the LP value of the node is closer to the best bound than the current incumbent.

The valid range for this integer option is $0 \leq \text{userheurobjfirst} < +\text{inf}$ and its default value is 0.

userjobid: Postfixes `gdxname`, `gdxnameinc`, and `gdxin`.

Possible value is any string.

userkeep: Calls `gamskeep` instead of `gams`.

The valid range for this integer option is $0 \leq \text{userkeep} \leq 1$ and its default value is 0.

MILP cutting planes in hybrid algorithm (B-Hyb)

2mir_cuts: Frequency (in terms of nodes) for generating 2-MIR cuts in branch-and-cut

If $k > 0$, cuts are generated every k nodes, if $-99 < k < 0$ cuts are generated every $-k$ nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if $k = -99$ generate cuts only at the root node, if $k = 0$ or 100 do not generate cuts. The valid range for this integer option is $-100 \leq \text{2mir_cuts} < +\text{inf}$ and its default value is 0.

Gomory_cuts: Frequency k (in terms of nodes) for generating Gomory cuts in branch-and-cut.

See the option `2mir_cuts` on how the value for the frequency is interpreted. The valid range for this integer option is $-100 \leq \text{Gomory_cuts} < +\text{inf}$ and its default value is -5 .

clique_cuts: Frequency (in terms of nodes) for generating clique cuts in branch-and-cut

See the option `2mir_cuts` on how the value for the frequency is interpreted. The valid range for this integer option is $-100 \leq \text{clique_cuts} < +\text{inf}$ and its default value is -5 .

cover_cuts: Frequency (in terms of nodes) for generating cover cuts in branch-and-cut

See the option `2mir_cuts` on how the value for the frequency is interpreted. The valid range for this integer option is $-100 \leq \text{cover_cuts} < +\text{inf}$ and its default value is -5 .

flow_covers_cuts: Frequency (in terms of nodes) for generating flow cover cuts in branch-and-cut

See the option `2mir_cuts` on how the value for the frequency is interpreted. The valid range for this integer option is $-100 \leq \text{flow_covers_cuts} < +\text{inf}$ and its default value is -5 .

lift_and_project_cuts: Frequency (in terms of nodes) for generating lift-and-project cuts in branch-and-cut. See the option `2mir_cuts` on how the value for the frequency is interpreted. The valid range for this integer option is $-100 \leq \text{lift_and_project_cuts} < +\text{inf}$ and its default value is 0.

mir_cuts: Frequency (in terms of nodes) for generating MIR cuts in branch-and-cut. See the option `2mir_cuts` on how the value for the frequency is interpreted. The valid range for this integer option is $-100 \leq \text{mir_cuts} < +\text{inf}$ and its default value is -5 .

reduce_and_split_cuts: Frequency (in terms of nodes) for generating reduce-and-split cuts in branch-and-cut. The valid range for this integer option is $-100 \leq \text{reduce_and_split_cuts} < +\text{inf}$ and its default value is 0.

NLP interface

warm_start: Select the warm start method. This will affect the function `getWarmStart()`, and as a consequence the warm starting in the various algorithms. The default value for this string option is "none". Possible values:

- none: No warm start
- optimum: Warm start with direct parent optimum
- interior_point: Warm start with an interior point of direct parent

NLP solution robustness

max_consecutive_failures: (temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree.

When $n > 0$, continue exploring a branch of the tree until n consecutive problems in the branch are unsolved (we call unsolved a problem for which Ipopt can not guarantee optimality within the specified tolerances). The valid range for this integer option is $0 \leq \text{max_consecutive_failures} < +\text{inf}$ and its default value is 10.

max_random_point_radius: Set max value r for coordinate of a random point.

When picking a random point coordinate i will be in the interval $[\min(\max(l,-r), u-r), \max(\min(u,r), l+r)]$ (where l is the lower bound for the variable and u is its upper bound). The valid range for this real option is $0 < \text{max_random_point_radius} < +\text{inf}$ and its default value is 100000.

num_iterations_suspect: Number of iterations over which a node is considered "suspect" (for debugging purposes only, see detailed documentation).

When the number of iterations to solve a node is above this number, the subproblem at this node is considered to be suspect and it will be outputted in a file (set to -1 to deactivate this). The valid range for this integer option is $-1 \leq \text{num_iterations_suspect} < +\text{inf}$ and its default value is -1 .

num_retry_unsolved_random_point: Number k of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).

When Ipopt fails to solve a continuous NLP sub-problem, if $k > 0$, the algorithm will try again to solve the failed NLP with k new randomly chosen starting points or until the problem is solved with success. The valid range for this integer option is $0 \leq \text{num_retry_unsolved_random_point} < +\text{inf}$ and its default value is 0.

random_point_perturbation_interval: Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point

The valid range for this real option is $0 < \text{random_point_perturbation_interval} < +\text{inf}$ and its default value is 1.

random_point_type: method to choose a random starting point

The default value for this string option is "Jon".

Possible values:

- Jon: Choose random point uniformly between the bounds
- Andreas: perturb the starting point of the problem within a prescribed interval
- Claudia: perturb the starting point using the perturbation radius suffix information

NLP solves in hybrid algorithm (B-Hyb)

nlp_solve_frequency: Specify the frequency (in terms of nodes) at which NLP relaxations are solved in B-Hyb.

A frequency of 0 amounts to to never solve the NLP relaxation. The valid range for this integer option is $0 \leq \text{nlp_solve_frequency} < +\text{inf}$ and its default value is 10.

nlp_solve_max_depth: Set maximum depth in the tree at which NLP relaxations are solved in B-Hyb.

A depth of 0 amounts to to never solve the NLP relaxation. The valid range for this integer option is $0 \leq \text{nlp_solve_max_depth} < +\text{inf}$ and its default value is 10.

nlp_solves_per_depth: Set average number of nodes in the tree at which NLP relaxations are solved in B-Hyb for each depth.

The valid range for this real option is $0 \leq \text{nlp_solves_per_depth} < +\text{inf}$ and its default value is $1 \cdot 10^{+30}$.

Nonconvex problems

max_consecutive_infeasible: Number of consecutive infeasible subproblems before aborting a branch.

Will continue exploring a branch of the tree until "max_consecutive_infeasible" consecutive problems are infeasibles by the NLP sub-solver. The valid range for this integer option is $0 \leq \text{max_consecutive_infeasible} < +\text{inf}$ and its default value is 0.

num_resolve_at_infeasibles: Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point.

The algorithm will solve all the infeasible nodes with k different random starting points and will keep the best local optimum found. The valid range for this integer option is $0 \leq \text{num_resolve_at_infeasibles} < +\text{inf}$ and its default value is 0.

num_resolve_at_node: Number k of tries to resolve a node (other than the root) of the tree with different starting point.

The algorithm will solve all the nodes with k different random starting points and will keep the best local optimum found. The valid range for this integer option is $0 \leq \text{num_resolve_at_node} < +\text{inf}$ and its default value is 0.

num_resolve_at_root: Number k of tries to resolve the root node with different starting points.

The algorithm will solve the root node with k random starting points and will keep the best local optimum found. The valid range for this integer option is $0 \leq \text{num_resolve_at_root} < +\text{inf}$ and its default value is 0.

Outer Approximation Decomposition (B-OA)

milp_subsolver: Choose the subsolver to solve MILP sub-problems in OA decompositions. The default value for this string option is "Cbc_D".

Possible values:

- Cbc_D: Coin Branch and Cut with its default
- Cbc_Par: Coin Branch and Cut with passed parameters

oa_dec_time_limit: Specify the maximum number of seconds spent overall in OA decomposition iterations. The valid range for this real option is $0 \leq \text{oa_dec_time_limit} < +\text{inf}$ and its default value is 30.

Outer Approximation cuts generation

add_only_violated_oa: Do we add all OA cuts or only the ones violated by current point? The default value for this string option is "no".

Possible values:

- no: Add all cuts
- yes: Add only violated Cuts

cut_strengthening_type: Determines if and what kind of cut strengthening should be performed. The default value for this string option is "none".

Possible values:

- none: No strengthening of cuts.
- sglobal: Strengthen global cuts.
- uglobal-slocal: Unstrengthened global and strengthened local cuts
- sglobal-slocal: Strengthened global and strengthened local cuts

disjunctive_cut_type: Determine if and what kind of disjunctive cuts should be computed. The default value for this string option is "none".

Possible values:

- none: No disjunctive cuts.
- most-fractional: If discrete variables present, compute disjunction for most-fractional variable

oa_cuts_scope: Specify if OA cuts added are to be set globally or locally valid. The default value for this string option is "global".

Possible values:

- local: Cuts are treated as globally valid
- global: Cuts are treated as locally valid

tiny_element: Value for tiny element in OA cut

We will remove "cleanly" (by relaxing cut) an element lower than this. The valid range for this real option is $-0 \leq \text{tiny_element} < +\text{inf}$ and its default value is $1 \cdot 10^{-08}$.

very_tiny_element: Value for very tiny element in OA cut
 Algorithm will take the risk of neglecting an element lower than this. The valid range for this real option is $-0 \leq \text{very_tiny_element} < +\text{inf}$ and its default value is $1 \cdot 10^{-17}$.

Strong branching setup

candidate_sort_criterion: Choice of the criterion to choose candidates in strong-branching
 The default value for this string option is "best-ps-cost".
 Possible values:

- best-ps-cost: Sort by decreasing pseudo-cost
- worst-ps-cost: Sort by increasing pseudo-cost
- most-fractional: Sort by decreasing integer infeasibility
- least-fractional: Sort by increasing integer infeasibility

maxmin_crit_have_sol: Weight towards minimum in of lower and upper branching estimates when a solution has been found.
 The valid range for this real option is $0 \leq \text{maxmin_crit_have_sol} \leq 1$ and its default value is 0.1.

maxmin_crit_no_sol: Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.
 The valid range for this real option is $0 \leq \text{maxmin_crit_no_sol} \leq 1$ and its default value is 0.7.

min_number_strong_branch: Sets minimum number of variables for strong branching (overriding trust)
 The valid range for this integer option is $0 \leq \text{min_number_strong_branch} < +\text{inf}$ and its default value is 0.

number_before_trust_list: Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.
 The valid range for this integer option is $-1 \leq \text{number_before_trust_list} < +\text{inf}$ and its default value is the value of "number_before_trust".

number_look_ahead: Sets limit of look-ahead strong-branching trials
 The valid range for this integer option is $0 \leq \text{number_look_ahead} < +\text{inf}$ and its default value is 0.

number_strong_branch_root: Maximum number of variables considered for strong branching in root node.
 The valid range for this integer option is $0 \leq \text{number_strong_branch_root} < +\text{inf}$ and its default value is 2147483647.

setup_pseudo_frac: Proportion of strong branching list that has to be taken from most-integer-infeasible list.
 The valid range for this real option is $0 \leq \text{setup_pseudo_frac} \leq 1$ and its default value is 0.5.

trust_strong_branching_for_pseudo_cost: Whether or not to trust strong branching results for updating pseudo costs.
 The default value for this string option is "yes". Possible values are "no" and "yes".

Ipopt

All Ipopt options are available in CoinBonmin, please refer to section 6.6 for a detailed description. The default value of the following Ipopt parameters are changed in Gams/CoinBonmin:

- `mu_strategy` and `mu_oracle` are set, respectively, to `adaptive` and `probing` by default.
- `gamma_phi` and `gamma_theta` are set to 10^{-8} and 10^{-4} respectively. This has the effect of reducing the size of the filter in the line search performed by Ipopt.
- `required_infeasibility_reduction` is set to 0.1. This increases the required infeasibility reduction when Ipopt enters the restoration phase and should thus help detect infeasible problems faster.
- `expect_infeasible_problem` is set to `yes` which enables some heuristics to detect infeasible problems faster.
- `warm_start_init_point` is set to `yes` when a full primal/dual starting point is available (generally for all the optimizations after the continuous relaxation has been solved).
- `print_level` is set to 0 by default to turn off Ipopt output.
- `bound_relax_factor` is set to 10^{-10} .

3 CoinCbc and CoinCbcD

GAMS/CoinCBC brings the open source LP/MIP solver CBC to the broad audience of GAMS users.

CBC (COIN-OR Branch and Cut) is an open-source mixed integer programming solver working with the COIN-OR LP solver CLP and the COIN-OR Cut generator library Cgl. The code has been written primarily by John J. Forrest, who is the COIN-OR project leader for Cbc.

For more information we refer to

- the CBC web site <https://projects.coin-or.org/Cbc>,
- the Cgl web site <https://projects.coin-or.org/Cgl>, and
- the CLP web site <https://projects.coin-or.org/Clp>.

Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

Gams/CbcD is an experimental in-core communication link. It offers in-core communication between GAMS and the solver, making potentially large model scratch files unnecessary. This link supports all features of the traditional link except for the GAMS branch-cut-heuristic facility (BCH).

3.1 Model requirements

The CBC link in GAMS supports continuous, binary, integer, semicontinuous, semiinteger variables, special ordered sets of type 1 and 2, and branching priorities (see chapter 17.1 of the GAMS User's Guide).

3.2 Usage of CoinCbc

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

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

The above statement should appear before the Solve statement. If CoinCBC was specified as the default solver during GAMS installation, the above statement is not necessary.

There are many parameters which can affect the performance the CBCs Branch and Cut Algorithm. First just try with default settings and look carefully at the log file. Did cuts help? Did they take too long? Look at the output to see which cuts were effective and then do some tuning (see the option [cuts](#)). If the [preprocessing](#) reduced the size of the problem or strengthened many coefficients then it is probably wise to leave it on. Switch off [heuristics](#) which did not provide solutions. The other major area to look at is the search. Hopefully good solutions were obtained fairly early in the search so the important point is to select the best variable to branch on. See whether strong branching did a good job - or did it just take a lot of iterations. Adjust the options [strongbranching](#) and [trustpseudocosts](#).

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

```
cuts root
perturbation off
```

It will cause CoinCBC to use cut generators only in the root node and turns off the perturbation of the LP relaxation.

GAMS/CoinCBC supports the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. The GAMS BCH facility automates all major steps necessary to define, execute, and control the use of user defined routines within the framework of general purpose MIP codes. Currently supported are user defined cut generators and heuristics. Please see the BCH documentation at <http://www.gams.com/docs/bch.htm> for further information.

3.3 Summary of CoinCBC Options

Among all CBC options, the following GAMS parameters are currently supported in CoinCBC: [reslim](#), [iterlim](#), [nodlim](#), [optca](#), [optcr](#), [cheat](#), [cutoff](#).

Currently CoinCBC understands the following options:

General Options

iterlim	iteration limit
names	specifies whether variable and equation names should be given to CBC
reslim	resource limit (CPU time in seconds)
special	options passed unseen to CBC
writemps	create MPS file for problem

LP Options

idiotcrash	idiot crash
sprintcrash	sprint crash
sifting	synonym for sprint crash
crash	use crash method to get dual feasible
maxfactor	maximum number of iterations between refactorizations
crossover	crossover to simplex algorithm after barrier
dualpivot	dual pivot choice algorithm
primalpivot	primal pivot choice algorithm
perturbation	perturbation of problem
scaling	scaling method
presolve	switch for initial presolve of LP

<code>tol_dual</code>	dual feasibility tolerance
<code>tol_primal</code>	primal feasibility tolerance
<code>tol_presolve</code>	tolerance used in presolve
<code>passpresolve</code>	how many passes to do in presolve
<code>startalg</code>	LP solver for root node

MIP Options

<code>mipstart</code>	whether it should be tried to use the initial variable levels as initial MIP solution
<code>strategy</code>	switches on groups of features
<code>tol_integer</code>	tolerance for integrality
<code>sollim</code>	limit on number of solutions
<code>strongbranching</code>	strong branching
<code>trustpseudocosts</code>	after howmany nodes we trust the pseudo costs
<code>coststrategy</code>	how to use costs as priorities
<code>nodestrategy</code>	how to select nodes
<code>preprocess</code>	integer presolve
<code>threads</code>	number of threads to use (available on Unix variants only)
<code>printfrequency</code>	frequency of status prints
<code>increment</code>	increment of cutoff when new incumbent
<code>nodelim</code>	node limit
<code>nodlim</code>	node limit
<code>optca</code>	absolute stopping tolerance
<code>optcr</code>	relative stopping tolerance
<code>cutoff</code>	cutoff for objective function value

MIP Options for Cutting Plane Generators

<code>cutdepth</code>	depth in tree at which cuts are applied
<code>cut_passes_root</code>	number of cut passes at root node
<code>cut_passes_tree</code>	number of cut passes at nodes in the tree
<code>cuts</code>	global switch for cutgenerators
<code>cliquecuts</code>	Clique Cuts
<code>flowcovercuts</code>	Flow Cover Cuts
<code>gomorycuts</code>	Gomory Cuts
<code>knapsackcuts</code>	Knapsack Cover Cuts
<code>liftandprojectcuts</code>	Lift and Project Cuts
<code>mircuts</code>	Mixed Integer Rounding Cuts
<code>twomircuts</code>	Two Phase Mixed Integer Rounding Cuts
<code>probingcuts</code>	Probing Cuts
<code>reduceandsplitcuts</code>	Reduce and Split Cuts
<code>residualcapacitycuts</code>	Residual Capacity Cuts

MIP Options for Heuristics

<code>heuristics</code>	global switch for heuristics
<code>combinesolutions</code>	combine solutions heuristic
<code>dins</code>	distance induced neighborhood search
<code>divingrandom</code>	turns on random diving heuristic
<code>divingcoefficient</code>	coefficient diving heuristic
<code>divingfractional</code>	fractional diving heuristic

divingguided	guided diving heuristic
divinglinesearch	line search diving heuristic
divingpseudocost	pseudo cost diving heuristic
divingvectorlength	vector length diving heuristic
feaspump	feasibility pump
feaspump_passes	number of feasibility passes
greedyheuristic	greedy heuristic
localtreesearch	local tree search heuristic
naiveheuristics	naive heuristics
pivotandfix	pivot and fix heuristic
randomizedrounding	randomized rounding heuristics
rens	relaxation enforced neighborhood search
rins	relaxed induced neighborhood search
roundingheuristic	rounding heuristic
vubheuristic	VUB heuristic

MIP Options for the GAMS Branch Cut and Heuristic Facility

usercutcall	The GAMS command line to call the cut generator
usercutfirst	Calls the cut generator for the first n nodes
usercutfreq	Determines the frequency of the cut generator model calls
usercutinterval	Determines interval when to apply multiplier for frequency of cut generator model calls
usercutmult	Determines the multiplier for the frequency of the cut generator model calls
usercutnewint	Calls the cut generator if the solver found a new integer feasible solution
usergdxin	The name of the GDX file read back into CBC
usergdxname	The name of the GDX file exported from the solver with the solution at the node
usergdxnameinc	The name of the GDX file exported from the solver with the incumbent solution
usergdxprefix	Prefixes usergdxin, usergdxname, and usergdxnameinc
userheurcall	The GAMS command line to call the heuristic
userheurfirst	Calls the heuristic for the first n nodes
userheurfreq	Determines the frequency of the heuristic model calls
userheurinterval	Determines interval when to apply multiplier for frequency of heuristic model calls
userheurmult	Determines the multiplier for the frequency of the heuristic model calls
userheurnewint	Calls the heuristic if the solver found a new integer feasible solution
userheurobjfirst	Calls heuristic if LP value of node is closer to best bound than current incumbent
userjobid	Postfixes gdxname, gdxnameinc and gdxin
userkeep	Calls gamskeep instead of gams

3.4 Detailed Descriptions of CoinCBC Options

General Options

iterlim (*integer*)

For an LP, this is the maximum number of iterations to solve the LP. For a MIP, this option is ignored.

(*default = GAMS iterlim*)

names (*integer*)

This option causes GAMS names for the variables and equations to be loaded into Cbc. These names will then be used for error messages, log entries, and so forth. Turning names off may help if memory is very tight.

(*default = 0*)

0 Do not load variable and equation names.

1 Load variable and equation names.

reslim (*real*)

Maximum CPU time in seconds.

(*default = GAMS reslim*)

writemps (*string*)

Write the problem formulation in MPS format. The parameter value is the name of the MPS file.

special (*string*)

This parameter let you specify CBC options which are not supported by the GAMS/CoinCBC interface.

The string value given to this parameter is split up into parts at each space and added to the array of parameters given to CBC (in front of the -solve command). Hence, you can use it like the command line parameters for the CBC standalone version.

LP Options

idiotcrash (*integer*)

This is a type of ‘crash’ which works well on some homogeneous problems. It works best on problems with unit elements and right hand sides but will do something to any model. It should only be used before the primal simplex algorithm.

A positive number determines the number of passes that idiotcrash is called.

(*default = -1*)

-1 Let CLP decide by itself whether to use it.

0 Switch this method off.

sprintcrash (*integer*)

For long and thin problems this method may solve a series of small problems created by taking a subset of the columns. Cplex calls it ‘sifting’.

A positive number determines the number of passes that sprintcrash is called.

(*default = -1*)

-1 Let CLP decide by itself whether to use it.

0 Switch this method off.

sifting (*integer*)

Synonym for [sprintcrash](#).

(*default = -1*)

crash (*string*)

Determines whether CLP should use a crash algorithm to find a dual feasible basis.

(*default = off*)

off Switch off the creation of dual feasible basis by the crash method.

on Switch on the creation of dual feasible basis by the crash method.

solow_halim Switch on a crash variant due to Solow and Halim.

halim_solow Switch on a crash variant due to Solow and Halim with modifications of John J. Forrest.

maxfactor (*integer*)

Maximum number of iterations between refactorizations in CLP.

If this is left at the default value of 200 then CLP will guess at a value to use. CLP may decide to refactorize earlier for accuracy.

(*default = 200*)

crossover (*integer*)

Determines whether CLP should crossover to the simplex algorithm after the barrier algorithm finished.

Interior point algorithms do not obtain a basic solution. This option will crossover to a basic solution suitable for ranging or branch and cut.

(*default = 1*)

0 Turn off crossover to simplex algorithm after barrier algorithm finished.

1 Turn on crossover to simplex algorithm after barrier algorithm finished.

dualpivot (*string*)

Choice of the pivoting strategy in the dual simplex algorithm.

(*default = auto*)

auto Let CLP use a variant of the steepest choice method which starts like *partial*, i.e., scans only a subset of the primal infeasibilities, and later changes to full pricing when the factorization becomes denser.

dantzig Let CLP use the pivoting strategy due to Dantzig.

steepest Let CLP use the steepest choice method.

partial Let CLP use a variant of the steepest choice method which scans only a subset of the primal infeasibilities to select the pivot step.

primalpivot (*string*)

Choice of the pivoting strategy in the primal simplex algorithm.

(*default = auto*)

auto Let CLP use a variant of the exact devex method.

dantzig Let CLP use the pivoting strategy due to Dantzig.

steepest Let CLP use the steepest choice method.

partial Let CLP use a variant of the exact devex method which scans only a subset of the primal infeasibilities to select the pivot step.

exact Let CLP use the exact devex method.

change Let CLP initially use Dantzig pivot method until the factorization becomes denser.

perturbation (*integer*)

Determines whether CLP should perturb the problem before starting. Perturbation helps to stop cycling, but CLP uses other measures for this. However, large problems and especially ones with unit elements and unit right hand sides or costs benefit from perturbation. Normally CLP tries to be intelligent, but you can switch this off.

(*default = 1*)

0 Turns off perturbation of LP.

1 Turns on perturbation of LP.

scaling (*string*)

Scaling can help in solving problems which might otherwise fail because of lack of accuracy. It can also reduce the number of iterations. It is not applied if the range of elements is small. Both methods do several passes alternating between rows and columns using current scale factors from one and applying them to the other.

(*default = auto*)

off Turns off scaling.

auto Let CLP choose the scaling method automatically. It decides for one of these methods depending on which gives the better ratio of the largest element to the smallest one.

equilibrium Let CLP use an equilibrium based scaling method which uses the largest scaled element.

geometric Let CLP use a geometric based scaling method which uses the squareroot of the product of largest and smallest element.

presolve (*integer*)

Presolve analyzes the model to find such things as redundant constraints, constraints which fix some variables, constraints which can be transformed into bounds, etc. For the initial solve of any problem this is worth doing unless you know that it will have no effect.

(*default = 1*)

0 Turns off the initial presolve.

1 Turns on the initial presolve.

tol_dual (*real*)

The maximum amount the dual constraints can be violated and still be considered feasible.

(*default = 1e-7*)

tol_primal (*real*)

The maximum amount the primal constraints can be violated and still be considered feasible.

(*default = 1e-7*)

tol_presolve (*real*)

The tolerance used in presolve.

(*default = 1e-8*)

passpresolve (*integer*)

Normally Presolve does 5 passes but you may want to do less to make it more lightweight or do more if improvements are still being made. As Presolve will return if nothing is being taken out, you should not normally need to use this fine tuning.

(*default = 5*)

startalg (*string*)

Determines the algorithm to use for an LP or the initial LP relaxation if the problem is a MIP.

(*default = dual*)

primal Let CLP use the primal simplex algorithm.

dual Let CLP use the dual simplex algorithm.

barrier Let CLP use a primal dual predictor corrector algorithm.

MIP Options

mipstart (*integer*)

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

(*default = 0*)

0 Do not use the initial variable levels.

1 Try to use the initial variable levels as a MIP starting solution.

strategy (*integer*)

Setting strategy to 1 (the default) uses Gomory cuts using tolerance of 0.01 at root, does a possible restart after 100 nodes if Cbc can fix many variables and activates a diving and RINS heuristic and makes feasibility pump more aggressive.

(default = 1)

- 0 Use this setting for easy problems.
- 1 This is the default setting.
- 2 Use this setting for difficult problems.

tol_integer (*real*)

For an optimal solution, no integer variable may be farther than this from an integer value.

(default = 1e-6)

sollim (*integer*)

A limit on number of feasible solutions that CBC should find for a MIP.

(default = -1)

- 1 No limit on the number of feasible solutions.

strongbranching (*integer*)

Determines the number of variables to look at in strong branching.

In order to decide which variable to branch on, the code will choose up to this number of unsatisfied variables and try minimal up and down branches. The most effective one is chosen. If a variable is branched on many times then the previous average up and down costs may be used - see the option [trustpseudocosts](#).

(default = 5)

trustpseudocosts (*integer*)

Using strong branching computes pseudo-costs. This parameter determines after how many branches for a variable we just trust the pseudo costs and do not do any more strong branching.

(default = 5)

coststrategy (*string*)

This parameter influence the branching variable selection.

If turned on, then the variables are sorted in order of their absolute costs, and branching is done first on variables with largest cost. This primitive strategy can be surprisingly effective.

(default = off)

off Turns off a specific cost strategy.

priorities Assigns highest priority to variables with largest absolute cost.

columnorder Assigns the priorities 1, 2, 3,.. with respect to the column ordering.

binaryfirst Handles two sets of priorities such that binary variables get high priority.

binarylast Handles two sets of priorities such that binary variables get low priority.

length Assigns high priority to variables that are at most nonzero.

nodestrategy (*string*)

This determines the strategy used to select the next node from the branch and cut tree.

(default = fewest)

hybrid Let CBC do first a breath search on nodes with a small depth in the tree and then switch to choose nodes with fewest infeasibilities.

fewest This will let CBC choose the node with the fewest infeasibilities.

- depth This will let CBC always choose the node deepest in tree. It gives minimum tree size but may take a long time to find the best solution.
- upfewest This will let CBC choose the node with the fewest infeasibilities and do up branches first.
- downfewest This will let CBC choose the node with the fewest infeasibilities and do down branches first.
- updepth This will let CBC choose the node deepest in tree and do up branches first.
- downdepth This will let CBC choose the node deepest in tree and do down branches first.

preprocess (*string*)

This option controls the MIP specific presolve routines. They try to reduce the size of the model in a similar way to presolve and also try to strengthen the model. This can be very useful and is worth trying.

(*default = on*)

- off Turns off the presolve routines.
- on Turns on the presolve routines.
- equal Turns on the presolve routines and let CBC turn inequalities with more than 5 elements into equalities (cliques) by adding slack variables.
- equalall Turns on the presolve routines and let CBC turn all inequalities into equalities by adding slack variables.
- sos This option let CBC search for rows with upper bound 1 and where all nonzero coefficients are 1 and creates special ordered sets if the sets are not overlapping and all integer variables (except for at most one) are in the sets.
- trysos This option is similar to sos, but allows any number integer variables to be outside of the sets.

threads (*integer*)

This option controls the multithreading feature of CBC, which is currently available only on Unix variants. A number between 1 and 100 sets the number of threads used for parallel branch and bound. A number $100 + n$ with n between 1 and 100 says that n threads are used to parallelize the branch and bound, but also heuristics such as RINS which do branch and bound on a reduced model also use threads. A number $200 + n$ with n between 1 and 100 says that n threads are used to parallelize the branch and bound, but also the cut generators at the root node (i.e., before threads are useful) are run in parallel. A number $300 + n$ with n between 1 and 100 combines the $100 + n$ and $200 + n$ options. A number $400 + n$ with n between 1 and 100 says that n threads are used in sub-trees. Thus, n threads are used to parallelize the branch and bound, but also heuristics use threads and the cut generators at the root node are run in parallel. The $100 + n$, $200 + n$, and $300 + n$ options are experimental.

printfrequency (*integer*)

Controls the number of nodes that are evaluated between status prints.

(*default = 0*)

- 0 Automatic choice, which is 100 for large problems and 1000 for small problems.

increment (*real*)

A valid solution must be at least this much better than last integer solution.

If this option is not set then it CBC will try and work one out. E.g., if all objective coefficients are multiples of 0.01 and only integer variables have entries in objective then this can be set to 0.01.

(*default = GAMS cheat*)

nodelim (*integer*)

Maximum number of nodes that are considered in the Branch and Bound.

(*default = GAMS nodlim*)

nodlim (*integer*)

Maximum number of nodes that are considered in the Branch and Bound. This option is overwritten by nodelim, if set.

(*default = GAMS nodlim*)

optca (real)

Absolute optimality criterion for a MIP. CBC stops if the gap between the best known solution and the best possible solution is less than this value.

(default = GAMS optca)

optcr (real)

Relative optimality criterion for a MIP. CBC stops if the relative gap between the best known solution and the best possible solution is less than this value.

(default = GAMS optcr)

cutoff (real)

CBC stops if the objective function values exceeds (in case of maximization) or falls below (in case of minimization) this value.

(default = GAMS cutoff)

MIP Options for Cutting Plane Generators**cutdepth (integer)**

If the depth in the tree is a multiple of cutdepth, then cut generators are applied.

Cut generators may be off, on only at the root, on if they look useful, or on at some interval. Setting this option to a positive value K let CBC call a cutgenerator on a node whenever the depth in the tree is a multiple of K.

(default = -1)

-1 Does not turn on cut generators because the depth of the tree is a multiple of a value.

cut_passes_root (integer)

Determines the number of rounds that the cut generators are applied in the root node.

A negative value $-n$ means that n passes are also applied if the objective does not drop.

(default = 100 passes if the MIP has less than 500 columns, 100 passes (but stop if the drop in the objective function value is small) if it has less than 5000 columns, and 20 passes otherwise)

cut_passes_tree (integer)

Determines the number of rounds that the cut generators are applied in the nodes of the tree other than the root node.

A negative value $-n$ means that n passes are also applied if the objective does not drop.

(default = 1)

cuts (string)

A global switch to turn on or off the cutgenerators.

This can be used to switch on or off all default cut generators. Then you can set individual ones off or on using the specific options.

(default = on)

off Turns off all cut generators.

on Turns on all default cut generators and CBC will try them in the branch and cut tree (see the option [cutdepth](#) on how to fine tune the behaviour).

root Let CBC generate cuts only at the root node.

ifmove Let CBC use cut generators in the tree if they look as if they are doing some good and moving the objective value.

forceon Turns on all default cut generators and force CBC to use the cut generator at every node.

cliquecuts (*string*)

Determines whether and when CBC should try to generate clique cuts. See the option [cuts](#) for an explanation on the different values.

Clique cuts are of the form “sum of a set of variables ≤ 1 ”.

Reference: M. Eso, Parallel branch and cut for set partitioning, Cornell University, 1999.

(*default = ifmove*)

flowcovercuts (*string*)

Determines whether and when CBC should try to generate flow cover cuts.

See the option [cuts](#) for an explanation on the different values.

The flow cover cut generator generates lifted simple generalized flow cover inequalities. Since flow cover inequalities are generally not facet-defining, they are lifted to obtain stronger inequalities. Although flow cover inequalities requires a special problem structure to be generated, they are quite useful for solving general mixed integer linear programs.

Reference: Z. Gu, G.L. Nemhauser, M.W.P. Savelsbergh, Lifted flow cover inequalities for mixed 0-1 integer programs, Math. Programming A 85 (1999) 439-467.

(*default = ifmove*)

gomorycuts (*string*)

Determines whether and when CBC should try to generate mixed-integer Gomory cuts.

See the option [cuts](#) for an explanation on the different values.

Reference: Laurence A. Wolsey, Integer Programming, Wiley, John & Sons, (1998) 124-132.

(*default = ifmove*)

knapsackcuts (*string*)

Determines whether and when CBC should try to generate knapsack cover cuts.

See the option [cuts](#) for an explanation on the different values.

The knapsack cover cut generator looks for a series of different types of minimal covers. If a minimal cover is found, it lifts the associated minimal cover inequality and adds the lifted cut to the cut set.

Reference: S. Martello, and P. Toth, Knapsack Problems, Wiley, 1990, p30.

(*default = ifmove*)

liftandprojectcuts (*string*)

Determines whether and when CBC should try to generate lift and project cuts. They might be expensive to compute, thus they are switched off by default.

See the option [cuts](#) for an explanation on the different values.

Reference: E. Balas and M. Perregaard, A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. Math. Program., 94(203,Ser. B):221-245,2003.

(*default = off*)

mircuts (*string*)

Determines whether and when CBC should try to generate mixed integer rounding cuts.

See the option [cuts](#) for an explanation on the different values.

Reference: H. Marchand and L. A. Wolsey, Aggregation and Mixed Integer Rounding to Solve MIPs, Operations Research, 49(3), (2001).

(*default = ifmove*)

twomircuts (*string*)

Determines whether and when CBC should try to generate two phase mixed integer rounding cuts.

See the option [cuts](#) for an explanation on the different values.

Reference: S. Dash, and O. Guenluek, Valid Inequalities Based on Simple Mixed-integer Sets, to appear in Math. Programming.

(*default = root*)

probingcuts (*string*)

Determines whether and when CBC should try to generate cuts based on probing.

Additional to the values for the option [cuts](#) three more values are possible here.

Reference: M. Savelsbergh, Preprocessing and Probing Techniques for Mixed Integer Programming Problems, ORSA Journal on Computing 6 (1994), 445.

(*default = ifmove*)

off Turns off Probing.

on Turns on Probing and CBC will try it in the branch and cut tree (see the option [cutdepth](#) how to fine tune this behaviour).

root Let CBC do Probing only at the root node.

ifmove Let CBC do Probing in the tree if it looks as if it is doing some good and moves the objective value.

forceon Turns on Probing and forces CBC to do Probing at every node.

forceonbut Turns on Probing and forces CBC to call the cut generator at every node, but does only probing, not strengthening etc.

forceonstrong If CBC is forced to turn Probing on at every node (by setting this option to force), but this generator produces no cuts, then it is actually turned on only weakly (i.e., just every now and then). Setting forceonstrong forces CBC strongly to do probing at every node.

forceonbutstrong This is like forceonstrong, but does only probing (column fixing) and turns off row strengthening, so the matrix will not change inside the branch and bound.

reduceandsplitcuts (*string*)

Determines whether and when CBC should try to generate reduced and split cuts.

See the option [cuts](#) for an explanation on the different values.

Reduce and split cuts are variants of Gomory cuts. Starting from the current optimal tableau, linear combinations of the rows of the current optimal simplex tableau are used for generating Gomory cuts. The choice of the linear combinations is driven by the objective of reducing the coefficients of the non basic continuous variables in the resulting row.

Reference: K. Anderson, G. Cornuejols, and Yanjun Li, Reduce-and-Split Cuts: Improving the Performance of Mixed Integer Gomory Cuts, Management Science 51 (2005).

(*default = off*)

residualcapacitycuts (*string*)

Determines whether and when CBC should try to generate residual capacity cuts.

See the option [cuts](#) for an explanation on the different values.

These inequalities are particularly useful for Network Design and Capacity Planning models.

References:

T.L. Magnanti, P. Mirchandani, and R. Vachani, The convex hull of two core capacitated network design problems, Math. Programming, 60 (1993), pp. 233-250.

A. Atamturk and D. Rajan, On splittable and unsplittable flow capacitated network design arc-set polyhedra, Math. Programming, 92 (2002), pp. 315-333.

(*default = off*)

MIP Options for Heuristics

heuristics (*integer*)

This parameter can be used to switch on or off all heuristics, except for the local tree search as it dramatically alters the search. Then you can set individual ones off or on.

(default = 1)

- 0 Turns all MIP heuristics off.
- 1 Turns all MIP heuristics on (except [local tree search](#)).

combinesolutions (*integer*)

This parameter control the use of a heuristic which does branch and cut on the given problem by just using variables which have appeared in one or more solutions. It is obviously only tried after two or more solutions.

(default = 1)

- 0 Turns the combine solutions heuristic off.
- 1 Turns the combine solutions heuristic on.

dins (*integer*)

This parameter control the use of the distance induced neighborhood search heuristic.

(default = 0)

- 0 Turns the distance induced neighborhood search off.
- 1 Turns the distance induced neighborhood search on.

divingrandom (*integer*)

This switches on a random diving heuristic at various times.

(default = 0)

- 0 . Turns the random diving heuristics off.
- 1 . Turns the random diving heuristics on.

divingcoefficient (*integer*)

This switches on the coefficient diving heuristic.

(default = 1)

- 0 . Turns the coefficient diving heuristics off.
- 1 . Turns the coefficient diving heuristics on.

divingfractional (*integer*)

This switches on the fractional diving heuristic.

(default = 0)

- 0 . Turns the fractional diving heuristics off.
- 1 . Turns the fractional diving heuristics on.

divingguided (*integer*)

This switches on the guided diving heuristic.

(default = 0)

- 0 . Turns the guided diving heuristics off.
- 1 . Turns the guided diving heuristics on.

divinglinesearch (*integer*)

This switches on the line search diving heuristic.

(default = 0)

- 0 . Turns the line search diving heuristics off.
- 1 . Turns the linesearch diving heuristics on.

divingpseudocost (*integer*)

This switches on the pseudo costs diving heuristic.

(default = 0)

- 0 . Turns the pseudo costs diving heuristics off.
- 1 . Turns the pseudo costs diving heuristics on.

divingvectorlength (*integer*)

This switches on the vector length diving heuristic.

(default = 0)

- 0 . Turns the vector length diving heuristics off.
- 1 . Turns the vector length diving heuristics on.

feaspump (*integer*)

This parameter control the use of the feasibility pump heuristic at the root.

This is due to Fischetti and Lodi and uses a sequence of LPs to try and get an integer feasible solution. Some fine tuning is available by the option [feaspump_passes](#). Reference: M. Fischetti, F. Glover, and A. Lodi, The feasibility pump, Math. Programming, 104 (2005), pp. 91-104.

(default = 1)

- 0 Turns the feasibility pump off.
- 1 Turns the feasibility pump on.

feaspump_passes (*integer*)

This fine tunes the feasibility pump heuristic by setting the number of passes.

(default = 20)

greedyheuristic (*string*)

This parameter control the use of a pair of greedy heuristic which will try to obtain a solution. It may just fix a percentage of variables and then try a small branch and cut run.

(default = on)

- off Turns off the greedy heuristic.
- on Turns on the greedy heuristic.
- root Turns on the greedy heuristic only for the root node.

localtreesearch (*integer*)

This parameter control the use of a local search algorithm when a solution is found.

It is from Fischetti and Lodi and is not really a heuristic although it can be used as one (with limited functionality). This heuristic is not controlled by the option [heuristics](#).

Reference: M. Fischetti and A. Lodi, Local Branching, Math. Programming B, 98 (2003), pp. 23-47.

(default = 0)

- 0 Turns the local tree search off.
- 1 Turns the local tree search on.

naiveheuristics (*integer*)

This parameter controls the use of some naive heuristics, e.g., fixing of all integers with costs to zero. `>`
(*default = 0*)

- 0 . Turns the naive heuristics off.
- 1 . Turns the naive heuristics on.

randomizedrounding (*integer*)

This parameter controls the use of the randomized rounding heuristic.
(*default = 0*)

- 0 . Turns the randomized rounding heuristic off.
- 1 . Turns the randomized rounding heuristic on.

rens (*integer*)

This parameter controls the use of the relaxation enforced neighborhood search heuristic. `>`
(*default = 0*)

- 0 . Turns the relaxation enforced neighborhood search off.
- 1 . Turns the relaxation enforced neighborhood search on.

pivotandfix (*integer*)

This parameter controls the use of the pivot and fix heuristic.
(*default = 0*)

- 0 . Turns the naive pivot and fix heuristic off.
- 1 . Turns the naive pivot and fix heuristic on.

rins (*integer*)

This parameter control the use of the relaxed induced neighborhood search heuristic.

This heuristic compares the current solution with the best incumbent, fixes all discrete variables with the same value, presolves the problem, and does a branch and bound for 200 nodes.

Reference: E. Danna, E. Rothberg, and C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, *Math. Programming*, 102 (1) (2005), pp. 71-91.

(*default = 0*)

- 0 Turns the relaxed induced neighborhood search off.
- 1 Turns the relaxed induced neighborhood search on.

roundingheuristic (*integer*)

This parameter control the use of a simple (but effective) rounding heuristic at each node of tree.
(*default = 1*)

- 0 Turns the rounding heuristic off.
- 1 Turns the rounding heuristic on.

vubheuristic (*integer*)

This parameter control the use of the VUB heuristic. If it is set (between -2 and 20), Cbc will try and fix some integer variables

(*default = -1*)

MIP Options for the GAMS Branch Cut and Heuristic Facility

usercutcall (*string*)

The GAMS command line (minus the gams executable name) to call the cut generator.

usercutfirst (*integer*)

Calls the cut generator for the first n nodes.

(default = 10)

usercutfreq (*integer*)

Determines the frequency of the cut generator model calls.

(default = 10)

usercutinterval (*integer*)

Determines the interval when to apply the multiplier for the frequency of the cut generator model calls. See `userheurinterval` for details.

(default = 100)

usercutmult (*integer*)

Determines the multiplier for the frequency of the cut generator model calls.

(default = 2)

usercutnewint (*integer*)

Calls the cut generator if the solver found a new integer feasible solution.

(default = 1)

0 . Do not call cut generator because a new integer feasible solution is found.

1 . Let CBC call the cut generator if a new integer feasible solution is found.

usergdxin (*string*)

The name of the GDX file read back into CBC.

(default = bchin.gdx)

usergdxname (*string*)

The name of the GDX file exported from the solver with the solution at the node.

(default = bchout.gdx)

usergdxnameinc (*string*)

The name of the GDX file exported from the solver with the incumbent solution.

(default = bchout_i.gdx)

usergdxprefix (*string*)

Prefixes to use for `usergdxin`, `usergdxname`, and `usergdxnameinc`.

userheurcall (*string*)

The GAMS command line (minus the gams executable name) to call the heuristic.

userheurfirst (*integer*)

Calls the heuristic for the first n nodes.

(default = 10)

userheurfreq (*integer*)

Determines the frequency of the heuristic model calls.

(default = 10)

userheurinterval (*integer*)

Determines the interval when to apply the multiplier for the frequency of the heuristic model calls. For example, for the first 100 (userheurinterval) nodes, the solver call every 10th (userheurfreq) node the heuristic. After 100 nodes, the frequency gets multiplied by 10 (userheurmultip), so that for the next 100 node the solver calls the heuristic every 20th node. For nodes 200-300, the heuristic get called every 40th node, for nodes 300-400 every 80th node and after node 400 every 100th node.

(default = 100)

userheurmultip (*integer*)

Determines the multiplier for the frequency of the heuristic model calls.

(default = 2)

userheurnewint (*integer*)

Calls the heuristic if the solver found a new integer feasible solution.

(default = 1)

- 0 . Do not call heuristic because a new integer feasible solution is found.
- 1 . Let CBC call the heuristic if a new integer feasible solution is found.

userheurobjfirst (*integer*)

Similar to userheurfirst but only calls the heuristic if the relaxed objective value promises a significant improvement of the current incumbent, i.e., the LP value of the node has to be closer to the best bound than the current incumbent.

(default = 0)

userjobid (*string*)

Postfixes to use for gdxname, gdxnameinc, and gdxin.

userkeep (*integer*)

Calls gamskeep instead of gams

(default = 0)

4 CoinCouenne

GAMS/CoinCouenne brings the open source MINLP solver Couenne to the broad audience of GAMS users.

Couenne (**C**onvex **O**ver and **U**nder **E**nvelopes for **N**onlinear **E**stimation) is an open-source solver for nonconvex mixed-integer nonlinear programming (MINLPs). The code is developed in a joined project of IBM, Carnegie Mellon University, and Lehigh University. The COIN-OR project leader for Couenne is Pietro Belotti.

Couenne solves convex and nonconvex MINLPs by an LP based spatial branch-and-bound algorithm that is similar to BARON. The implementation extends Bonmin by routines to compute valid linear outer approximations for nonconvex problems and methods for bound tightening and branching on nonlinear continuous variables.

For more information we refer to

- the Couenne web site <https://projects.coin-or.org/Couenne> and
- the paper P. Belotti, J. Lee, L. Liberti, F. Margot and A. Waechter, Branching and bounds tightening techniques for non-convex MINLP, *IBM Research Report* RC24620, 2008.

Most of the Couenne documentation in the section is taken from the Couenne manual available on the Couenne web site.

4.1 Model requirements

Couenne can handle mixed-integer nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. The Couenne link in GAMS supports continuous, binary, and integer variables, but no special ordered sets, semi-continuous or semi-integer variables (see chapter 17.1 of the GAMS User's Guide).

If GAMS/CoinCouenne is called for a linear model, the interface directly calls CBC.

4.2 Usage of CoinCouenne

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

```
Option MINLP = CoinCouenne;      { or LP, RMIP, MIP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP }
```

The above statement should appear before the Solve statement. If CoinCouenne was specified as the default solver during GAMS installation, the above statement is not necessary.

4.3 Specification of CoinCouenne Options

A Couenne option file contains Ipopt, Bonmin, and Couenne options, for clarity all Bonmin options should be preceded with the prefix “bonmin.” and all Couenne options should be preceded with the prefix “couenne.”. All Ipopt and many Bonmin options are available in CoinCouenne, please refer to sections 6.6 and 2.4 for a detailed description. The scheme to name option files is the same as for all other GAMS solvers. Specifying `optfile=1` let Gams/CoinCouenne read `coincouenne.opt`, `optfile=2` corresponds to `coincouenne.op2`, and so on. The format of the option file is the same as for Ipopt (see Section 6.5).

GAMS/CoinCouenne understands currently the following GAMS parameters: `reslim` (time limit), `nodlim` (node limit), `cutoff`, `optca` (absolute gap tolerance), and `optcr` (relative gap tolerance). You can set them either on the command line, e.g. `nodlim=1000`, or inside your GAMS program, e.g. `Option nodlim=1000;`.

4.4 Description of CoinCouenne Options

The following tables gives the list of options together with their types, default values and availability in each of the four main algorithms. The column labeled ‘type’ indicates the type of the parameter (‘F’ stands for float, ‘I’ for integer, and ‘S’ for string). The column labeled ‘default’ indicates the global default value.

Option	type	default
Output options		
<code>boundtightening_print_level</code>	I	0
<code>branching_print_level</code>	I	0
<code>convexifying_print_level</code>	I	0
<code>disjcuts_print_level</code>	I	4
<code>nlpheur_print_level</code>	I	4
<code>problem_print_level</code>	I	2
<code>reformulate_print_level</code>	I	4
Reformulation and Linearization options		
<code>convexification_cuts</code>	I	1
<code>convexification_type</code>	S	current-point-only
<code>convexification_points</code>	I	4
<code>violated_cuts_only</code>	S	yes
<code>delete_redundant</code>	S	yes
<code>use_quadratic</code>	S	no

Option	type	default
Branching options		
cont_var_priority	I	2000
branch_conv_cuts	S	yes
branch_fbbt	S	yes
branch_pt_select	S	mid-point
branch_pt_select_cube	S	branch_pt_select
branch_pt_select_div	S	branch_pt_select
branch_pt_select_exp	S	branch_pt_select
branch_pt_select_log	S	branch_pt_select
branch_pt_select_negpow	S	branch_pt_select
branch_pt_select_pow	S	branch_pt_select
branch_pt_select_prod	S	branch_pt_select
branch_pt_select_sqr	S	branch_pt_select
branch_pt_select_trig	S	branch_pt_select
branch_lp_clamp	F	0.2
branch_lp_clamp_cube	F	0.2
branch_lp_clamp_div	F	0.2
branch_lp_clamp_exp	F	0.2
branch_lp_clamp_log	F	0.2
branch_lp_clamp_negpow	F	0.2
branch_lp_clamp_pow	F	0.2
branch_lp_clamp_prod	F	0.2
branch_lp_clamp_sqr	F	0.2
branch_lp_clamp_trig	F	0.2
branch_midpoint_alpha	F	0.25
branching_object	S	var_obj
red_cost_branching	S	no
pseudocost_mult	S	interval_br_rev
pseudocost_mult_lp	S	no
Bound tightening options		
feasibility_bt	S	yes
aggressive_fbbt	S	yes
optimality_bt	S	yes
redcost_bt	S	yes
log_num_abt_per_level	I	2
log_num_obbt_per_level	I	1
Disjunctive cut options		
minlp_disj_cuts	I	0
disj_depth_level	I	5
disj_depth_stop	I	20
disj_active_cols	S	no
disj_active_rows	S	no
disj_cumulative	S	no
disj_init_number	I	10
disj_init_perc	F	0.5
Nonlinear solver options		
local_optimization_heuristic	S	yes
log_num_local_optimization_per_level	I	2
Tolerance options		
feas_tolerance	F	10^{-5}

Option	type	default
MIP cut generators options		
Gomory_cuts	I	0
clique_cuts	I	0
cover_cuts	I	0
flow_covers_cuts	I	0
lift_and_project_cuts	I	0
mir_cuts	I	0
2mir_cuts	I	0
probing_cuts	I	0
reduce_split_cuts	I	0

Output options

boundtightening_print_level: Output level for bound tightening code in Couenne

The valid range for this integer option is $-2 \leq \text{boundtightening_print_level} \leq 12$ and its default value is 0.

branching_print_level: Output level for braching code in Couenne

The valid range for this integer option is $-2 \leq \text{branching_print_level} \leq 12$ and its default value is 0.

convexifying_print_level: Output level for convexifying code in Couenne

The valid range for this integer option is $-2 \leq \text{convexifying_print_level} \leq 12$ and its default value is 0.

disjcuts_print_level: Output level for disjunctive cuts in Couenne

The valid range for this integer option is $-2 \leq \text{disjcuts_print_level} \leq 12$ and its default value is 4.

nlphour_print_level: Output level for NLP heuristic in Couenne

The valid range for this integer option is $-2 \leq \text{nlphour_print_level} \leq 12$ and its default value is 4.

problem_print_level: Output level for problem manipulation code in Couenne. An option value of 4 prints out the initial problem, while a value of 7 prints the reformulated problem as well.

The valid range for this integer option is $-2 \leq \text{problem_print_level} \leq 12$ and its default value is 2.

reformulate_print_level: Output level for reformulating problems in Couenne

The valid range for this integer option is $-2 \leq \text{reformulate_print_level} \leq 12$ and its default value is 4.

Reformulation and Linearization options

convexification_cuts: Specify the frequency (in terms of nodes) at which Couenne ECP cuts are generated.

A frequency of 0 amounts to never solve the NLP relaxation. The valid range for this integer option is $-99 \leq \text{convexification_cuts}$ and its default value is 1.

convexification_type: Determines in which point the linear over/under-estimators are generated

The default value for this string option is "current-point-only".

Possible values:

- current-point-only: Only at current optimum of relaxation
- uniform-grid: Points chosen in a uniform grid between the bounds of the problem
- around-current-point: At points around current optimum of relaxation

convexification_points: Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point.

For the lower envelopes of convex functions, this is the number of points where a supporting hyperplane is generated. This only holds for the initial linearization, as all other linearizations only add at most one cut per expression.

The valid range for this integer option is $0 \leq \text{convexification_points}$ and its default value is 4.

violated_cuts_only: Yes if only violated convexification cuts should be added.

The default value for this string option is "yes".

Possible values: "yes" and "no".

delete_redundant: Eliminate redundant variables, which appear in the problem as $x_k = x_h$.

The default value for this string option is in most cases "yes". If the GAMS interface observes tiny constants in the nonlinear expressions, then the default is "no".

Possible values:

- no: Keep redundant variables, making the problem a bit larger
- yes: Eliminate redundant variables (the problem will be equivalent, only smaller)

use_quadratic: disable decomposition of quadratic expressions

If enabled, then quadratic forms are not reformulated and therefore decomposed as a sum of auxiliary variables, each associated with a bilinear term, but rather taken as a whole expression. Envelopes for these expressions are generated through α -convexification. The default value for this string option is "no".

Possible values:

- no: Use an auxiliary for each bilinear term
- yes: Create only one auxiliary for a quadratic expression

Branching options

cont_var_priority: Priority of continuous variable branching

When branching, this is compared to the priority of integer variables, whose priority is fixed to 1000. Higher values mean smaller priority, so if this parameter is set to 1001 or higher, if a branch-and-bound node has at least one integer variable whose value is fractional, then branching will be performed on that variable. The valid range for this integer option is $1 \leq \text{cont_var_priority}$ and its default value is 2000.

branch_conv_cuts: Apply convexification cuts before branching (for now only within strong branching).

After applying a branching rule and before resolving the subproblem, generate a round of linearization cuts with the new bounds enforced by the rule. The default value for this string option is "yes".

Possible values: "yes" and "no".

branch_fbbt: Apply bound tightening before branching

After applying a branching rule and before re-solving the subproblem, apply Bound Tightening. The default value for this string option is "yes".

Possible values: "yes" and "no".

branch_pt_select: Chooses branching point selection strategy.

The default value for this string option is "mid-point".

Possible values:

- lp-clamped: LP point clamped in $[k, 1 - k]$ of the bound intervals (k defined by lp_clamp)

- **lp-central:** LP point if within $[k, 1 - k]$ of the bound intervals, middle point otherwise (k defined by `branch_lp_clamp`)
- **balanced:** minimizes max distance from curve to convexification
- **min-area:** minimizes total area of the two convexifications
- **mid-point:** convex combination of current point and mid point

branch_pt_select_cube: Chooses branching point selection strategy for cube operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_div: Chooses branching point selection strategy for division operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_exp: Chooses branching point selection strategy for exp operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_log: Chooses branching point selection strategy for log operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_negpow: Chooses branching point selection strategy for negpow operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_pow: Chooses branching point selection strategy for power operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_prod: Chooses branching point selection strategy for product operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_sqr: Chooses branching point selection strategy for square operator. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_pt_select_trig: Chooses branching point selection strategy for trigonometric operators. The default value for this string option is the value of “branch_pt_select”. Possible values: see “branch_pt_select”.

branch_lp_clamp: Defines safe interval percentage for using LP point as a branching point. The valid range for this real option is $0 \leq \text{branch_lp_clamp} \leq 1$ and its default value is 0.2.

branch_lp_clamp_cube: Defines safe interval percentage for using LP point as a branching point for cube. The valid range for this real option is $0 \leq \text{branch_lp_clamp_cube} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_div: Defines safe interval percentage for using LP point as a branching point for div. The valid range for this real option is $0 \leq \text{branch_lp_clamp_div} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_exp: Defines safe interval percentage for using LP point as a branching point for exp. The valid range for this real option is $0 \leq \text{branch_lp_clamp_exp} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_log: Defines safe interval percentage for using LP point as a branching point for log. The valid range for this real option is $0 \leq \text{branch_lp_clamp_log} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_negpow: Defines safe interval percentage for using LP point as a branching point for neg-pow. The valid range for this real option is $0 \leq \text{branch_lp_clamp_negpow} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_pow: Defines safe interval percentage for using LP point as a branching point for power. The valid range for this real option is $0 \leq \text{branch_lp_clamp_pow} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_prod: Defines safe interval percentage for using LP point as a branching point for products. The valid range for this real option is $0 \leq \text{branch_lp_clamp_prod} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_sqr: Defines safe interval percentage for using LP point as a branching point for square. The valid range for this real option is $0 \leq \text{branch_lp_clamp_sqr} \leq 0.5$ and its default value is 0.2.

branch_lp_clamp_trig: Defines safe interval percentage for using LP point as a branching point for trigonometric expressions. The valid range for this real option is $0 \leq \text{branch_lp_clamp_trig} \leq 0.5$ and its default value is 0.2.

branch_midpoint_alpha: Defines convex combination of mid point and current LP point: $b = \alpha x_{lp} + (1 - \alpha)(lb + ub)/2$. The valid range for this real option is $0 \leq \text{branch_midpoint_alpha} \leq 1$ and its default value is 0.25.

branching_object: Defines the source of infeasibility of a problem.

The option “var_obj” indicates that infeasibility is associated with each variable. For example, suppose the optimal solution to the LP relaxation is denoted by (x^*, w^*) . If two auxiliary variables $w_2 = f_2(x_1)$ and $w_3 = f_3(x_1)$ have an optimal value in the LP solution such that $w_2^* \neq f_2(x_1^*)$ and $w_3^* \neq f_3(x_1^*)$, then x_1 will be associated an “infeasibility” measure dependent on $|w_2^* - f_2(x_1^*)|$ and $|w_3^* - f_3(x_1^*)|$. With the option “expr_obj”, the infeasibility is attributed to auxiliaries w_2 and w_3 , but this is not recommended. Finally, using “vt_obj” allows to use Violation Transfer, a branching variable selection technique used in BARON. The default value for this string option is “var_obj”.

Possible values:

- vt_obj: use Violation Transfer from Tawarmalani and Sahinidis (BARON)
- var_obj: use one object for each variable
- expr_obj: use one object for each nonlinear expression

red_cost_branching: Apply Reduced Cost Branching (instead of the Violation Transfer)

Setting this option requires setting “branching_object” to “vt_obj”. The default value for this string option is “no”.

Possible values:

- no: Use Violation Transfer with $\sum |\pi_i a_{ij}|$
- yes: Use Reduced cost branching with $|\sum \pi_i a_{ij}|$

pseudocost_mult: Multipliers of pseudocosts for estimating and updating estimation of bound
The default value for this string option is “interval_br_rev”.

Possible values:

- infeasibility: infeasibility of the variable (see “branching_object”)
- projectDist: distance between current LP point and resulting branches’ LP points
- interval_lp: width of the interval between variable bounds and current LP point
- interval_lp_rev: similar to interval_lp, reversed
- interval_br: width of the interval between variable bounds and branching point
- interval_br_rev: similar to interval_br, reversed

pseudocost_mult_lp: Use distance between LP points to update multipliers of pseudocosts after simulating branching

The default value for this string option is “no”.

Possible values: “yes” and “no”

Bound tightening options

feasibility_bt: Feasibility-based (cheap) bound tightening (FBBT)

A pre-processing technique to reduce the bounding box, before the generation of linearization cuts. This is a quick and effective way to reduce the solution set, and it is highly recommended to keep it active. The default value for this string option is “yes”.

Possible values: “yes” and “no”.

aggressive_fbbt: Aggressive feasibility-based bound tightening (to use with NLP points)

Aggressive FBBT is a version of probing that also allows to reduce the solution set, although it is not as quick as FBBT. It can be applied up to a certain depth of the B&B tree – see “log_num_abt_per_level”. In general, this option is useful but can be switched off if a problem is too large and seems not to benefit from it. The default value for this string option is “yes”.

Possible values: “yes” and “no”

optimality_bt: Optimality-based (expensive) bound tightening (OBBT)

This is another bound reduction technique aiming at reducing the solution set by looking at the initial LP relaxation. This technique is computationally expensive, and should be used only when necessary. The default value for this string option is “yes”.

Possible values: “yes” and “no”

redcost_bt: Reduced cost bound tightening

This bound reduction technique uses the reduced costs of the LP in order to infer better variable bounds. The default value for this string option is “yes”.

Possible values: “yes” and “no”

log_num_abt_per_level: Specify the frequency (in terms of nodes) for aggressive bound tightening.

If -1 , apply at every node (expensive!). If 0 , apply at the root node only. If $k \geq 0$, apply with probability 2^{k-d} , where d is the current depth of the B&B tree. The valid range for this integer option is $-1 \leq \text{log_num_abt_per_level}$ and its default value is 2 .

log_num_obbt_per_level: Specify the frequency (in terms of nodes) for optimality-based bound tightening. If -1 , apply at every node (expensive!). If 0 , apply at the root node only. If $k \geq 0$, apply with probability 2^{k-d} , where d is the current depth of the B&B tree. The valid range for this integer option is $-1 \leq \text{log_num_obbt_per_level}$ and its default value is 1 .

Disjunctive cut options

minlp_disj_cuts: The frequency (in terms of nodes) at which Couenne disjunctive cuts are generated. A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number $-n$ means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether. The valid range for this integer option is $-99 \leq \text{minlp_disj_cuts}$ and its default value is 0 .

disj_depth_level: Depth of the B&B tree when to start decreasing the number of objects that generate disjunctions.

This has a similar behavior as `log_num_obbt_per_level`. A value of -1 means that generation can be done at all nodes. The valid range for this integer option is $-1 \leq \text{disj_depth_level}$ and its default value is 5 .

disj_depth_stop: Depth of the B&B tree where separation of disjunctive cuts is stopped.

A value of -1 means that generation is not stopped. The default value for this integer option is 20 and its valid range is $-1 \leq \text{disj_depth_stop}$.

disj_active_cols: Only include violated variable bounds in the Cut Generating LP (CGLP).

This reduces the size of the CGLP, but may produce less efficient cuts. The default value for this string option is "no".

Possible values: "yes" and "no".

disj_active_rows: Only include violated linear inequalities in the CGLP.

This reduces the size of the CGLP, but may produce less efficient cuts. The default value for this string option is "no".

Possible values: "yes" and "no".

disj_cumulative: Add previous disjunctive cut to the current CGLP.

When generating disjunctive cuts on a set of disjunctions $1, 2, \dots, k$, introduce the cut relative to the previous disjunction $i - 1$ in the CGLP used for disjunction i . Notice that, although this makes the cut generated more efficient, it increases the rank of the disjunctive cut generated. The default value for this string option is "no".

Possible values: "yes" and "no"

disj_init_number: Maximum number of disjunction to consider at each iteration.

-1 means no limit. The valid range for this integer option is $-1 \leq \text{disj_init_number}$ and its default value is 10 .

disj_init_perc: The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions.

The valid range for this real option is $0 \leq \text{disj_init_perc} \leq 1$ and its default value is 0.5 .

Nonlinear solver options

local_optimization_heuristic: Do we search for local solutions of NLP's

If enabled, a heuristic based on Ipopt is used to find feasible solutions for the problem. It is highly recommended that this option is left enabled, as it would be difficult to find feasible solutions otherwise. The default value for

this string option is "yes".

Possible values: "yes" and "no"

log_num_local_optimization_per_level: At what depth of the B&B tree should the calls to the heuristic be reduced.

The behavior is similar to "log_num_obbt_per_level": The parameter specifies the logarithm of the number of local optimizations to perform on average on a given depth of the tree. The nodes at a given depth are randomly selected. If for a given level there are less nodes than this number, then NLPs are solved for every node. For example if this parameter is set to 8, then NLP's are solved for all nodes until level 8, then for half the node at level 9, 1/4 at level 10, ... Value -1 specify to perform local searches at all nodes. The valid range for this integer option is $-1 \leq \text{log_num_local_optimization_per_level}$ and its default value is 2.

Tolerance options

feas_tolerance: Feasibility tolerance for constraints/auxiliary variables.

The valid range for this real option is $-\text{inf} < \text{feas_tolerance}$ and its default value is 10^{-5} .

MIP cut generator options

For the following options, the frequency k has the following meaning: If $k > 0$, then cuts are generated every k nodes. If $-99 < k < 0$, then cuts are generated every $-k$ nodes but Cbc may decide to stop generating cuts if not enough are generated at the root node. If $k = -99$, then cuts are generate only at the root node. If $k = 0$ or $k = -100$, then do not generate cuts. The valid range for these integer options is $-100 \leq k$ and their default value is 0.

Gomory_cuts: Frequency k (in terms of nodes) for generating Gomory cuts in branch-and-cut.

clique_cuts: Frequency k (in terms of nodes) for generating clique cuts in branch-and-cut.

cover_cuts: Frequency k (in terms of nodes) for generating cover cuts in branch-and-cut.

flow_covers_cuts: Frequency k (in terms of nodes) for generating flow cover cuts in branch-and-cut.

lift_and_project_cuts: Frequency k (in terms of nodes) for generating lift and project cuts in branch-and-cut.

mir_cuts: Frequency k (in terms of nodes) for generating MIR cuts in branch-and-cut.

2mir_cuts: Frequency k (in terms of nodes) for generating 2-MIR cuts in branch-and-cut.

probing_cuts: Frequency k (in terms of nodes) for generating probing cuts in branch-and-cut.

reduce_split_cuts: Frequency k (in terms of nodes) for generating reduce split cuts in branch-and-cut.

5 CoinGlpk

GAMS/CoinGlpk brings the open source LP/MIP solver Glpk from the GNU Open Software foundation to the broad audience of GAMS users.

The code has been written primarily by A. Makhorin. The interface uses the OSI Glpk interface written by Vivian De Smedt, Braden Hunsaker, and Lou Hafer.

For more information we refer to

- the Glpk web site <http://www.gnu.org/software/glpk/glpk.html> and
- the Osi web site <https://projects.coin-or.org/Osi>.

Most of the Glpk documentation in the section was taken from the Glpk manual.

5.1 Model requirements

Glpk supports continuous, binary, and integer variables, but no special ordered sets, semi-continuous or semi-integer variables (see chapter 17.1 of the GAMS User's Guide). Also branching priorities are not supported.

5.2 Usage of CoinGlpk

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

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

The above statement should appear before the Solve statement. If CoinGlpk was specified as the default solver during GAMS installation, the above statement is not necessary.

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

```
factorization gives
```

It will cause CoinGlpk to use Givens rotation updates for the factorization. (This option setting might help to avoid numerical difficulties in some cases.)

5.3 Summary of CoinGlpk Options

Among the following Glpk options, only the GAMS parameters `reslim`, `iterlim`, and `optcr` are currently supported in CoinGlpk. Note, that the option `optca` to set the absolute MIP gap is currently not supported.

<code>startalg</code>	LP solver for root node
<code>scaling</code>	scaling method
<code>pricing</code>	pricing method
<code>factorization</code>	basis factorization method
<code>initbasis</code>	method for initial basis
<code>tol_dual</code>	dual feasibility tolerance
<code>tol_primal</code>	primal feasibility tolerance
<code>tol_integer</code>	integer feasibility tolerance
<code>backtracking</code>	backtracking heuristic
<code>presolve</code>	LP presolver
<code>cuts</code>	turning on or off all cut generators
<code>covercuts</code>	cover cuts
<code>cliquecuts</code>	clique cuts
<code>gomorycuts</code>	gomorys mixed-integer cuts
<code>mircuts</code>	mixed-integer rounding cuts

<code>reslim_fixedrun</code>	resource limit for solve with fixed discrete variables
<code>names</code>	indicates whether row and column names should be given to glpk
<code>solvefinal</code>	switch for solve of the problem with fixed discrete variables
<code>reslim</code>	resource limit
<code>iterlim</code>	iteration limit
<code>optcr</code>	relative stopping tolerance on MIP gap
<code>writemps</code>	create MPS file for problem

5.4 Detailed Descriptions of CoinGlpk Options

startalg (*string*)

This option determines whether a primal or dual simplex algorithm should be used to solve an LP or the root node of a MIP.

(*default = primal*)

primal Let GLPK use a primal simplex algorithm.

dual Let GLPK use a dual simplex algorithm.

scaling (*string*)

This option determines the method how the constraint matrix is scaled. Note that scaling is only applied when the `presolver` is turned off, which is on by default.

(*default = meanequilibrium*)

off Turn off scaling.

equilibrium Let GLPK use an equilibrium scaling method.

mean Let GLPK use a geometric mean scaling method.

meanequilibrium Let GLPK use first a geometric mean scaling, then an equilibrium scaling.

pricing (*string*)

Sets the pricing method for both primal and dual simplex.

(*default = steepestedge*)

textbook Use a textbook pricing rule.

steepestedge Use a steepest edge pricing rule.

factorization (*string*)

Sets the method for the LP basis factorization.

If you observe that GLPK reports numerical instabilities than you could try to use a more stable factorization method.

(*default = forresttomlin*)

forresttomlin Does a LU factorization followed by Forrest-Tomlin updates. This method is fast, but less stable than others.

bartelsgolub Does a LU factorization followed by a Schur complement and Bartels-Golub updates. This method is slower than Forrest-Tomlin, but more stable.

givens Does a LU factorization followed by a Schur complement and Givens rotation updates. This method is slower than Forrest-Tomlin, but more stable.

initbasis (*string*)

Sets the method that computes the initial basis. Setting this option has only effect if the `presolver` is turned off, which is on by default.

(*default = advanced*)

standard Uses the standard initial basis of all slacks.

advanced Computes an advanced initial basis.

bixby Uses Bixby's initial basis.

user Uses basis provided by GAMS. Note that a feasible basis need to be given, otherwise Glpk will fail.

tol_dual (real)

Absolute tolerance used to check if the current basis solution is dual feasible.

(default = 1e-7)

tol_primal (real)

Relative tolerance used to check if the current basis solution is primal feasible.

(default = 1e-7)

tol_integer (real)

Absolute tolerance used to check if the current basis solution is integer feasible.

(default = 1e-5)

backtracking (string)

Determines which method to use for the backtracking heuristic.

(default = bestprojection)

depthfirst Let GLPK use a depth first search.

breadthfirst Let GLPK use a breadth first search.

bestprojection Let GLPK use a best projection heuristic.

presolve (integer)

Determines whether the LP presolver should be used.

(default = 1)

0 Turns off the LP presolver.

1 Turns on the LP presolver.

cuts (integer)

Determines which cuts generator to use: none, all, or user-defined

(default = 0)

-1 . Turn off all cut generators

0 . Turn on or off each cut generators separately

1 . Turn on all cut generators

covercuts (integer)

Whether to enable cover cuts.

(default = 1)

0 . Turn off cover cuts

1 . Turn on cover cuts

cliquecuts (integer)

Whether to enable clique cuts.

(default = 1)

0 . Turn off clique cuts

- 1 . Turn on clique cuts

gomorycuts (*integer*)

Whether to enable Gomory's mixed-integer linear cuts.

(*default = 1*)

- 0 . Turn off gomory cuts
- 1 . Turn on gomory cuts

mircuts (*integer*)

Whether to enable mixed-integer rounding cuts.

(*default = 0*)

- 0 . Turn off mir cuts
- 1 . Turn on mir cuts

reslim_fixedrun (*real*)

Maximum time in seconds for solving the MIP with fixed discrete variables.

(*default = 1000*)

names (*integer*)

This option causes GAMS names for the variables and equations to be loaded into Glpk. These names will then be used for error messages, log entries, and so forth. Turning names off may help if memory is very tight.

(*default = 0*)

- 0 . Do not load variable and equation names.
- 1 . Load variable and equation names.

solvefinal (*integer*)

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

(*default = 1*)

- 0 . Do not solve the fixed problem
- 1 . Solve the fixed problem and return duals

reslim (*real*)

Maximum time in seconds.

(*default = GAMS reslim*)

iterlim (*integer*)

Maximum number of simplex iterations.

(*default = GAMS iterlim*)

optcr (*real*)

Relative optimality criterion for a MIP. The search is stopped when the relative gap between the incumbent and the bound given by the LP relaxation is smaller than this value.

(*default = GAMS optcr*)

writemps (*string*)

Write an MPS problem file. The parameter value is the name of the MPS file.

6 CoinIopt and CoinIoptD

GAMS/CoinIopt brings the open source NLP solver Iopt to the broad audience of GAMS users.

Iopt (**I**nterior **P**oint **O**ptimizer) is an open-source solver for large-scale nonlinear programming. The code has been written primarily by Andreas Wächter, who is the COIN-OR project leader for Iopt.

For more information we refer to

- the Iopt web site <https://projects.coin-or.org/Iopt> and
- the *implementation paper* A. Wächter and L. T. Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, *Mathematical Programming* 106(1), pp. 25-57, 2006.

Most of the Iopt documentation in the section was taken from the Iopt manual available on the Iopt web site.

Gams/IoptD is an experimental in-core communication link. It offers in-core communication between GAMS and the solver, making potentially large model scratch files unnecessary. This link supports all features of the traditional link except.

6.1 Model requirements

Iopt can handle nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable.

6.2 Usage of CoinIopt

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

```
Option NLP = CoinIopt;      { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP }
```

The above statement should appear before the Solve statement. If CoinIopt was specified as the default solver during GAMS installation, the above statement is not necessary.

6.3 The linear solver in Iopt

The performance and robustness of Iopt on larger models heavily relies on the used solver for sparse symmetric indefinite linear systems. GAMS/CoinIopt includes the sparse solver MUMPS 4.8.3 (<http://graal.ens-lyon.fr/MUMPS>). The user can provide the Parallel Sparse Direct Solver PARDISO or routines from the Harwell Subroutine Library (HSL) as shared (or dynamic) libraries to replace MUMPS.

Using Harwell Subroutine Library routines with GAMS/CoinIopt

If you have routines from the HSL available and want Gams/CoinIopt to use them, you can provide them in a shared library. GAMS/CoinIopt can use MA27, MA28, MA57, and MC19. By telling Iopt to use one of these routines (see options `linear_solver`, `linear_system_scaling`, `nlp_scaling_method`, `dependency_detector`) GAMS/CoinIopt attempts to load the required routines from the library `libhsl.so` (Unix-Systems), `libhsl.dylib` (MacOS X), or `libhsl.dll` (Windows). You can also specify the path and name for this library with the option `hsl_library`.

For example,

```
linear_solver ma27
hsl_library /my/path/to/the/hsl/lib/myhsl/lib.so
```

tells Ipopt to use the linear solver MA27 from the HSL library `myhsl.lib.so` under the specified path.

The HSL routines MA27, MA28, and MC19 are available at <http://www.cse.clrc.ac.uk/nag/hsl>. Note that it is your responsibility to ensure that you are entitled to download and use these routines! You can build a shared library using the ThirdParty/HSL project at COIN-OR.

Using PARDISO routines with GAMS/CoinIpopt

If you have the linear solver PARDISO available, then you can tell Gams/CoinIpopt to use by setting the `linear_solver` option to `pardiso`. GAMS/CoinIpopt then attempts to load the library `libpardiso.so` (Unix-Systems), `libpardiso.dylib` (MacOS X), or `libpardiso.dll` (Windows). You can also specify the path and name for this library with the option `pardiso_library`.

For example,

```
linear_solver pardiso
pardiso_library /my/path/to/the/pardisolib/mypardisolib.so
```

tells Ipopt to use the linear solver PARDISO from the library `mypardisolib.so` under the specified path.

PARDISO is available as compiled shared library for several platforms at <http://www.pardiso-project.org>. Note that it is your responsibility to ensure that you are entitled to download and use this package!

6.4 Output

This section describes the standard Ipopt console output. The output is designed to provide a quick summary of each iteration as Ipopt solves the problem.

Before Ipopt starts to solve the problem, it displays the problem statistics (number of nonzero-elements in the matrices, number of variables, etc.). Note that if you have fixed variables (both upper and lower bounds are equal), Ipopt may remove these variables from the problem internally and not include them in the problem statistics.

Following the problem statistics, Ipopt will begin to solve the problem and you will see output resembling the following,

```
iter   objective   inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0   1.6109693e+01  1.12e+01  5.28e-01   0.0  0.00e+00   -   0.00e+00  0.00e+00   0
   1   1.8029749e+01  9.90e-01  6.62e+01   0.1  2.05e+00   -   2.14e-01  1.00e+00f  1
   2   1.8719906e+01  1.25e-02  9.04e+00  -2.2  5.94e-02   2.0  8.04e-01  1.00e+00h  1
```

and the columns of output are defined as

iter The current iteration count. This includes regular iterations and iterations while in restoration phase. If the algorithm is in the restoration phase, the letter 'r' will be appended to the iteration number.

objective The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.

inf_pr The scaled primal infeasibility at the current point. During the restoration phase, this value is the primal infeasibility of the original problem at the current point.

inf_du The scaled dual infeasibility at the current point. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.

lg(mu) \log_{10} of the value of the barrier parameter μ .

||d|| The infinity norm (max) of the primal step (for the original variables x and the internal slack variables s). During the restoration phase, this value includes the values of additional variables, p and n .

lg(rg) \log_{10} of the value of the regularization term for the Hessian of the Lagrangian in the augmented system.

alpha_du The stepsize for the dual variables.

alpha_pr The stepsize for the primal variables.

ls The number of backtracking line search steps.

When the algorithm terminates, IPOPT will output a message to the screen based on the return status of the call to `Optimize`. The following is a list of the possible output messages to the console, and a brief description.

Optimal Solution Found.

This message indicates that IPOPT found a (locally) optimal point within the desired tolerances.

Solved To Acceptable Level.

This indicates that the algorithm did not converge to the “desired” tolerances, but that it was able to obtain a point satisfying the “acceptable” tolerance level as specified by `acceptable-*` options. This may happen if the desired tolerances are too small for the current problem.

Converged to a point of local infeasibility. Problem may be infeasible.

The restoration phase converged to a point that is a minimizer for the constraint violation (in the ℓ_1 -norm), but is not feasible for the original problem. This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point). The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem. If you believe that the NLP is feasible, it might help to start the optimization from a different point.

Search Direction is becoming Too Small.

This indicates that Ipopt is calculating very small step sizes and making very little progress. This could happen if the problem has been solved to the best numerical accuracy possible given the current scaling.

Iterates diverging; problem might be unbounded.

This message is printed if the max-norm of the iterates becomes larger than the value of the option `diverging_iterates_tol`. This can happen if the problem is unbounded below and the iterates are diverging.

Stopping optimization at current point as requested by user.

This message is printed if either the time limit or the domain violation limit is reached.

Maximum Number of Iterations Exceeded.

This indicates that Ipopt has exceeded the maximum number of iterations as specified by the option `max_iter`.

Restoration Failed!

This indicates that the restoration phase failed to find a feasible point that was acceptable to the filter line search for the original problem. This could happen if the problem is highly degenerate or does not satisfy the constraint qualification, or if an external function in GAMS provides incorrect derivative information.

Error in step computation (regularization becomes too large?)!

This message is printed if Ipopt is unable to compute a search direction, despite several attempts to modify the iteration matrix. Usually, the value of the regularization parameter then becomes too large.

Problem has too few degrees of freedom.

This indicates that your problem, as specified, has too few degrees of freedom. This can happen if you have too many equality constraints, or if you fix too many variables (Ipopt removes fixed variables).

Not enough memory.

An error occurred while trying to allocate memory. The problem may be too large for your current memory and swap configuration.

INTERNAL ERROR: Unknown SolverReturn value - Notify IPOPT Authors.

An unknown internal error has occurred. Please notify the authors of the GAMS/CoinIpopt link or IPOPT (refer to <https://projects.coin-or.org/GAMSlinks> or <https://projects.coin-or.org/Ipopt>).

6.5 Specification of CoinIopt Options

Ipopt has many options that can be adjusted for the algorithm (see Section 6.6). Options are all identified by a string name, and their values can be of one of three types: Number (real), Integer, or String. Number options are used for things like tolerances, integer options are used for things like maximum number of iterations, and string options are used for setting algorithm details, like the NLP scaling method. Options can be set by creating a `ipopt.opt` file in the directory you are executing Ipopt.

The `ipopt.opt` file is read line by line and each line should contain the option name, followed by whitespace, and then the value. Comments can be included with the `#` symbol. Don't forget to ensure you have a newline at the end of the file. For example,

```
# This is a comment

# Turn off the NLP scaling
nlp_scaling_method none

# Change the initial barrier parameter
mu_init 1e-2

# Set the max number of iterations
max_iter 500
```

is a valid `ipopt.opt` file.

Note, that GAMS/CoinIopt overwrites the Ipopt default setting for the parameters `bound_relax_factor` (set to 0.0) and `mu_strategy` (set to adaptive). You can change these values by specifying these options in your Ipopt options file.

GAMS/CoinIopt understand currently the following GAMS parameters: `reslim` (time limit), `iterlim` (iteration limit), `domlim` (domain violation limit). You can set them either on the command line, e.g. `iterlim=500`, or inside your GAMS program, e.g. `Option iterlim=500;`.

6.6 Detailed Description of CoinIopt Options

Output

print_level: Output verbosity level.

Sets the default verbosity level for console output. The larger this value the more detailed is the output. The valid range for this integer option is $-2 \leq \text{print_level} \leq 12$ and its default value is 5.

file_print_level: Verbosity level for output file.

NOTE: This option only works when read from the `ipopt.opt` options file! Determines the verbosity level for the file specified by `"output_file"`. By default it is the same as `"print_level"`. The valid range for this integer option is $0 \leq \text{file_print_level} \leq 12$ and its default value is 5.

output_file: File name of desired output file (leave unset for no file output).

NOTE: This option only works when read from the `ipopt.opt` options file! An output file with this name will be written (leave unset for no file output). The verbosity level is by default set to `"print_level"`, but can be overridden with `"file_print_level"`. The file name is changed to use only small letters. The default value for this string option is "".

Possible value is any acceptable standard file name.

print_info_string: Enables printing of additional info string at end of iteration output.

This string contains some insider information about the current iteration. The default value for this string option

is "no".

Possible values:

- no: don't print string
- yes: print string at end of each iteration output

print_options_documentation: Switch to print all algorithmic options.

If selected, the algorithm will print the list of all available algorithmic options with some documentation before solving the optimization problem. The default value for this string option is "no".

Possible values:

- no: don't print list
- yes: print list

print_timing_statistics: Switch to print timing statistics.

If selected, the program will print the CPU usage (user time) for selected tasks. The default value for this string option is "no".

Possible values:

- no: don't print statistics
- yes: print all timing statistics

print_user_options: Print all options set by the user.

If selected, the algorithm will print the list of all options set by the user including their values and whether they have been used. In some cases this information might be incorrect, due to the internal program flow. The default value for this string option is "yes".

Possible values:

- no: don't print options
- yes: print options

Convergence

acceptable_compl_inf_tol: "Acceptance" threshold for the complementarity conditions.

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also `acceptable_tol`. The valid range for this real option is $0 < \text{acceptable_compl_inf_tol} < +\text{inf}$ and its default value is 0.01.

acceptable_constr_viol_tol: "Acceptance" threshold for the constraint violation.

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also `acceptable_tol`. The valid range for this real option is $0 < \text{acceptable_constr_viol_tol} < +\text{inf}$ and its default value is 0.01.

acceptable_dual_inf_tol: "Acceptance" threshold for the dual infeasibility.

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also `acceptable_tol`. The valid range for this real option is $0 < \text{acceptable_dual_inf_tol} < +\text{inf}$ and its default value is $1 \cdot 10^{+10}$.

acceptable_iter: Number of "acceptable" iterates before triggering termination.

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled. The valid range for this integer option is $0 \leq \text{acceptable_iter} < +\text{inf}$ and its default value is 15.

acceptable_obj_change_tol: "Acceptance" stopping criterion based on objective function change.

If the relative change of the objective function (scaled by $\text{Max}(1, -f(x))$) is less than this value, this part of the acceptable tolerance termination is satisfied; see also `acceptable_tol`. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility. The valid range for this real option is $0 \leq \text{acceptable_obj_change_tol} < +\text{inf}$ and its default value is $1 \cdot 10^{+20}$.

acceptable_tol: "Acceptable" convergence tolerance (relative).

Determines which (scaled) overall optimality error is considered to be "acceptable." There are two levels of termination criteria. If the usual "desired" tolerances (see `tol`, `dual_inf_tol` etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy. The valid range for this real option is $0 < \text{acceptable_tol} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

compl_inf_tol: Desired threshold for the complementarity conditions.

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold. The valid range for this real option is $0 < \text{compl_inf_tol} < +\text{inf}$ and its default value is 0.0001.

constr_viol_tol: Desired threshold for the constraint violation.

Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold. The default value is for this real option is 0.0001 and its valid range is $0 < \text{constr_viol_tol} < +\text{inf}$.

diverging_iterates_tol: Threshold for maximal value of primal iterates.

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging. The valid range for this real option is $0 < \text{diverging_iterates_tol} < +\text{inf}$ and its default value is $1 \cdot 10^{+20}$.

dual_inf_tol: Desired threshold for the dual infeasibility.

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold. The valid range for this real option is $0 < \text{dual_inf_tol} < +\text{inf}$ and its default value is 1.

max_cpu_time: Maximum number of CPU seconds.

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message. The valid range for this integer option is $0 \leq \text{max_cpu_time} < +\text{inf}$ and its default value is the value of the GAMS parameter `reslim`, which default value is 1000.

max_iter: Maximum number of iterations.

The algorithm terminates with an error message if the number of iterations exceeded this number. The valid range for this integer option is $0 \leq \text{max_iter} < +\text{inf}$ and its default value is the value of the GAMS parameter `iterlim`, which default value is ∞ .

s_max: Scaling threshold for the NLP error.

(See paragraph after Eqn. (6) in the implementation paper.) The valid range for this real option is $0 < \text{s_max} < +\text{inf}$ and its default value is 100.

tol: Desired convergence tolerance (relative).

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "primal_inf_tol", and "cmlpl_inf_tol" are met. (This is ε_{tol} in Eqn. (6) in the implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc. The valid range for this real option is $0 < \text{tol} < +\text{inf}$ and its default value is $1 \cdot 10^{-08}$.

Initialization

bound_frac: Desired minimum relative distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is κ_2 in Section 3.6 of the implementation paper.) The valid range for this real option is $0 < \text{bound_frac} \leq 0.5$ and its default value is 0.01.

bound_mult_init_method: Initialization method for bound multipliers

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, the each value is initialized to the the value of "mu_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution. The default value for this string option is "constant". Possible values:

- constant: set all bound multipliers to the value of bound_mult_init_val
- mu-based: initialize to mu_init/x_slack

bound_mult_init_val: Initial value for the bound multipliers.

All dual variables corresponding to bound constraints are initialized to this value. The valid range for this real option is $0 < \text{bound_mult_init_val} < +\text{inf}$ and its default value is 1.

bound_push: Desired minimum absolute distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is κ_1 in Section 3.6 of the implementation paper.) The valid range for this real option is $0 < \text{bound_push} < +\text{inf}$ and its default value is 0.01.

constr_mult_init_max: Maximum allowed least-square guess of constraint multipliers.

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero. The valid range for this real option is $0 \leq \text{constr_mult_init_max} < +\text{inf}$ and its default value is 1000.

least_square_init_duals: Least square initialization of all dual variables

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method". The default value for this string option is "no".

Possible values:

- no: use bound_mult_init_val and least-square equality constraint multipliers

- yes: overwrite user-provided point with least-square estimates

least_square_init_primal: Least square initialization of the primal variables

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. The default value for this string option is "no".

Possible values:

- no: take user-provided point
- yes: overwrite user-provided point with least-square estimates

slack_bound_frac: Desired minimum relative distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is κ_2 in Section 3.6 of the implementation paper.) The valid range for this real option is $0 < \text{slack_bound_frac} \leq 0.5$ and its default value is 0.01.

slack_bound_push: Desired minimum absolute distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is κ_1 in Section 3.6 of the implementation paper.) The valid range for this real option is $0 < \text{slack_bound_push} < +\text{inf}$ and its default value is 0.01.

Hessian Approximation

hessian_approximation: Indicates what Hessian information is to be used.

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm. The default value for this string option is "exact".

Possible values:

- exact: Use second derivatives provided by the NLP.
- limited-memory: Perform a limited-memory quasi-Newton approximation

hessian_approximation_space: Indicates in which subspace the Hessian information is to be approximated.

The default value for this string option is "nonlinear-variables".

Possible values:

- nonlinear-variables: only in space of nonlinear variables.
- all-variables: in space of all variables (without slacks)

limited_memory_init_val: Value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant". The valid range for this real option is $0 < \text{limited_memory_init_val} < +\text{inf}$ and its default value is 1.

limited_memory_init_val_max: Upper bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant". The valid range for this real option is $0 < \text{limited_memory_init_val_max} < +\text{inf}$ and its default value is $1 \cdot 10^{+08}$.

limited_memory_init_val_min: Lower bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant". The valid range for this real option is $0 < \text{limited_memory_init_val_min} < +\text{inf}$ and its default value is $1 \cdot 10^{-08}$.

limited_memory_initialization: Initialization strategy for the limited memory quasi-Newton approximation.

Determines how the diagonal Matrix B.0 as the first term in the limited memory approximation should be computed. The default value for this string option is "scalar1".

Possible values:

- scalar1: $\sigma = s^T y / s^T s$
- scalar2: $\sigma = y^T y / s^T y$
- constant: $\sigma = \text{limited_memory_init_val}$

limited_memory_max_history: Maximum size of the history for the limited quasi-Newton Hessian approximation.

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation. The valid range for this integer option is $0 \leq \text{limited_memory_max_history} < +\text{inf}$ and its default value is 6.

limited_memory_max_skipping: Threshold for successive iterations where update is skipped.

If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset. The valid range for this integer option is $1 \leq \text{limited_memory_max_skipping} < +\text{inf}$ and its default value is 2.

limited_memory_update_type: Quasi-Newton update formula for the limited memory approximation.

Determines which update formula is to be used for the limited-memory quasi-Newton approximation. The default value for this string option is "bfgs".

Possible values:

- bfgs: BFGS update (with skipping)
- sr1: SR1 (not working well)

Linear Solver

hsl_library: path and filename of HSL library for dynamic load

Specify the path to a library that contains HSL routines and can be load via dynamic linking. Note, that you still need to specify to use the corresponding routines (ma27, ...) by setting the corresponding options, e.g., "linear_solver".

linear_scaling_on_demand: Flag indicating that linear scaling is only done if it seems required.

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end. The default value for this string option is "yes".

Possible values:

- no: Always scale the linear system.
- yes: Start using linear system scaling if solutions seem not good.

linear_solver: Linear solver used for step computations.

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, that GAMS/CoinIpopt has been compiled with the MUMPS library only. To use MA27, MA57, Pardiso, the user need to provide them (see Section 6.3). The default value for this string option is "mumps".

Possible values:

- ma27: use the Harwell routine MA27
- ma57: use the Harwell routine MA57
- pardiso: use the Pardiso package
- mumps: use MUMPS package

linear_system_scaling: Method for scaling the linear system.

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. This option is only available if Ipopt has been compiled with MC19. The default value for this string option is "mc19".

Possible values:

- none: no scaling will be performed
- mc19: use the Harwell routine MC19

pardiso_library: path and filename of PARDISO library for dynamic load

Specify the path to a PARDISO library that and can be load via dynamic linking. Note, that you still need to specify to pardiso as linear_solver.

MA27 Linear Solver

ma27_ignore_singularity: Enables MA27's ability to solve a linear system even if the matrix is singular.

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt's heuristic of small perturbation of the lower diagonal of the KKT matrix. The default value for this string option is "no".

Possible values:

- no: Don't have MA27 solve singular systems
- yes: Have MA27 solve singular systems

ma27_la_init_factor: Real workspace memory for MA27.

The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. The valid range for this real option is $1 \leq \text{ma27_la_init_factor} < +\text{inf}$ and its default value is 5.

ma27_liw_init_factor: Integer workspace memory for MA27.

The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. The default value for this real option is 5 and its valid range is $1 \leq \text{ma27_liw_init_factor} < +\text{inf}$.

ma27_meminc_factor: Increment factor for workspace size for MA27.

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. The valid range for this real option is $1 \leq \text{ma27_meminc_factor} < +\text{inf}$ and its default value is 10.

ma27_pivtol: Pivot tolerance for the linear solver MA27.

A smaller number pivots for sparsity, a larger number pivots for stability. The valid range for this real option is $0 < \text{ma27_pivtol} < 1$ and its default value is $1 \cdot 10^{-08}$.

ma27_pivtolmax: Maximum pivot tolerance for the linear solver MA27.

Ipop may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. The valid range for this real option is $0 < \text{ma27_pivtolmax} < 1$ and its default value is 0.0001.

ma27_skip_inertia_check: Always pretend inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control. The default value for this string option is "no".

Possible values:

- no: check inertia
- yes: skip inertia check

MA28 Linear Solver

ma28_pivtol: Pivot tolerance for linear solver MA28.

This is used when MA28 tries to find the dependent constraints. The valid range for this real option is $0 < \text{ma28_pivtol} \leq 1$ and its default value is 0.01.

MA57 Linear Solver

ma57_pivtol: Pivot tolerance for the linear solver MA57.

A smaller number pivots for sparsity, a larger number pivots for stability. The valid range for this real option is $0 < \text{ma57_pivtol} < 1$ and its default value is $1 \cdot 10^{-08}$.

ma57_pivtolmax: Maximum pivot tolerance for the linear solver MA57.

Ipop may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system. The valid range for this real option is $0 < \text{ma57_pivtolmax} < 1$ and its default value is 0.0001.

ma57_pre_alloc: Safety factor for work space memory allocation for the linear solver MA57.

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. The valid range for this real option is $1 \leq \text{ma57_pre_alloc} < +\text{inf}$ and its default value is 3.

ma57_pivot_order: Controls pivot order in MA57.

This is INCTL(6) in MA57. The valid range for this integer option is $0 < \text{ma57_pivot_order} < 5$ and its default value is 5.

Mumps Linear Solver

mumps_dep_tol: Pivot threshold for detection of linearly dependent constraints in MUMPS.

When MUMPS is used to determine linearly dependent constraints, this determines the threshold for a pivot to be considered zero. This is CNTL(3) in MUMPS. The valid range for this real option is $-\text{inf} < \text{mumps_dep_tol} < +\text{inf}$ and its default value is -1.

mumps_mem_percent: Percentage increase in the estimated working space for MUMPS.

In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of `mumps_mem_percent` may help use the workspace more efficiently. The valid range for this integer option is $0 \leq \text{mumps_mem_percent} < +\text{inf}$ and its default value is 1000.

mumps_permuting_scaling: Controls permuting and scaling in MUMPS

This is ICTL(6) in MUMPS. The valid range for this integer option is $0 \leq \text{mumps_permuting_scaling} \leq 7$ and its default value is 7.

mumps_pivot_order: Controls pivot order in MUMPS

This is ICTL(7) in MUMPS. The valid range for this integer option is $0 \leq \text{mumps_pivot_order} \leq 7$ and its default value is 7.

mumps_pivtol: Pivot tolerance for the linear solver MUMPS.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS. The valid range for this real option is $0 \leq \text{mumps_pivtol} \leq 1$ and its default value is $1 \cdot 10^{-06}$.

mumps_pivtolmax: Maximum pivot tolerance for the linear solver MUMPS.

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS. The valid range for this real option is $0 \leq \text{mumps_pivtolmax} \leq 1$ and its default value is 0.1.

mumps_scaling: Controls scaling in MUMPS

This is ICTL(8) in MUMPS. The valid range for this integer option is $-2 \leq \text{mumps_scaling} \leq 7$ and its default value is 7.

Pardiso Linear Solver

pardiso_dropping_schur_exponent:

The valid range for this integer option is $-\text{inf} < \text{pardiso_dropping_schur_exponent} < +\text{inf}$ and its default value is -3.

pardiso_dropping_factor_exponent:

The valid range for this integer option is $-\text{inf} < \text{pardiso_dropping_factor_exponent} < +\text{inf}$ and its default value is -3.

pardiso_inverse_norm_factor:

The valid range for this integer option is $-\text{inf} < \text{pardiso_inverse_norm_factor} < +\text{inf}$ and its default value is 500.

pardiso_iter_tol_exponent:

The valid range for this integer option is $-\text{inf} < \text{pardiso_iter_tol_exponent} < +\text{inf}$ and its default value is -14.

pardiso_iterative:

The default value for this string option is "no".

pardiso_matching_strategy: Matching strategy to be used by Pardiso

This is IPAR(13) in Pardiso manual. The default value for this string option is "complete+2x2".

Possible values:

- complete: Match complete (IPAR(13)=1)
- complete+2x2: Match complete+2x2 (IPAR(13)=2)
- constraints: Match constraints (IPAR(13)=3)

pardiso_max_iter:

The valid range for this integer option is $-\text{inf} < \text{pardiso_max_iter} < +\text{inf}$ and its default value is 500.

pardiso_msglvl: Pardiso message level

This determines the amount of analysis output from the Pardiso solver. This is MSGLVL in the Pardiso manual.

The valid range for this integer option is $0 < \text{pardiso_msglvl} < +\text{inf}$ and its default value is 0.

pardiso_out_of_core_power: Enables out-of-core variant of Pardiso

Setting this option to a positive integer k makes Pardiso work in the out-of-core variant where the factor is split in 2^k subdomains. This is IPARM(50) in the Pardiso manual. The valid range for this integer option is $0 \leq \text{pardiso_out_of_core_power} < +\text{inf}$ and its default value is 0.

pardiso_redo_symbolic_fact_only_if_inertia_wrong: Toggle for handling case when elements were perturbed by Pardiso.

The default value for this string option is "no".

Possible values:

- no: Always redo symbolic factorization when elements were perturbed
- yes: Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

pardiso_repeated_perturbation_means_singular: Interpretation of perturbed elements.

The default value for this string option is "no".

Possible values:

- no: Don't assume that matrix is singular if elements were perturbed after recent symbolic factorization
- yes: Assume that matrix is singular if elements were perturbed after recent symbolic factorization

pardiso_skip_inertia_check: Always pretend inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control. The default value for this string option is "no".

Possible values:

- no: check inertia
- yes: skip inertia check

NLP

bound_relax_factor: Factor for initial relaxation of the bounds.

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then then bounds relaxation is disabled. (See Eqn.(35) in the implementation paper.) The valid range for this real option is $0 \leq \text{bound_relax_factor} < +\text{inf}$ and its default value is 10^{-10} .

check_derivatives_for_naninf: Indicates whether it is desired to check for Nan/Inf in derivative matrices. Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. The default value for this string option is "no".

Possible values:

- no: Don't check (faster).
- yes: Check Jacobians and Hessian for Nan and Inf.

dependency_detection_with_rhs: Indicates if the right hand sides of the constraints should be considered during dependency detection.

The default value for this string option is "no".

Possible values:

- no: only look at gradients
- yes: also consider right hand side

dependency_detector: Indicates which linear solver should be used to detect linearly dependent equality constraints.

The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well. The default value for this string option is "none".

Possible values:

- none: don't check; no extra work at beginning
- mumps: use MUMPS
- ma28: use MA28

fixed_variable_treatment: Determines how fixed variables should be handled.

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to "bound_relax_factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables. The default value for this string option is "make_parameter".

Possible values:

- make_parameter: Remove fixed variable from optimization variables
- make_constraint: Add equality constraints fixing variables
- relax_bounds: Relax fixing bound constraints

honor_original_bounds: Indicates whether final points should be projected into original bounds.

Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization. The default value for this string option is "yes".

Possible values:

- no: Leave final point unchanged
- yes: Project final point back into original bounds

kappa_d: Weight for linear damping term (to handle one-sided bounds). (see Section 3.7 in the implementation paper.) The valid range for this real option is $0 \leq \text{kappa_d} < +\text{inf}$ and its default value is $1 \cdot 10^{-05}$.

NLP Scaling

nlp_scaling_constr_target_gradient: Target value for constraint function gradient size.

If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides `nlp_scaling_max_gradient` for the constraint functions. The valid range for this real option is $0 \leq \text{nlp_scaling_constr_target_gradient} < +\text{inf}$ and its default value is 0.

nlp_scaling_max_gradient: Maximum gradient after NLP scaling.

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is `g_max` in Section 3.8 of the implementation paper.) Note: This option is only used if `nlp_scaling_method` is chosen as "gradient-based". The valid range for this real option is $0 < \text{nlp_scaling_max_gradient} < +\text{inf}$ and its default value is 100.

nlp_scaling_method: Select the technique used for scaling the NLP.

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes ("scaling_factor") The default value for this string option is "gradient-based".

Possible values:

- none: no problem scaling will be performed
- gradient-based: scale the problem so the maximum gradient at the starting point is `scaling_max_gradient`
- equilibration-based: scale the problem so that first derivatives are of order 1 at random points (only available with MC19)

nlp_scaling_obj_target_gradient: Target value for objective function gradient size.

If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides `nlp_scaling_max_gradient` for the objective function. The valid range for this real option is $0 \leq \text{nlp_scaling_obj_target_gradient} < +\text{inf}$ and its default value is 0.

obj_scaling_factor: Scaling factor for the objective function.

This option sets a scaling factor for the objective function. The scaling is seen internally by Ipopt but the unscaled objective is reported in the console output. If additional scaling parameters are computed (e.g. user-scaling or gradient-based), both factors are multiplied. If this value is chosen to be negative, Ipopt will maximize the objective function instead of minimizing it. The valid range for this real option is $-\text{inf} < \text{obj_scaling_factor} < +\text{inf}$ and its default value is 1.

Barrier Parameter Update

adaptive_mu_globalization: Globalization strategy for the adaptive mu selection mode.

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option `"mu_strategy"` is chosen as "adaptive".) The default value for this string option is "obj-constr-filter".

Possible values:

- `kkt-error`: nonmonotone decrease of kkt-error
- `obj-constr-filter`: 2-dim filter for objective and constraint violation
- `never-monotone-mode`: disables globalization

`adaptive_mu_kkt_norm_type`: Norm used for the KKT error in the adaptive mu globalization strategies. When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the `QualityFunctionMuOracle`. The default value for this string option is "2-norm-squared".

Possible values:

- 1-norm: use the 1-norm (abs sum)
- 2-norm-squared: use the 2-norm squared (sum of squares)
- max-norm: use the infinity norm (max)
- 2-norm: use 2-norm

`adaptive_mu_kkterror_red_fact`: Sufficient decrease factor for "kkt-error" globalization strategy. For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease. The valid range for this real option is $0 < \text{adaptive_mu_kkterror_red_fact} < 1$ and its default value is 0.9999.

`adaptive_mu_kkterror_red_iters`: Maximum number of iterations requiring sufficient progress. For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode. The valid range for this integer option is $0 \leq \text{adaptive_mu_kkterror_red_iters} < +\text{inf}$ and its default value is 4.

`adaptive_mu_monotone_init_factor`: Determines the initial value of the barrier parameter when switching to the monotone mode.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode "average_compl" is chosen for `fixed_mu_oracle`, the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

The valid range for this real option is $0 < \text{adaptive_mu_monotone_init_factor}$ and its default value is 0.8.

`adaptive_mu_restore_previous_iterate`: Indicates if the previous iterate should be restored if the monotone mode is entered.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes). The default value for this string option is "no".

Possible values:

- no: don't restore accepted iterate
- yes: restore accepted iterate

`barrier_tol_factor`: Factor for mu in barrier stop test.

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. (This is κ_ε in the implementation paper). The valid range for this real option is $0 < \text{barrier_tol_factor} < +\text{inf}$ and its default value is 10.

filter_margin_fact: Factor determining width of margin for obj-constr-filter adaptive globalization strategy. When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: $(\text{new obj})_i - (\text{filter obj})_i - \text{filter_margin_fact} * (\text{new constr-viol})_i$ OR $(\text{new constr-viol})_i - (\text{filter constr-viol})_i - \text{filter_margin_fact} * (\text{new constr-viol})_i$. For the description of the "kkt-error-filter" option see "filter_max_margin". The valid range for this real option is $0 < \text{filter_margin_fact} < 1$ and its default value is $1 \cdot 10^{-05}$.

filter_max_margin: Maximum width of margin in obj-constr-filter adaptive globalization strategy. The valid range for this real option is $0 < \text{filter_max_margin} < +\text{inf}$ and its default value is 1.

fixed_mu_oracle: Oracle for the barrier parameter when switching to fixed mode. Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy".) The default value for this string option is "average_compl". Possible values:

- probing: Mehrotra's probing heuristic
- loqo: LOQO's centrality rule
- quality-function: minimize a quality function
- average_compl: base on current average complementarity

mu_allow_fast_monotone_decrease: Allow skipping of barrier problem if barrier test is already met. If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter. The default value for this string option is "yes". Possible values:

- no: Take at least one iteration per barrier problem
- yes: Allow fast decrease of mu if barrier test it met

mu_init: Initial value for the barrier parameter. This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if "mu_strategy" is chosen as "monotone") The valid range for this real option is $0 < \text{mu_init} < +\text{inf}$ and its default value is 0.1.

mu_linear_decrease_factor: Determines linear decrease rate of barrier parameter. For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of $\text{mu} * \text{mu_linear_decrease_factor}$ and $\text{mu}^{\text{superlinear_decrease_power}}$. (This is κ_μ in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode. The valid range for this real option is $0 < \text{mu_linear_decrease_factor} < 1$ and its default value is 0.2.

mu_max: Maximum value for barrier parameter. This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option "mu_strategy" is chosen as "adaptive".) The valid range for this real option is $0 < \text{mu_max} < +\text{inf}$ and its default value is 100000.

mu_max_fact: Factor for initialization of maximum value for barrier parameter. This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option "mu_strategy" is chosen as "adaptive".) The valid range for this real option is $0 < \text{mu_max_fact} < +\text{inf}$ and its default value is 1000.

mu_min: Minimum value for barrier parameter.

This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of $1e-11$ and $\min(\text{"tol"}, \text{"compl_inf_tol"}) / (\text{"barrier_tol_fact- or"} + 1)$, which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".) The valid range for this real option is $0 < \text{mu_min} < +\text{inf}$ and its default value is $1 \cdot 10^{-11}$.

mu_oracle: Oracle for a new barrier parameter in the adaptive strategy.

Determines how a new barrier parameter is computed in each "free-mode" iteration of the adaptive barrier parameter strategy. (Only considered if "adaptive" is selected for option "mu_strategy"). The default value for this string option is "quality-function".

Possible values:

- probing: Mehrotra's probing heuristic
- loqo: LOQO's centrality rule
- quality-function: minimize a quality function

mu_strategy: Update strategy for barrier parameter.

Determines which barrier parameter update strategy is to be used. The default value for this string option is "adaptive".

Possible values:

- monotone: use the monotone (Fiacco-McCormick) strategy
- adaptive: use the adaptive update strategy

mu_superlinear_decrease_power: Determines superlinear decrease rate of barrier parameter.

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of $\text{mu}^{\text{"mu_linear_decrease_factor"}}$ and $\text{mu}^{\text{"superlinear_decrease_power"}}$. (This is θ_μ in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode. The valid range for this real option is $1 < \text{mu_superlinear_decrease_power} < 2$ and its default value is 1.5.

quality_function_balancing_term: The balancing term included in the quality function for centrality.

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option "mu_oracle" is set to "quality-function".) The default value for this string option is "none".

Possible values:

- none: no balancing term is added
- cubic: $\text{Max}(0, \text{Max}(\text{dual_inf}, \text{primal_inf}) - \text{compl})^3$

quality_function_centrality: The penalty term for centrality that is included in quality function.

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the xi in the Loqo update rule. (Only used if option "mu_oracle" is set to "quality-function".) The default value for this string option is "none".

Possible values:

- none: no penalty term is added
- log: complementarity * the log of the centrality measure
- reciprocal: complementarity * the reciprocal of the centrality measure
- cubed-reciprocal: complementarity * the reciprocal of the centrality measure cubed

quality_function_max_section_steps: Maximum number of search steps during direct search procedure determining the optimal centering parameter.

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".) The valid range for this integer option is $0 \leq \text{quality_function_max_section_steps}$ and its default value is 8.

quality_function_norm_type: Norm used for components of the quality function.

(Only used if option "mu_oracle" is set to "quality-function".) The default value for this string option is "2-norm-squared".

Possible values:

- 1-norm: use the 1-norm (abs sum)
- 2-norm-squared: use the 2-norm squared (sum of squares)
- max-norm: use the infinity norm (max)
- 2-norm: use 2-norm

quality_function_section_qf_tol: Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".) The valid range for this real option is $0 \leq \text{quality_function_section_qf_tol} < 1$ and its default value is 0.

quality_function_section_sigma_tol: Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".) The valid range for this real option is $0 \leq \text{quality_function_section_sigma_tol} < 1$ and its default value is 0.01.

sigma_max: Maximum value of the centering parameter.

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".) The valid range for this real option is $0 < \text{sigma_max} < +\text{inf}$ and its default value is 100.

sigma_min: Minimum value of the centering parameter.

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".) The valid range for this real option is $0 \leq \text{sigma_min} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

tau_min: Lower bound on fraction-to-the-boundary parameter tau.

(This is τ_{\min} in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode. The valid range for this real option is $0 < \text{tau_min} < 1$ and its default value is 0.99.

Line Search

accept_every_trial_step: Always accept the first trial step.

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees. The default value for this string option is "no".

Possible values:

- no: don't arbitrarily accept the full step
- yes: always accept the full step

alpha_for_y: Method to determine the step size for constraint multipliers.

This option determines how the step size (`alpha_y`) will be calculated when updating the constraint multipliers. The default value for this string option is "primal".

Possible values:

- primal: use primal step size
- bound_mult: use step size for the bound multipliers (good for LPs)
- min: use the min of primal and bound multipliers
- max: use the max of primal and bound multipliers
- full: take a full step of size one
- min_dual_infeas: choose step size minimizing new dual infeasibility
- safe_min_dual_infeas: like "min_dual_infeas", but safeguarded by "min" and "max"
- primal-and-full: use the primal step size, and full step if $\delta_x \neq \alpha_{\text{for_y_tol}}$
- dual-and-full: use the dual step size, and full step if $\delta_x \neq \alpha_{\text{for_y_tol}}$
- acceptor: Call LSAccepter to get step size for y

alpha_for_y_tol: Tolerance for switching to full equality multiplier steps.

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance. The valid range for this real option is $0 \leq \alpha_{\text{for_y_tol}} < +\text{inf}$ and its default value is 10.

alpha_min_frac: Safety factor for the minimal step size (before switching to restoration phase).

(This is γ_{α} in Eqn. (20) in the implementation paper.) The valid range for this real option is $0 < \alpha_{\text{min_frac}} < 1$ and its default value is 0.05.

alpha_red_factor: Fractional reduction of the trial step size in the backtracking line search.

At every step of the backtracking line search, the trial step size is reduced by this factor. The valid range for this real option is $0 < \alpha_{\text{red_factor}} < 1$ and its default value is 0.5.

constraint_violation_norm_type: Norm to be used for the constraint violation in the line search.

Determines which norm should be used when the algorithm computes the constraint violation in the line search. The default value for this string option is "1-norm".

Possible values:

- 1-norm: use the 1-norm
- 2-norm: use the 2-norm
- max-norm: use the infinity norm

corrector_compl_avrg_red_fact: Complementarity tolerance factor for accepting corrector step (unsupported!).

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted. The valid range for this real option is $0 < \text{corrector_compl_avrg_red_fact} < +\text{inf}$ and its default value is 1.

corrector_type: The type of corrector steps that should be taken (unsupported!).

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried. The default value for this string option is "none".

Possible values:

- none: no corrector
- affine: corrector step towards $\mu=0$
- primal-dual: corrector step towards current μ

delta: Multiplier for constraint violation in the switching rule.

(See Eqn. (19) in the implementation paper.) The valid range for this real option is $0 < \text{delta} < +\text{inf}$ and its default value is 1.

eta_phi: Relaxation factor in the Armijo condition.

(See Eqn. (20) in the implementation paper) The valid range for this real option is $0 < \text{eta_phi} < 0.5$ and its default value is $1 \cdot 10^{-08}$.

filter_reset_trigger: Number of iterations that trigger the filter reset.

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset. The valid range for this integer option is $1 \leq \text{filter_reset_trigger} < +\text{inf}$ and its default value is 5.

gamma_phi: Relaxation factor in the filter margin for the barrier function.

(See Eqn. (18a) in the implementation paper.) The valid range for this real option is $0 < \text{gamma_phi} < 1$ and its default value is $1 \cdot 10^{-08}$.

gamma_theta: Relaxation factor in the filter margin for the constraint violation.

(See Eqn. (18b) in the implementation paper.) The valid range for this real option is $0 < \text{gamma_theta} < 1$ and its default value is $1 \cdot 10^{-05}$.

kappa_sigma: Factor limiting the deviation of dual variables from primal estimates.

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction. The valid range for this real option is $0 < \text{kappa_sigma} < +\text{inf}$ and its default value is $1 \cdot 10^{+10}$.

kappa_soc: Factor in the sufficient reduction rule for second order correction.

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.) The valid range for this real option is $0 < \text{kappa_soc} < +\text{inf}$ and its default value is 0.99.

max_filter_resets: Maximal allowed number of filter resets

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place. The valid range for this integer option is $0 \leq \text{max_filter_resets} < +\text{inf}$ and its default value is 5.

max_soc: Maximum number of second order correction trial steps at each iteration.

Choosing 0 disables the second order corrections. (This is p^{max} of Step A-5.9 of Algorithm A in the implementation paper.) The valid range for this integer option is $0 \leq \text{max_soc} < +\text{inf}$ and its default value is 4.

nu_inc: Increment of the penalty parameter.

The valid range for this real option is $0 < \text{nu_inc} < +\text{inf}$ and its default value is 0.0001.

nu_init: Initial value of the penalty parameter.

The valid range for this real option is $0 < \text{nu_init} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

obj_max_inc: Determines the upper bound on the acceptable increase of barrier objective function.

Trial points are rejected if they lead to an increase in the barrier objective function by more than `obj_max_inc` orders of magnitude. The valid range for this real option is $1 < \text{obj_max_inc} < +\text{inf}$ and its default value is 5.

recalc_y: Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than `recalc_y_feas_tol`. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default. The default value for this string option is "no".

Possible values:

- no: use the Newton step to update the multipliers
- yes: use least-square multiplier estimates

recalc_y_feas_tol: Feasibility threshold for recomputation of multipliers.

If `recalc_y` is chosen and the current infeasibility is less than this value, then the multipliers are recomputed. The valid range for this real option is $0 < \text{recalc_y_feas_tol} < +\text{inf}$ and its default value is $1 \cdot 10^{-06}$.

rho: Value in penalty parameter update formula.

The valid range for this real option is $0 < \text{rho} < 1$ and its default value is 0.1.

s_phi: Exponent for linear barrier function model in the switching rule.

(See Eqn. (19) in the implementation paper.) The valid range for this real option is $1 < \text{s_phi} < +\text{inf}$ and its default value is 2.3.

s_theta: Exponent for current constraint violation in the switching rule.

(See Eqn. (19) in the implementation paper.) The valid range for this real option is $1 < \text{s_theta} < +\text{inf}$ and its default value is 1.1.

skip_corr_if_neg_curv: Skip the corrector step in negative curvature iteration (unsupported!).

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive". The default value for this string option is "yes".

Possible values:

- no: don't skip
- yes: skip

skip_corr_in_monotone_mode: Skip the corrector step during monotone barrier parameter mode (unsupported!).

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive". The default value for this string option is "yes".

Possible values:

- no: don't skip
- yes: skip

slack_move: Correction size for very small slacks.

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is $\text{mach_eps}^{3/4}$. (See also end of Section 3.5 in the implementation paper - but actual implementation might be somewhat different.) The valid range for this real option is $0 \leq \text{slack_move} < +\text{inf}$ and its default value is $1.81899 \cdot 10^{-12}$.

theta_max_fact: Determines upper bound for constraint violation in the filter.

The algorithmic parameter `theta_max` is determined as `theta_max_fact` times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than `theta_max` is unacceptable to the filter (see Eqn. (21) in the implementation paper). The valid range for this real option is $0 < \text{theta_max_fact} < +\text{inf}$ and its default value is 10000.

theta_min_fact: Determines constraint violation threshold in the switching rule.

The algorithmic parameter `theta_min` is determined as `theta_min_fact` times the maximum of 1 and the constraint violation at initial point. The switching rule treats an iteration as an h-type iteration whenever the current constraint violation is larger than `theta_min` (see paragraph before Eqn. (19) in the implementation paper). The valid range for this real option is $0 < \text{theta_min_fact} < +\text{inf}$ and its default value is 0.0001.

tiny_step_tol: Tolerance for detecting numerically insignificant steps.

If the search direction in the primal variables (`x` and `s`) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision. The valid range for this real option is $0 \leq \text{tiny_step_tol} < +\text{inf}$ and its default value is $2.22045 \cdot 10^{-15}$.

tiny_step_y_tol: Tolerance for quitting because of numerically insignificant steps.

If the search direction in the primal variables (`x` and `s`) is, in relative terms for each component, repeatedly less than `tiny_step_tol`, and the step in the `y` variables is smaller than this threshold, the algorithm will terminate. The valid range for this real option is $0 \leq \text{tiny_step_y_tol} < +\text{inf}$ and its default value is 0.01.

watchdog_shortened_iter_trigger: Number of shortened iterations that trigger the watchdog.

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure. The valid range for this integer option is $0 \leq \text{watchdog_shortened_iter_trigger} < +\text{inf}$ and its default value is 10.

watchdog_trial_iter_max: Maximum number of watchdog iterations.

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point. The valid range for this integer option is $1 \leq \text{watchdog_trial_iter_max} < +\text{inf}$ and its default value is 3.

Restoration Phase

bound_mult_reset_threshold: Threshold for resetting bound multipliers after the restoration phase.

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the

threshold specified by this option, the multipliers are all reset to 1. The valid range for this real option is $0 \leq \text{bound_mult_reset_threshold} < +\text{inf}$ and its default value is 1000.

constr_mult_reset_threshold: Threshold for resetting equality and inequality multipliers after restoration phase.

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored. The valid range for this real option is $0 \leq \text{constr_mult_reset_threshold} < +\text{inf}$ and its default value is 0.

evaluate_orig_obj_at_resto_trial: Determines if the original objective function should be evaluated at restoration phase trial points.

Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly. The default value for this string option is "yes".

Possible values:

- no: skip evaluation
- yes: evaluate at every trial point

expect_infeasible_problem: Enable heuristics to quickly detect an infeasible problem.

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically. The default value for this string option is "no".

Possible values:

- no: the problem probably be feasible
- yes: the problem has a good chance to be infeasible

expect_infeasible_problem_ctol: Threshold for disabling "expect_infeasible_problem" option.

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0. The valid range for this real option is $0 \leq \text{expect_infeasible_problem_ctol} < +\text{inf}$ and its default value is 0.001.

max_resto_iter: Maximum number of successive iterations in restoration phase.

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number. The valid range for this integer option is $0 \leq \text{max_resto_iter} < +\text{inf}$ and its default value is 3000000.

max_soft_resto_iters: Maximum number of iterations performed successively in soft restoration phase.

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called. The valid range for this integer option is $0 \leq \text{max_soft_resto_iters} < +\text{inf}$ and its default value is 10.

required_infeasibility_reduction: Required reduction of infeasibility before leaving restoration phase.

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option. The valid range for this real option is $0 \leq \text{required_infeasibility_reduction} < 1$ and its default value is 0.9.

resto_penalty_parameter: Penalty parameter in the restoration phase objective function.

This is the parameter ρ in equation (31a) in the Ipopt implementation paper. The valid range for this real option is $0 \leq \text{resto_penalty_parameter} < +\text{inf}$ and its default value is 1000.

soft_resto_pderror_reduction_factor: Required reduction in primal-dual error in the soft restoration phase.

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase. The valid range for this real option is $0 \leq \text{soft_resto_pderror_reduction_factor} < +\text{inf}$ and its default value is 0.9999.

start_with_resto: Tells algorithm to switch to restoration phase in first iteration.

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure. The default value for this string option is "no".

Possible values:

- no: don't force start in restoration phase
- yes: force start in restoration phase

Step Calculation

fast_step_computation: Indicates if the linear system should be solved quickly.

If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster. The default value for this string option is "no".

Possible values:

- no: Verify solution of linear system by computing residuals.
- yes: Trust that linear systems are solved well.

first_hessian_perturbation: Size of first x-s perturbation tried.

The first value tried for the x-s perturbation in the inertia correction scheme. (This is δ_0 in the implementation paper.) The valid range for this real option is $0 < \text{first_hessian_perturbation} < +\text{inf}$ and its default value is 0.0001.

jacobian_regularization_exponent: Exponent for μ in the regularization for rank-deficient constraint Jacobians.

The valid range for this real option is $0 \leq \text{jacobian_regularization_exponent} < +\text{inf}$ and its default value is 0.25. (This is κ_c in the implementation paper.)

jacobian_regularization_value: Size of the regularization for rank-deficient constraint Jacobians.

The valid range for this real option is $0 \leq \text{jacobian_regularization_value} < +\text{inf}$ and its default value is $1 \cdot 10^{-08}$. (This is $\bar{\delta}_c$ in the implementation paper.)

max_hessian_perturbation: Maximum value of regularization parameter for handling negative curvature.

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum

value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is δ_w^{\max} in the implementation paper.) The valid range for this real option is $0 < \text{max_hessian_perturbation} < +\text{inf}$ and its default value is $1 \cdot 10^{+20}$.

max_refinement_steps: Maximum number of iterative refinement steps per linear system solve. Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps. The valid range for this integer option is $0 \leq \text{max_refinement_steps} < +\text{inf}$ and its default value is 10.

mehrotra_algorithm: Indicates if we want to do Mehrotra's algorithm.

If set to yes, Ipopt runs as Mehrotra's predictor-corrector algorithm. This works usually very well for LPs and convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive mu strategy with the "probing" oracle, and uses "corrector_type=affine" without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult". The default value for this string option is "no".

Possible values:

- no: Do the usual Ipopt algorithm.
- yes: Do Mehrotra's predictor-corrector algorithm.

min_hessian_perturbation: Smallest perturbation of the Hessian block.

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. (This is δ_w^{\min} in the implementation paper.) The valid range for this real option is $0 \leq \text{min_hessian_perturbation} < +\text{inf}$ and its default value is $1 \cdot 10^{-20}$.

min_refinement_steps: Minimum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.) The valid range for this integer option is $0 \leq \text{min_refinement_steps} < +\text{inf}$ and its default value is 1.

neg_curv_test_tol: Tolerance for heuristic to ignore wrong inertia.

If positive, incorrect inertia in the augmented system is ignored, and we test if the direction is a direction of positive curvature. This tolerance determines when the direction is considered to be sufficiently positive. The valid range for this real option is $0 < \text{neg_curv_test_tol} < +\text{inf}$ and its default value is 0.

perturb_always_cd: Active permanent perturbation of constraint linearization.

This options makes the delta_c and delta_d perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular. The default value for this string option is "no".

Possible values:

- no: perturbation only used when required
- yes: always use perturbation

perturb_dec_fact: Decrease factor for x-s perturbation.

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is κ_w^- in the implementation paper.) The valid range for this real option is $0 < \text{perturb_dec_fact} < 1$ and its default value is 0.333333.

perturb_inc_fact: Increase factor for x-s perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is κ_w^+ in the implementation paper.) The valid range for this real option is $1 < \text{perturb_inc_fact} < +\text{inf}$ and its default value is 8.

perturb_inc_fact_first: Increase factor for x-s perturbation for very first perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for for the first perturbation than that used for the remaining perturbations. (This is $\bar{\kappa}_w^+$ in the implementation paper.) The valid range for this real option is $1 < \text{perturb_inc_fact_first} < +\text{inf}$ and its default value is 100.

residual_improvement_factor: Minimal required reduction of residual test ratio in iterative refinement.

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted. The valid range for this real option is $0 < \text{residual_improvement_factor} < +\text{inf}$ and its default value is 1.

residual_ratio_max: Iterative refinement tolerance

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed). The valid range for this real option is $0 < \text{residual_ratio_max} < +\text{inf}$ and its default value is $1 \cdot 10^{-10}$.

residual_ratio_singular: Threshold for declaring linear system singular after failed iterative refinement.

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular. The valid range for this real option is $0 < \text{residual_ratio_singular} < +\text{inf}$ and its default value is $1 \cdot 10^{-05}$.

Warm Start

warm_start_bound_frac: same as bound_frac for the regular initializer.

The valid range for this real option is $0 < \text{warm_start_bound_frac} \leq 0.5$ and its default value is 0.001.

warm_start_bound_push: same as bound_push for the regular initializer.

The valid range for this real option is $0 < \text{warm_start_bound_push} < +\text{inf}$ and its default value is 0.001.

warm_start_entire_iterate: Tells algorithm whether to use the GetWarmStartIterate method in the NLP.

The default value for this string option is "no".

Possible values:

- no: call GetStartingPoint in the NLP
- yes: call GetWarmStartIterate in the NLP

warm_start_init_point: Warm-start for initial point

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.) The default value for this string option is "no".

Possible values:

- no: do not use the warm start initialization
- yes: use the warm start initialization

warm_start_mult_bound_push: same as `mult_bound_push` for the regular initializer.

The valid range for this real option is $0 < \text{warm_start_mult_bound_push} < +\text{inf}$ and its default value is 0.001.

warm_start_mult_init_max: Maximum initial value for the equality multipliers.

The valid range for this real option is $-\text{inf} < \text{warm_start_mult_init_max} < +\text{inf}$ and its default value is $1 \cdot 10^{+06}$.

warm_start_slack_bound_frac: same as `slack_bound_frac` for the regular initializer.

The valid range for this real option is $0 < \text{warm_start_slack_bound_frac} \leq 0.5$ and its default value is 0.001.

warm_start_slack_bound_push: same as `slack_bound_push` for the regular initializer.

The valid range for this real option is $0 < \text{warm_start_slack_bound_push} < +\text{inf}$ and its default value is 0.001.

7 CoinOS

GAMS/CoinOS brings the Optimization Services project to the broad audience of GAMS users.

OS (Optimization Services) is an initiative to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. The code has been written primarily by Horand Gassmann, Jun Ma, and Kipp Martin. Kipp Martin is the COIN-OR project leader for OS.

For more information we refer to

- the official OS web site <http://www.optimizationservices.org>,
- the OS developer web site <https://projects.coin-or.org/OS>, and
- the OS manual available on the OS websites.

The OS link in GAMS allows you to convert instances of GAMS models into the OS instance language (OSiL) format, let an Optimization Services Server solve your instances remotely, or solve your models locally via the OS solver interfaces (currently Bonmin, Cbc, Clp, Couenne, Glpk, and Ipopt).

7.1 Model requirements

OS supports continuous, binary, and integer variables, linear and nonlinear equations.

7.2 Usage of CoinOS

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

```
Option MINLP = CoinOS;      { or LP, RMIP, MIP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP }
```

The above statement should appear before the Solve statement.

By default, for a given instance of a GAMS model, CoinOS selects a suitable solver, let the solver solve this instance (locally) via the corresponding OS solver interface and returns the solution to GAMS. For continuous linear models (LP and RMIP), CoinOS chooses Clp. For continuous nonlinear models (NLP, DNLP, RMINLP, QCP, RMIQCP), CoinOS chooses Ipopt. For mixed-integer linear models (MIP), CoinOS chooses Cbc. For mixed-integer nonlinear models (MIQCP, MINLP), CoinOS chooses Bonmin. You can change the choice of the solver by setting the “solver” option.

For a remove solve, you have to specify the URL of an Optimization Services Server via the option “service”. Remove solves are experimental in CoinOS.

Solver options can be specified with an OSoL (Optimization Services Options Language) file. The `osol` file is specified with the “`readosol`” option.

7.3 Detailed Descriptions of CoinOS Options

readosol (*string*)

Specifies the name of an option file in OSoL format that is given to the solver or an OS server.

writeosil (*string*)

Specifies the name of a file in which the GAMS model instance should be writing in OSiL format.

writeosrl (*string*)

Specifies the name of a file in which the result of a solve process (solution, status, ...) should be writing in OSrL format.

service (*string*)

Specifies the URL of an Optimization Services Server. If specified, then this server is contacted and the service method specified under `service_method` is executed. Note that by default the server executes CBC to solve an instance. You can change the solver with the `solver` option.

service_method (*string*)

Specifies the method to execute on a server.

(*default = solve*)

`solve` Specifies that an OS server should solve the provided instance and return the result. CoinOS will wait until the OS server returns the result.

`getJobID` Specifies that an available Job identifier should be requested from an OS server.

`knock` Specifies that the status of a solve process is requested from an OS server. The `knock` service method requires an OSpL file as input, see option `readospl`.

`kill` Specifies that a solve process should be interrupted on an OS server. You should specify an OSoL file containing the JobID by using the `readosil` option.

`send` Specifies that the model instance is send to an OS server and the server should solve this instance. CoinOS does not wait until the OS server returns the result but returns when submission of the instance is completed.

`retrieve` Specifies that an optimization result should be requested from an OS server. You should specify an OSoL file containing the JobID by using the `readosil` option.

solver (*string*)

Specifies the solver that is used to solve an instance. Valid values are Clp, Cbc, Glpk, Ipopt, Bonmin, Couenne.

readospl (*string*)

Specifies the name of an OSpL file to use for the `knock` method.

writeospl (*string*)

Specifies the name of an OSpL file in which the answer from the `knock` or `kill` method is written.

8 CoinScip

GAMS/CoinScip brings the MIP solver from the Constrained Integer Programming framework SCIP to the broad audience of academic GAMS users.

The code is developed at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and has been written primarily by T. Achterberg. It is distributed under the ZIB Academic License.

For more information we refer to

- the SCIP web site <http://scip.zib.de> and

- the Ph.D. thesis “Constraint Integer Programming” by Tobias Achterberg, Berlin 2007.

GAMS/CoinScip uses the COIN-OR linear solver CLP from J.J. Forrest as LP solver, see [Section 3](#).

8.1 Model requirements

SCIP supports continuous, binary, and integer variables, special ordered sets, and branching priorities. Semi-continuous or semi-integer variables (see chapter 17.1 of the GAMS User’s Guide) and indicator constraints are not supported by the interface yet.

8.2 Usage of CoinScip

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

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

The above statement should appear before the Solve statement. If CoinScip was specified as the default solver during GAMS installation, the above statement is not necessary.

GAMS/CoinScip supports the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. The GAMS BCH facility automates all major steps necessary to define, execute, and control the use of user defined routines within the framework of general purpose MIP codes. Currently supported are user defined cut generators and heuristics and the incumbent reporting callback. Please see the BCH documentation at <http://www.gams.com/docs/bch.htm> for further information.

Information on the use of BCH callback routines is displayed in an extra column in the SCIP iteration output. The first number in this column (below the “BCH” in the header) is the number of callbacks to GAMS models that have been made so far (accumulated from cutgeneration, heuristic, and incumbent callbacks). The number below “cut” or “cuts” gives the number of cutting planes that have been generated by the users cutgenerator. Finally, the number below “sol” or “sols” gives the number of primal solutions that have been generated by the users heuristic. If SCIP accepts a heuristic solution as new incumbent solution, it prints a ‘G’ in the first column of the iteration output.

8.3 Specification of CoinScip Options

GAMS/CoinScip currently supports the GAMS parameters reslim, iterlim, nodlim, optcr, and optca.

Further, for a MIP solve the user can specify options by a SCIP options file. A SCIP options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs). Further, string values have to be enclosed in quotation marks.

A small example for a coinscip.opt file is:

```
separating/maxrounds      = 0
separating/maxroundsroot = 0
gams/solvefinal           = FALSE
gams/usercutcall          = "bchcutgen.gms"
```

It causes GAMS/CoinScip to turn off all cut generators, to skip the final solve of the MIP with fixed discrete variables, and to use a user defined cut generator.

8.4 Description of CoinScip options

SCIP supports a large set of options. Sample option files can be obtained from

<http://www.gams.com/~svigerske/scip1.2>

Further, there is a set of options that are specific to the GAMS/CoinScip interface, most of them for control of the GAMS BCH facility.

gams/names (*integer*)

This option causes GAMS names for the variables and equations to be loaded into SCIP. These names will then be used for error messages, log entries, and so forth. Turning names off may help if memory is very tight.

(*default = FALSE*)

FALSE Do not load variable and equation names.

TRUE Load variable and equation names.

gams/solvefinal (*integer*)

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

(*default = TRUE*)

FALSE Do not solve the fixed problem.

TRUE Solve the fixed problem and return duals.

gams/mipstart (*integer*)

This option controls the use of advanced starting values for mixed integer programs. A setting of TRUE indicates that the variable level values should be checked to see if they provide an integer feasible solution before starting optimization.

(*default = TRUE*)

FALSE Do not use the initial variable levels.

TRUE Try to use the initial variable levels as a MIP starting solution.

gams/print_statistics (*integer*)

This option controls the printing of solve statistics after a MIP solve. Turning on this option indicates that statistics like the number of generated cuts of each type or the calls of heuristics are printed after the MIP solve.

(*default = FALSE*)

FALSE Do not print statistics.

TRUE Print statistics.

gams/usercutcall (*string*)

The GAMS command line (minus the gams executable name) to call the cut generator.

gams/usercutfirst (*integer*)

Calls the cut generator for the first n nodes.

(*default = 10*)

gams/usercutfreq (*integer*)

Determines the frequency of the cut generator model calls.

(*default = 10*)

gams/usercutinterval (integer)

Determines the interval when to apply the multiplier for the frequency of the cut generator model calls. See gams/userheurinterval for details.

(default = 100)

gams/usercutmult (integer)

Determines the multiplier for the frequency of the cut generator model calls.

(default = 2)

gams/usercutnewint (integer)

Calls the cut generator if the solver found a new integer feasible solution.

(default = TRUE)

FALSE Do not call cut generator because a new integer feasible solution is found.

TRUE Let SCIP call the cut generator if a new integer feasible solution is found.

gams/usergdxin (string)

The name of the GDX file read back into SCIP.

(default = bchin.gdx)

gams/usergdxname (string)

The name of the GDX file exported from the solver with the solution at the node.

(default = bchout.gdx)

gams/usergdxnameinc (string)

The name of the GDX file exported from the solver with the incumbent solution.

(default = bchout_i.gdx)

gams/usergdxprefix (string)

Prefixes to use for gams/usergdxin, gams/usergdxname, and gams/usergdxnameinc.

gams/userheurcall (string)

The GAMS command line (minus the gams executable name) to call the heuristic.

gams/userheurfirst (integer)

Calls the heuristic for the first n nodes.

(default = 10)

gams/userheurfreq (integer)

Determines the frequency of the heuristic model calls.

(default = 10)

gams/userheurinterval (integer)

Determines the interval when to apply the multiplier for the frequency of the heuristic model calls. For example, for the first 100 (gams/userheurinterval) nodes, the solver call every 10th (gams/userheurfreq) node the heuristic. After 100 nodes, the frequency gets multiplied by 10 (gams/userheurmultip), so that for the next 100 node the solver calls the heuristic every 20th node. For nodes 200-300, the heuristic get called every 40th node, for nodes 300-400 every 80th node and after node 400 every 100th node.

(default = 100)

gams/userheurmultip (integer)

Determines the multiplier for the frequency of the heuristic model calls.

(default = 2)

gams/userheurnewint (*integer*)

Calls the heuristic if the solver found a new integer feasible solution.

(*default = TRUE*)

FALSE Do not call heuristic because a new integer feasible solution is found.

TRUE Let SCIP call the heuristic if a new integer feasible solution is found.

gams/userheurobjfirst (*integer*)

Similar to gams/userheurfirst but only calls the heuristic if the relaxed objective value promises a significant improvement of the current incumbent, i.e., the LP value of the node has to be closer to the best bound than the current incumbent.

(*default = FALSE*)

gams/userjobid (*string*)

Postfixes to use for gams/gdxname, gams/gdxnameinc, and gams/gdxin.

gams/userkeep (*integer*)

Calls gamskeep instead of gams

(*default = FALSE*)

9 CoinCplex, CoinGurobi, CoinMosek, CoinXpress

GAMS/CoinCplex, GAMS/CoinGurobi, GAMS/CoinMosek, GAMS/CoinXpress bring the open source Open Solver Interface (OSI) to the broad audience of GAMS users.

These “bare bone” solver links allow users to solve their GAMS models with a standalone license of Cplex, Gurobi, Mosek, or Xpress. The links use the COIN-OR Open Solver Interface (OSI) to communicate with these solvers. The Osi/Cplex link has been written primarily by Tobias Achterberg, the Osi/Gurobi link has been written primarily by Stefan Vigerske, the Osi/Mosek link has been written primarily by Bo Jensen, and the Osi/Xpress link has been written primarily by John Doe. Matthew Saltzman is the COIN-OR project leader for OSI.

For more information we refer to the OSI web site <https://projects.coin-or.org/Osi>.

9.1 Model requirements

The OSI links support linear equations and continuous, binary, and integer variables. Semicontinuous and Semi-integer variables, special ordered sets, branching priorities, and indicator constraints are not supported by OSI.

9.2 Usage of these links

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

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

The above statement should appear before the Solve statement.

The links only support the general GAMS options reslim, optca (except for Gurobi), optcr, nodlim, and iterlim. In addition an option file in the format required by the solver can be provided via the GAMS optfile option.