COIN-OR

Stefan Vigerske, Humboldt University Berlin, Germany

Contents

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 [19]. 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. 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:

- Bonmin: Basic Open-source Nonlinear Mixed Integer programming
  (model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- CBC: COIN-OR Branch and Cut
  (model types: LP, RMIP, MIP)
- Couenne: Convex Over and Under Envelopes for Nonlinear Estimation
  (model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- Ipopt: Interior Point Optimizer
  (model types: LP, RMIP, DNLP, NLP, RMINLP, QCP, RMIQCP)
- OS: Optimization Services
  (model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- OsiCplex, OsiGlpk, OsiGurobi, OsiMosek, OsiSoplex, OsiXpress: Open Solver Interface
  (model types: LP, RMIP, MIP)

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

2 Bonmin

Bonmin (**B**asic **O**pen-source **N**onlinear **M**ixed **In**teger programming) is an open-source solver for mixed-integer nonlinear programming (MINLPs). The code has been developed as part of a collaboration between Carnegie Mellon University and IBM Research. The COIN-OR project leader for Bonmin is Pierre Bonami.

Bonmin implements six different algorithms for solving MINLPs:

- 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 [17]; 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 [12, 13]; this algorithm is similar to the one implemented in the solver DICOPT
- B-QG: an outer-approximation based branch-and-cut algorithm 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 [21].
- 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 [11]
- B-ECP: a Kelley's outer-approximation based branch-and-cut algorithm inspired by the settings used in the solver FilMINT [1]
- B-iFP: an iterated feasibility pump algorithm [7]

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 (especially when used with CPLEX as MIP subproblem solver) is much faster than B-Hyb and others where B-BB is interesting. B-QG and B-ECP corresponds mainly to a specific parameter setting of B-Hyb but they can be faster in some cases. B-iFP is more tailored at finding quickly good solutions to very hard convex

MINLP. For **nonconvex** MINLPs, it is strongly recommended to use B-BB (the outer-approximation algorithms have not been tailored to treat nonconvex problems at this point). Although even B-BB is only a heuristic for such problems, several options are available to try and improve the quality of the solutions it provides.

For more information we refer to [8, 9, 11, 7] and the Bonmin web site `https://projects.coin-or.org/Bonmin`. Most of the Bonmin documentation in this section is taken from the Bonmin manual [10].

2.1 Model requirements

Bonmin can handle mixed-integer nonlinear programming models which functions 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/Bonmin is called for a model with only continuous variables, the interface switches over to Ipopt. If GAMS/Bonmin is called for a model with only linear equations, the interface switches over to CBC.

2.2 Usage

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

```
Option MINLP = BONMIN;    { or Option MIQCP = BONMIN; }
```

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

GAMS/Bonmin currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/Bonmin with BCH, please consider to use a GAMS system of version $\leq$ 23.3, available at http://www.gams.com/download/download_old.htm.

Specification of Options

A Bonmin option file contains both Ipopt and Bonmin options, for clarity all Bonmin options should be preceded with the prefix "`bonmin.`". The scheme to name option files is the same as for all other GAMS solvers. Specifying `optfile=1` let GAMS/Bonmin read `bonmin.opt`, `optfile=2` corresponds to `bonmin.op2`, and so on. The format of the option file is the same as for Ipopt (see Section 5.2).

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`, `B-ECP`, or `B-iFP` (its default value is `B-BB`). Depending on the value of this option, certain other options may be available or not, cf. Section 2.3.

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

```
bonmin.algorithm        B-Hyb
bonmin.oa_log_level     4
print_level             6
```

This sets the algorithm to be used to the hybrid algorithm, the level of outer approximation related output to 4, and sets the print level for Ipopt to 6.

GAMS/Bonmin 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). One can set them either on the command line, e.g. `nodlim=1000`, or inside your GAMS program, e.g. `Option nodlim=1000;`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the linear algebra routines of Ipopt.

Passing options to local search based heuristics and OA generators

Several parts of the algorithms in Bonmin are based on solving a simplified version of the problem with another instance of Bonmin: Outer Approximation Decomposition (called in `B-Hyb` at the root node) and Feasibility Pump for MINLP (called in `B-Hyb` or `B-BB` at the root node), RINS, RENS, Local Branching.

In all these cases, one can pass options to the sub-algorithm used through the option file. The basic principle is that the "`bonmin.`" prefix is replaced with a prefix that identifies the sub-algorithm used:

- to pass options to Outer Approximation Decomposition: `oa_decomposition.`,
- to pass options to Feasibility Pump for MINLP: `pump_for_minlp.`,
- to pass options to RINS: `rins.`,
- to pass options to RENS: `rens.`,
- to pass options to Local Branching: `local_branch`.

For example, to run a maximum of 60 seconds of feasibility pump (FP) for MINLP until 6 solutions are found at the beginning of the hybrid algorithm, one sets the following options:

```
bonmin.algorithm              B-Hyb
bonmin.pump_for_minlp         yes   # tells to run FP for MINLP
pump_for_minlp.time_limit     60    # set a time limit for the pump
pump_for_minlp.solution_limit 6     # set a solution limit
```

Note that the actual solution and time limit will be the minimum of the global limits set for Bonmin.

A slightly more complicated set of options may be used when using RINS. Say for example that one wants to run RINS inside `B-BB`. Each time RINS is called one wants to solve the small-size MINLP generated using B-QG (one may run any algorithm available in Bonmin for solving an MINLP) and wants to stop as soon as `B-QG` found one solution. To achieve this, one sets the following options

```
bonmin.algorithm      B-BB
bonmin.heuristic_rins yes
rins.algorithm        B-QG
rins.solution_limit   1
```

This example shows that it is possible to set any option used in the sub-algorithm to be different than the one used for the main algorithm.

In the context of outer-approximation (OA) and feasibility pump for MINLP, a standard MILP solver is used. Several options are available for configuring this MILP solver. Bonmin allows a choice of different MILP solvers through the option `bonmin.milp_solver`. Values for this option are: `Cbc_D` which uses CBC with its default settings, `Cbc_Par` which uses a version of CBC that can be parameterized by the user, and `Cplex` which uses CPLEX with its default settings. The options that can be set in `Cbc_Par` are the number of strong-branching candidates, the number of branches before pseudo costs are to be trusted, and the frequency of the various cut generators, c.f. Section 2.3 for details. To use the `Cplex` option, a valid CPLEX licence (standalone or GAMS/CPLEX) is required.

Getting good solutions to nonconvex problems

To solve a problem with nonconvex constraints, one should only use the branch-and-bound algorithm `B-BB`.

A few options have been designed in Bonmin specifically to treat problems that do not have a convex continuous relaxation. In such problems, the solutions obtained from Ipopt are not necessarily globally optimal, but are only locally optimal. Also the outer-approximation constraints are not necessarily valid inequalities for the problem. No specific heuristic method for treating nonconvex problems is implemented yet within the OA framework. But for the pure branch-and-bound `B-BB`, a few options have been implemented while having in mind that lower bounds provided by Ipopt should not be trusted and with the goal of trying to get good solutions. Such options are at a very experimental stage.

First, in the context of nonconvex problems, Ipopt may find different local optima when started from different starting points. The two options `num_resolve_at_root` and `num_resolve_at_node` allow for solving the root node or each node of the tree, respectively, with a user-specified number of different randomly-chosen starting points, saving the best solution found. Note that the function to generate a random starting point is very naïve: it chooses a random point (uniformly) between the bounds provided for the variable. In particular if there are some functions that can not be evaluated at some points of the domain, it may pick such points, and so it is not robust in that respect.

Secondly, since the solution given by Ipopt does not truly give a lower bound, the fathoming rule can be changed to continue branching even if the solution value to the current node is worse than the best-known solution. This is achieved by setting `allowable_gap` and `allowable_fraction_gap` and `cutoff_decr` to negative values.

Ipopt options changed by Bonmin

Ipopt has a very large number of options, see Section 5.4 to get a complete description. To use Ipopt more efficiently in the context of MINLP, Bonmin changes some Ipopt options from their default values, which may help to improve Ipopt's warm-starting capabilities and its ability to prove quickly that a subproblem is infeasible. These are settings that Ipopt does not use for ordinary NLP problems. Note that options set by the user in an option file will override these settings.

- `mu_strategy` and `mu_oracle` are set, respectively, to `adaptive` and `probing` by default. These are strategies in Ipopt for updating the barrier parameter. They were found to be more efficient in the context of MINLP.
- `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 to 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 (except for the root node, which print level is controlled by the Bonmin option `nlp_log_at_root`).
- `bound_relax_factor` is set to 10^{-10}. All of the bounds of the problem are relaxed by this factor. This may cause some trouble when constraint functions can only be evaluated within their bounds. In such cases, this option should be set to 0.

2.3 Detailed Options Description

The following tables gives the list of options together with their types, default values, and availability in each of the main algorithms. The column labeled '`Cbc_Par`' indicates the options that can be used to parametrize the MLIP subsolver in the context of OA and FP.

Table 1.1: List of options and compatibility with the different algorithms.

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
Algorithm choice									
algorithm	string	B-BB	✓	✓	✓	✓	✓	✓	✓
Branch-and-bound options									
allowable_fraction_gap	$\mathbb{Q}$	GAMS `optcr`	✓	✓	✓	✓	✓	✓	✓
allowable_gap	$\mathbb{Q}$	GAMS `optca`	✓	✓	✓	✓	✓	✓	✓

continued on next page

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
cutoff	$\mathbb{Q}$	GAMS `cutoff`	✓	✓	✓	✓	✓	✓	✓
cutoff_decr	$\mathbb{Q}$	10^{-5}	✓	✓	✓	✓	✓	✓	✓
enable_dynamic_nlp	string	no	–	–	✓	✓	✓	–	–
integer_tolerance	$\mathbb{Q}$	10^{-6}	✓	✓	✓	✓	✓	✓	✓
iteration_limit	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	✓
nlp_failure_behavior	string	stop	✓	–	–	–	–	–	–
node_comparison	string	best-bound	✓	✓	✓	✓	✓	✓	–
node_limit	$\mathbb{Z}$	GAMS `nodlim`	✓	✓	✓	✓	✓	✓	✓
num_cut_passes	$\mathbb{Z}$	1	–	–	✓	✓	✓	–	–
num_cut_passes_at_root	$\mathbb{Z}$	20	–	–	✓	✓	✓	–	–
number_before_trust	$\mathbb{Z}$	8	✓	✓	✓	✓	✓	✓	✓
number_strong_branch	$\mathbb{Z}$	20	✓	✓	✓	✓	✓	✓	✓
solution_limit	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	✓
time_limit	$\mathbb{Q}$	GAMS `reslim`	✓	✓	✓	✓	✓	✓	✓
tree_search_strategy	string	probed-dive	✓	✓	✓	✓	✓	✓	–
variable_selection	string	strong-branching	✓	–	–	–	–	–	–
ECP cuts generation									
ecp_abs_tol	$\mathbb{Q}$	10^{-6}	–	–	✓	✓	–	–	–
ecp_max_rounds	$\mathbb{Z}$	5	–	–	✓	✓	–	–	–
ecp_probability_factor	$\mathbb{Q}$	10	–	–	✓	✓	–	–	–
ecp_rel_tol	$\mathbb{Q}$	0	–	–	✓	✓	–	–	–
filmint_ecp_cuts	$\mathbb{Z}$	0	–	–	✓	✓	–	–	–
Feasibility checker using OA cuts									
feas_check_cut_types	string	outer-approx	–	–	✓	✓	✓	–	–
feas_check_discard_policy	string	detect-cycles	–	–	✓	✓	✓	–	–
generate_benders_after_so_many_oa	$\mathbb{Z}$	5000	–	–	✓	✓	✓	–	–
MILP Solver									
cpx_parallel_strategy	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
milp_solver	string	Cbc_D	–	–	–	–	–	–	✓
milp_strategy	string	find_good_sol	–	–	–	–	–	–	✓
number_cpx_threads	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
MILP cutting planes in hybrid algorithm (B-Hyb)									
2mir_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
Gomory_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
clique_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
cover_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
flow_cover_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
lift_and_project_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
mir_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
reduce_and_split_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
MINLP Heuristics									
feasibility_pump_objective_norm	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	–
fp_pass_infeasible	string	no	✓	✓	✓	✓	✓	✓	✓
heuristic_RINS	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_MIP_fractional	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_MIP_vectorLength	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_fractional	string	no	✓	✓	✓	✓	✓	✓	–

continued on next page

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
heuristic_dive_vectorLength	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_feasibility_pump	string	no	✓	✓	✓	✓	✓	✓	–
pump_for_minlp	string	no	✓	✓	✓	✓	✓	✓	–
NLP interface									
warm_start	string	none	✓	–	–	–	–	–	–
NLP solution robustness									
max_consecutive_failures	$\mathbb{Z}$	10	✓	–	–	–	–	–	–
max_random_point_radius	$\mathbb{Q}$	100000	✓	–	–	–	–	–	–
num_iterations_suspect	$\mathbb{Z}$	-1	✓	✓	✓	✓	✓	✓	✓
num_retry_unsolved_random_point	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	✓
random_point_perturbation_interval	$\mathbb{Q}$	1	✓	–	–	–	–	–	–
random_point_type	string	Jon	✓	–	–	–	–	–	–
NLP solves in hybrid algorithm (B-Hyb)									
nlp_solve_frequency	$\mathbb{Z}$	10	–	–	–	✓	–	–	–
nlp_solve_max_depth	$\mathbb{Z}$	10	–	–	–	✓	–	–	–
nlp_solves_per_depth	$\mathbb{Q}$	10^{100}	–	–	–	✓	–	–	–
Nonconvex problems									
coeff_var_threshold	$\mathbb{Q}$	0.1	✓	–	–	–	–	–	–
dynamic_def_cutoff_decr	string	no	✓	–	–	–	–	–	–
first_perc_for_cutoff_decr	$\mathbb{Q}$	-0.02	✓	–	–	–	–	–	–
max_consecutive_infeasible	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_infeasibles	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_node	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_root	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
second_perc_for_cutoff_decr	$\mathbb{Q}$	-0.05	✓	–	–	–	–	–	–
Outer Approximation Decomposition (B-OA)									
oa_decomposition	string	no	–	–	✓	✓	✓	–	–
Outer Approximation cuts generation									
add_only_violated_oa	string	no	–	✓	✓	✓	✓	✓	✓
oa_cuts_scope	string	global	–	✓	✓	✓	✓	✓	✓
tiny_element	$\mathbb{Q}$	10^{-8}	–	✓	✓	✓	✓	✓	✓
very_tiny_element	$\mathbb{Q}$	10^{-17}	–	✓	✓	✓	✓	✓	✓
Output									
bb_log_interval	$\mathbb{Z}$	100	✓	✓	✓	✓	✓	✓	✓
bb_log_level	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	✓
fp_log_frequency	$\mathbb{Q}$	100	–	–	✓	✓	–	–	–
fp_log_level	$\mathbb{Z}$	1	–	–	✓	✓	–	–	–
lp_log_level	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
milp_log_level	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
nlp_log_at_root	$\mathbb{Z}$	5	✓	✓	✓	✓	✓	✓	–
nlp_log_level	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	✓
oa_cuts_log_level	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
oa_log_frequency	$\mathbb{Q}$	100	✓	–	–	✓	✓	–	–
oa_log_level	$\mathbb{Z}$	1	✓	–	–	✓	✓	–	–
Strong branching setup									
candidate_sort_criterion	string	best-ps-cost	✓	✓	✓	✓	✓	✓	–
maxmin_crit_have_sol	$\mathbb{Q}$	0.1	✓	✓	✓	✓	✓	✓	–

continued on next page

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
maxmin_crit_no_sol	$\mathbb{Q}$	0.7	✓	✓	✓	✓	✓	✓	–
min_number_strong_branch	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	–
number_before_trust_list	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	–
number_look_ahead	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	–	–
number_strong_branch_root	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	–
setup_pseudo_frac	$\mathbb{Q}$	0.5	✓	✓	✓	✓	✓	✓	–
trust_strong_branching_for_pseudo_cost	string	yes	✓	✓	✓	✓	✓	✓	–

In the following we give a detailed list of Bonmin options. The value on the right denotes the default value.

Algorithm choice

`algorithm` (`B-BB, B-OA, B-QG, B-Hyb, B-Ecp, B-iFP`) `B-BB`
Choice of the algorithm.
This will preset some of the options of bonmin depending on the algorithm choice.

`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.

`B-iFP` Iterated Feasibility Pump for MINLP.

Branch-and-bound options

`allowable_fraction_gap` (real) 0.1
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.

`allowable_gap` (real) 0
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.

`cutoff` ($-10^{100} \leq$ real $\leq 10^{100}$) 10^{100}
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.

`cutoff_decr` ($-10^{10} \leq$ real $\leq 10^{10}$) 10^{-5}
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).

`enable_dynamic_nlp` (`no, yes`) `no`
Enable dynamic linear and quadratic rows addition in nlp

`integer_tolerance` ($0 <$ real) 10^{-6}
Set integer tolerance.
Any number within that value of an integer is considered integer.

`iteration_limit` ($0 \leq$ integer) ∞
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.

`nlp_failure_behavior` (`stop, fathom`) `stop`
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

whithin the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal.

`stop` Stop when failure happens.

`fathom` Continue when failure happens.

`node_comparison` (`best-bound, depth-first, breadth-first, dynamic, best-guess`) `best-bound`
Choose the node selection strategy.
Choose the strategy for selecting the next node to be processed.

`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` ($0 \leq$ integer) ∞
Set the maximum number of nodes explored in the branch-and-bound search.

`num_cut_passes` ($0 \leq$ integer) 1
Set the maximum number of cut passes at regular nodes of the branch-and-cut.

`num_cut_passes_at_root` ($0 \leq$ integer) 20
Set the maximum number of cut passes at regular nodes of the branch-and-cut.

`number_before_trust` ($0 \leq$ integer) 8
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.

`number_strong_branch` ($0 \leq$ integer) 20
Choose the maximum number of variables considered for strong branching.
Set the number of variables on which to do strong branching.

`solution_limit` ($0 \leq$ integer) ∞
Abort after that much integer feasible solution have been found by algorithm
value 0 deactivates option

`time_limit` ($0 \leq$ real) 1000
Set the global maximum computation time (in secs) for the algorithm.

`tree_search_strategy` (`top-node, dive, probed-dive, dfs-dive, dfs-dive-dynamic`) `probed-dive`
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.

`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` (`most-fractional, strong-branching, reliability-branching, curvature-estimator, qp-strong-branching, lp-strong-branching, nlp-strong-branching, osi-simple, osi-strong, random`) `strong-branching`
Chooses variable selection strategy

`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

ECP cuts generation

`ecp_abs_tol` ($0 \leq$ real) 10^{-6}
Set the absolute termination tolerance for ECP rounds.

`ecp_max_rounds` ($0 \leq$ integer) 5
Set the maximal number of rounds of ECP cuts.

`ecp_probability_factor` (real) 10
Factor appearing in formula for skipping ECP cuts.
Choosing -1 disables the skipping.

`ecp_rel_tol` ($0 \leq$ real) 0
Set the relative termination tolerance for ECP rounds.

`filmint_ecp_cuts` ($0 \leq$ integer) 0
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.

Feasibility checker using OA cuts

`feas_check_cut_types` (`outer-approx, Benders`) `outer-approx`
Choose the type of cuts generated when an integer feasible solution is found
If it seems too much memory is used should try Benders to use less

`outer-approx` Generate a set of Outer Approximations cuts.

`Benders` Generate a single Benders cut.

`feas_check_discard_policy` (`detect-cycles, keep-all, treated-as-normal`) `detect-cycles`
How cuts from feasibility checker are discarded
Normally to avoid cycle cuts from feasibility checker should not be discarded in the node where they are generated. However Cbc sometimes does it if no care is taken which can lead to an infinite loop in Bonmin (usualy on simple problems). To avoid this one can instruct Cbc to never discard a cut but if we do that for all cuts it can lead to memory problems. The default policy here is to detect cycles and only then impose to Cbc to keep the cut. The two other alternative are to instruct Cbc to keep all cuts or to just ignore the problem and hope for the best

`detect-cycles` Detect if a cycle occurs and only in this case force not to discard.

`keep-all` Force cuts from feasibility checker not to be discarded (memory hungry but sometimes better).

`treated-as-normal` Cuts from memory checker can be discarded as any other cuts (code may cycle then)

`generate_benders_after_so_many_oa` ($0 \leq$ integer) 5000
Specify that after so many oa cuts have been generated Benders cuts should be generated instead.
It seems that sometimes generating too many oa cuts slows down the optimization compared to Benders due to the size of the LP. With this option we specify that after so many OA cuts have been generated we should switch to Benders cuts.

MILP Solver

`cpx_parallel_strategy` ($-1 \leq$ integer ≤ 1) 0
Strategy of parallel search mode in CPLEX.
-1 = opportunistic, 0 = automatic, 1 = deterministic (refer to CPLEX documentation)

`milp_solver` (`Cbc_D, Cbc_Par, Cplex`) `Cbc_D`
Choose the subsolver to solve MILP sub-problems in OA decompositions.
To use Cplex, a valid license is required.

`Cbc_D` Coin Branch and Cut with its default

`Cbc_Par` Coin Branch and Cut with passed parameters

`Cplex` IBM CPLEX

`milp_strategy` (`find_good_sol, solve_to_optimality`) `find_good_sol`
Choose a strategy for MILPs.

`find_good_sol` Stop sub milps when a solution improving the incumbent is found

`solve_to_optimality` Solve MILPs to optimality

`number_cpx_threads` ($0 \leq$ integer) 0
Set number of threads to use with cplex.
(refer to CPLEX documentation)

MILP cutting planes in hybrid algorithm (B-Hyb)

`2mir_cuts` ($-100 \leq$ integer) 0
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.

`Gomory_cuts` ($-100 \leq$ integer) -5
Frequency k (in terms of nodes) for generating Gomory cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`clique_cuts` ($-100 \leq$ integer) -5
Frequency (in terms of nodes) for generating clique cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`cover_cuts` ($-100 \leq$ integer) 0
Frequency (in terms of nodes) for generating cover cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`flow_cover_cuts` ($-100 \leq$ integer) -5
Frequency (in terms of nodes) for generating flow cover cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`lift_and_project_cuts` ($-100 \leq$ integer) 0
Frequency (in terms of nodes) for generating lift-and-project cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`mir_cuts` ($-100 \leq$ integer) -5
Frequency (in terms of nodes) for generating MIR cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`reduce_and_split_cuts` ($-100 \leq$ integer) 0
Frequency (in terms of nodes) for generating reduce-and-split cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

MINLP Heuristics

`feasibility_pump_objective_norm` ($1 \leq$ integer ≤ 2) 1
Norm of feasibility pump objective function

`fp_pass_infeasible` (`no, yes`) `no`
Say whether feasibility pump should claim to converge or not

`no` When master MILP is infeasible just bail out (don't stop all algorithm). This is the option for using in B-Hyb.

`yes` Claim convergence, numerically dangerous.

`heuristic_RINS` (`no`, `yes`) `no`
if yes runs the RINS heuristic

`heuristic_dive_MIP_fractional` (`no`, `yes`) `no`
if yes runs the Dive MIP Fractional heuristic

`heuristic_dive_MIP_vectorLength` (`no`, `yes`) `no`
if yes runs the Dive MIP VectorLength heuristic

`heuristic_dive_fractional` (`no`, `yes`) `no`
if yes runs the Dive Fractional heuristic

`heuristic_dive_vectorLength` (`no`, `yes`) `no`
if yes runs the Dive VectorLength heuristic

`heuristic_feasibility_pump` (`no`, `yes`) `no`
whether the heuristic feasibility pump should be used

`pump_for_minlp` (`no`, `yes`) `no`
if yes runs FP for MINLP

NLP interface

`warm_start` (`none`, `optimum`, `interior_point`) `none`
Select the warm start method
This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms.

`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` ($0 \leq$ integer) 10
(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).

`max_random_point_radius` ($0 <$ real) 100000
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)

`num_iterations_suspect` ($-1 \leq$ integer) -1
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 outputed in a file (set to -1 to deactivate this).

`num_retry_unsolved_random_point` ($0 \leq$ integer) 0
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.

`random_point_perturbation_interval` ($0 <$ real) 1
Amount by which starting point is perturbed when choosing to pick random point by perturbating starting point

`random_point_type` (`Jon`, `Andreas`, `Claudia`) `Jon`
method to choose a random starting point

`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` ($0 \leq$ integer) 10
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.

`nlp_solve_max_depth` ($0 \leq$ integer) 10
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.

`nlp_solves_per_depth` ($0 \leq$ real) 10^{100}
Set average number of nodes in the tree at which NLP relaxations are solved in B-Hyb for each depth.

Nonconvex problems

`coeff_var_threshold` ($0 \leq$ real) 0.1
Coefficient of variation threshold (for dynamic definition of cutoff_decr).

`dynamic_def_cutoff_decr` (`no, yes`) `no`
Do you want to define the parameter cutoff_decr dynamically?

`first_perc_for_cutoff_decr` (real) -0.02
The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically.

`max_consecutive_infeasible` ($0 \leq$ integer) 0
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.

`num_resolve_at_infeasibles` ($0 \leq$ integer) 0
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.

`num_resolve_at_node` ($0 \leq$ integer) 0
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.

`num_resolve_at_root` ($0 \leq$ integer) 0
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.

`second_perc_for_cutoff_decr` (real) -0.05
The percentage used when, the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically.

Outer Approximation Decomposition (B-OA)

`oa_decomposition` (`no, yes`) `no`
If yes do initial OA decomposition

Outer Approximation cuts generation

`add_only_violated_oa` (`no, yes`) `no`
Do we add all OA cuts or only the ones violated by current point?

`no` Add all cuts

`yes` Add only violated Cuts

`oa_cuts_scope` (`local, global`) `global`
Specify if OA cuts added are to be set globally or locally valid

`local` Cuts are treated as locally valid

`global` Cuts are treated as globally valid

`tiny_element` ($-0 \leq$ real) 10^{-8}
Value for tiny element in OA cut
We will remove "cleanly" (by relaxing cut) an element lower than this.

`very_tiny_element` ($-0 \leq$ real) 10^{-17}
Value for very tiny element in OA cut
Algorithm will take the risk of neglecting an element lower than this.

Output

`bb_log_interval` ($0 \leq$ integer) 100
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.

`bb_log_level` ($0 \leq$ integer ≤ 5) 1
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

`fp_log_frequency` ($0 <$ real) 100
display an update on lower and upper bounds in FP every n seconds

`fp_log_level` ($0 \leq$ integer ≤ 2) 1
specify FP iterations log level.
Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose

`lp_log_level` ($0 \leq$ integer ≤ 4) 0
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

`milp_log_level` ($0 \leq$ integer ≤ 4) 0
specify MILP solver log level.
Set the level of output of the MILP subsolver in OA : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high

`nlp_log_at_root` ($0 \leq$ integer ≤ 12) 5
Specify a different log level for root relaxtion.

`nlp_log_level` ($0 \leq$ integer ≤ 2) 1
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

`oa_cuts_log_level` ($0 \leq$ integer) 0
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.

`oa_log_frequency` ($0 <$ real) 100
display an update on lower and upper bounds in OA every n seconds

`oa_log_level` ($0 \leq$ integer ≤ 2) 1
specify OA iterations log level.
Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose

Strong branching setup

`candidate_sort_criterion` (`best-ps-cost, worst-ps-cost, most-fractional, least-fractional`) `best-ps-cost`
Choice of the criterion to choose candidates in strong-branching

`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` ($0 \leq$ real ≤ 1) 0.1
Weight towards minimum in of lower and upper branching estimates when a solution has been found.

`maxmin_crit_no_sol` ($0 \leq$ real ≤ 1) 0.7
Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.

`min_number_strong_branch` ($0 \leq$ integer) 0
Sets minimum number of variables for strong branching (overriding trust)

`number_before_trust_list` ($-1 \leq$ integer) 0
Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.
The default value is that of "number_before_trust"

`number_look_ahead` ($0 \leq$ integer) 0
Sets limit of look-ahead strong-branching trials

`number_strong_branch_root` ($0 \leq$ integer) ∞
Maximum number of variables considered for strong branching in root node.

`setup_pseudo_frac` ($0 \leq$ real ≤ 1) 0.5
Proportion of strong branching list that has to be taken from most-integer-infeasible list.

`trust_strong_branching_for_pseudo_cost` (`no, yes`) `yes`
Whether or not to trust strong branching results for updating pseudo costs.

3 CBC

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.

For more information we refer to the website of CBC, CGL, and CLP: https://projects.coin-or.org/Cbc, https://projects.coin-or.org/Cgl, https://projects.coin-or.org/Clp. Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

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

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

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

The above statement should appear before the Solve statement. If CBC 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.

Specification of Options

The GAMS/CBC 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 `cbc.opt`.

```
cuts root
perturbation off
```

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

GAMS/CBC currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/CBC with BCH, please consider to use a GAMS system of version $\leq$ 23.3, available at http://www.gams.com/download/download_old.htm.

3.3 Options

Among all CBC options, the following GAMS parameters are currently supported in CBC: reslim, iterlim, nodlim, optca, optcr, cheat, cutoff, threads (only on non-Windows systems).

In the following, we summarize all available CBC 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
tol_dual	dual feasibility tolerance
tol_primal	primal feasibility tolerance
tol_presolve	tolerance used in presolve
passpresolve	how many passes to do in presolve
startalg	LP solver for root node

MIP Options

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

MIP Options for Cutting Plane Generators

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

MIP Options for Heuristics

heuristics	global switch for heuristics
combinesolutions	combine solutions heuristic
dins	distance induced neighborhood search
divingrandom	turns on random diving heuristic
divingcoefficient	coefficient diving heuristic
divingfractional	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 heuristis
rens	relaxation enforced neighborhood search
rins	relaxed induced neighborhood search
roundingheuristic	rounding heuristic
vubheuristic	VUB heuristic

In the following, we give a detailed description of all available CBC 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.

(default = GAMS threads)

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.¡BR>

(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)

4 Couenne

Couenne (**C**onvex **O**ver and **U**nder **En**velopes for **N**onlinear **E**stimation) is an open-source solver for nonconvex mixed-integer nonlinear programming (MINLPs). The code has been developed originally in a cooperation of Carnegie Mellon University and IBM Research, and now at 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 the algorithm used in 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 variables.

For more information on the algorithm we refer to [4, 6] and the Couenne web site https://projects.coin-or.org/Couenne. Most of the Couenne documentation in this section is taken from the Couenne manual [5].

4.1 Model requirements

Couenne can handle mixed-integer nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. Further, an algebraic description of the model need to be made available, which makes the use of some GAMS functions and user-specified external functions impossible. 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/Couenne is called for a linear model, the interface directly calls CBC.

4.2 Usage

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

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

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

Specification of 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 Couenne, please refer to the Sections 5.4 and 2.3 for a detailed description. The scheme to name option files is the same as for all other GAMS solvers. Specifying `optfile=1` let GAMS/Couenne read `couenne.opt`, `optfile=2` corresponds to `couenne.op2`, and so on. The format of the option file is the same as for Ipopt (see Section 5.2).

GAMS/Couenne understands currently the following GAMS parameters: `reslim` (time limit), `nodlim` (node limit), `cutoff`, `optca` (absolute gap tolerance), and `optcr` (relative gap tolerance). One can set them either on the command line, e.g. `nodlim=1000`, or inside your GAMS program, e.g. `Option nodlim=1000;`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the linear algebra routines of Ipopt.

4.3 Detailed Options Description

In the following we give a detailed list of options available for Couenne solely. The value on the right denotes the default value. Note that options specific to Ipopt and Bonmin are not listed her, see Sections 2.3 and 5.4 instead.

`2mir_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating 2mir_cuts 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.

`Gomory_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating Gomory_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`aggressive_fbbt` (`no, yes`) `yes`
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.

`art_cutoff` (real) ∞
Artificial cutoff.

`art_lower` (real) $-\infty$
Artificial lower bound.

`boundtightening_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for bound tightening code in Couenne

`branch_conv_cuts` (`no, yes`) `yes`
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.

`branch_fbbt` (`no, yes`) `yes`
Apply bound tightening before branching
After applying a branching rule and before re-solving the subproblem, apply Bound Tightening.

`branch_lp_clamp` ($0 \leq$ real ≤ 1) 0.2
Defines safe interval percentage for using LP point as a branching point.

`branch_lp_clamp_cube` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_div` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_exp` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_log` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_negpow` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_pow` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_prod` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_sqr` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_trig` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_midpoint_alpha` ($0 \leq$ real ≤ 1) 0.25
Defines convex combination of mid point and current LP point: b = alpha x_lp + (1-alpha) (lb+ub)/2.

`branch_pt_select` (`lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `mid-point`
Chooses branching point selection strategy

- `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
- `no-branch` do not branch, return null infeasibility; for testing purposes only

`branch_pt_select_cube` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator cube. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_div` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`

Chooses branching point selection strategy for operator div. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_exp` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator exp. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_log` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator log. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_negpow` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator negpow. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_pow` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator pow. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_prod` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator prod. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_sqr` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator sqr. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_trig` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator trig. Default is to use the value of `branch_pt_select` (value `common`).

`branching_object` (`vt_obj`, `var_obj`, `expr_obj`) `var_obj`
type of branching object for variable selection

- `vt_obj` use Violation Transfer from Tawarmalani and Sahinidis
- `var_obj` use one object for each variable
- `expr_obj` use one nonlinear expression

`branching_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for braching code in Couenne

`check_lp` (`no`, `yes`) `no`
Check all LPs through an independent call to OsiClpSolverInterface::initialSolve()

`clique_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating clique_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`cont_var_priority` ($1 \leq$ integer) 2000
Priority of continuous variable branching
When branching, this is compared to the priority of integer variables, whose priority is fixed to 1000, and SOS, whose priority is 10. 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.

`convexification_cuts` ($-99 \leq$ integer) 1
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.

`convexification_points` ($0 \leq$ integer) 4
Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point.

`convexification_type` (`current-point-only, uniform-grid, around-current-point`) `current-point-only`
Determines in which point the linear over/under-estimator are generated
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.

- `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

`convexifying_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for convexifying code in Couenne

`cover_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating cover_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`delete_redundant` (`no, yes`) `yes`
Eliminate redundant variables, which appear in the problem as x_k = x_h

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

`disj_active_cols` (`yes, no`) `no`
Only include violated variable bounds in the Cut Generating LP (CGLP).
This reduces the size of the CGLP, but may produce less efficient cuts.

`disj_active_rows` (`yes, no`) `no`
Only include violated linear inequalities in the CGLP.
This reduces the size of the CGLP, but may produce less efficient cuts.

`disj_cumulative` (`yes, no`) `no`
Add previous disjunctive cut to current CGLP.
When generating disjunctive cuts on a set of disjunctions 1, 2, ..., 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.

`disj_depth_level` ($-1 \leq$ integer) 5
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.

`disj_depth_stop` ($-1 \leq$ integer) 20
Depth of the B&B tree where separation of disjunctive cuts is stopped.
A value of -1 means that generation can be done at all nodes

`disj_init_number` ($-1 \leq$ integer) 10
Maximum number of disjunction to consider at each iteration.
-1 means no limit.

`disj_init_perc` ($0 \leq$ real ≤ 1) 0.5
The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions.

`disjcuts_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for disjunctive cuts in Couenne

`display_stats` (`yes, no`) `no`
display statistics at the end of the run

`enable_lp_implied_bounds` (`no, yes`) `no`
Enable OsiSolverInterface::tightenBounds () – warning: it has caused some trouble to Couenne

`estimate_select` (`normal, product`) `normal`
How the min/max estimates of the subproblems' bounds are used in strong branching

`normal` as usual in literature

`product` use their product

`feas_tolerance` (real) 10^{-5}
Tolerance for constraints/auxiliary variables
Default value is zero.

`feasibility_bt` (`no, yes`) `yes`
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.

`flow_covers_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating flow_covers_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`lift_and_project_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating lift_and_project_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`local_optimization_heuristic` (`no, yes`) `yes`
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.

`log_num_abt_per_level` ($-1 \leq$ integer) 2
Specify the frequency (in terms of nodes) for aggressive bound tightening.
If -1, apply at every node (expensive!). If 0, apply at root node only. If k>=0, apply with probability $2^{(k-level)}$, level being the current depth of the B&B tree.

`log_num_local_optimization_per_level` ($-1 \leq$ integer) 2
Specify the logarithm of the number of local optimizations to perform on average for each level of given depth of the tree.
Solve as many nlp's at the nodes for each level of the tree. Nodes are randomly selected. If for a given level there are less nodes than this number nlp are solved for every nodes. For example if parameter is 8, nlp's are solved for all node until level 8, then for half the node at level 9, 1/4 at level 10.... Value -1 specify to perform at all nodes.

`log_num_obbt_per_level` ($-1 \leq$ integer) 1
Specify the frequency (in terms of nodes) for optimality-based bound tightening.
If -1, apply at every node (expensive!). If 0, apply at root node only. If k>=0, apply with probability $2^{(k-level)}$, level being the current depth of the B&B tree.

`lp_solver` (`clp, cplex`) `clp`
Linear Programming solver for the linear relaxation.

`clp` Use the Coin-OR Open Source solver CLP

`cplex` Use the commercial solver Cplex (license is needed)

`minlp_disj_cuts` ($-99 \leq$ integer) 0
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.

`mir_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating mir_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`nlpheur_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for NLP heuristic in Couenne

`opt_window` (real) ∞
Window around known optimum.

`optimality_bt` (`no, yes`) `yes`
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.

`orbital_branching` (`yes, no`) `no`
detect symmetries and apply orbital branching

`probing_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating probing_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`problem_print_level` ($-2 \leq$ integer ≤ 12) 1
Output level for problem manipulation code in Couenne

`pseudocost_mult` (`infeasibility, projectDist, interval_lp, interval_lp_rev, interval_br, interval_br_rev`) `interval_br_rev`
Multipliers of pseudocosts for estimating and update estimation of bound

- `infeasibility` infeasibility returned by object
- `projectDist` distance between current LP point and resulting branches' LP points
- `interval_lp` width of the interval between bound and current lp point
- `interval_lp_rev` similar to interval_lp, reversed
- `interval_br` width of the interval between bound and branching point
- `interval_br_rev` similar to interval_br, reversed

`pseudocost_mult_lp` (`yes, no`) `no`
Use distance between LP points to update multipliers of pseudocosts after simulating branching

`red_cost_branching` (`no, yes`) `no`
Apply Reduced Cost Branching (instead of the Violation Transfer) – MUST have vt_obj enabled

- `no` Use Violation Transfer with $\sum |\pi\_ia\_ij|$
- `yes` Use Reduced cost branching with $|\sum \pi\_ia\_ij|$

`redcost_bt` (`no, yes`) `yes`
Reduced cost bound tightening
This bound reduction technique uses the reduced costs of the LP in order to infer better variable bounds.

`reduce_split_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating reduce_split_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`reformulate_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for reformulating problems in Couenne

`use_quadratic` (`no, yes`) `no`
Use quadratic expressions and related exprQuad class
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 alpha-convexification.

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

`violated_cuts_only` (`no, yes`) `yes`
Yes if only violated convexification cuts should be added

5 Ipopt

IPOPT (**I**nterior **P**oint **Opt**imizer) 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 IPOPT.

IPOPT implements an interior point line search filter method. For more information on the algorithm we refer to [22, 23] and the IPOPT web site https://projects.coin-or.org/Ipopt. Most of the IPOPT documentation in the section was taken from the IPOPT manual [18].

GAMS/IPOPT uses MUMPS 4.9 [2, 3] as linear solver, cf. http://graal.ens-lyon.fr/MUMPS.

5.1 Model requirements

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

5.2 Usage

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

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

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

The linear solver in Ipopt

The performance and robustness of IPOPT on larger models heavily relies on the used solver for sparse symmetric indefinite linear systems. GAMS/IPOPT includes the sparse solver MUMPS 4.9 [2, 3]. The user can provide the routines from the Harwell Subroutine Library (HSL) as shared (or dynamic) libraries to replace MUMPS.

Using Harwell Subroutine Library routines with GAMS/Ipopt. GAMS/IPOPT can use the HSL routines `MA27`, `MA28`, `MA57`, and `MC19` when provided as shared library. By telling IPOPT to use one of these routines (see options `linear_solver`, `linear_system_scaling`, `nlp_scaling_method`, `dependency_detector`), GAMS/IPOPT attempts to load the required routines from the library `libhsl.so` (Unix-Systems), `libhsl.dylib` (MacOS X), or `libhsl.dll` (Windows), respectively. 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/hsllib/myhsllib.so
```

tells IPOPT to use the linear solver MA27 from the HSL library `myhsllib.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.

Specification of Options

IPOPT has many options that can be adjusted for the algorithm (see Section 5.4). 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.

GAMS/IPOPT 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;`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the basic linear algebra routines.

5.3 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 mu.

$\|d\|$ 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 divering; 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 messages 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/Ipopt link or Ipopt (refer to https://projects.coin-or.org/GAMSlinks or https://projects.coin-or.org/Ipopt).

5.4 Detailed Options Description

Barrier Parameter Update

`adaptive_mu_globalization` (`kkt-error, obj-constr-filter, never-monotone-mode`) `obj-constr-filter`
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".)

`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` (`1-norm, 2-norm-squared, max-norm, 2-norm`) `2-norm-squared`
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.

`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` ($0 <$ real < 1) 0.9999
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.

`adaptive_mu_kkterror_red_iters` ($0 \leq$ integer) 4
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.

`adaptive_mu_monotone_init_factor` ($0 <$ real) 0.8
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 and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

`adaptive_mu_restore_previous_iterate` (`no, yes`) `no`
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).

`no` don't restore accepted iterate

`yes` restore accepted iterate

`barrier_tol_factor` ($0 <$ real) 10
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 kappa_epsilon in implementation paper).

`filter_margin_fact` ($0 <$ real < 1) 10^{-5}
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: (new obj) < (filter obj) - filter_margin_fact*(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact*(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

`filter_max_margin` (0 < real) 1
Maximum width of margin in obj-constr-filter adaptive globalization strategy.

`fixed_mu_oracle` (`probing, loqo, quality-function, average_compl`) `average_compl`
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".)

`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` (`no, yes`) `yes`
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.

`no` Take at least one iteration per barrier problem

`yes` Allow fast decrease of mu if barrier test it met

`mu_init` (0 < real) 0.1
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")

`mu_linear_decrease_factor` (0 < real < 1) 0.2
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 mu*"mu_linear_decrease_factor" and mu"$\hat{}$"superlinear_decrease_power". (This is kappa_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

`mu_max` (0 < real) 100000
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".)

`mu_max_fact` (0 < real) 1000
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".)

`mu_min` (0 < real) 10^{-11}
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("tol","compl_inf_tol")/("barrier_tol_factor"+1), which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".)

`mu_oracle` (`probing, loqo, quality-function`) `quality-function`
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").

`probing` Mehrotra's probing heuristic

`loqo` LOQO's centrality rule

`quality-function` minimize a quality function

`mu_strategy` (`monotone, adaptive`) `adaptive`
Update strategy for barrier parameter.
Determines which barrier parameter update strategy is to be used.

`monotone` use the monotone (Fiacco-McCormick) strategy

`adaptive` use the adaptive update strategy

`mu_superlinear_decrease_power` ($1 <$ real < 2) 1.5
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 mu*"mu_linear_decrease_factor" and mu"superlinear_decrease_power". (This is theta_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

`quality_function_balancing_term` (`none, cubic`) `none`
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".)

`none` no balancing term is added

`cubic` Max(0,Max(dual_inf,primal_inf)-compl)$\hat{}$3

`quality_function_centrality` (`none, log, reciprocal, cubed-reciprocal`) `none`
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".)

`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` ($0 \leq$ integer) 8
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".)

`quality_function_norm_type` (`1-norm, 2-norm-squared, max-norm, 2-norm`) `2-norm-squared`
Norm used for components of the quality function.
(Only used if option "mu_oracle" is set to "quality-function".)

`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` ($0 \leq$ real < 1) 0
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".)

`quality_function_section_sigma_tol` ($0 \leq$ real < 1) 0.01
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".)

`sigma_max` ($0 <$ real) 100
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".)

`sigma_min` ($0 \leq$ real) 10^{-6}
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".)

`tau_min` ($0 <$ real < 1) 0.99
Lower bound on fraction-to-the-boundary parameter tau.
(This is tau_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Convergence

`acceptable_compl_inf_tol` ($0 <$ real) 0.01
"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.

`acceptable_constr_viol_tol` ($0 <$ real) 0.01
"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.

`acceptable_dual_inf_tol` ($0 <$ real) 10^{10}
"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.

`acceptable_iter` ($0 \leq$ integer) 15
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.

`acceptable_obj_change_tol` ($0 \leq$ real) 10^{20}
"Acceptance" stopping criterion based on objective function change.
If the relative change of the objective function (scaled by 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.

`acceptable_tol` ($0 <$ real) 10^{-6}
"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.

`compl_inf_tol` ($0 <$ real) 0.0001
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.

`constr_viol_tol` ($0 <$ real) 0.0001
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.

`diverging_iterates_tol` ($0 <$ real) 10^{20}
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.

`dual_inf_tol` (0 < real) 1
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.

`max_cpu_time` (0 < real) 1000
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.

`max_iter` (0 ≤ integer) ∞
Maximum number of iterations.
The algorithm terminates with an error message if the number of iterations exceeded this number.

`mu_target` (0 ≤ real) 0
Desired value of complementarity.
Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of "mu_target" specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

`s_max` (0 < real) 100
Scaling threshold for the NLP error.
(See paragraph after Eqn. (6) in the implementation paper.)

`tol` (0 < real) 10^{-8}
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 "cmpl_inf_tol" are met. (This is epsilon_tol in Eqn. (6) in implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

Hessian Approximation

`hessian_approximation` (`exact, limited-memory`) `exact`
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.

`exact` Use second derivatives provided by the NLP.

`limited-memory` Perform a limited-memory quasi-Newton approximation

`hessian_approximation_space` (`nonlinear-variables, all-variables`) `nonlinear-variables`
Indicates in which subspace the Hessian information is to be approximated.

`nonlinear-variables` only in space of nonlinear variables.

`all-variables` in space of all variables (without slacks)

`limited_memory_aug_solver` (`sherman-morrison, extended`) `sherman-morrison`
Strategy for solving the augmented system for low-rank Hessian.

`sherman-morrison` use Sherman-Morrison formula

`extended` use an extended augmented system

`limited_memory_init_val` (0 < real) 1
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".

`limited_memory_init_val_max` ($0 <$ real) 10^8
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".

`limited_memory_init_val_min` ($0 <$ real) 10^{-8}
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".

`limited_memory_initialization` (`scalar1, scalar2, scalar3, scalar4, constant`) `scalar1`
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.

`scalar1` sigma = $s^T y / s^T s$

`scalar2` sigma = $y^T y / s^T y$

`scalar3` arithmetic average of scalar1 and scalar2

`scalar4` geometric average of scalar1 and scalar2

`constant` sigma = limited_memory_init_val

`limited_memory_max_history` ($0 \leq$ integer) 6
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.

`limited_memory_max_skipping` ($1 \leq$ integer) 2
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.

`limited_memory_update_type` (`bfgs, sr1`) `bfgs`
Quasi-Newton update formula for the limited memory approximation.
Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

`bfgs` BFGS update (with skipping)

`sr1` SR1 (not working well)

Initialization

`bound_frac` ($0 <$ real ≤ 0.5) 0.01
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 kappa_2 in Section 3.6 of implementation paper.)

`bound_mult_init_method` (`constant, mu-based`) `constant`
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.

`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` ($0 <$ real) 1
Initial value for the bound multipliers.
All dual variables corresponding to bound constraints are initialized to this value.

`bound_push` ($0 <$ real) 0.01
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 kappa_1 in Section 3.6 of implementation paper.)

`constr_mult_init_max` ($0 \leq$ real) 1000
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.

`least_square_init_duals` (`no, yes`) `no`
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".

`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` (`no, yes`) `no`
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.

`no` take user-provided point

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

`slack_bound_frac` ($0 <$ real ≤ 0.5) 0.01
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 kappa_2 in Section 3.6 of implementation paper.)

`slack_bound_push` ($0 <$ real) 0.01
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 kappa_1 in Section 3.6 of implementation paper.)

Line Search

`accept_after_max_steps` ($-1 \leq$ integer) -1
Accept a trial point after maximal this number of steps.
Even if it does not satisfy line search conditions.

`accept_every_trial_step` (`no, yes`) `no`
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.

`no` don't arbitrarily accept the full step

`yes` always accept the full step

`alpha_for_y` (`primal, bound-mult, min, max, full, min-dual-infeas, safer-min-dual-infeas, primal-and-full dual-and-full, acceptor`) `primal`
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.

`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

`safer-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 <= alpha_for_y_tol

`dual-and-full` use the dual step size, and full step if delta_x <= alpha_for_y_tol

`acceptor` Call LSAcceptor to get step size for y

`alpha_for_y_tol` ($0 \leq$ real) 10
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.

`alpha_min_frac` ($0 <$ real < 1) 0.05
Safety factor for the minimal step size (before switching to restoration phase).
(This is gamma_alpha in Eqn. (20) in the implementation paper.)

`alpha_red_factor` ($0 <$ real < 1) 0.5
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.

`constraint_violation_norm_type` (`1-norm, 2-norm, max-norm`) `1-norm`
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.

`1-norm` use the 1-norm

`2-norm` use the 2-norm

`max-norm` use the infinity norm

`corrector_compl_avrg_red_fact` ($0 <$ real) 1
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.

`corrector_type` (`none, affine, primal-dual`) `none`
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.

`none` no corrector

`affine` corrector step towards mu=0

`primal-dual` corrector step towards current mu

`delta` ($0 <$ real) 1
Multiplier for constraint violation in the switching rule.
(See Eqn. (19) in the implementation paper.)

`eta_phi` ($0 <$ real < 0.5) 10^{-8}
Relaxation factor in the Armijo condition.
(See Eqn. (20) in the implementation paper)

`filter_reset_trigger` ($1 \leq$ integer) 5
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.

`gamma_phi` ($0 <$ real < 1) 10^{-8}
Relaxation factor in the filter margin for the barrier function.
(See Eqn. (18a) in the implementation paper.)

`gamma_theta` ($0 <$ real < 1) 10^{-5}
Relaxation factor in the filter margin for the constraint violation.

(See Eqn. (18b) in the implementation paper.)

`kappa_sigma` ($0 <$ real) 10^{10}
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.

`kappa_soc` ($0 <$ real) 0.99
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.)

`max_filter_resets` ($0 \leq$ integer) 5
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.

`max_soc` ($0 \leq$ integer) 4
Maximum number of second order correction trial steps at each iteration.
Choosing 0 disables the second order corrections. (This is p$\hat{\text{m}}$ax of Step A-5.9 of Algorithm A in the implementation paper.)

`nu_inc` ($0 <$ real) 0.0001
Increment of the penalty parameter.

`nu_init` ($0 <$ real) 10^{-6}
Initial value of the penalty parameter.

`obj_max_inc` ($1 <$ real) 5
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.

`recalc_y` (`no, yes`) `no`
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.

`no` use the Newton step to update the multipliers

`yes` use least-square multiplier estimates

`recalc_y_feas_tol` ($0 <$ real) 10^{-6}
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.

`rho` ($0 <$ real < 1) 0.1
Value in penalty parameter update formula.

`s_phi` ($1 <$ real) 2.3
Exponent for linear barrier function model in the switching rule.
(See Eqn. (19) in the implementation paper.)

`s_theta` ($1 <$ real) 1.1
Exponent for current constraint violation in the switching rule.
(See Eqn. (19) in the implementation paper.)

`skip_corr_if_neg_curv` (`no, yes`) `yes`
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".

`no` don't skip

`yes` skip

`skip_corr_in_monotone_mode` (`no, yes`) `yes`
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".

`no` don't skip

`yes` skip

`slack_move` ($0 \leq$ real) $1.81899 \cdot 10^{-12}$
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 mach_eps$\hat{3}/4$. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

`theta_max_fact` ($0 <$ real) 10000
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).

`theta_min_fact` ($0 <$ real) 0.0001
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 rules 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).

`tiny_step_tol` ($0 \leq$ real) $2.22045 \cdot 10^{-15}$
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.

`tiny_step_y_tol` ($0 \leq$ real) 0.01
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.

`watchdog_shortened_iter_trigger` ($0 \leq$ integer) 10
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.

`watchdog_trial_iter_max` ($1 \leq$ integer) 3
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.

Linear Solver

`hsl_library` (`string`)
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` (`no, yes`) `yes`
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.

`no` Always scale the linear system.

`yes` Start using linear system scaling if solutions seem not good.

`linear_solver` (`ma27, ma57, mumps`) `ma27`
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 in order to use MA27 or MA57, a library with HSL code need to be provided (see Section 5.2 and the option "hsl_library").

`ma27` use the Harwell routine MA27

`ma57` use the Harwell routine MA57

`mumps` use MUMPS package

`linear_system_scaling` (`none, mc19, slack-based`) `mc19`
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 value is only available if Ipopt has been compiled with MC19.

`none` no scaling will be performed

`mc19` use the Harwell routine MC19

`slack-based` use the slack values

MA27 Linear Solver

`ma27_ignore_singularity` (`no, yes`) `no`
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.

`no` Don't have MA27 solve singular systems

`yes` Have MA27 solve singular systems

`ma27_la_init_factor` ($1 \leq$ real) 5
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. This option is only available if Ipopt has been compiled with MA27.

`ma27_liw_init_factor` ($1 \leq$ real) 5
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. This option is only available if Ipopt has been compiled with MA27.

`ma27_meminc_factor` ($1 \leq$ real) 10
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. This option is only available if Ipopt has been compiled with MA27.

`ma27_pivtol` ($0 <$ real < 1) 10^{-8}
Pivot tolerance for the linear solver MA27.
A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

`ma27_pivtolmax` ($0 <$ real < 1) 0.0001
Maximum pivot tolerance for the linear solver MA27.

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 MA27.

`ma27_skip_inertia_check` (`no, yes`) `no`
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.

`no` check inertia

`yes` skip inertia check

MA28 Linear Solver

`ma28_pivtol` ($0 <$ real ≤ 1) 0.01
Pivot tolerance for linear solver MA28.
This is used when MA28 tries to find the dependent constraints.

MA57 Linear Solver

`ma57_automatic_scaling` (`no, yes`) `yes`
Controls MA57 automatic scaling
This option controls the internal scaling option of MA57.This is ICNTL(15) in MA57.

`no` Do not scale the linear system matrix

`yes` Scale the linear system matrix

`ma57_block_size` ($1 \leq$ integer) 16
Controls block size used by Level 3 BLAS in MA57BD
This is ICNTL(11) in MA57.

`ma57_node_amalgamation` ($1 \leq$ integer) 16
Node amalgamation parameter
This is ICNTL(12) in MA57.

`ma57_pivot_order` ($0 \leq$ integer ≤ 5) 5
Controls pivot order in MA57
This is ICNTL(6) in MA57.

`ma57_pivtol` ($0 <$ real < 1) 10^{-8}
Pivot tolerance for the linear solver MA57.
A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.

`ma57_pivtolmax` ($0 <$ real < 1) 0.0001
Maximum pivot tolerance for the linear solver MA57.
Ipopt may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

`ma57_pre_alloc` ($1 \leq$ real) 1.05
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. This option is only available if Ipopt has been compiled with MA57.

`ma57_small_pivot_flag` ($0 \leq$ integer ≤ 1) 0
If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.
This is ICNTL(16) in MA57.

Mumps Linear Solver

`mumps_dep_tol` (real) -1
Pivot threshold for detection of linearly dependent constraints in MUMPS.
When MUMPS is used to determine linearly dependent constraints, this is determines the threshold for a pivot

to be considered zero. This is CNTL(3) in MUMPS.

`mumps_mem_percent` ($0 \leq$ integer) 1000
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. On the other hand, if memory requirement are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

`mumps_permuting_scaling` ($0 \leq$ integer ≤ 7) 7
Controls permuting and scaling in MUMPS
This is ICNTL(6) in MUMPS.

`mumps_pivot_order` ($0 \leq$ integer ≤ 7) 7
Controls pivot order in MUMPS
This is ICNTL(7) in MUMPS.

`mumps_pivtol` ($0 \leq$ real ≤ 1) 10^{-6}
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.

`mumps_pivtolmax` ($0 \leq$ real ≤ 1) 0.1
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.

`mumps_scaling` ($-2 \leq$ integer ≤ 77) 77
Controls scaling in MUMPS
This is ICNTL(8) in MUMPS.

NLP

`bound_relax_factor` ($0 \leq$ real) 10^{-10}
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 implementation paper.)

`check_derivatives_for_naninf` (`no, yes`) `no`
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. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MORE_DETAILED; so beware of large output!

`no` Don't check (faster).

`yes` Check Jacobians and Hessian for Nan and Inf.

`dependency_detection_with_rhs` (`no, yes`) `no`
Indicates if the right hand sides of the constraints should be considered during dependency detection

`no` only look at gradients

`yes` also consider right hand side

`dependency_detector` (`none, mumps, wsmp, ma28`) `none`
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.

`none` don't check; no extra work at beginning

`mumps` use MUMPS

`wsmp` use WSMP

`ma28` use MA28

`fixed_variable_treatment` (`make_parameter, make_constraint, relax_bounds`) `make_parameter`
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.

`make_parameter` Remove fixed variable from optimization variables

`make_constraint` Add equality constraints fixing variables

`relax_bounds` Relax fixing bound constraints

`honor_original_bounds` (`no, yes`) `yes`
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.

`no` Leave final point unchanged

`yes` Project final point back into original bounds

`jac_c_constant` (`no, yes`) `no`
Indicates whether all equality constraints are linear
Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

`no` Don't assume that all equality constraints are linear

`yes` Assume that equality constraints Jacobian are constant

`jac_d_constant` (`no, yes`) `no`
Indicates whether all inequality constraints are linear
Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

`no` Don't assume that all inequality constraints are linear

`yes` Assume that equality constraints Jacobian are constant

`kappa_d` ($0 \leq$ real) 10^{-5}
Weight for linear damping term (to handle one-sided bounds).
(see Section 3.7 in implementation paper.)

`num_linear_variables` ($0 \leq$ integer) 0
Number of linear variables
When the Hessian is approximated, it is assumed that the first num_linear_variables variables are linear. The Hessian is then not approximated in this space. If the get_number_of_nonlinear_variables method in the TNLP is implemented, this option is ignored.

NLP Scaling

`nlp_scaling_constr_target_gradient` ($0 \leq$ real) 0
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.

`nlp_scaling_max_gradient` ($0 <$ real) 100
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".

`nlp_scaling_method` (`none, user-scaling, gradient-based, equilibration-based`) `gradient-based`
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")

`none` no problem scaling will be performed

`user-scaling` scaling parameters will come from the user

`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_min_value` ($0 \leq$ real) 10^{-8}
Minimum value of gradient-based scaling values.
This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

`nlp_scaling_obj_target_gradient` ($0 \leq$ real) 0
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.

Output

`print_eval_error` (`no, yes`) `no`
whether to print information about function evaluation errors into the listing file

`print_info_string` (`no, yes`) `no`
Enables printing of additional info string at end of iteration output.
This string contains some insider information about the current iteration.

`no` don't print string

`yes` print string at end of each iteration output

`print_level` ($0 \leq$ integer ≤ 12) 5
Output verbosity level.
Sets the default verbosity level for console output. The larger this value the more detailed is the output.

`print_timing_statistics` (`no, yes`) `no`
Switch to print timing statistics.
If selected, the program will print the CPU usage (user time) for selected tasks.

`no` don't print statistics

`yes` print all timing statistics

`replace_bounds` (`no, yes`) `no`
Indicates if all variable bounds should be replaced by inequality constraints
This option must be set for the inexact algorithm

`no` leave bounds on variables

`yes` replace variable bounds by inequality constraints

Restoration Phase

`bound_mult_reset_threshold` ($0 \leq$ real) 1000
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.

`constr_mult_reset_threshold` ($0 \leq$ real) 0
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.

`evaluate_orig_obj_at_resto_trial` (`no, yes`) `yes`
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.

`no` skip evaluation

`yes` evaluate at every trial point

`expect_infeasible_problem` (`no, yes`) `no`
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.

`no` the problem probably be feasible

`yes` the problem has a good chance to be infeasible

`expect_infeasible_problem_ctol` ($0 \leq$ real) 0.001
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.

`expect_infeasible_problem_ytol` ($0 <$ real) 10^8
Multiplier threshold for activating "expect_infeasible_problem" option.
If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

`max_resto_iter` ($0 \leq$ integer) 3000000
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.

`max_soft_resto_iters` ($0 \leq$ integer) 10
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.

`required_infeasibility_reduction` ($0 \leq$ real < 1) 0.9
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.

`resto_penalty_parameter` ($0 <$ real) 1000
Penalty parameter in the restoration phase objective function.
This is the parameter rho in equation (31a) in the Ipopt implementation paper.

`soft_resto_pderror_reduction_factor` ($0 \leq$ real) 0.9999
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.

`start_with_resto` (`no, yes`) `no`
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.

`no` don't force start in restoration phase

`yes` force start in restoration phase

Step Calculation

`fast_step_computation` (`no, yes`) `no`
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.

`no` Verify solution of linear system by computing residuals.

`yes` Trust that linear systems are solved well.

`first_hessian_perturbation` ($0 <$ real) 0.0001
Size of first x-s perturbation tried.
The first value tried for the x-s perturbation in the inertia correction scheme.(This is delta_0 in the implementation paper.)

`jacobian_regularization_exponent` ($0 \leq$ real) 0.25
Exponent for mu in the regularization for rank-deficient constraint Jacobians.
(This is kappa_c in the implementation paper.)

`jacobian_regularization_value` ($0 \leq$ real) 10^{-8}
Size of the regularization for rank-deficient constraint Jacobians.
(This is bar delta_c in the implementation paper.)

`max_hessian_perturbation` ($0 <$ real) 10^{20}
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 delta_wmax in the implementation paper.)

`max_refinement_steps` ($0 \leq$ integer) 10
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.

`mehrotra_algorithm` (`no, yes`) `no`
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".

`no` Do the usual Ipopt algorithm.

`yes` Do Mehrotra's predictor-corrector algorithm.

`min_hessian_perturbation` ($0 \leq$ real) 10^{-20}
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 delta_wmin in implementation paper.)

`min_refinement_steps` ($0 \leq$ integer) 1
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.)

`neg_curv_test_tol` ($0 <$ real) 0
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.

`perturb_always_cd` (`no, yes`) `no`
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.

`no` perturbation only used when required

`yes` always use perturbation

`perturb_dec_fact` ($0 <$ real < 1) 0.333333
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 kappa_w$\hat{-}$ in the implementation paper.)

`perturb_inc_fact` ($1 <$ real) 8
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 kappa_w$\hat{+}$ in the implementation paper.)

`perturb_inc_fact_first` ($1 <$ real) 100
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$\hat{+}$ in the implementation paper.)

`residual_improvement_factor` ($0 <$ real) 1
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.

`residual_ratio_max` ($0 <$ real) 10^{-10}
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).

`residual_ratio_singular` ($0 <$ real) 10^{-5}
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.

Warm Start

`warm_start_bound_frac` ($0 <$ real ≤ 0.5) 0.001
same as bound_frac for the regular initializer.

`warm_start_bound_push` ($0 <$ real) 0.001
same as bound_push for the regular initializer.

`warm_start_entire_iterate` (`no, yes`) `no`
Tells algorithm whether to use the GetWarmStartIterate method in the NLP.

`no` call GetStartingPoint in the NLP

`yes` call GetWarmStartIterate in the NLP

`warm_start_init_point` (`no`, `yes`) `no`
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.)

`no` do not use the warm start initialization

`yes` use the warm start initialization

`warm_start_mult_bound_push` ($0 <$ real) 0.001
same as mult_bound_push for the regular initializer.

`warm_start_mult_init_max` (real) 10^6
Maximum initial value for the equality multipliers.

`warm_start_same_structure` (`no`, `yes`) `no`
Indicates whether a problem with a structure identical to the previous one is to be solved.
If "yes" is chosen, then the algorithm assumes that an NLP is now to be solved, whose structure is identical to one that already was considered (with the same NLP object).

`no` Assume this is a new problem.

`yes` Assume this is problem has known structure

`warm_start_slack_bound_frac` ($0 <$ real ≤ 0.5) 0.001
same as slack_bound_frac for the regular initializer.

`warm_start_slack_bound_push` ($0 <$ real) 0.001
same as slack_bound_push for the regular initializer.

6 Optimization Services

OS (**O**ptimization **S**ervices) 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 OS manual [20], the papers [14, 15, 16], and the web sites `http://www.optimizationservices.org` and `https://projects.coin-or.org/OS`.

The OS link in GAMS allows you to convert instances of GAMS models into the OS instance language (OSiL) format and let an Optimization Services Server solve your instances remotely.

6.1 Model requirements

OS supports continuous, binary, and integer variables, linear and nonlinear equations. Special ordered sets, semicontinuous or semiinteger variables, and indicator constraints are currently not supported. Initial values are currently not supported by the GAMS/OS link.

6.2 Usage

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

```
Option MINLP = OS;      { 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, nothing happens. To solve an instance remotely, you have to specify the URL of an Optimization Services Server via the option `service`. Usually, the server chooses an appropriate solver for your instance, depending on their availability on the server. A fully equipped server

chooses CLP for continuous linear models (LP and RMIP), IPOPT for continuous nonlinear models (NLP, DNLP, RMINLP, QCP, RMIQCP), CBC for mixed-integer linear models (MIP), and BONMIN for mixed-integer nonlinear models (MIQCP, MINLP). An easy way to influence the choice of the solver on the server is the `solver` option.

Further options can be provided in an OSoL (Optimization Services Options Language) file, which is specified via the the `readosol` option. An example OSoL file looks like

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
                          http://www.optimizationservices.org/schemas/2.0/OSoL.xsd">
<optimization>
  <solverOptions numberOfSolverOptions="3">
    <solverOption name="cuts" solver="cbc" value="off" />
    <solverOption name="max_active_nodes" solver="symphony"  value="2" />
    <solverOption name="max_iter" solver="ipopt" type="integer" value="2000"/>
  </solverOptions>
</optimization>
</osol>
```

It specifies that if CBC is used, then cutting planes are disabled, if SYMPHONY is used, then at most 2 nodes should be active, and if IPOPT is used, then a limit of 2000 iterations is imposed.

By default, the call to the server is a *synchronous* call. The GAMS process will wait for the result and then display the result. This may not be desirable when solving large optimization models. In order to use the remote solver service in an *asynchronous* fashion, one can make use of the GAMS Grid Computing Facility, see Appendix I in the GAMS manual.

6.3 Detailed Options Descriptions

readosol (*string*)

Specifies the name of an option file in OSoL format that is given to the OS server. This way it is possible to pass options directly to the solvers interfaced by OS.

writeosil (*string*)

Specifies the name of a file in which the GAMS model instance should be writting in OSiL format.

writeosrl (*string*)

Specifies the name of a file in which the result of a solve process (solution, status, ...) should be writting in OSrL format.

service (*string*)

Specifies the URL of an Optimization Services Server. The GAMS model is converted into OSiL format, send to the server, and the result translated back into GAMS format. Note that by default the server chooses a solver that is appropriate to the model type. You can change the solver with the solver option.

solver (*string*)

Specifies the solver that is used to solve an instance on the OS server.

7 OsiCplex, OsiGlpk, OsiGurobi, OsiMosek, OsiSoplex, OsiXpress

The "bare bone" solver links GAMS/OSICPLEX, GAMS/OSIGLPK, GAMS/OSIGUROBI, GAMS/OSIMOSEK, GAMS/OSISOPLEX, and GAMS/OSIXPRESS allow users to solve their GAMS models with GLPK, SOPLEX, or 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 OsiCplex link has been written primarily by Tobias Achterberg, the OsiGlpk link has been written by Vivian De Smedt, Braden Hunsaker, and Lou Hafer, the OsiGurobi link has been written primarily by Stefan Vigerske, the OsiMosek link has been written primarily by Bo Jensen, and the OsiSoplex link has been written primarily by Tobias Achterberg, Ambros M. Gleixner, and Wei Huang, and the OsiXpress 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`.

7.1 Model requirements

The OSI links support linear equations and continuous, binary, and integer variables. Semicontinuous and Semiinteger variables, special ordered sets, branching priorities, and indicator constraints are not supported by OSI. OsiSoplex solves only LPs, no MIPs.

7.2 Usage

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

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

The above statement should appear before the Solve statement.

The links support the general GAMS options `reslim`, `optca` (except for OsiGlpk), `optcr`, `nodlim`, `iterlim`, and `threads` (except for OsiGlpk and OsiSoplex). For OsiCplex, OsiGurobi, OsiMosek, and OsiXpress an option file in the format required by the solver can be provided via the GAMS `optfile` option. For OsiGlpk, an GAMS option file can be provided. See Section 7.3 for details.

If a MIP is solved via one of the OSI links, only primal solution values are reported by default. To receive also the dual values for the LP that is obtained from the MIP by fixing all discrete variables, the GAMS option `integer1` can be set to a nonzero value. Note that this may lead to solving another LP after the MIP solve has finished.

Setting the GAMS option `integer2` to a nonzero value makes variable and equation names available to the solver. This option may be useful for debugging purposes.

Setting the GAMS option `integer3` to a nonzero value leads to writing the model instance to a file in MPS format before starting the solution process. The name of the MPS file is chosen to be the name of the GAMS model file with the extension `.gms` replaced by `.mps`. This option may be useful for debugging purposes.

For OsiCplex, OsiGurobi, and OsiXpress, setting the GAMS option `integer4` to a nonzero value leads to passing the variable level values (`.l` suffix) to the MIP solver as initial solution. This is analog to the `mipstart` option of the full CPLEX and GUROBI links and the `loadmipsol` option of the full XPRESS link.

7.3 Option files

OsiCplex Options

In an OsiCplex option file, each line lists one option setting, where the option name and value are separated by space.
Example:

```
CPX_PARAM_MIPEMPHASIS           2
CPX_PARAM_HEURFREQ              42
CPX_PARAM_MIPDISPLAY            4
```

OsiGlpk Options

The OsiGlpk option 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 osiglpk.opt.

```
factorization givens
```

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

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 poor performce, then you may try setting the factorization method to forresttomlin.

(default = givens)

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.

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 Gomorys 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 (*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 stoped when the relative gap between the incumbent and the bound given by the LP relaxation is smaller than this value.

(default = GAMS optcr)

OsiGurobi Options

In an OsiCplex option file, each line lists one option setting, where the option name and value are separated by space.

Example:

```
Cuts 2
Heuristics 0.1
```

OsiMosek Options

An OsiMosek option file begins with the line `BEGIN MOSEK` and terminates with `END MOSEK`. Comments are introduced with an '%', empty lines are ignored. Each other line starts with a MOSEK parameter value, followed by space, and a value for that parameter.

Example:

```
BEGIN MOSEK
% disable probing and solve the root node by the interior point solver
MSK_IPAR_MIO_PRESOLVE_PROBING MSK_OFF
MSK_IPAR_MIO_ROOT_OPTIMIZER   MSK_OPTIMIZER_INTPNT
END MOSEK
```

OsiXpress Options

In an OsiXpress option file, each line lists one option setting, where the option name and value are separated by an equal sign.

Example:

```
MIPLOG = 3
HEURFREQ = 2
```

COIN-OR References

[1] K. Abhishek, S. Leyffer, and J.T. Linderoth. FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs. *INFORMS Journal On Computing*, 22(4):555–567, 2010.

[2] Patrick R. Amestoy, Iain S. Duff, Jacko Koster, and Jean-Yves L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–24, 2001.

[3] Patrick R. Amestoy, Abdou Guermouche, Jean-Yves L'Excellent, and Stephane Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[4] P. Belotti. Disjunctive cuts for non-convex MINLP. In Jon Lee and Sven Leyffer, editors, *Mixed-integer nonlinear optimization: Algorithmic advances and applications*, IMA volumes in Mathematics and its Applications. Springer, 2011. to appear.

[5] Pietro Belotti. *COUENNE: a user's manual*. https://projects.coin-or.org/Couenne.

[6] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009.

[7] Pierre Bonami, Gérard Cornuéjols, Andrea Lodi, and François Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009.

[8] Pierre Bonami and João P. M. Gonçalves. Primal heuristics for mixed integer nonlinear programs. Technical Report RC24639, IBM Research, 2008.

[9] Pierre Bonami, Mustafa Kilinç, and Jeffrey Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In Jon Lee and Sven Leyffer, editors, *Mixed-integer nonlinear optimization: Algorithmic advances and applications*, IMA volumes in Mathematics and its Applications. Springer, 2011. to appear.

[10] Pierre Bonami and Jon Lee. *BONMIN Users' Manual*, 1.3 edition, November 2009. https://projects.coin-or.org/Bonmin.

[11] Pierre Bonami, Andreas Wächter, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, and Nicolas W. Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[12] Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[13] Roger Fletcher and Sven Leyffer. Solving Mixed Integer Nonlinear Programs by Outer Approximation. *Mathematical Programming*, 66(3(A)):327–349, 1994.

[14] Robert Fourer, Jun Ma, and Kipp Martin. Optimization services: A framework for distributed optimization. *Operations Research*, accepted, 2009. http://www.optimizationservices.org/.

[15] Robert Fourer, Jun Ma, and Kipp Martin. OSiL: An instance language for optimization. *Computational Optimization and Applications*, 45(1):181–203, 2010.

[16] Horand Gassmann, Jun Ma, Kipp Martin, and Wayne Sheng. *Optimization Services 2.1 User's Manual*, March 2010. https://projects.coin-or.org/OS.

[17] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[18] Yoshiaki Kawajir, Carl Laird, and Andreas Wächter. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*, 1597 edition, November 2009. https://projects.coin-or.org/Ipopt.

[19] Robin Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, 2003. http://www.coin-or.org.

[20] Jun Ma. *Optimization Services (OS)*. PhD thesis, Industrial Engineering and Management Sciences, Northwestern University, 2005.

[21] Ignacio Quesada and Ignacio E. Grossmann. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & Chemical Engineering*, 16:937–947, 1992.

[22] Andreas Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, January 2002.

[23] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. http://projects.coin-or.org/Ipopt.