Mathematical Programming manuscript No.
(will be inserted by the editor)

Arnold Neumaier · Oleg Shcherbina · Waltraud Huyer · Tamás Vinkó

A comparison of complete global optimization solvers

the date of receipt and acceptance should be inserted later

Abstract. Results are reported of testing a number of existing state of the art solvers for global constrained optimization and constraint satisfaction on a set of over 1000 test problems in up to 1000 variables, collected from the literature.

The test problems are available online in AMPL and were translated into the input formats of the various solvers using routines from the COCONUT environment. These translators are available online, too.

1. Overview

This paper presents test results for software performing a complete search to solve global optimization or constraint satisfaction problems.

In contrast to local or heuristic searches, a **complete search** checks *all* points in the search region for feasibility, and all feasible points for global optimality. A solver that performs a complete search – apart from rounding error issues – is called a **complete solver**. Since the search region for continuous global optimization problems contains an infinite number of points, analytic techniques are needed to make decisions about infinitely many points simultaneously. This is usually (but not always) done in a branch-and-bound framework.

As the recent survey NEUMAIER [25] of complete solution techniques in global optimization documents, there are now about a dozen solvers for constrained global optimization that claim to solve global optimization and/or constraint satisfaction problems to global optimality by performing a complete search.

Within the COCONUT project [30,31], we evaluated many of the existing software packages for global optimization and constraint satisfaction problems. This is the first time that different constrained global optimization and constraint satisfaction algorithms are compared on a systematic basis and with a test set that allows to derive statistically significant conclusions. We tested the global solvers BARON, COCOS, GlobSol, ICOS, LGO, LINGO, OQNLP, Premium Solver, and the local solver MINOS.

The testing process turned out to be extremely time-consuming, due to various reasons not initially anticipated. A lot of effort went into creating appropriate

Arnold Neumaier, Oleg Shcherbina, Waltraud Huyer: Fakultät für Mathematik, Universität Wien, Nordbergstr. 15, A-1090 Wien, Austria

Tamás Vinkó: Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged, H-6720 Szeged, Aradi vértanúk tere 1., Hungary

interfaces, making the comparison fair and reliable, and making it possible to process a large number of test examples in a semiautomatic fashion.

In a recent paper about testing local optimization software, Dolan & Moré [7,8] write: *We realize that testing optimization software is a notoriously difficult problem and that there may be objections to the testing presented in this report. For example, performance of a particular solver may improve significantly if non-default options are given. Another objection is that we only use one starting point per problem and that the performance of a solver may be sensitive to the choice of starting point. We also have used the default stopping criteria of the solvers. This choice may bias results but should not affect comparisons that rely on large time differences. In spite of these objections, we feel that it is essential that we provide some indication of the performance of optimization solvers on interesting problems.*

These difficulties are also present with our benchmarking studies. Section 2 describes our testing methodology. We use a large test set of over 1000 problems from various collections. Our main performance criterium is currently how often the attainment of the global optimum, or the infeasibility of a problem, is correctly or incorrectly claimed (within some time limit). All solvers are tested with the default options suggested by the providers of the codes, with the request to stop at a time limit or after the solver believed that first global solution was obtained.

These are very high standards, much more demanding than what had been done by anyone before. Thorough comparisons are indeed very rare, due to the difficulty of performing extensive and meaningful testing. Indeed, we know of only two comparative studies [18,23] in global optimization ranging over more than perhaps a dozen examples, and both are limited to bound constrained black box optimization. (See also Huyer [15] for some further tests.)

Only recently rudimentary beginnings were made elsewhere in testing constrained global optimization [12]. On the other hand, there are a number of reports about comparing codes in local optimization [1,4,7,8,14,17,28], and there is an extensive web site [22] with wide-ranging comparative results on local constrained optimization codes.

Section 3 describes the tests done on the most important state of the art global solvers. Shortly expressed, the result is the following:

Among the currently available global solvers, BARON is the fastest and most robust one, with OQNLP being close. None of the current global solvers is fully reliable, with one exception: For pure constraint satisfaction problems, ICOS, while slower than BARON, has excellent reliability properties when it is able to finish the complete search. Models in dimension < 100 are solved with a success rate (global optimum found) of over 90% by BARON while (within half an hour of CPU time) less than two thirds of the larger models are solved.

OQNLP, the best of the stochastic solvers, had solved the maximal number of problems a year ago, but is now in most respects second to the newest version of BARON; moreover, it is much slower and cannot

offer information about when the search is completed. However, on the models with > 100 variables, OQNLP still solves (within the imposed time limit) the highest percentage (72%) of problems.

The best solver, BARON, was able to complete the search in over two third of the models with less than 100 variables (for larger problems only about one third, within the time limit of 30 minutes), but it lost the global minimum in about 4 percent of the cases.

The final Section 4 concludes with various remarks, including guidelines for developers of global optimization codes derived from our experience with the initial versions of the packages tested.

Much more detailed results than can be given here are available online at [3].

Acknowledgments. We are happy to acknowledge cooperation with various people. Dan Fylstra (Frontline Solver), Tony Gau (LINGO), Baker Kearfott (GlobSol), Yahia Lebbah (ICOS), Alex Meeraus (GAMS), János Pintér (LGO), and Nick Sahinidis (BARON) provided valuable support in obtaining and running their software. The first round of testing GlobSol was done by Boglárka Tóth (Szeged, Hungary). Mihály Markót tested and debugged preliminary versions of the COCOS strategy.

The results presented are part of work done in the context of the COCONUT project [30,31] sponsored by the European Union, with the goal of integrating various existing complete approaches to global optimization into a uniform whole. Funding by the European Union under the IST Project Reference Number IST-2000-26063 within the FET Open Scheme is gratefully acknowledged.

2. Testing

Introduction. We present test results for the global optimization systems BARON, COCOS, GlobSol, ICOS, LGO/GAMS, LINGO, OQNLP Premium Solver, and for comparison the local solver MINOS. All tests were made on the COCONUT benchmarking suite described in Shcherbina et al. [33].

Our experience with the solvers tested and preliminary test results were communicated to the developers of the solvers and lead to significant improvements in the robustness and user-friendliness of several solvers – the present results are based on the last available version of each solver.

For generalities on benchmarking and the associated difficulties, in particular for global optimization, see Shcherbina et al. [33]. Here we concentrate on the documentation of the testing conditions used and on the interpretation of the results obtained. For the interpretation of the main results, see the overview in Section 1.

The test set. A good benchmark must be one that can be interfaced with all existing solvers, in a way that a sufficient number of comparative results can be obtained. There are various smaller-scale benchmark projects for partial

domains, in particular the benchmarks for local optimization by MITTELMANN [22]. A very recent web site, the GAMS Global Library [13] started collecting real life global optimization problems with industrial relevance, but currently most problems on this site are without computational results. Our benchmark (described in more detail in [33]) includes most of the problems from these collections.

The test set consists of 1322 models varying in dimension (number of variables) between 1 and over 1000, coded in the modeling language AMPL [9]. They are sorted by size and source (library). Size k denotes models whose number of variables (after creating the corresponding DAG and simplifying it) has k decimal digits. Library 1 (from Global Library [13]) and Library 2 (from CUTE [5], in the version of VANDERBEI [36]) consist of global (and some local) optimization problems with nonempty feasible domain, while Library 3 (from EPFL [33]) consists of pure constraint satisfaction problems (constant objective function), almost all being feasible. The resulting 12 model classes are labeled as `lib2s1` (= size 1 models from Library 2), etc..

	Number of test models				
Number of variables	$1-9$	$10-99$	$100-999$	≥ 1000	any
	size 1	size 2	size 3	size 4	total
Library 1	84	90	44	48	266
Library 2	347	100	93	187	727
Library 3	225	76	22	6	329
total	656	266	159	241	1322

We restricted testing to models with less than 1000 variables since the models of size 3 already pose so many difficulties that working on the (much more CPU time consuming) larger models is likely to give no additional insight for the current generation of solvers.

We also excluded a small number of models from these test sets because of difficulties unrelated to the solvers. In particular, the functions `if`, `log10`, `tan`, `atan`, `asin`, `acos` and `acosh` are currently not supported by the `ampl2dag` converter underlying all our translators into the various solver input formats. Since they are used in the models `ColumnDesign-original`, `FatigueDesign-original`, `djtl`, `hubfit` (`if`), `bearing` (`log10`), `yfit`, `yfitu` (`tan`), `artif`, `helix`, `s332`, `TrussDesign-full`, `TrussDesign01` (`atan`), `dallasl`, `dallasm`, `dallass` (`asin`), `chebyqad`, `cresc4`, `cresc50`, `cresc100`, `cresc132` (`acos`), `coshfun` (`acosh`), these models were excluded. A few of the problems in Library 3 (pure constraint satisfaction problems) in fact contained objective functions, and hence were excluded, too. This was the case for the models `h78`, `h80`, `h81`, `logcheb`, `median_exp`, `median_nonconvex`, `robotarm`, `steenbre`. A few other models, namely `ex8_3_12`, `ex8_3_14`, `concon`, `mconcon`, `osborneb`, showed strange behavior, making us suspect that it is due to unspotted bugs in our converters.

The models where none of the solvers found a feasible point and some other attempts to get one failed, are regarded in the following as being infeasible (though some of these might possess undiscovered feasible points).

The computers. Because of the large number of models to be solved, we performed our tests on a number of different computers called Lisa, Hektor, Zenon, Theseus and Bagend. Their brand and their general performance characteristics are displayed below. The standard unit time STU, redefined in SHCHERBINA et al. [33], is essentially equivalent to 10^8 standard unit times according to DIXON & SZEGÖ [6]. For Bogomips and Linpack, see

`http://www.tldp.org/HOWTO/mini/BogoMips-2.html`
`http://www.netlib.org/benchmark/linpackjava`

(The high Linpack entry for Zenon is apparently caused by an inefficient Windows environment.)

Computer	CPU type	OS	CPU/MHz	Bogomips	STU/sec	Linpack
Lisa	AMD Athlon XP2000+	Linux	1678.86	3348.88	50	7.42
Hektor	AMD Athlon XP1800+	Linux	1544.51	3080.19	53	6.66
Zenon	AMD Family 6 Model 4	Windows NT 4.0	1001	—	74	46.78
Theseus	Pentium III	Linux	1000.07	1992.29	130	4.12
Bagend	AMD Athlon MP2000+	Linux	1666.72	3329.22	36	5.68

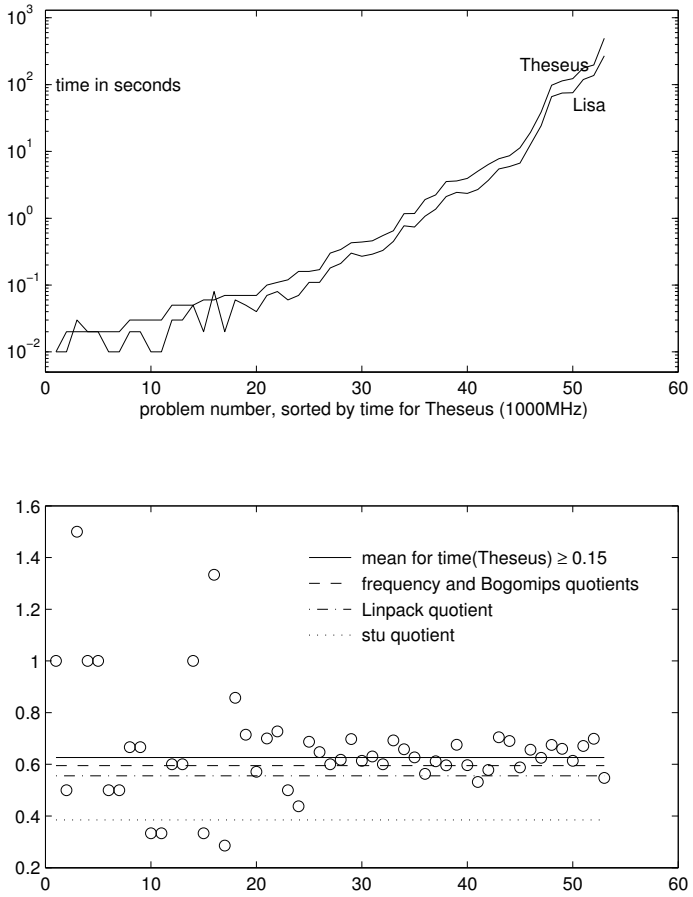
To decide on the best way to compare across computers, we ran the models from `lib1s1` with BARON on both Lisa and Theseus, and compared the resulting ratios of CPU times with the ratios of performance indices, given by the following table.

	Lisa	Theseus	Ratios	Inverse ratios
Frequency	1678.86	1000.07	1.68	0.60
Bogomips	3348.88	1992.29	1.68	0.59
STU	50.00	130.00	0.38	2.60
Linpack	7.42	4.12	1.80	0.56

As Figure 1 with the results shows, the appropriate index to use is the frequency. We therefore measure times in multiples of 1000 Mcycles, obtained by multiplying the CPU time by the nominal frequency of the CPU in MHz, and dividing the result by 1000. Figure 1 also shows that small times are not well comparable; we therefore decided to round the resulting numbers t to 1 digit after the decimal point if $t < 10$, and to the nearest integer if $t \geq 10$. For tiny times where this would result in a zero time, we use instead $t = 0.05$.

The solvers. The following tables summarize some of the main properties of these solvers, as far as known to us. Missing information is indicated by a question mark, and partial applicability by a + or − in parentheses; the dominant technique (if any) exploited by the solver is denoted by ++.

Fig. 1. Times and timing ratios for `lib1s1` with BARON



The first two rows give the name of the solvers and the access language used to pass the problem description. The next two rows indicate whether it is possible to specify integer constraints (although we don't test this feature), and whether it is necessary to specify a finite search box within which all functions can be evaluated without floating point exceptions.

The next three rows indicate whether black box function evaluation is supported, whether the search is complete (i.e., is claimed to cover the whole search region if the arithmetic is exact and sufficiently fast) or even rigorous (i.e., the results are claimed to be valid with mathematical certainty even in the presence of rounding errors).

Solver	Minos	LGO	BARON	ICOS	GlobSol
access language	GAMS	GAMS	GAMS	AMPL	Fortran90
optimization?	+	+	+	−	+
integer constraints	−	+	+	−	−
search bounds	−	required	recommended	−	required
black box eval.	+	+	−	−	−
complete	−	(−)	+	+	+
rigorous	−	−	−	+	+
local	++	+	+	+	(+)
CP	−	−	+	++	+
other interval	−	−	−	+	++
convex/LP	−	−	++	+	−
dual	+	−	+	−	−
available	+	+	+	+	+
free	−	−	−	(+)	+

Solver	Premium Solver	LINGO Global	αBB	GloptiPoly	OQNLP
access language	Visual Basic	LINGO	MINOPT	Matlab	GAMS
optimization?	+	+	+	(+)	+
integer constraints	+	+	+	−	+
search bounds	+	−	?	−	+
black box eval.	−	−	−	−	+
complete	+	+	+	+	−
rigorous	(+)	−	−	−	−
local	+	+	+	−	+
CP	+	+	−	−	−
other interval	++	+	+	−	−
convex/LP	+	++	++	+	−
dual	−	+	−	++	−
available	+	+	−	+	+
free	−	−	−	+	−

Note that general theorems forbid a complete finite search if black box functions are part of the problem formulation, and that a rigorous search is necessarily complete. In view of the goals of the COCONUT project we were mainly interested in complete solvers. However, we were curious how (some) incomplete solvers perform. Five further rows indicate the mathematical techniques used to do the global search. We report whether local optimization techniques, constraint propagation, other interval techniques, convex analysis and linear programming (LP), or dual (multiplier) techniques are part of the toolkit of the solver.

The final two rows indicate whether the code is available (we include in this list of properties the solver αBB because of its good reported properties, although we failed to obtain a copy of the code), and whether it is free (in the public domain).

In this paper, we study the behavior of the solvers BARON/GAMS (version 7.2, released July 7, 2004) [29,34,35], COCOS (beta test version 1.0, released September 20, 2004), GlobSol (version released 11 September 2003) [19], ICOS (beta-test version, released March 29, 2004) [20], LGO/GAMS [27], LINGO (version 9.0, released October 12, 2004) [21], OQNLP/GAMS [11], Premium Solver (Interval Global Solver from the Premium Solver Platform of Frontline Systems, Version 5) [10]. (GloptiPoly is limited to polynomial systems of dimension < 20, and was not tested.) Note that our tests apply to the combination of solver plus interface. For LINGO we used the converter from GAMS. In a few cases, the failures reported are due to problems in the GAMS interface rather than the solver.

To enable us to assess how difficult it is (i) to find a global minimum, and (ii) to verify it as global – in many instances, part (ii) is significantly harder than part (i) –, results (without timings) from the local solver MINOS [24] are also included in our comparison.

ICOS only handles pure constraint satisfaction problems, and hence was tested only on Library 3. Two of the solvers (BARON and ICOS) also allow the generation of multiple solutions, but due to the lack of a reliable basis for comparison, this feature has not been tested. Two of the solvers (BARON and LINGO) allow one to pose integer constraints, and two (LINGO and Premium Solver) allows nonsmooth expressions. Neither of these features has been tested in this study.

Passing the models. The access to all test models is through an AMPL interface, which translates the AMPL model definition into the internal form of a directed acyclic graph (DAG) which is labelled in such a way as to provide a unique description of the model to be solved. This internal description could be simplified by a program `dag_simplify` which performs simple presolve and DAG simplification tasks. Moreover, all maximization problems are converted to minimization problems, with objective multiplied by -1. This preprocessing ensures that all solvers start with a uniform level of description of the model. The DAG is then translated into the input format required by each solver. (For the tests, we switched off the model simplification stage, since it is not yet efficiently implemented.)

A testing environment was created to make as much as possible of the testing work automatic. We had to rerun many calculations for many models whenever bugs were fixed, new versions of a solver became available, new solvers were added, improvements in the testing statistics were made, etc.; this would have been impossible without the support of such a testing environment.

The simplifier and the translators from AMPL into the input formats for the solvers tested are available in the COCONUT environment (Schichl [32]). The remainder of the test environment is not fully automatic and hence not publicly available.

Performance criteria. All solvers are tested with the default options suggested by the providers of the codes. (Most solvers may be made to work sig-

nificantly better by tuning the options to particular model classes; hence the view given by our comparisons may look more pessimistic than the view users get who spend time and effort on tuning a solver to their models. However, in a large-scale, automated comparison, it is impossible to do such tuning.)

The timeout limit used was (scaled to a 1000 MHz machine) around 180 seconds CPU time for models of size 1, 900 seconds for models of size 2, and 1800 seconds for models of size 3 (except for GlobSol and Premium Solver which had slightly different time limits, the results stemming from earlier runs). The precise value changed between different runs since we experimented with different units for measuring time on different machines. But changing (not too much) the value for the timeout limit hardly affects the cumulative results, since the overwhelming majority of the models was either completed very quickly, or extremely slow.

The solvers LGO and GlobSol required a bounded search region, and we bounded each variable between -1000 and 1000, except in a few cases where this leads to a loss of the global optimum.

The reliability of claimed results is the most poorly documented aspect of current global optimization software. Indeed, as was shown by Neumaier & Shcherbina [26] as part of the current project, even famous state-of-the-art solvers like CPLEX 8.0 (and many other commercial MILP codes) may lose an integral global solution of an innocent-looking mixed integer linear program. We use the following five categories to describe the quality claimed:

Sign	Description
X	model not accepted by the solver
I	model claimed infeasible by the solver
G	result claimed to be a global optimizer
L	result claimed to be a local (possibly global) optimizer
U	unresolved (no solution found or error message)
T	timeout reached (qualifies L and U)

Note that the unresolved case may contain cases where a feasible but nonoptimal point was found, but the system stopped before claiming a local or global optimum.

Checking for best function value. The program `solcheck` from the COCONUT Environment checks the feasibility of putative solutions of solver results. This was necessary since we found lots of inconsistencies where different solvers produced different results, and we needed a way of checking whether the problem was in the solver or in our interface to it. A point was considered to be (nearly) feasible if each constraint $c(x) \in [\underline{c}, \overline{c}]$ was satisfied within an absolute error of `tol` for bounds with absolute value < 1, and a relative error of `tol` for all other bounds. Equality constraints were handled by the same recipe with $\underline{c} = \overline{c}$.

To evaluate the test results, the best function value is needed for each model. We checked in all cases the near feasibility of the best points used to verify the

claim of global optimality or feasibility. In a later stage of testing we intend to prove rigorously the existence of a nearby feasible point. More specifically:

The results of all solvers tested were taken into account, and the best function value was chosen from the minimum of the (nearly) feasible solutions by any solver. For models where this did not give a (nearly) feasible point, we tried to find feasible points by ad hoc means, which were sometimes successful. If there was still no feasible solution for a given model, the (local) solution with the minimal residual was chosen (but the result marked as infeasible).

To test which accuracy requirements on the constraint residuals were adequate, we counted the number of solutions of BARON and LINGO on `lib1s1` which were accepted as feasible with various `solcheck` tolerances `tol`. Based on the results given in the following table, it was decided that a tolerance of `tol`$= 10^{-5}$ was adequate. (The default tolerances used for running BARON and LINGO were 10^{-6}.)

solver	`tol`	all	accepted	+G	G!	G?	I?
BARON	1e-4	91	88	75	36	1	0
	1e-5	91	88	75	36	1	0
	1e-6	91	88	56	24	13	0
	1e-7	91	88	49	19	18	0
LINGO	1e-4	91	91	82	66	3	0
	1e-5	91	91	81	65	4	0
	1e-6	91	91	77	63	6	0
	1e-7	91	91	52	41	28	0

3. Test results

Notation in the tables. In the summary statistic tables the following notation is used:

Column	Description
library	describes the library
all	library/size
accepted	the number of models accepted by the solver
+G	number of models for which the global solution was found
G!	number of models for which the global solution was correctly claimed to be found
G?	number of models for which a global solution was claimed but the true global solution was in fact significantly better or the global solution is reported but in fact that is an infeasible point
I?	number of models for which the model was claimed infeasible although a feasible point exists

For models where a local optimizer finds the global optimum (without knowing it), the purpose of a global code is to check whether there is indeed no better point; this may well be the most time-consuming part of a complete search. For the remaining models the search for the global optimum is already hard. We therefore split the evaluation into

- 'easy location models', where the local optimizer MINOS found a feasible point with the global objective function value, and
- 'hard location models', all others where MINOS failed.

For the easy and hard models (according to this classification), the claimed status is given in the columns; the rows contain a comparison with the true status:

Column	Description
wrong	number of wrong claims, i.e. the sum of G? and I? from the summary statistic table
+G	how often the solution found was in fact global
−G	how often it was in fact not global
I	how many models are in fact infeasible

For the purposes of comparison in view of roundoff, we rounded function values to 3 significant decimal digits, and regarded function values of $< 10^{-4}$ as zero (but print them in the tables as nonzeros) when the best known function value was zero. Otherwise, we regard the global minimum as achieved if the printed (rounded) values agree.

For each library detailed tables for all models and all solvers tested, and many more figures (of the same type as the few presented here) are also available, for reasons of space they are presented online on the web [3]. There we give a complete list of results we currently have on the nine model classes (i.e., excluding the models with 1000 or more variables, and the few models described before.)

GlobSol and Premium Solver. To test GlobSol, we used an evaluation version of LAHEY Fortran 95 compiler. Note that we had difficulties with the Intel Fortran compiler.

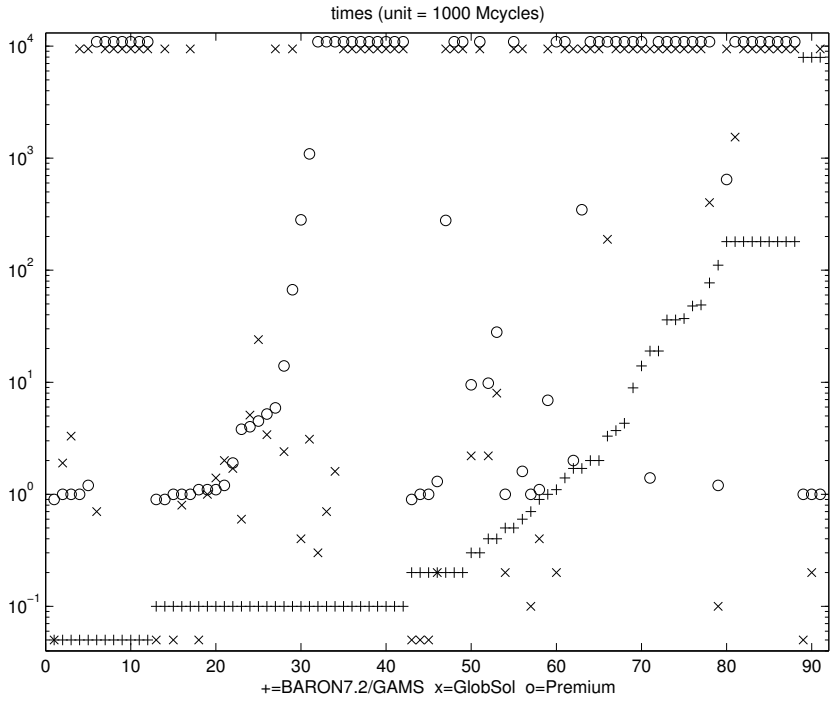
In the first round we tested GlobSol on Library 1 size 1 problems (containing 91 problems) with the same time limit as used for the other solvers. GlobSol failed to solve most of the problems within the strict time limit. For this reason we decided to use a very permissive time limit (even then, only half the accepted problems were solved).

The same tests as for GlobSol were performed for Premium Solver, with similar performance.

Figure 2 compares the global solvers GlobSol, Premium Solver, and BARON on the size 1 problems from Library 1. The figure contains timing results for the models described in the figure caption, sorted by the time used by BARON.

Conversion times for putting the models into the format required by the solvers are not included. Times (given in units of 1000 Mcycles) below 0.05 are places on the bottom border of each figure, models for which the global minimum was not found by the solver get a dummy time above the timeout value, and are placed at the top border of each figure, in slightly different heights for the different solvers. In this way one can assess the successful completion of the global optimization task.

Fig. 2. Times for `lib1s1`, all models, GlobSol and Premium Solver vs. BARON



In a few cases, GlobSol and Premium Solver found solutions where BARON failed, which suggests that BARON would benefit from some of the advanced interval techniques implemented in GlobSol and Premium Solver.

However, GlobSol and Premium Solver are much less efficient in both time and solving capacity than BARON. To a large extent this may be due to the fact that both GlobSol and Premium Solve strive to achieve mathematical rigor, resulting in significant slowdown due to the need of rigorously validated techniques. (There may be also systematic reasons in the comparison, since GlobSol does not output a best approximation to a global solution but boxes, from which we had to extract a test point.)

Moreover, the theoretically rigorous performance guarantees are not borne out, in that both solvers make some wrong claims, probably due to lack of care in programming.

In view of these results, we refrained from further testing GlobSol and Premium Solver.

The summary statistics on `lib1s1` can be found in the following tables.

GlobSol summary statistics						
library	all	accepted	+G	G!	G?	I?
`lib1s1`	91	77	39	38	20	0

Premium summary statistics						
library	all	accepted	+G	G!	G?	I?
`lib1s1`	91	75	45	31	10	1

A more detailed table gives more information. Included is an evaluation of the status claimed about model feasibility and the global optimum, and the true status (based on the best point known to us).

GlobSol on `lib1s1`								
status	all	wrong	easy location			hard location		
			+G	−G	I	+G	−G	I
all	91	20	30	34	0	9	18	0
G	58	20	30	13	0	8	7	0
U	19	0	0	16	0	1	2	0
X	14	0	0	5	0	0	9	0

Premium Solver on `lib1s1`								
status	all	wrong	easy location			hard location		
			+G	−G	I	+G	−G	I
all	91	11	36	28	0	9	18	0
G	41	10	25	9	0	6	1	0
L	12	0	6	2	0	2	2	0
LT	9	0	4	3	0	1	1	0
U	12	0	1	2	0	0	9	0
X	16	0	0	11	0	0	5	0
I	1	1	0	1	0	0	0	0

COCOS. The COCONUT environment is an open domain global optimization platform. Apart from the translators used for the present comparison, it contains a configurable solver COCOS with many modules that can be combined to yield various combination strategies for global optimization. We tested the strategy called by "`cocos -hopt +bs +lp_solve <model>.dag`".

COCOS summary statistics						
library	all	accepted	+G	G!	G?	I?
lib1s1	91	91	51	36	12	0
lib1s2	80	80	26	11	14	0
lib1s3	41	41	3	0	2	0
lib2s1	324	324	182	98	29	0
lib2s2	99	99	37	10	5	0
lib2s3	95	95	11	2	1	0
lib3s1	217	217	143	48	2	0
lib3s2	69	69	38	19	0	0
lib3s3	22	22	4	3	0	0

Reliability analysis for COCOS	
	global minimum found/accepted
size 1	$376/632 \approx 59\%$
size 2	$101/248 \approx 41\%$
size 3	$18/158 \approx 11\%$
all	$495/1038 \approx 48\%$
	correctly claimed global/accepted
size 1	$182/632 \approx 29\%$
size 2	$40/248 \approx 16\%$
size 3	$5/158 \approx 3\%$
all	$227/1038 \approx 22\%$
	wrongly claimed global/claimed global
size 1	$43/225 \approx 19\%$
size 2	$19/59 \approx 32\%$
size 3	$3/8 \approx 38\%$
all	$65/292 \approx 22\%$
	claimed infeasible/accepted and feasible
size 1	$0/626 = 0\%$
size 2	$0/241 = 0\%$
size 3	$0/147 = 0\%$
all	$0/1014 = 0\%$

ICOS. ICOS is a pure constraint solver, which currently cannot handle models with an objective function, and hence was tested only on Library 3. (An enhanced version of ICOS, capable also of rigorously solving global optimization problems is under development.)

ICOS also claims to provide mathematically rigorous results, and indeed, it is the only complete solver tested that did not make any false claims in our tests.

ICOS summary statistics						
library	all	accepted	+G	G!	G?	I?
lib3s1	217	207	145	68	0	0
lib3s2	69	63	34	12	0	0
lib3s3	22	20	5	0	0	0

Reliability analysis for ICOS (on pure CSPs only)	
	global minimum found/accepted
size 1	145/207 ≈ 70%
size 2	34/63 ≈ 54%
size 3	5/20 ≈ 25%
all	184/290 ≈ 63%
	correctly claimed global/accepted
size 1	68/207 ≈ 33%
size 2	12/63 ≈ 19%
size 3	0/20 = 0%
all	80/290 ≈ 28%
	wrongly claimed global/claimed global
size 1	0/68 = 0%
size 2	0/12 = 0%
all	0/80 = 0%
	claimed infeasible/accepted and feasible
size 1	0/201 = 0%
size 2	0/59 = 0%
size 3	0/18 = 0%
all	0/278 = 0%

BARON, LINGO, OQNLP, LGO and MINOS. The following tables contain the summary statistics for the performance of the other solvers tested, apart from COCOS.

BARON7.2/GAMS summary statistics						
library	all	accepted	+G	G!	G?	I?
lib1s1	91	88	88	64	0	0
lib1s2	80	77	71	46	3	0
lib1s3	41	33	23	5	1	0
lib2s1	324	296	254	206	11	0
lib2s2	99	89	82	48	2	0
lib2s3	95	87	51	25	6	0
lib3s1	217	195	182	180	3	3
lib3s2	69	63	57	57	2	1
lib3s3	22	20	14	13	1	0

LINGO9 summary statistics						
library	all	accepted	+G	G!	G?	I?
lib1s1	91	91	84	70	3	0
lib1s2	80	80	53	42	14	1
lib1s3	41	41	12	1	2	1
lib2s1	324	324	260	232	26	1
lib2s2	99	99	71	45	10	0
lib2s3	95	95	49	26	11	0
lib3s1	217	217	189	189	15	0
lib3s2	69	69	55	55	9	0
lib3s3	22	22	10	9	4	0

OQNLP/GAMS summary statistics						
library	all	accepted	+G	G!	G?	I?
lib1s1	91	91	83	0	0	1
lib1s2	80	80	70	0	0	1
lib1s3	41	28	12	0	0	5
lib2s1	324	315	272	0	0	1
lib2s2	99	95	90	0	0	0
lib2s3	95	83	68	0	0	1
lib3s1	217	213	196	0	0	3
lib3s2	69	67	47	0	0	2
lib3s3	22	19	11	0	0	3

LGO/GAMS summary statistics						
library	all	accepted	+G	G!	G?	I?
lib1s1	91	85	65	0	0	0
lib1s2	80	78	39	0	0	8
lib1s3	41	31	4	0	0	12
lib2s1	324	309	234	0	0	4
lib2s2	99	94	61	0	0	15
lib2s3	95	57	23	0	0	10
lib3s1	217	212	155	0	0	46
lib3s2	69	66	35	0	0	21
lib3s3	22	11	3	0	0	4

MINOS/GAMS summary statistics						
library	all	accepted	+G	G!	G?	I?
lib1s1	91	91	64	0	0	0
lib1s2	80	80	47	4	0	4
lib1s3	41	41	19	1	0	4
lib2s1	324	323	245	15	1	12
lib2s2	99	97	80	4	2	3
lib2s3	95	92	42	1	0	8
lib3s1	217	213	155	3	0	27
lib3s2	69	68	35	0	1	12
lib3s3	22	21	11	1	0	4

The corresponding reliability analysis tables are as follows.

Reliability analysis for BARON 7.2	
	global minimum found/accepted
size 1	524/579 ≈ 91%
size 2	210/229 ≈ 92%
size 3	88/140 ≈ 63%
all	821/950 ≈ 86%
	correctly claimed global/accepted
size 1	450/579 ≈ 78%
size 2	151/229 ≈ 66%
size 3	43/140 ≈ 31%
all	644/950 ≈ 68%
	wrongly claimed global/claimed global
size 1	14/464 ≈ 3%
size 2	7/158 ≈ 4%
size 3	8/51 ≈ 16%
all	29/675 ≈ 4%
	claimed infeasible/accepted and feasible
size 1	3/571 ≈ 1%
size 2	1/222 ≈ 0%
size 3	0/128 = 0%
all	4/921 ≈ 0.4%

Reliability analysis for LINGO9	
	global minimum found/accepted
size 1	533/632 ≈ 84%
size 2	179/248 ≈ 72%
size 3	71/158 ≈ 45%
all	783/1038 ≈ 75%
	correctly claimed global/accepted
size 1	491/632 ≈ 78%
size 2	142/248 ≈ 57%
size 3	36/158 ≈ 23%
all	669/1038 ≈ 64%
	wrongly claimed global/claimed global
size 1	44/535 ≈ 8%
size 2	33/175 ≈ 19%
size 3	17/53 ≈ 32%
all	94/763 ≈ 12%
	claimed infeasible/accepted and feasible
size 1	1/624 ≈ 0%
size 2	1/241 ≈ 0%
size 3	1/143 ≈ 0%
all	3/1008 ≈ 0.3%

Reliability analysis for OQNLP	
	global minimum found/accepted
size 1	551/619 ≈ 89%
size 2	207/242 ≈ 86%
size 3	91/130 ≈ 72%
all	847/993 ≈ 86%
	claimed infeasible/accepted and feasible
size 1	5/611 ≈ 1%
size 2	3/235 ≈ 1%
size 3	9/124 ≈ 8%
all	17/944 ≈ 2%

Reliability analysis for LGO	
	global minimum found/accepted
size 1	454/606 ≈ 75%
size 2	135/238 ≈ 57%
size 3	30/99 ≈ 30%
all	619/943 ≈ 66%
	claimed infeasible/accepted and feasible
size 1	50/598 ≈ 8%
size 2	44/229 ≈ 18%
size 3	26/91 ≈ 30%
all	120/918 ≈ 13%

Reliability analysis for MINOS	
	global minimum found/accepted
size 1	464/627 ≈ 74%
size 2	162/245 ≈ 66%
size 3	72/154 ≈ 47%
all	698/1026 ≈ 68%
	correctly claimed global/accepted
size 1	18/627 ≈ 3%
size 2	8/245 ≈ 3%
size 3	3/154 ≈ 2%
all	29/1026 ≈ 3%
	wrongly claimed global/claimed global
size 1	1/19 ≈ 5%
size 2	3/11 ≈ 27%
size 3	0/3 = 0%
all	4/33 ≈ 12%
	claimed infeasible/accepted and feasible
size 1	39/619 ≈ 6%
size 2	19/238 ≈ 8%
size 3	16/151 ≈ 11%
all	74/1008 ≈ 7%

Finally, we compare all solvers on `lib1s1` (problems from GlobalLib with less than 10 variables).

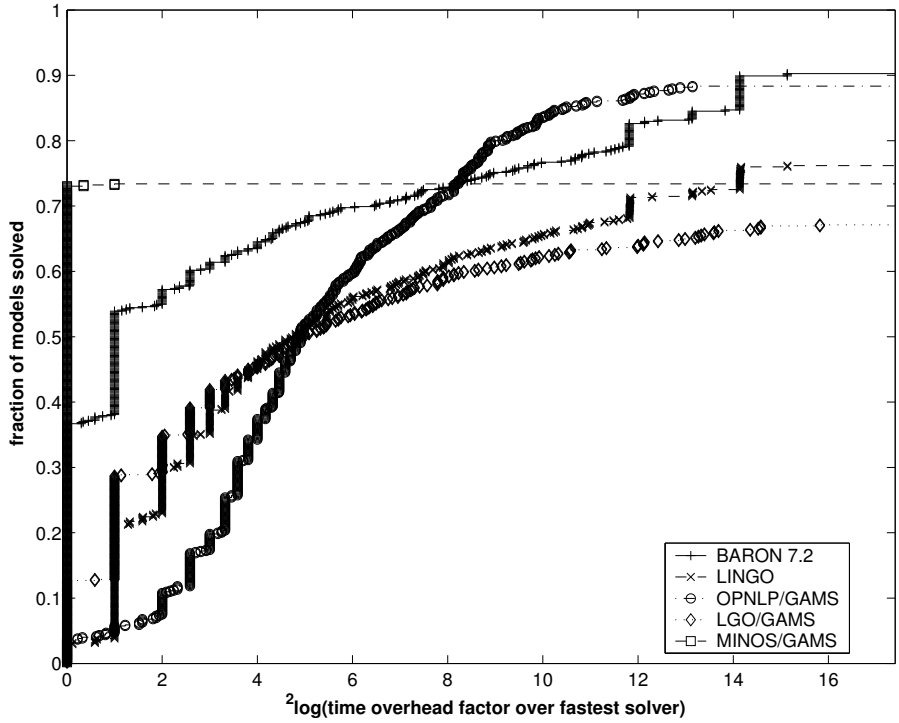
`lib1s1` summary statistics						
library	all	accepted	+G	G!	G?	I?
BARON	91	88	88	64	0	0
LINGO	91	91	83	70	6	0
OQNLP	91	91	83	0	0	1
LGO	91	85	65	0	0	0
MINOS	91	91	64	0	0	0
COCOS	91	91	51	36	12	0
Premium	91	75	45	31	10	1
GlobSol	91	77	39	38	20	0

4. Conclusions

The results speak for themselves, and the main conclusions were already given in the opening section. Here we add a few more observations.

- The most remarkable observation is that the models from Library 1, which were collected specifically as test problems for global optimization, do not behave much differently from those of Library 2, which were collected as test problems for local optimization routines. In particular, **many problems that were solved in the past only as local optimization problems were in fact global problems where the global minimizer is not easily found.**
- The GAMS solvers LGO and OQNLP are very cautious, never claiming a global minimum. This reflects the observed unreliability of the internal claims (as seen by studying the logfile) of the LGO version used by GAMS, and the decision of GAMS rather to err on the conservative side.
- It is on first sight surprising that under GAMS, the local solver MINOS sometimes claims to have found a global result. This is the case, e.g., because some models are recognized as linear programs for which every local solution is global. (The cases with G? are caused by too inaccurate approximate solutions.)
- In a few cases, solvers reported infeasibility, although the point they found was considered feasible by `solcheck`.
- Conversely, a number of the wrong claims of globality (especially of LINGO) are caused by the fact that an approximate minimizer was found but that the constraints were not satisfied with the accuracy one could have expected from the solver settings – some residual was larger than 10^{-5}, although the requested tolerance was 10^{-6}.
- In the mean time, BARON/GAMS had some bugs fixed, which eliminates all wrong claims to infeasibility and reduces the rate of wrong claims of global optimality to $12/675 = 1.8\%$. This raises the hope that the next official release has a much improved reliability.

Fig. 3. Performance profiles for reaching the global optimum

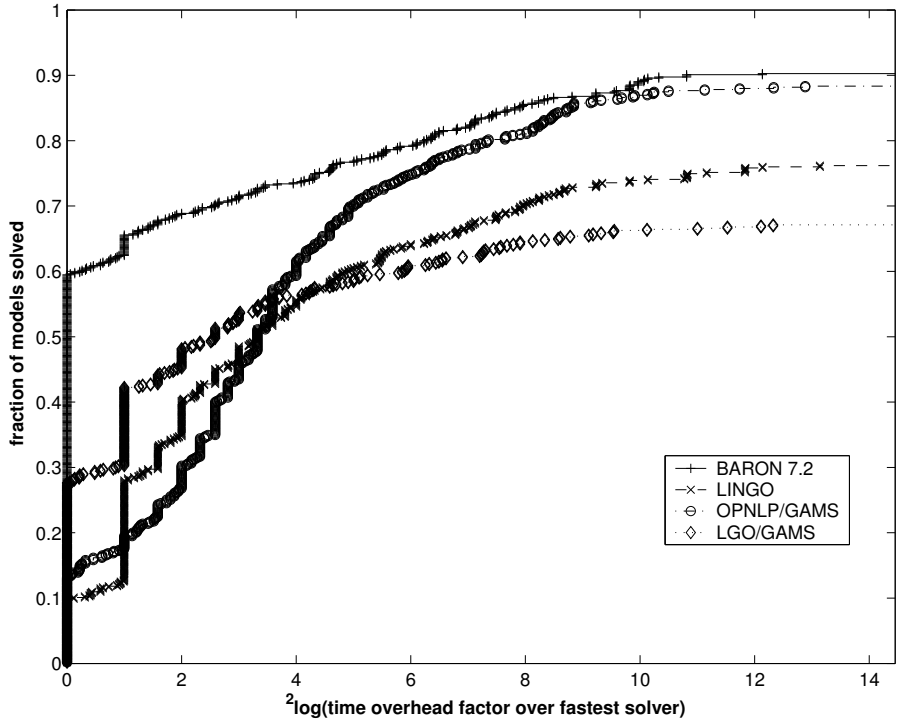


Performance profiles. Upon request by a reviewer, we also add some performance profiles (introduced by Dolan & Moré [8]). However, the profiles must be interpreted with much caution since often a global solver finds the global minimum quite early and then spends a lot of time checking whether there is another one. Unless a time limit is reached, BARON and LINGO quit only after they completed the search, while OQNLP and LGO quit according to some statistical criterion, and MINOS quits directly after finishing the local optimization. This is reflected in a severe dependence of the performance profiles on the solvers selected; cf. Figures 3 and 4.

Figure 3 displays the fraction of problems solved to global optimality within a factor 2^τ of the time the best solver (from BARON, LINGO, OQNLP, LGO and MINOS) needed for the problem, among all problems accepted by all these solvers. Figure 4 displays the same, but with MINOS excluded. (The numbers at $\tau = 0$ add up to more than 100% because of the way we rounded tiny times – as discussed in Section 2 –, which resulted in many ties for the best times, which were multiply counted. The steps in the profiles also come from this rounding procedure.)

Similarly, Figure 5 displays the fraction of problems where BARON or LINGO finished the global search successfully within a factor 2^τ of the time the better

Fig. 4. Performance profiles for reaching the global optimum

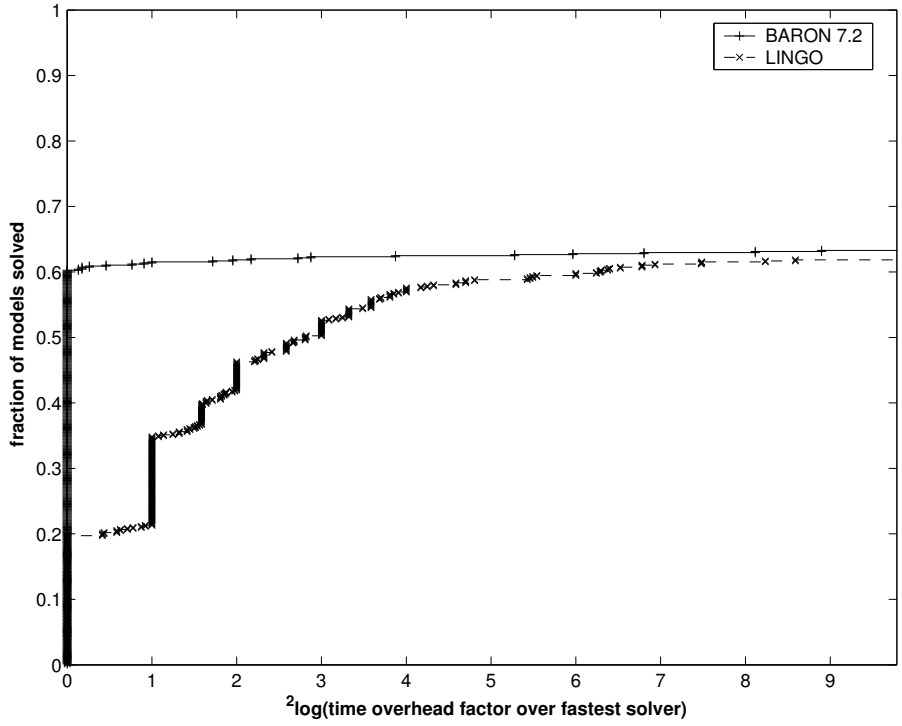


of the two solvers needed for finishing the global search, among all problems accepted by both solvers.

Guidelines for code developers. Our extensive testing revealed a number of desirable properties of solvers, that would have saved much of our time, and that code developers should consider taking into account:

- Solvers should never enter infinite loops; there should be an option to stop after (approx.) a given time limit.
- A constant objective should not cause difficulties.
- Solvers which have access to the computational tree should *not* fail because of overflow, underflow, exceptions, exp(1000), $1/0$, $0 * \log(0)$, $\log(0)$.
- Avoid confusing messages (e.g., detected infeasibility should not be labelled "success").
- Output messages should be meaningful to the user (e.g., "numerical difficulties encountered" $\Longrightarrow$ Results still OK??).
- There should be an informative final quality claim (such as the XIGLU classification used here).
- A large number of problems have not all variables bounded; so solvers should be able to address this. If a solver needs finite bounds to perform well, these should be set by default to reasonable values where none are provided by the model, and a warning should be given to the user.

Fig. 5. Performance profiles for reaching the global optimum



References

1. R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart, Designing and reporting on computational experiments with heuristic methods, Journal of Heuristics 1 (1995), 9–32.
`http://www.research.att.com/~mgcr/abstracts/guidelines.html`
2. F. Benhamou and F. Goualard, Universally Quantified Interval Constraints. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'2000)*, pages 67–82, 2000.
3. COCONUT test results, WWW-directory, 2004.
`http://www.mat.univie.ac.at/~neum/glopt/coconut/tests/figures/`
4. H.P. Crowder, R.S. Dembo and J.M. Mulvey, On reporting Computational Experiments with Mathematical Software, ACM Transactions on Mathematical Software, 5 (1979), 193–203.
5. N.I.M. Gould, D. Orban and Ph.L. Toint, CUTEr, a constrained and unconstrained testing environment, revisited, WWW-document, 2001.
`http://cuter.rl.ac.uk/cuter-www/problems.html`
6. L.C.W. Dixon and G.P. Szegö, The Global Optimization Problem: An Introduction, pp. 1–15 in: Towards Global Optimization 2, North-Holland, Amsterdam 1978.
7. E.D. Dolan and J.J. Moré, Benchmarking Optimization Software with COPS, Tech. Report ANL/MCS-246, Argonne Nat. Lab., November 2000.
`http://www-unix.mcs.anl.gov/~more/cops`
8. E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles, Math. Programming 91 (2002), 201–213. `http://www-unix.mcs.anl.gov/~more/cops`
9. R. Fourer, D.M. Gay and B.W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, Duxbury Press, Brooks/Cole Publishing Company, 1993.
`http://www.ampl.com/cm/cs/what/ampl/`

10. Frontline Systems, Inc., Solver Technology - Global Optimization, WWW-document (2003).
http://www.solver.com/technology5.htm
11. GAMS Solver descriptions, GAMS/OQNLP, WWW-document (2003).
http://www.gams.com/solvers/solvers.htm#OQNLP
12. GAMS World, WWW-document, 2002.
http://www.gamsworld.org
13. GLOBAL Library, WWW-document, 2002.
http://www.gamsworld.org/global/globallib.htm
14. H.J. Greenberg, Computational testing: Why, how, and how much, ORSA Journal on Computing 2 (1990), 94–97.
15. W. Huyer, A comparison of some algorithms for bound constrained global optimization, WWW-document (2004).
http://www.mat.univie.ac.at/~neum/glopt/contrib/compbound.pdf
16. ILOG. *ILOG Solver. Reference Manual*. 2002.
17. R.H.F. Jackson, P.T. Boggs, S.G. Nash and S. Powell, Guidelines for reporting results of computational experiments. Report of the ad hoc committee, Mathematical Programming 49 (1990/91), 413–426.
18. E. Janka, Vergleich stochastischer Verfahren zur globalen Optimierung, Diplomarbeit, Mathematisches Inst., Universität Wien, 1999.
A shorter online version in English language is at
http://www.mat.univie.ac.at/~neum/glopt/janka/gopt_eng.html.
19. R.B. Kearfott, Rigorous Global Search: Continuous Problems, Kluwer, Dordrecht 1996.
www.mscs.mu.edu/~globsol
20. Y. Lebbah, ICOS (Interval COnstraints Solver), WWW-document (2003).
http://www-sop.inria.fr/coprin/ylebbah/icos/
21. Lindo Systems, Inc., New LINGO 8.0, WWW-document (2003).
http://www.lindo.com/table/lgofeatures8t.html
22. H. Mittelmann, Benchmarks. WWW-document, 2002.
http://plato.la.asu.edu/topics/benchm.html
23. M. Mongeau, H. Karsenty, V. Rouzé, J.-B. Hiriart-Urruty, Comparison of public-domain software for black box global optimization, Optimization Methods and Software 13 (2000), 203–226.
http://mip.ups-tlse.fr/publi/rapp99/99.50.html
24. B.A. Murtagh and M.A. Saunders, MINOS 5.4 User's Guide, Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, December 1983 (revised February 1995).
http://www.sbsi-sol-optimize.com/Minos.htm
25. A. Neumaier, Complete Search in Continuous Global Optimization and Constraint Satisfaction, pp. 271–369 in: Acta Numerica 2004 (A. Iserles, ed.), Cambridge University Press 2004.
26. A. Neumaier and O. Shcherbina, Safe bounds in linear and mixed-integer programming, Math. Programming A 99 (2004), 283-296.
http://www.mat.univie.ac.at/~neum/papers.html#mip
27. J.D. Pinter, Global Optimization in Action, Kluwer, Dordrecht 1996.
http://www.dal.ca/~jdpinter/l_s_d.html
28. H. D. Ratliff and W. Pierskalla, Reporting Computational Experience in Operations Research, Operations Research 29 (2) (1981), xi–xiv.
29. H.S. Ryoo and N.V. Sahinidis, A branch-and-reduce approach to global optimization, J. Global Optim. 8 (1996), 107–139.
http://archimedes.scs.uiuc.edu/baron/baron.html
30. H. Schichl, Global optimization in the COCONUT project. in: Proceedings of the Dagstuhl Seminar "Numerical Software with Result Verification", Springer Lecture Notes in Computer Science 2991, Springer, Berlin, 2004.
31. H. Schichl, Mathematical Modeling and Global Optimization, Habilitation Thesis (2003), Cambridge Univ. Press, to appear.
http://www.mat.univie.ac.at/~herman/papers/habil.ps
32. H. Schichl, The COCONUT Environment. Web site (2004).
http://www.mat.univie.ac.at/coconut-environment/
33. O. Shcherbina, A. Neumaier, Djamila Sam-Haroud, Xuan-Ha Vu and Tuan-Viet Nguyen, Benchmarking global optimization and constraint satisfaction codes, pp.211–222 in: Ch. Bliek, Ch. Jermann and A. Neumaier (eds.), Global Optimization and Constraint Satisfaction, Springer, Berlin 2003.

34. M. Tawarmalani and N.V. Sahinidis, Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications, Kluwer, Dordrecht 2002.
35. M. Tawarmalani and N.V. Sahinidis, Global optimization of mixed-integer nonlinear programs: A theoretical and computational study, Math. Programming 99 (2004), 563–591.
36. B. Vanderbei, Nonlinear Optimization Models, WWW-document.
http://www.orfe.princeton.edu/∼rvdb/ampl/nlmodels/