Benchmarking Global Optimization and Constraint Satisfaction Codes

Oleg Shcherbina and Arnold Neumaier,
University of Vienna, Austria

Djamila Sam-Haroud, Xuan-Ha Vu and Tuan-Viet Nguyen,
Swiss Federal Institute of Technology, Lausanne, Switzerland

July 9, 2003

Abstract. A benchmarking suite describing over 1000 optimization problems and constraint satisfaction problems covering problems from different traditions is described, annotated with best known solutions, and accompanied by recommended benchmarking protocols for comparing test results.

1 Introduction

Global optimization problems and constraint satisfaction problems are NP-hard, and a widely held conjecture states that no polynomial, i.e., universally fast, algorithms solving such problems exist.

It is thus necessary to assemble a sufficient set of relevant and well-categorized problems in order to be able to evaluate experimentally different approaches, techniques, and/or solution strategies, and to compare them according to various performance measures.

The objective of the current work is to provide a **benchmark** consisting of a comprehensive suite of representative problems, covering as far as possible all the categories of problems that are relevant either from a scientific, technical, or industrial point of view.

A good benchmark must be one that can be interfaced with our framework and with other, existing systems, 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 [11]. A very recent web site by GAMS World [6] started collecting real life global optimization problems with industrial relevance, but currently most problems on this site are without computational results. Our benchmark includes a large part of the problems from these two projects.

All problems in our benchmark are represented in a common format suitable for automatic execution on global optimization and constraint satisfaction software. To ensure that benchmarking results are comparable across different solvers and environments, a **benchmarking protocol** is defined, whose execution on the benchmark will provide a number of objective performance measures for any implementation of global optimization and constraint satisfaction techniques.

With the present benchmarking suite, it is the first time that a large benchmark

- is made publicly available
- in a uniform, widely accessible format,
- covering problems from different traditions – nonlinear programming, global optimization, and constraint satisfaction,
- including most problems from the more restricted traditional collections of benchmarking problems as particular cases,
- checked for consistence,
- annotated with type and complexity information,
- (almost) complete with best known solutions, and
- accompanied by benchmarking protocols for comparing test results.

2 Description of the benchmarking suite

The delivered benchmarking suite is a comprehensive collection of 329 constraint satisfaction and 993 optimization problems from academic, industrial and real-life environments. Executable versions of these test problems, as well as information on their sources, are publicly available at http://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark.html

The problems range in difficulty from easy to very challenging, their sizes from a few number of variables to over 1000 variables. These test problems come from both the research literature and a wide spectrum of applications, including:

- Chemical engineering (pooling and blending, separation, heat exchanger network design, phase and chemical equilibrium, reactor network synthesis, etc.)
- Computational chemistry (including molecular design)
- Civil engineering problems
- Robotics problems
- Operations research problems
- Economics problems (including Nash equilibrium, Walrasian equilibrium, and traffic assignment problems)
- Multicommodity network flow problems
- Process design problems
- Stability analysis problems
- VLSI chip design problems
- Portfolio optimization problems

Older collections covered. The problem suite incorporates as integral part most problems from the CUTE test collection [8] (covering among others the Argonne test set, the Hock and Schittkowski collection, the Dembo network problems, the Gould quadratic programs, etc.), from the handbook of test problems in local and global optimization [2], from the GLOBAL Library [6], and from the Numerica [15] test problem collection; in addition many other constraint satisfaction problems from the literature.

Some problems (e.g. linear or convex quadratic programs) are known to be solvable to global optimality by local optimization alone. They were retained in the benchmark to be able to measure the overhead which global solvers have in order to prove optimality, compared with local solvers which are usually significantly faster and find of course for convex problems a global minimum.

Common Format. All test problems are coded in the AMPL modeling language. AMPL [3] is a flexible and convenient algebraic modeling language enabling rapid prototyping and model development. It is of widespread use in the optimization community, as attested by the large number of existing interfaces with state-of-the-art optimization solvers `http://www.ampl.com/solvers.html`. Unfortunately, no current modeling system allows the input of interval data reflecting uncertainties in the parameters specified. Such a facility would be important for fully rigorous search.

The CUTE problems existed already in AMPL format, coded by Hande Y. Benson, and made available in the collection of AMPL files by Vanderbei [14]. All other problems were either newly coded in AMPL, or automatically translated from existing GAMS models using GMS2XX [7] and GAMS/Convert [4].

As far as reasonable, all problems were checked for correctness; inconsistencies with information available from other sources were removed if possible. Information about (approximate) solutions and putative global minimizers and minima were provided in all but a few cases (where runtime constraints became active).

A few problems from the collections mentioned are missing in the present benchmark because of our inability to get a valid authentic version (e.g., the 'handbook' [2] contains numerous inconsistencies), or because of the presence of constructs or functions not supported by our current system (such as `if`, `erf`).

The AMPL code for all problems in the benchmark is available online at the web site mentioned, through links from tables with one line summaries of problem characteristics.

Typology of problems. The main criteria for categorizing the test problems are mathematical properties reflecting their degree of formal complexity. This is not equivalent with computational complexity, but typically more complex objectives and constraints require more complex algorithms to handle them.

A second criterion is the problems closeness to applicability in real-life applications.

Each problem is classified as in the CUTE collection of test problems, except that problems with a V code for the number of variables or constraints have the default value for that number after the V. In addition, a few more types of constraints are distinguished, and an optional type characterizing the solution set is provided.

The classification is a string of the form OCSD-KIT-n-m in which the letters are replaced by appropriate codes, as follows:

```
O = objective code:
        C = Constant
        L = Linear
        N = No objective
        Q = Quadratic
        S = Sum of squares
        O = Other (none of the above)
C = constraint type:
        B = Bounds on variables
        L = Linear
        N = linear Network
        Q = Quadratic
        U = Unconstrained
        X = only fiXed variables
        P = Polynomial
        T = Trigonometric
        O = Other (none of the above)
S = smoothness:
        R = Twice continuously differentiable
        I = Other
D = degree of differentiability:  0, 1, or 2

K = kind of the problem:
        A = Academic
        M = Modeling
        R = Real application
I = internal variables:
        Y = yes, problems has useful internal variables
        N = no useful internal variables
T = Type of solution set:
       I = Isolated
       N = Non-isolated
```

```
    U = Unknown

n = number of variables, or: V = varies, followed by default
m = number of constraints, or: V = varies, followed by default
```

In most cases, we also report the number of nonzeros and nonlinear nonzeros in an internal representation used in the GAMS system [5]. These give additional complexity information.

Solutions. For all problems with nonconstant objective function we provide on the WWW site the function value of the best point found, in many cases the global optimum.

Many test problems in the current benchmarking suite contain floating-point constants. Unfortunately, the AMPL software currently does not allow to control the rounding errors in the conversion to an internal representations. For this reason, the solutions reported in the current benchmarking suite are approximate only. Usually the solutions should be affected only in insignificant decimal places but there may be a nontrivial effect in case of degeneracies. This must be kept in mind when comparing our solution information to the literature, or to rigorous solvers which have a mathematically rigorous input/output interface.

The information about the solutions, their status (feasible/local minimum/global minimum) and accuracy (approximate/verified) will be updated as we run the benchmark with verifying global solvers.

3 Benchmarking protocol

The benchmarking suite is designed to allow researchers convenient testing of their own algorithms, and to post the results for comparison with other algorithms.

The benchmarking protocol defines the experimentation procedure and the criteria used to measure *efficiency* of the algorithm. It can be carried out within a limited amount of work, and hence be checked in regular intervals. We decided to create a benchmarking protocol that works on the full set of problems, while allowing code developers to assess progress without endless testing.

A well-designed benchmarking protocol must be able to assess work in progress as well as the final results obtained. Since it may be impractical to frequently repeat testing on problems of realistic size, the **benchmarking protocol for assessing work in progress** is designed to be executable in a limited amount of time (approximately 24h clock time on a fast computer), and gives information about successes and weaknesses to guide the further development. The **benchmarking protocol for assessing a release of a code** has all the time limits specified below multiplied by a factor of 10.

In order to compare benchmark results across different platforms, we specify that each benchmark involves the computation of a **standard unit time**, as suggested in the first global optimization benchmark by DIXON & SZEGÖ [1] in 1974. To make it unambiguous and large enough for today's computers, we define it as the CPU time needed to carry out the C++ program defined in the appendix, compiled without optimization or debugging option. It evaluates the 4-dimensional Shekel5 test function at 10^8 specified points.

Timeout limits and runtimes are given in multiples of standard unit times. We are aware of the fact that items not accounted for in our standard unit time affect the performance of global optimization methods. The above choice was made on the assumption that people who want to switch from local to global optimization are most likely to compare against performance in terms of equivalent function values.

For branch and bound codes, time spent on memory management is relevant also. Since people solving optimization problems locally only or using heuristics like simulated annealing, use floating points only, and use little memory, total runtime (and not number of function values or number of iterations) seems to be the most relevant unit to compare with – it tells how much slower or faster the complete solver is compared with heuristics.

In the following,we describe the information used and produced by the protocol (parameters, parameter settings, stopping rules and output).

Parameters and rules The parameters and rules used by the benchmarking protocols are defined as follows:

- The two main query options defined are: running *incomplete* and *complete* search; each search option may be executed in either *approximate* or *rigorous* verification mode, on any of four *complexity classes*. This gives $2 * 2 * 4 = 16$ cases to consider.
  - A successful *incomplete search* shall mean:

  - (for problems with objective function) running the search until the best known function value was found for the first time to within a relative accuracy of 10^{-6} or an absolute accuracy of 10^{-9}, whichever is more generous;
  - (for problems without objective function) running the search until the first feasible point is found to within a relative accuracy of 10^{-3} or an absolute accuracy of 10^{-6}, whichever is more generous.
- A successful *complete search* shall mean:
  - (for problems with objective function) locating all global optimizers, or asserting correctly that the objective function is unbounded below on the search space.
  - (for problems without objective function and a discrete solution set) finding all feasible points, or asserting correctly that none exists.
  - (for problems without objective function and nonisolated solutions) finding an explicit description of the feasible point set within a relative accuracy of at most 5 percent of the maximal side of the interval hull of the feasible point set.
- in *approximate verification mode*,
  - the solution(s) must satisfy all constraints to within a relative accuracy of approximately 10^{-3} or an absolute accuracy of approximately 10^{-6}, whichever is more generous.
- in *rigorous verification mode*,
  - the solution(s) must be enclosed in boxes of a relative accuracy of approximately 10^{-3} or an absolute accuracy of approximately 10^{-6}, whichever is more generous, and any such box is guaranteed with certainty to contain a feasible point, in spite of rounding errors made (but there may be additional boxes neither excluded nor guaranteed to contain a solution),
  - (for problems with objective function) the global minimum value is to be enclosed with an approximate relative accuracy of 10^{-6} or an absolute accuracy of 10^{-9}, whichever is more generous.
- All problems are assigned to one of four complexity classes according to the number of variables defining the problem:
  - *size* 1, with $0-9$ variables,

    - *size* 2, with $10 - 99$ variables,
    - *size* 3, with $100 - 999$ variables,
    - *size* 4, with ≥ 1000 variables.
  - The problems are sorted into three libraries, two with optimization problems, one with constraint satisfaction problems.

    For testing, the problems of each library are arranged in a fixed order of increasing dimension within each complexity class. This order will probably change with time, to reflect the currently unknown real difficulty of the test problems. Thus the file defining the ordering contains a version letter which should be quoted when publishing results using the benchmark. Version A of the required ordering is specified in

    `http://www.mat.univie.ac.at/~neum/glopt/coconut/probclasses.txt`

    This gives a total of $3 * 4 = 12$ lists of problems, to be executed with up to four possible pairs of query options.

  In some cases it is unreasonable that the full goal is achieved (for ill-conditioned solutions, or for non-isolated solution sets); in these cases, suitably relaxed goals may be specified in comments to the solution statistic provided.

- Each problem of a given complexity class gets its own timeout limits for both complete and incomplete search; and each complexity class within each of the three libraries gets total timeout limits for both complete and incomplete search.
- Part of the documentation of running the benchmark should contain:
  - the program name and version,
  - if applicable, the compiler used for creating the executable,
  - the standard unit time (and optionally other timing information),
  - the tuning parameter settings in the algorithm used,
  - starting points used (if there is a choice in the algorithm), and
  - the stopping rule used.

  In particular, for each particular benchmark run, all problems should be handled by the same set of tuning parameters and stopping rules. If only part of the benchmark is tested, the reasons should be given.

- For each query option and each tested problem, the following information (if available) should be reported:
  - success or failure with respect to
    * timeout limit reached,
    * incorrect solution(s) found, or
    * best known function value not reached;
  - function value reached (for problems with objective function).
    If a better function value than the best known one is found, this should be mentioned and the value and the coordinates of the best point found recorded; the improved values will be used in later versions of the benchmark.
  - solution(s) found
  - time needed for search
  - accuracy reached at timeout (for first solution)
  - number of nodes generated in the search
  - size of remaining search space at timeout. This size is given by the residual dimension d, the residual size s, the interval hull, and the 10-logarithm of the d-volume of the interval hull of the remaining search space. (d is defined as the residual dimension of the hull of the remaining search space, and s is defined as the 10-logarithm of the sum of d-volumes of the boxes remaining.)

  (The residual dimension of a box is the number d of component intervals of positive width. The d-volume is the product of these widths.)

  The following global statistics should be reported for each query option and each complexity class:

  - number of problems correctly solved (best known function value reached or improved/correct solution(s) identified)
  - number of timeout failures (incomplete search space/first solution not found/best known function not reached at timeout)
  - number of problems incorrectly solved (for problems without objective functions)
  - mean of all residual dimensions

– mean of all residual sizes.

- For problems without objective functions and with non-isolated solutions, it makes no sense to individually enumerate all (uncountably many) solutions; instead the following alternative output should be computed:

  – inner and outer approximations of the solution sets as lists of boxes.

  In this case, the quality of the approximations will be reported by the volumes of the inner and the outer approximations, their quotient (if an inner approximation has nonzero volume), and by the space requirements needed for storing all the boxes. (The volume quotient will often be zero, namely if the solution set has measure zero.)

For solvers which do not provide all the output requested, the available subset of the requested output should be provided, and the limitations should be explicitly mentioned.

The benchmarking protocols are fully defined by the preceding, including the reported criteria, except for the timeout limits.

The **benchmarking protocol for assessing work in progress** is designed to be executable on the total benchmarking suite in a limited amount of time. It is implemented by setting the timeout limits to the values given in the table below, in standard time units (stu = between about 1 and 7 minutes on our machines). The timeout limit for each set is set to one tenth of the sum of the individual timeout limits.

Number of variables		$1-9$	$10-99$	$100-999$	≥ 1000	total
timeout(stu)/problem		2	10	20	40	–
Library 1	# problems	84	90	44	48	266
	timeout(stu)/set	16.8	90	88	192	386.8
Library 2	# problems	347	100	93	187	727
	timeout(stu)/set	69.4	100	186	748	1103.4
Library 3	# problems	225	76	22	6	329
	timeout(stu)/set	45.0	76	44	24	189.0
total	# problems	656	266	159	241	1322
	timeout(stu)/set	131.2	266	318	964	1679.2

This gives – for each choice of options – a total running time of about a week on our slowest (1stu=431s), and of about 23 hours on our fastest (1stu=53s) computer. Clearly, one can expect to solve only a limited number of problems within these tight time limits, especially at larger dimensions.

The **benchmarking protocol for assessing a release of a code** is implemented by running the benchmarking suite twice, setting the timeout limits first to the values in the table, and then to these values multiplied by a factor of 10.

Note: This is Version A of the benchmarking protocol. Later versions of the protocol may contain minor or major changes to the above setting, reflecting experience gained with using the protocol on various solvers and platforms.

4 Testing the performance of current solvers

The solvers we started to test are MINOS [10] (a local solver, for comparison only), LGO [12], BARON [13], Numerica [15], and GlobSol [9]. Further solvers we hope to test if time permits are four recent new developments Premium Interval, LINDO global, αBB, GloptiPoly, and OptQuest.

The tables below 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 ++.

Solver	Minos	LGO	BARON	Numerica	GlobSol
Access language	GAMS	C	GAMS	Numerica	Fortran90
Integer constraints	−	+	+	−	−
search bounds	−	required	recommended	−	required
black box eval.	+	+	−	−	−
complete	−	(−)	+	+	+
rigorous	−	−	−	+	+
local	++	+	+	(+)	(+)
CP	−	−	+	++	+
other interval	−	−	−	+	++
convex	−	−	++	−	−
dual	+	−	+	−	−
available	+	+	+	+	+
free	−	−	−	(−)	+

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 the benchmark does not 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). 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.

Solver	Premium Interval	LINDO Global	αBB	GloptiPoly	OptQuest
Access language	Visual Basic	LINGO	MINOPT	Matlab	Visual Basic
Integer constraints	+	+	+	+	+
search bounds	+	?	?	−	+
black box eval.	−	−	−	−	+
complete	+	+	+	+	−
rigorous	(+)	−	−	−	−
local	+	+	+	−	+
CP	+	+	−	−	−
interval	++	+	+	−	−
convex	+	++	++	+	−
dual	−	+	−	++	−
available	+	+	−	+	+
free	−	−	−	+	−

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, or dual (multiplier) techniques are part of the toolkit of the solver.

The final two rows indicate whether the code is available, and whether it is free (in the public domain).

Appendix. The standard unit time

The **standard unit time** is defined as the cpu time needed to carry out the C++ program `shekel5.cpp` available from

`http://www.mat.univie.ac.at/~neum/glopt/coconut/`

compiled without any optimization or debugging option; in particular with

```
g++ -o shekel5.x ./shekel5.cpp
```

on UNIX/LINUX systems and with

```
cl -o shekel5.exe ./shekel5.cpp
```

on MS/DOS systems. The program evaluates the 4-dimensional Shekel5 test function at 10^8 specified points. (Reflecting the increased speed of modern computers, this is a factor of 10^5 larger than the standard unit time used in the first global optimization benchmark by DIXON & SZEGÖ [1] in 1974.)

References

[1] L.C.W. Dixon and G.P. Szegö, The global optimization problem: an introduction, pp. 1–15 in: L.C.W. Dixon and G.P. Szegö (eds.), Towards Global Optimisation 2, North-Holland, Amsterdam 1978.

[2] C.A. Floudas, P.M. Pardalos, C.S. Adjiman, W.R. Esposito, Z.H. Gümüs, S.T. Harding, J.L. Klepeis, C.A. Meyer and C.A. Schweiger, Handbook of Test Problems in Local and Global Optimization, Kluwer, Dordrecht 1999.

[3] 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/`

[4] GAMS/Convert 4.0 User Notes, PDF-document, 2002.
`http://www.gams.com/docs/contributed/convert.pdf`

[5] GAMS World, WWW-document, 2002.
`http://www.gamsworld.org`

[6] GLOBAL Library, WWW-document, 2002.
`http://www.gamsworld.org/global/globallib.htm`

[7] GMS2XX Translator, WWW-document, 2002.
http://www.gamsworld.org/translate.htm

[8] 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

[9] R.B. Kearfott, Rigorous Global Search: Continuous Problems, Kluwer, Dordrecht 1996.
www.mscs.mu.edu/ globsol

[10] 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

[11] H. Mittelmann, Benchmarks. WWW-document, 2002.
http://plato.la.asu.edu/topics/benchm.html

[12] J.D. Pinter, Global Optimization in Action, Kluwer, Dordrecht 1996.
http://www.dal.ca/~jdpinter/l_s_d.html

[13] M. Tawarmalani and N.V. Sahinidis, Convexification and global optimization in continuous mixed-integer nonlinear programming, Kluwer, Dordrecht 2002.
http://archimedes.scs.uiuc.edu/baron/baron.html

[14] B. Vanderbei, Nonlinear Optimization Models, WWW-document.
http://www.orfe.princeton.edu/∼rvdb/ampl/nlmodels/

[15] P. Van Hentenryck, L. Michel and Y. Deville, Numerica. A Modeling Language for Global Optimization, MIT Press, Cambridge, MA 1997.