Optimization and Engineering manuscript No.
(will be inserted by the editor)

The Optimization Test Environment

Ferenc Domes · Martin Fuchs · Hermann Schichl · Arnold Neumaier

Received: date / Accepted: date

Abstract The Optimization Test Environment is an interface to efficiently test different optimization solvers. It is designed as a tool for both developers of solver software and practitioners who just look for the best solver for their specific problem class. It enables users to:

- Choose and compare diverse solver routines;
- Organize and solve large test problem sets;
- Select interactively subsets of test problem sets;
- Perform a statistical analysis of the results, automatically produced as LaTeX, PDF, and JPG output.

The Optimization Test Environment is free to use for research purposes.

Keywords test environment · optimization · solver benchmarking · solver comparison

F. Domes
University of Vienna, Faculty of Mathematics, Nordbergstr. 15, 1090 Wien, Austria
Tel.: +43-1-4277-50666
E-mail: ferenc.domes@univie.ac.at

M. Fuchs
CERFACS, Parallel Algorithms Team, 42 Avenue Gaspard Coriolis, 31057 Toulouse, France
Tel.: +33-5-6119-3045
E-mail: martin.fuchs81@gmail.com
Corresponding author

H. Schichl
University of Vienna, Faculty of Mathematics, Nordbergstr. 15, 1090 Wien, Austria
Tel.: +43-1-4277-50666
E-mail: hermann.schichl@esi.ac.at

A. Neumaier
University of Vienna, Faculty of Mathematics, Nordbergstr. 15, 1090 Wien, Austria
Tel.: +43-1-4277-50661
E-mail: arnold.neumaier@univie.ac.at

1 Introduction

Testing is a crucial part of software development in general, and hence also in optimization. Unfortunately, it is often a time consuming and little exciting activity. This naturally motivated us to increase the efficiency in testing solvers for optimization problems and to automate as much of the procedure as possible.

The procedure typically consists of three basic tasks: organize possibly large test problem sets (also called test libraries); choose solvers and solve selected test problems with selected solvers; analyze, check and compare the results. The Optimization Test Environment is a graphical user interface (GUI) that enables to manage the first two tasks interactively, and the third task automatically.

The Optimization Test Environment is particularly designed for users who seek to

1. adjust solver parameters, or
2. compare solvers on single problems, or
3. compare solvers on suitable test sets.

The first point concerns a situation in which the user wants to improve parameters of a particular solver manually, see, e.g., [9]. The second point is interesting in many real-life applications in which a good solution algorithm for a particular problem is sought, e.g., in [5,16,25] (all for black box problems). The third point targets general benchmarks of solver software. It often requires a selection of subsets of large test problem sets (based on common characteristics, like similar problem size), and afterwards running all available solvers on these subsets with problem class specific default parameters, e.g., timeout. Finally all tested solvers are compared with respect to some performance measure.

In the literature, such comparisons typically exist for **black box** problems only, see, e.g., [26] for global optimization, or the large online collection [24], mainly for local optimization. Since in most real-life applications models are given as black box functions (e.g., the three examples we mentioned in the last paragraph) it is popular to focus comparisons on this problem class. However, the popularity of **modeling languages** like AMPL and GAMS, cf. [3,15,22], that formulate objectives and constraints algebraically, is increasing. Thus first steps are made towards comparisons of global solvers using modeling languages, e.g., on the Gamsworld website [17], which offers test sets and tools for comparing solvers with interface to GAMS.

One main difficulty of solver comparison is to determine a reasonable criterion to **measure the performance** of a solver. Our concept of comparison is to count for each solver the number of global numerical solutions found, and the number of wrong and correct claims for the solutions. Here we consider the term global numerical solution as the best solution found among all solvers. We also produce several more results and enable the creation of performance profiles [6,27].

Further rather technical difficulties come with duplicate test problems, the identification of which is an open task for future versions of the Optimization Test Environment.

A severe showstopper of many current test environments is that it is uncomfortable to use them, i.e., the library and solver management are not very user-friendly, and features like automated LaTeX table creation are missing. Test environments like CUTEr [19] provide an excellent test library, some kind of modeling language

(in this case SIF) with associated interfaces to the solvers to be tested. The unpleasant rest is up to the user. However, our interpretation of the term test environment also requests to analyze and summarize the results **automatically** in a way that it can be used easily as a basis for numerical experiments in scientific publications. A similar approach is used in Libopt [18], available for Unix/Linux, but not restricted to optimization problems. It provides test library management, library subset selection, solve tasks, all as (more or less user-friendly) console commands only. Also it is able to produce performance profiles from the results automatically. The main drawback is the limited amount of supported solvers.

Our approach to developing the Optimization Test Environment is inspired by the experience made during the comparisons reported in [30], in which the Coconut Environment benchmark [34] is run on several different solvers. The goal is to create an easy-to-use library and solver management tool, with an intuitive GUI, and an easy, multi-platform installation. Hence the core part of the Optimization Test Environment is **interactive**. We have dedicated particular effort to the interactive library subset selection, determined by criteria such as a minimum number of constraints, or a maximum number of integer variables or similar. Also the solver selection is done interactively.

The modular part of the Optimization Test Environment is mainly designed as **scripts** without having fixed a scripting language, so it is possible to use Perl, Python, etc. according to the preference of the user. The scripts are interfaces from the Optimization Test Environment to solvers. They have a simple structure as their task is simply to call a solve command for selected solvers, or simplify the solver output to a unified format for the Optimization Test Environment. A collection of already existing scripts for several solvers, including setup instructions, is available on the Optimization Test Environment website [10]. We explicitly **encourage** people who have implemented a solve script or analyze script for the Optimization Test Environment to send it to the authors who will add it to the website. By the use of scripts the modular part becomes very flexible. For many users default scripts are convenient, but just a few modifications in a script allow for non-default adjustment of solver parameters without the need to manipulate code of the Optimization Test Environment. This may significantly improve the performance of a solver.

As **problem representation** we use Directed Acyclic Graphs (DAGs) from the Coconut Environment [20]. We have decided to choose this format as the Coconut Environment already contains automatic conversion tools from many modeling languages to DAGs and vice versa. The Optimization Test Environment is thus independent from any choice of a modeling language. Nevertheless benchmark problem collections, e.g., given in AMPL such as COPS [7], can be easily converted to DAGs by our conversion tools. The other direction would be, e.g., a conversion from a test set consisting merely of Coconut Environment DAGs to AMPL problems and solving them via the previously mentioned solve scripts on the NEOS server [29].

The analyzer of the COPS test set allows for solution checks and iterative refinement of solver tolerances, cf. [8]. The DAG format enables us to go in the same direction as we are also internally performing a check of the solutions. With the present DAG format, the Optimization Test Environment currently excludes test problems that are created in a black box fashion.

The summarizing part of the Optimization Test Environment is managing **automated tasks** which have to be performed manually in many former test environments. These tasks include the automatic check of solutions mentioned, and the generation of LaTeX tables that can be copied and pasted easily in numerical result sections of scientific publications. As mentioned we test especially whether global numerical solutions are obtained and correctly claimed. The results of the Optimization Test Environment also allow for the automated creation of performance profiles.

This paper is organized as follows. In Section 2 we give an overview of our notation for optimization problem formulations. Section 3 presents the functionality of the Optimization Test Environment. Finally we demonstrate the capabilities of the Optimization Test Environment with numerical tests in Section 4.

The last section includes a benchmark of eight solvers for constrained global optimization and constraint satisfaction problems using three libraries with more than 1000 problems in up to about 20000 variables, arising from the Coconut Environment benchmark [34]. The test libraries and the results are also available online on the Optimization Test Environment website [10]. This paper focuses on the presentation of the Optimization Test Environment rather than on the benchmark. However, we intend to collect benchmark results from the Optimization Test Environment on our website, towards a complete comparison of solvers.

The tested solvers in alphabetical order are: BARON 8.1.5 [31,35] (global solver), Cocos [20] (global), COIN with Ipopt 3.6/Bonmin 1.0 [23] (local solver), CONOPT 3 [12,13] (local), KNITRO 5.1.2 [4] (local), Lindoglobal 6.0 [32] (global), MINOS 5.51 [28] (local), Pathnlp 4.7 [14] (local). Counting the number of global optimal solutions found among all solvers the best solver for global optimization is currently Baron. Among the local solvers Coin (Ipopt/Bonmin) performed best. Lindoglobal had the most correctly claimed global numerical solutions, however, it made also the most mistakes claiming a global numerical solution. More details can be found in Section 4.

2 Formulating optimization problems

We consider optimization problems that can be formulated as follows:

$$\begin{aligned} \min\ & f(x) \\ \text{s.t.}\ & x \in \mathbf{x}, \\ & F(x) \in \mathbf{F}, \\ & x_i \in \mathbb{Z} \text{ for } i \in I, \end{aligned} \tag{1}$$

where $\mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\}$ is a **box** in $\mathbb{R}^n$, $f : \mathbf{x} \to \mathbb{R}$ is the **objective function**, $F : \mathbf{x} \to \mathbb{R}^m$ is a vector of **constraint functions** $F_1(x), \ldots, F_m(x)$, $\mathbf{F}$ is a box in $\mathbb{R}^m$ specifying the **constraints** on $F(x)$, $I \subseteq \{1, \ldots, n\}$ is the index set defining the **integer components** of x. Inequalities between vectors are interpreted componentwise.

Since $\underline{x} \in (\mathbb{R} \cup \{-\infty\})^n$, $\overline{x} \in (\mathbb{R} \cup \{\infty\})^n$, the definition of a box includes two-sided bounds, one-sided bounds, and unbounded variables.

An optimization problem is called **bound constrained** if $m = 0$ and is called **unconstrained** if in addition to this $\mathbf{x} = \mathbb{R}^n$.

There are several different classes of optimization problems that are special cases of (1), differentiated by the properties of f, F, and I. If $f = \text{const}$, problem (1) is a so-called **constraint satisfaction problem** (CSP). If f and F are linear and $I = \emptyset$ then we have a **linear programming** problem (LP). If f or one component of F is nonlinear and $I = \emptyset$ we have a **nonlinear programming** problem (NLP). If I is not empty and $I \neq \{1, \ldots, n\}$ one speaks of **mixed-integer programming** (MIP), MILP in the linear case, and MINLP in the nonlinear case. If $I = \{1, \ldots, n\}$ we deal with **integer programming**.

A **solution** of (1) is a point $\widehat{x} \in C := \{x \in \mathbf{x} \mid x_I \in \mathbb{Z}, F(x) \in \mathbf{F}\}$ with

$$f(\widehat{x}) = \min_{x \in C} f(x),$$

i.e., the solutions are the **global** minimizers of f over the **feasible domain** C. A **local** minimizer satisfies $f(\widehat{x}) \leq f(x)$ only for all $x \in C$ in a neighborhood of $\widehat{x}$. The problem (1) is called **infeasible** if $C = \emptyset$.

A **solver** is a program that seeks an approximate global, local, or feasible solution of an optimization problem. **Global solvers** are designed to find global minimizers, **local solvers** focus on finding local minimizers, **rigorous solvers** guarantee to include all global minimizers in their output set even in the presence of rounding errors. In the Optimization Test Environment we declare the result x_s, f_s of a solver a **numerically feasible solution** if the solver claims it to be feasible and we find that the result has a sufficiently small **feasibility distance**

$$\mathrm{d}_{\mathrm{feas,p}}(x_s, f_s) \leq \alpha \tag{2}$$

for a small **tolerance** level α. As a default value for α we use 0. Intuitively the distance to feasibility $d : \mathbb{R}^n \to \mathbb{R}$ of a point x could be defined as $d(x) = \min_{y \in C} \|x - y\|_p$, i.e., the minimum distance of x from the feasible domain C in the p-norm. However, this definition would not be appropriate for a computational check of feasibility, since it imposes a further optimization problem.

Instead we introduce a componentwise violation of the constraints v and infer a feasibility distance from v. To reduce the sensitivity of the feasibility distance to scaling issues we first define the intervals

$$\mathbf{u} = [-\varepsilon \max(\|x_s\|_\infty, \kappa), \varepsilon \max(\|x_s\|_\infty, \kappa)], \tag{3}$$

$$\mathbf{x}_s = x_s + \mathbf{u}, \tag{4}$$

with the parameters ε (set to 10^{-6} by default), and κ (set to 1 by default). The interval $\mathbf{u}$ has the minimal width 2ε and becomes wider if $\|x_s\|_\infty$ becomes larger. Note that this is the most tolerant way to compensate the loss of significant digits during the computational process. On the one hand using relative errors for small $\|x_s\|_\infty$ makes no sense, on the other hand for large $\|x_s\|_\infty$ the use of absolute errors would be too restrictive to get a feasible solution.

Then we compute the **objective violation** $v_o(x_s, f_s)$ as follows. We add to $f(x_s)$ an interval with a width that grows with the width of $\mathbf{u}$ and with the range of the gradient f' in the interval $\mathbf{x}_s$, cf. Figure 1. If f_s is contained in this interval, with respect to the tolerance α in (2), the objective violation is deemed reasonably small. Thus we consider both a badly scaled x_s and f. We get

$$v_o(x_s, f_s) := \langle f(x_s) + f'(\mathbf{x}_s)\mathbf{u} - f_s \rangle, \tag{5}$$

where all operations are in interval arithmetics, and $\langle \mathbf{x} \rangle$ denotes the **mignitude** of the interval $\mathbf{x}$, i.e.,

$$\langle \mathbf{x} \rangle := \begin{cases} \min(|\underline{x}|, |\overline{x}|) & \text{if } 0 \notin [\underline{x}, \overline{x}], \\ 0 & \text{otherwise }, \end{cases} \tag{6}$$

i.e., the smallest absolute value within the interval $[\underline{x}, \overline{x}]$. If $\mathbf{x}$ is multidimensional the mignitude operates componentwise. Similar to the objective violation we get the **constraint violations**

$$v_c(x_s, f_s) := \langle F(x_s) + F'(\mathbf{x}_s)\mathbf{u} - \mathbf{F} \rangle. \tag{7}$$

Finally we define the **box constraint violations** by

$$v_b(x_s, f_s) := \langle \mathbf{x}_s - \mathbf{x} \rangle, \tag{8}$$

and the complete componentwise violation is given by

$$v(x_s, f_s) = (v_o(x_s, f_s), v_c(x_s, f_s), v_b(x_s, f_s)). \tag{9}$$

We define the **feasibility distance** $\mathrm{d}_{\mathrm{feas,p}} : \mathbb{R}^{n+1} \to \mathbb{R}$ by

$$\mathrm{d}_{\mathrm{feas,p}}(x_s, f_s) := \|v(x_s, f_s)\|_p. \tag{10}$$

Since $\mathrm{d}_{\mathrm{feas},\infty} \leq \mathrm{d}_{\mathrm{feas,p}}$ for all p with $1 \leq p \leq \infty$ we decided to choose $p = \infty$ in the Optimization Test Environment to check for feasibility via (2). In the remainder of the paper the feasibility check is also referred to as the **solution check**.

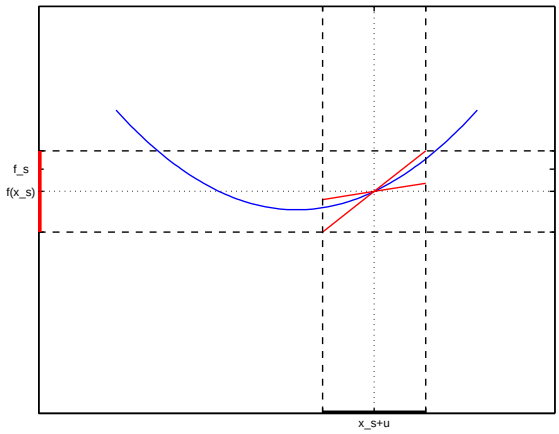

Fig. 1 Illustration of (5). If f_s is contained in the interval marked in red it passes the objective violation check.

Let J_{feas} be the set of all solver results that have passed the solution check, i.e., $J_{\mathrm{feas}} = J_{\mathrm{feas}\,x} \times J_{\mathrm{feas}\,f}$, $J_{\mathrm{feas}\,x} = \{x_1, \ldots, x_N\}$, $J_{\mathrm{feas}\,f} = \{f_1, \ldots, f_N\}$, $\mathrm{d}_{\mathrm{feas,p}}(x_s, f_s) \leq \alpha$ for all $(x_s, f_s) \in J_{\mathrm{feas}}$, and let $J_{\mathrm{inf}} = J_{\mathrm{inf}\,x} \times J_{\mathrm{inf}\,f}$ be the set of all solver results that did not pass the solution check. We define a **global numerical solution** $(x_{\mathrm{opt}}, f_{\mathrm{opt}})$ as the best numerically feasible solution found by all solvers used, i.e.,

$(x_{\text{opt}}, f_{\text{opt}}) \in J_{\text{feas}}$ and $f_{\text{opt}} \leq f_j$ for all $f_j \in J_{\text{feas}\,f}$. Another feasible solution $(\widetilde{x}, \widetilde{f}) \in J_{\text{feas}}$ is also considered global if $\widetilde{f}$ is sufficiently close to f_{opt}, i.e.,

$$\widetilde{f} \leq \begin{cases} f_{\text{opt}} + \beta & \text{if } |f_{\text{opt}}| \leq \kappa, \\ f_{\text{opt}} + \beta\,|f_{\text{opt}}| & \text{otherwise}\ , \end{cases} \tag{11}$$

with κ as above and with the tolerance β which is set to 10^{-6} by default. One can consider κ as a threshold between the use of absolute and relative error. For the special case of $\kappa = 1$ the condition reads

$$\widetilde{f} \leq f_{\text{opt}} + \beta \max(|f_{\text{opt}}|\,, 1). \tag{12}$$

We define a **local numerical solution** as a feasible, non-global solver result.

We define the **best point** found as

$$(x_{\text{best}}, f_{\text{best}}) = \begin{cases} (x_{\text{opt}}, f_{\text{opt}}) & \text{if } J_{\text{feas}} \neq \emptyset, \\ \arg\min_{(x,f)\in J_{\text{inf}}} \mathrm{d}_{\text{feas,p}}(x, f) & \text{if } J_{\text{feas}} = \emptyset, \end{cases} \tag{13}$$

i.e., the global numerical solution if a feasible solution has been found, otherwise the solver result with the minimal feasibility distance.

The best points of each problem of a test problem set are contained in the so-called **Best solvers** list.

To assess the location of the global numerical solution we distinguish between hard and easy locations. The best point is considered as a **hard location** if it could not be found by a particular user-defined local solver, otherwise it is considered to be an **easy location**.

The user who wishes to solve an optimization problem should become familiar with one of the several existing **modeling languages**. A modeling language is an interface between a solver software and the (user-provided) formal description of an optimization problem in the fashion of (1). Prominent successful modeling languages are AMPL [15] and GAMS [3], but there are many more such as AIMMS, LINGO, LPL, MPL, see [22].

The Optimization Test Environment provides an easy interface to set up arbitrary modeling languages and solvers to manage and solve optimization problems.

3 Functionality

This section presents the functionality of the Optimization Test Environment illustrated by a small test set example. For more details the interested reader is referred to the Optimization Test Environment user's manual [11].

3.1 Installation

The installation of the Optimization Test Environment is straightforward: Download the Optimization Test Environment available on-line at [10]. Install it by running the installer in Windows and following the instructions, or by unzipping

the `tar.gz` file in Linux. Afterwards one can start the Optimization Test Environment at the unzip location via `java -jar TestEnvironment.jar`. The graphical user interface (GUI) of the Optimization Test Environment is programmed in Java, hence Java JRE 6, Update 13 or later is required to be installed. This is the only prerequisite needed. Figure 2 shows the current GUI appearance on a Windows machine.

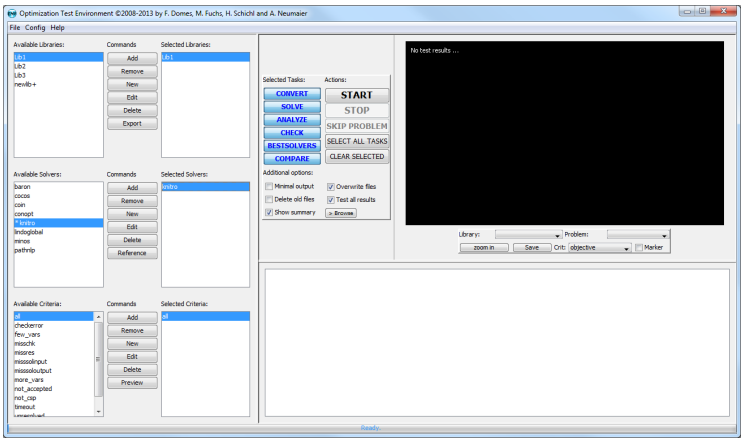

Fig. 2 GUI appearance on a Windows computer.

The Optimization Test Environment does not include any solver software. Installation of a solver and obtaining a valid license is independent of the Optimization Test Environment and up to the user.

3.2 Adding a new test library

After starting the Optimization Test Environment, the first step is to add a set of optimization problems, what we call a **test library**. The problems are assumed to be given as `.dag` files, an input format originating from the Coconut Environment [20]. In case one does not have the problems given as `.dag` files, but as AMPL code, one needs to convert the AMPL model to `.dag` files first which is possible via the Coconut Environment or easily via a converter script separately available on the Optimization Test Environment website [10]. Also one can find a huge collection of test problem sets from the Coconut Benchmark [33] given as `.dag` files on the Optimization Test Environment website.

We use the following three problems as an illustrative example 'newlib' of a test library.

3.2.1 Simple example test library 'newlib'

Problem t1: The intersection of two unit circles around $x = (\pm 0.5, 0)^T$, cf. Figure 3. The problem formulation is as follows:

$$\begin{aligned} \min_x \ & x_2 \\ \text{s.t.} \ & (x_1 - 0.5)^2 + x_2^2 = 1, \\ & (x_1 + 0.5)^2 + x_2^2 = 1, \\ & x_1 \in [-3, 3], x_2 \in [-3, 3]. \end{aligned} \tag{14}$$

The feasible points of (14) are $x = (0, \pm\sqrt{3}/2)^T$. Minimizing x_2 results in the optimal solution $\widehat{x} = (0, -\sqrt{3}/2)^T \approx (0, -0.8660)^T$.

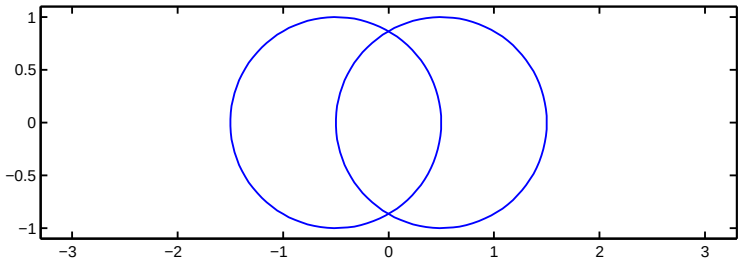

Fig. 3 The feasible domain of (14) consists of the intersection points of two unit circles around $x = (\pm 0.5, 0)^T$.

Problem t2: Two touching unit circles around $x = (\pm 1, 0)^T$, cf. Figure 4. The problem formulation is as follows:

$$\begin{aligned} \min_x \ & 1 \\ \text{s.t.} \ & (x_1 - 1)^2 + x_2^2 = 1, \\ & (x_1 + 1)^2 + x_2^2 = 1, \\ & x_1 \in [-3, 3], x_2 \in [-3, 3]. \end{aligned} \tag{15}$$

The only feasible point of (15) is $x = (0, 0)^T$.

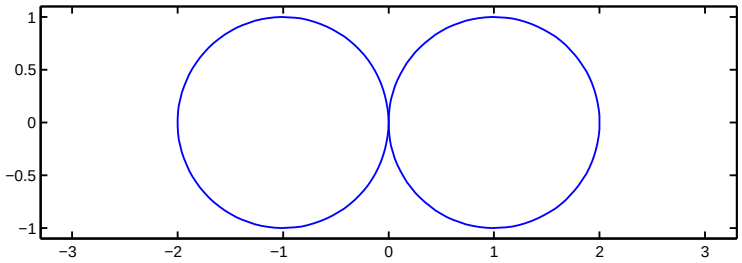

Fig. 4 The only feasible point of (15) is at $x = (0, 0)^T$.

Problem t3: Two disjoint unit circles around $x = (\pm 2, 0)^T$, cf. Figure 5. The problem formulation is as follows:

$$\begin{aligned} \min_x \ & 1 \\ \text{s.t.} \ & (x_1-2)^2+x_2^2=1, \\ & (x_1+2)^2+x_2^2=1, \\ & x_1 \in [-3,3], x_2 \in [-3,3]. \end{aligned} \tag{16}$$

There is no feasible solution for (16).

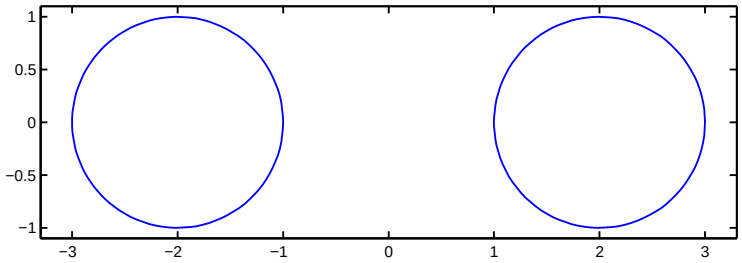

Fig. 5 Obviously there is no feasible point for (16).

In our examples we actually know the results of the three problems. In this case the user can optionally provide a reference solution.

3.3 Reference solutions and .res files

A reference solution is a known solution for one of the test problems in our test problem library. It can be provided simply as a file with extension `.res`. The generic format of `.res` files within the Optimization Test Environment for our example problem `t1.dag` reads as follows:

```
modelstatus = 0
x(1) = 0
x(2) = -0.8660254037844386
obj = -0.8660254037844386
infeas = 0.00
nonopt = 0.00
```

There are several fields that can be entered in `.res` files. Table 1 gives a survey of all possible entries.

Possible values for the model status in the `.res` file are shown in Table 2. The variables `x(1)`, ..., `x(n)` correspond to the variables in the problem (`.dag`) file, enumerated in the same sequence, starting with index 1. Note that in case of rigorous solvers $\widehat{x}$ and $f(\widehat{x})$ can also be interval enclosures of the solution, e.g., `x(1) = [-1e-8,1e-8]`. In the current Optimization Test Environment version we focus on comparing non-rigorous solvers. Thus the full functionality of solvers providing verified interval enclosures of solutions cannot be assessed yet (e.g., by the size of the enclosures). If only an interval solution is given by a rigorous solver we use the upper bound of `obj` for our comparisons. Since this could be

Table 1 Entries in `.res` files.

`modelstatus`	solver model status, see Table 2
`x(i)`	solver output for $\widehat{x}_i$, $i = 1, \ldots, n$
`obj`	solver output for $f(\widehat{x})$
`infeas`	feasibility distance provided by the solver
`nonopt`	0 if $\widehat{x}$ claimed to be at least locally optimal, 1 otherwise
`time`	used CPU time to solve the problem
`splits`	number of splits made, e.g., in branching algorithms

disadvantageous for rigorous solvers it is recommended to provide points for $\widehat{x}$ and $f(\widehat{x})$, and provide interval information separately using the additional fields

```
xi(1) = [<value>,<value>]
xi(2) = [<value>,<value>]
...
xi(n) = [<value>,<value>]
obji = [<value>,<value>]
```

in the `.res` file, describing the interval hull of the feasible domain as well as an interval enclosure of the objective function. In future versions of the Optimization Test Environment we intend to use this information to compare rigorous solvers.

Table 2 Modelstatus values.

0	Global numerical solution found
1	Local solution found
2	Unresolved problem
3	The problem was not accepted
-1	Timeout, local solution found
-2	Timeout, unresolved problem
-3	Problem has been found infeasible

If we compare a given reference solution – which could possibly be wrong – with further solutions from different solvers, we treat the reference solution like all the solver solutions (x_s, f_s), cf. Section 2. This concerns in particular the construction of J_{feas}, x_{opt}, and x_{best}.

3.4 Adding a solver

We solve our example test library using a KNITRO Student Edition with AMPL interface in its demo version (both are freely available on the internet). We use the predefined solver configuration for KNITRO from the Optimization Test Environment website [10] which provides the solver configurations and installation guides for many well-known solvers. We only need to modify the path names for the AMPL and KNITRO path in the solver configuration window.

The solver interfaces of the Optimization Test Environment are designed as a combination of the GUI and **scripts** without having fixed a scripting language, so it is possible to use Perl, Python, etc. according to the preference of the user. The scripts have a simple structure and contain a solve command and the algorithmic parameters for the solver concerned. To run the same solver with different parameter sets the solver should be added multiple times with different script configurations.

Remark 1 Some solvers tend to be imprecise in the interpretation of their timeout parameter. Also the precision claimed by a solver may differ from the actual precision. In contrast to COPS [8] we do not adjust the solver tolerances adaptively. If a solver claims a given precision, we choose ε in our solution check accordingly.

After calling the solver, an analyze-script is used to generate `.res` files from the solver output. In order to perform a statistical analysis of the results complete `.res` files **must** be produced even if no feasible solution has been found or a timeout has been reached or similar.

Writing the solve and analyze scripts for solvers that are not available on the Optimization Test Environment website requires basic knowledge about scripting string manipulations which is an easy task for advanced users and especially for solver developers.

Note that the original solver output is also kept and stored. Moreover, some solvers can be called setting a command line flag in order to generate `.res` files directly without the need of an analyze script.

We explicitly **encourage** people who have implemented a solve or analyze script for the Optimization Test Environment to send it to the authors, who will add it to the Optimization Test Environment website.

3.5 Choosing criteria

The selection of criteria to filter test problems is another core feature of the Optimization Test Environment. The criteria are used to specify subsets of test libraries. There are many possibilities to specify the selection of test problems from the libraries by way of connected logical expressions, e.g., `[variables<=10] and [int-vars<=5]` restricts the selected test libraries to those problems with $n \leq 10$, $|I| \leq 5$. A criterion is defined by such a connection of logical expressions. The types of logical expressions that can be adjusted are shown in Table 3. The creation of a criterion in the Optimization Test Environment GUI is straightforward using the 'Condition' box together with the 'Logicals' box of the 'Criteria editor'.

A click on the **Preview** button in the main Optimization Test Environment window shows all selected criteria in correspondence to all selected problems that meet the criteria.

One important feature of criteria is their use in setting **timeouts** in connection with further conditions like problem size. For example, to set up the timeout as in Table 4, we create 4 criteria:

size1: `[variables>=1]and[variables<=9]and[timeout==180]`
size2: `[variables>=10]and[variables<=99]and[timeout==900]`
size3: `[variables>=100]and[variables<=999]and[timeout==1800]`
size4: `[variables>=1000]and[timeout==1800]`

Table 3 Types of logical expressions.

`problem name`	string of the problem name
`binary-vars`	number of binary variables
`edges`	number of edges in the DAG
`nodes`	number of nodes in the DAG
`objective`	boolean statement
`int-vars`	number of integer variables
`constraints`	number of constraints
`variables`	number of variables
`solution status`	the modelstatus, cf. Table 2
`check error`	$d_{feas,p}$ in the solution check
`solver input file missing`	boolean statement
`solver output file missing`	boolean statement
`res file missing`	boolean statement
`chk file missing`	boolean statement
`timeout`	maximum allowed CPU time

and add them to the selected criteria. Any task on the selected test library will now be performed first on problems with $1 \leq n \leq 9$ and timeout 180 s, then on problems with $10 \leq n \leq 99$ and timeout 900 s, then problems with $100 \leq n \leq 999$ and timeout 1800 s, and eventually problems with $n \geq 1000$ and timeout 1800 s.

Table 4 Benchmark timeout settings depending on problem size.

size	n	**timeout**
1	1-9	3 min
2	10-99	15 min
3	100-999	30 min
4	≥ 1000	30 min

The Optimization Test Environment already includes a collection of predefined criteria in the 'Available criteria' box, such as, e.g., `few_vars` to choose problems with only a few variables setting a timeout of 2 minutes, or `misssoloutput` to choose problems for which a solver output file is missing.

3.6 Solve the test problems

Finally selected problems can be solved by selected solvers, the results are analyzed by the Optimization Test Environment, the output files (in LaTeX, pdf, and jpg) are written, and a pdf file shows up that summarizes the results.

There are essentially three different kinds of generated output pdfs. The `problems.pdf` analyzes the performance of the selected solvers on each problem from the selected test libraries. The `solvers.pdf` summarizes the performances of each se-

lected solver according to each selected test library. The **summary** pdfs are similar to `solvers.pdf`, but they additionally provide a solver summary concerning the whole test set that consists of all selected test libraries.

The results for the example test library can be found in Section 4.1.

3.7 Performance profiles

Apart from the result tables we also generate performance profiles that are displayed in the top right corner of the GUI and also stored as jpg files. The plots can be saved separately, and additionally a mat file is generated to facilitate manipulation of the plot.

The concept of performance profiles has been gaining importance in the past decade, cf. [6]. A standard choice of the performance measure is the time. However, since we are rather interested in the global numerical solutions, we pick the objective function value as a performance measure. Profiles comparing objective function values are already discussed, e.g., in [1,2,36]. The fact that objective function values can be negative or zero imposes difficulties in the classical framework of performance profiles. The solutions need to be normalized in some way. We define the performance measure m as follows.

Let N_p be the number of problems in a test library and $(x_{s1}, f_{s1}), \ldots, (x_{sN_p}, f_{sN_p})$ the results of solver s on that library. Let $J_{\mathrm{feas}i}$ be the set containing solutions that have passed the solution check for problem i, with respect to Section 2. Let

$$\underline{f}_i := \min_{(x_{\sigma i}, f_{\sigma i}) \in J_{\mathrm{feas}i}} f_{\sigma i}, \tag{17}$$

$$\overline{f}_i := \max_{(x_{\sigma i}, f_{\sigma i}) \in J_{\mathrm{feas}i}} f_{\sigma i}, \tag{18}$$

$$\mathrm{wid} := \begin{cases} \overline{f}_i - \underline{f}_i & \text{if } \overline{f}_i > \underline{f}_i, \\ 1 & \text{otherwise.} \end{cases} \tag{19}$$

For $i = 1, \ldots, N_p$ we define the performance m of solver s on problem i as

$$m_{si} := \begin{cases} 1 & \text{if } J_{\mathrm{feas}i} = \emptyset \text{ or } (x_{si}, f_{si}) \text{ global w.r.t. (11)}, \\ \frac{f_{si} - \underline{f}_i}{\mathrm{wid}} + 1 & \text{if } (x_{si}, f_{si}) \in J_{\mathrm{feas}i} \text{ and not global}, \\ \mathrm{failfac} + 1 & \text{otherwise}, \end{cases} \tag{20}$$

with the user-defined parameter failfac > 1 which is set to 1.5 by default.

The best performance among all solvers for problem i is given by

$$K_i = \min_s m_{si} = 1. \tag{21}$$

The performance profile of solver s is defined as the empirical cumulative distribution function of the relative performances $r_{si} := \frac{m_{si}}{K_i}$, i.e.,

$$\mathrm{Profile}_s(t) = \frac{1}{N_p} \sum_{i=1}^{N_p} \chi_{\{\tau | r_{si} \leq \tau\}}(t) \tag{22}$$

where $\chi_{\mathcal{A}}(t)$ is the characteristic function of the set $\mathcal{A}$. In our case the ratios r and the performance measures m are identical since $K_i = 1$ for all i.

Figure 6 illustrates an example of a performance profile. The value of $\text{Profile}_s(1)$ shows the fraction of problems for which solver s performed best. The value of $\text{Profile}_s(t)$ for $2 \leq t < \text{failfac} + 1$ is the number problems that could be solved by solver s and is typically interpreted as the robustness of s. One particularity of our profiles are the steps at $t = 2$ that arise from the normalization in (20). The size of these steps corresponds to the fraction of problems for which solver s performed worst without failing.

4 Numerical results

In this section we present two cases of test results produced with the Optimization Test Environment. All LaTeX tables containing the results are automatically generated. The first subsection gives the results for the example of KNITRO on the test library 'newlib', cf. Section 3.2. The second subsection illustrates the strength of the Optimization Test Environment in benchmarking solvers.

In the generated LaTeX tables we use the legend given in Table 5.

Table 5 Legend.

TABLE LEGEND	
General symbols:	
all	the number of problems given to the solver
acc	problems accepted by the solver
wr	number of wrong claims (the sum of F?, G?, I?, L?, see below)
easy	problems classified as easy
hard	problems classified as hard
n	number of variables
m	number of constraints
fbest	best objective function value known
obj	objective function value
Solution status claimed by the solver (stat):	
G	result claimed to be a global solution
L	result claimed to be a local solution
I	the problem claimed to be infeasible
TL	timeout reached but local solution claimed
U	unresolved (no solution found or error message)
X	model not accepted by the solver
Solver independent information:	
F+	feasible solution known
F-	no feasible solution known
Solution validation results:	
G+	result is a global solution
G-	result is not a global solution
F?	wrong (in)feasibility claim, [(F+ and I) or (F+ but feasibility check failed and (G or L or TL))]
G!	correctly claimed global solution, [G and G+]
G?	wrongly claimed global solution, [G and G- and not F?]
L?	wrongly claimed local solution, [L and F-]
I!	correctly claimed infeasibility, [I and F-]
I?	wrongly claimed infeasibility, [I and F+]

4.1 Results for 'newlib'

The AMPL code of (14), (15), and (16), respectively, reads as follows.

t1.mod:
```
var x1 >=-3, <=3;
var x2 >=-3, <=3;

minimize obj: x2;

s.t. c1: (x1-0.5)^2+x2^2=1;
s.t. c2: (x1+0.5)^2+x2^2=1;
```

t2.mod:
```
var x1 >=-3, <=3;
var x2 >=-3, <=3;

minimize obj: 1;

s.t. c1: (x1-1)^2+x2^2=1;
s.t. c2: (x1+1)^2+x2^2=1;
```

t3.mod:
```
var x1 >=-3, <=3;
var x2 >=-3, <=3;

minimize obj: 1;

s.t. c1: (x1-2)^2+x2^2=1;
s.t. c2: (x1+2)^2+x2^2=1;
```

The AMPL code is converted to `.dag` files via the converter script mentioned.

The results for the example test library 'newlib' are shown in Table 6. We see that KNITRO has found correct solutions for the problems t1 and t2 and could not resolve the infeasible problem t3. The Optimization Test Environment treated the two feasible solutions for the problems t1 and t2 as global numerical solutions since no better solution was found by any other solver (KNITRO being the only solver tested on newlib). The problem t3 was treated as infeasible since no feasible solution was found among all solvers used. As KNITRO is a local solver it could not claim any solution as global or infeasible. The global numerical solutions identified by the Optimization Test Environment are classified as easy locations as they were found by the local reference solver, i.e., KNITRO itself.

4.2 Solver benchmark

We have run the benchmark on the libraries Lib1, Lib2, and Lib3 of the Coconut Environment benchmark [34], containing more than 1000 test problems. We have removed some test problems from the 2003 benchmark that had incompatible DAG formats. Thus we have ended up with in total 1286 test problems.

The tested solvers in alphabetical order are: BARON 8.1.5 [31,35] (global solver), Cocos [20] (global), COIN with Ipopt 3.6/Bonmin 1.0 [23] (local solver), CONOPT 3 [12,13] (local), KNITRO 5.1.2 [4] (local), Lindoglobal 6.0 [32] (global), MINOS 5.51 [28] (local), Pathnlp 4.7 [14] (local). For the benchmark we run all solvers on the full set of test problems with default options and fixed timeouts from the Coconut Environment benchmark, cf. Table 4.

Additionally for solvers inside GAMS we used the GAMS options

Table 6 Results for 'newlib'.

knitro on `Newlib`								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
acc	3	2	1	-	2	-	-	-
L	2	2	-	-	2	-	-	-
U	1	-	1	-	-	-	-	-

knitro summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
newlib	3	3	-	2	-	-	-	-	-	-
total	3	3	-	2	-	-	-	-	-	-

name	n	m	fbest	knitro	
				st	tst
t1	2	2	-8.660e-01	L	G+
t2	2	2	CSP	L	G+
t3	2	2	CSP	U	-

```
decimals = 8, limrow = 0, limcol = 0, sysout = on, optca=1e-6, optcr=0
```

and for solvers inside AMPL we used the option

```
presolve 0
```

The parameters inside the Optimization Test Environment described in Section 2 have been fixed as follows: $\alpha = 0$, $\varepsilon = 10^{-4}$, $\beta = 10^{-6}$, $\kappa = 1$. As default local solver we have chosen KNITRO to distinguish between easy and hard locations. Hence in case of KNITRO the Optimization Test Environment status G- never occurs for easy locations and G+ never occurs for hard locations by definition.

We have run the benchmark on an Intel Core 2 Duo 3 GHz machine with 4 GB of RAM. We should keep in mind that running the benchmark on faster computers will probably improve the results of all solvers but the relative improvement may vary between different solvers (cf., e.g., [21]).

The results are shown in Tables 7 to 12, automatically generated by the Optimization Test Environment. The performance of every solver on Lib1 is given in Table 7 and Table 8. Table 9 and Table 10 summarize the performance of every solver on all the test libraries. Moreover, we automatically plot a performance profile comparing all solvers on all test problems, cf. Figure 6.

Cocos and KNITRO accepted (almost) all test problems. Also the other solvers accepted the majority of the problems. Minos accepted the smallest number of problems, i.e., 81% of the problems. A typical reason why some solvers reject a problem is that the constraints of the objective function could not be evaluated at the starting point $x = 0$ because of the occurrence of expressions like $1/x$ or $\log(x)$. Some solvers like Baron also reject problems in which sin or cos occur in any expression.

The difference between global and local solvers is clearly visible in the reliability of claiming global numerical solutions. Looking at the tables 7 and 8 the global solvers are obviously superior in this respect, especially on hard locations.

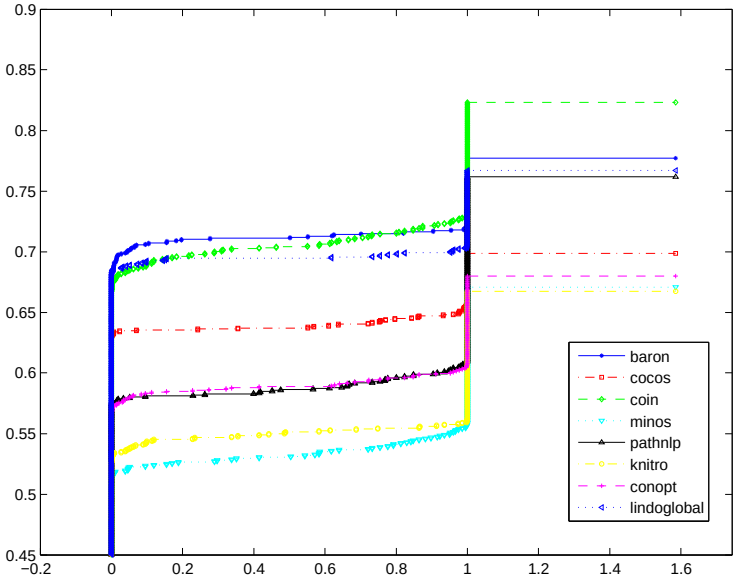

Fig. 6 Performance profile comparing all solvers on all test problems (log2 scaled).

Table 11 provides the reliability statistics of the solvers, showing the ratio of global numerical solutions found over the number of accepted feasible problems, the ratio of correctly claimed global numerical solutions over the number of global numerical solutions found, and the ratio of wrong solutions found over the number of accepted problems.

Lindoglobal has the best score (79%) in the number of correctly claimed global numerical solutions among the global numerical solutions found. Cocos is second with 76%, and Baron is third with 69%. But it should be remarked that Lindoglobal made 14% wrong solution claims as opposed to Baron with 8%. Not surprisingly, the local solvers had only very bad scores in claiming global numerical solutions, since they are not global solvers. On the other hand, they had a low percentage of wrong solutions, between 3% and 8% (except for KNITRO). The local solvers did not have zero score in claiming global numerical solutions since for some LP problems they are able to claim globality of the solution.

We also give an example of a detailed survey of problems in Table 12. It shows the solver status and Optimization Test Environment status for each solver on some problems of size 1 from Lib1. One can see, e.g., that for the first problem 'chance' all solvers have found the global numerical solution except for Cocos and Pathnlp which did not resolve the problem. BARON and Lindoglobal correctly claimed the global numerical solution. We see how easily the Optimization Test Environment can be used to compare results on small or big problem sizes. We could also study comparisons, e.g., only among MINLPs by using the selection of criteria, cf. Section 3.5.

Baron has found the most global numerical solutions among all accepted feasible problems. The local solver Coin (Ipopt/Bonmin) also performed very well in this respect, almost at the same level as the global solver Lindoglobal. Hence Coin

Table 7 Performance of each solver on Lib1.

baron on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	36	-	-	-	-	-	-	-
acc	227	223	4	33	57	25	89	52
G	103	102	1	19	38	16	44	4
L	103	103	-	12	19	6	45	33
I	3	2	1	2	-	-	-	2
U	18	16	2	-	-	3	-	13

cocos on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
acc	263	252	11	27	67	22	72	91
G	149	145	4	22	52	5	53	35
I	6	4	2	3	-	2	-	2
TL	32	32	-	2	10	-	19	3
U	76	71	5	-	5	15	-	51

coin on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	25	-	-	-	-	-	-	-
acc	238	234	4	5	68	17	70	79
G	3	3	-	1	2	-	-	1
L	214	214	-	4	66	17	70	61
U	21	17	4	-	-	-	-	17

conopt on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	25	-	-	-	-	-	-	-
acc	238	234	4	17	67	18	50	99
G	3	3	-	1	2	-	-	1
L	191	191	-	16	65	11	50	65
U	44	40	4	-	-	7	-	33

(Ipopt/Bonmin) would be a strong local reference solver providing a tougher definition of hard locations. The other solvers are not far behind, except for KNITRO with 49%. However, it should be noted that for license reasons we used the quite old KNITRO version 5.1.2 (this may also explain the high number of wrong solutions which were often quite close to a correct solution – to avoid this an iterative refinement of solver tolerances as in COPS is planned, also see Remark 1).

Table 8 Performance of each solver on LIB1 ctd.

knitro on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	4	-	-	-	-	-	-	-
acc	259	248	11	43	89	-	-	159
L	213	211	2	43	87	-	-	124
TU	13	12	1	-	-	-	-	12
U	33	25	8	-	2	-	-	23

lindoglobal on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	25	-	-	-	-	-	-	-
acc	238	234	4	26	67	18	80	69
G	113	113	-	8	49	6	45	13
L	62	62	-	4	14	1	33	14
I	18	17	1	14	-	5	-	12
U	45	42	3	-	4	6	2	30

minos on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	29	-	-	-	-	-	-	-
acc	234	230	4	2	63	18	44	105
G	3	3	-	1	2	-	-	1
L	180	180	-	1	61	12	43	64
U	51	47	4	-	-	6	1	40

pathnlp on Lib1								
stat	all	F+	F-	F?	easy		hard	
					G+	G-	G+	G-
X	30	-	-	-	-	-	-	-
acc	233	229	4	2	61	19	54	95
G	3	3	-	1	2	-	-	1
L	182	182	-	1	59	16	53	54
U	48	44	4	-	-	3	1	40

New results with updated versions are continuously uploaded to the OPTIMIZATION TEST ENVIRONMENT website [10].

Acknowledgements Partial funding of the project is gratefully appreciated: Ferenc Domes was supported through the research grant FS 506/003 of the University of Vienna. Hermann Schichl was supported through the research grant P18704-N13 of the Austrian Science Foundation (FWF).

Table 9 Performance summary of every solver on the test libraries.

baron summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	227	37	146	82	1	33	2	-	2
Lib2	715	635	43	414	232	2	23	16	1	3
Lib3	306	273	11	252	252	5	8	-	-	3
total	1284	1135	91	812	566	8	64	18	1	8

cocos summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	263	53	139	105	2	27	22	-	4
Lib2	715	715	83	336	228	2	31	45	-	7
Lib3	306	306	28	272	238	3	22	1	-	5
total	1284	1284	164	747	571	7	80	68	-	16

coin summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	238	5	138	2	-	5	-	-	-
Lib2	715	674	46	439	20	-	35	5	6	-
Lib3	306	297	3	214	3	-	2	-	1	-
total	1284	1209	54	791	25	-	42	5	7	-

conopt summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	238	17	117	2	-	17	-	-	-
Lib2	715	678	86	377	21	-	77	4	5	-
Lib3	306	290	2	173	4	-	1	-	1	-
total	1284	1206	105	667	27	-	95	4	6	-

Furthermore, we would like to acknowledge the help of Oleg Shcherbina in several solver and test library issues. We thank Nick Sahinidis, Alexander Meeraus, and Michael Bussieck for the support with several solver licenses. Thanks to Mihaly Markot who has resolved several issues with Cocos, and to Yahia Lebbah for his comments.

References

1. M. Ali, C. Khompatraporn, and Z. Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, 31(4):635–672, 2005.
2. C. Audet, C. Dang, and D. Orban. *Software Automatic Tuning, From Concepts to State-of-the-Art Results*, chapter Algorithmic Parameter Optimization of the DFO Method with the OPAL Framework. Springer, 2010. In press, preprint available on-line at: `http://www.gerad.ca/~orban/papers.html`.
3. A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, 1988.
4. R. Byrd, J. Nocedal, and R. Waltz. *Large-Scale Nonlinear Optimization*, chapter KNITRO: An Integrated Package for Nonlinear Optimization, pages 35–59. Springer, 2006.

Table 10 Performance summary of every solver on the test libraries ctd.

knitro summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	259	45	89	-	-	43	-	2	-
Lib2	715	703	179	337	-	-	171	-	8	-
Lib3	306	305	44	177	-	-	42	-	2	-
total	1284	1267	268	603	-	-	256	-	12	-

lindoglobal summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	238	54	147	94	1	26	11	-	17
Lib2	715	686	118	386	274	4	55	43	1	19
Lib3	306	297	11	272	272	4	7	1	-	3
total	1284	1221	183	805	640	9	88	55	1	39

minos summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	234	2	107	2	-	2	-	-	-
Lib2	715	510	29	298	22	-	21	2	6	-
Lib3	306	296	1	194	4	-	1	-	-	-
total	1284	1040	32	599	28	-	24	2	6	-

pathnlp summary statistics										
library	all	acc	wr	G+	G!	I!	F?	G?	L?	I?
Lib1	263	233	2	115	2	-	2	-	-	-
Lib2	715	683	42	371	17	-	33	3	6	-
Lib3	306	296	-	200	4	-	-	-	-	-
total	1284	1212	44	686	23	-	35	3	6	-

5. S. Cox, R. Haftka, C. Baker, B. Grossman, W. Mason, and L. Watson. A comparison of global optimization methods for the design of a high-speed civil transport. *Journal of Global Optimization*, 21(4):415–432, 2001.
6. E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
7. E. Dolan, J. Moré, and T. Munson. Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-273, Mathematics and Computer Science Division, Argonne National Laboratory, 2004.
8. E. Dolan, J. Moré, and T. Munson. Optimality measures for performance profiles. *SIAM Journal on Optimization*, 16(3):891–909, 2006.
9. F. Domes. GloptLab - A configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. *Optimization Methods and Software*, 24(4-5):727–747, 2009.
10. F. Domes. Test Environment. Web document, `http://www.mat.univie.ac.at/~dferi/testenv.html`, 2013.
11. F. Domes, M. Fuchs, and H. Schichl. The Optimization Test Environment. Web document, `http://www.mat.univie.ac.at/~dferi/testenv_download.html`, 2013.
12. A. Drud. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming*, 31(2):153–191, 1985.
13. A. Drud. CONOPT – a large-scale GRG code. *ORSA Journal on Computing*, 6(2):207–216, 1994.

Table 11 Reliability analysis. Percentage of global numerical solutions found/number of accepted feasible problems (G+/F+), percentage of correctly claimed global numerical solutions/number of global numerical solutions found (G!/G+), percentage of wrong solutions/number of accepted problems (wr/acc).

Reliability analysis on accepted problems			
solver	G+/F+	G!/G+	wr/acc
baron	73%	69%	8%
cocos	60%	76%	12%
coin	66%	3%	4%
conopt	56%	4%	8%
knitro	49%	0%	21%
lindoglobal	67%	79%	14%
minos	58%	4%	3%
pathnlp	57%	3%	3%

Table 12 Status of each solver on some problems of size 1 of LIB1.

name	n	m	fbest	baron		cocos		coin		conopt	
				st	tst	st	tst	st	tst	st	tst
chance	4	3	2.989e+01	G	G!	U	-	L	G+	L	G+
circle	3	10	4.574e+00	G	G!	G	G!	L	G+	U	-
dispatch	4	2	3.155e+03	G	G!	G	G!	L	G+	L	G+
ex14-1-1	3	4	-1.400e-07	G	G!	G	G!	L	G+	L	G+
ex14-1-2	6	9	-9.920e-09	G	G!	G	G!	L	G+	L	G+
ex14-1-3	3	4	-9.964e-09	G	G!	G	G!	L	G+	L	G+
name	n	m	fbest	knitro		lindoglobal		minos		pathnlp	
				st	tst	st	tst	st	tst	st	tst
chance	4	3	2.989e+01	L	G+	G	G!	L	G+	U	-
circle	3	10	4.574e+00	L	F?	I	I?	U	-	U	-
dispatch	4	2	3.155e+03	L	G+	G	G!	L	G+	L	G+
ex14-1-1	3	4	-1.400e-07	L	G+	G	G!	L	G+	L	G+
ex14-1-2	6	9	-9.920e-09	L	F+	G	G!	L	G+	L	G+
ex14-1-3	3	4	-9.964e-09	L	G+	G	G!	L	G+	L	F+

14. M. Ferris and T. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12(1):207–227, 1999.
15. R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press/Brooks/Cole Publishing Company, 2002.
16. K. Fowler, J. Reese, C. Kees, J. Dennis, C. Kelley, C. Miller, C. Audet, A. Booker, G. Couture, R. Darwin, M. Farthing, D. Finkel, J. Gablonsky, G. Gray, and T. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, 2008.
17. Gamsworld. Performance tools. Web document, `http://gamsworld.org/performance/tools.htm`, 2009.
18. J. Gilbert and X. Jonsson. LIBOPT - An environment for testing solvers on heterogeneous collections of problems - The manual, version 2.1. Technical Report RT-331 revised, INRIA, 2009.
19. N. Gould, D. Orban, and P. Toint. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.
20. H. Schichl et al. The COCONUT Environment, 2000–2013. Software.
21. D. Johnson. A theoreticians guide to the experimental analysis of algorithms. *American Mathematical Society*, 220(5-6):215–250, 2002.
22. J. Kallrath. *Modeling languages in mathematical optimization*. Kluwer Academic Publishers, 2004.

23. R. Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, 2003.
24. H. Mittelmann. Benchmarks. Web document, `http://plato.asu.edu/sub/benchm.html`, 2009.
25. C. Moles, P. Mendes, and J. Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003.
26. M. Mongeau, H. Karsenty, V. Rouze, and J. Hiriart-Urruty. Comparison of public-domain software for black box global optimization. *Optimization Methods and Software*, 13(3):203–226, 2000.
27. J. Moré and S. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.
28. B. Murtagh and M. Saunders. MINOS 5.5 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, 1983. Available on-line at: `http://www.sbsi-sol-optimize.com/manuals/Minos%20Manual.pdf`.
29. NEOS. NEOS Server for Optimization `http://www.neos-server.org/neos/`, 2013.
30. A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinko. A comparison of complete global optimization solvers. *Mathematical programming*, 103(2):335–356, 2005.
31. N. Sahinidis and M. Tawarmalani. BARON 7.2.5: Global optimization of mixed-integer nonlinear programs. User's Manual, 2005. Available on-line at: `http://www.gams.com/dd/docs/solvers/baron.pdf`.
32. L. Schrage. *Optimization Modeling with LINGO*. LINDO Systems, 2008.
33. O. Shcherbina. COCONUT benchmark. Web document, `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`, 2013.
34. O. Shcherbina, A. Neumaier, D. Sam-Haroud, X. Vu, and T. Nguyen. *Global Optimization and Constraint Satisfaction*, chapter Benchmarking global optimization and constraint satisfaction codes, pages 211–222. Springer, 2003.
35. M. Tawarmalani and N. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.
36. A. Vaz and L. Vicente. A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*, 39(2):197–219, 2007.