Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique

The Optimization Test Environment

Ferenc Domes, Martin Fuchs, Hermann Schichl

Technical Report TR/PA/10/15

Publications of the Parallel Algorithms Team
http://www.cerfacs.fr/algor/publications/

The Optimization Test Environment

Ferenc Domes[1], Martin Fuchs[2*], Hermann Schichl[1]

[1]University of Vienna, Faculty of Mathematics, Wien, Austria
[2]CERFACS, Parallel Algorithms Team, Toulouse, France
[*]corresponding author: martin.fuchs81@gmail.com

February 17, 2010

Abstract. The 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 and PDF output.

The Test Environment is free to use for research purposes.

Keywords. test environment, optimization, solver benchmarking, solver comparison

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 automatize 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 Test Environment is a graphical user interface (GUI) that enables to manage the first two tasks interactively, and the third task automatically.

The 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., [7]. 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 [3, 12, 21] (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., [22] for global optimization, or the large online collection [20], 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. [1, 11, 18], 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 [13], 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. For our comparisons we will count for each solver the number of global solutions found, and the number of wrong and correct claims for the solutions.

Here we consider the term global solution as the best solution found among all solvers. We also produce several more results and enable the creation of performance profiles [4, 23].

Further rather technical difficulties come with duplicate test problems, the identification of which is an open task for future versions of the 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 [15] provide a 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 [14], 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, restricted to black box optimization.

Our approach to developing the TEST ENVIRONMENT is inspired by the experience made during the comparisons reported in [25], in which the COCONUT Environment benchmark [29] 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, platform independent installation. Hence the core part of the 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 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 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 TEST ENVIRONMENT. A collection of already existing scripts for several solvers, including setup instructions, is available on the TEST ENVIRONMENT website [6]. We explicitly **encourage** people who have implemented a solve script or analyze script for the 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 TEST ENVIRONMENT. This may significantly improve the performance of a solver.

As **problem representation** we use Directed Acyclic Graphs (DAGs) from the COCONUT Environment [16]. We have decided to choose this format as there already exist automatic conversion tools inside the COCONUT Environment from many modeling languages to DAGs and vice versa. The TEST ENVIRONMENT is thus independent from any choice of a modeling language. Nevertheless benchmark problem collections, e.g., given in AMPL such as COPS [5], can be easily converted to DAGs. With the DAG format, the present version of the TEST ENVIRONMENT excludes test problems that are created in a black box fashion.

The summarizing part of the TEST ENVIRONMENT is managing **automated tasks** which have to be performed manually in many former test environments. These tasks include an automatic check of solutions, 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 solutions are obtained and correctly claimed. The results of the TEST ENVIRONMENT also allow for the automated creation of performance profiles which is left as an open task for future versions.

This paper is organized as follows. In Section 2 we give an overview of our notation for optimization problem formulations. Section 3 can be regarded as a tutorial for the TEST ENVIRONMENT, while in Section 4 we present advanced features. Finally we demonstrate the capabilities of the TEST ENVIRONMENT with numerical tests in Section 5.

The last section includes a benchmark of six 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 [29]. The test libraries and the results are also available online on the TEST ENVIRONMENT website [6]. This paper focuses on the presentation of the TEST ENVIRONMENT software rather than on the benchmark. However, we intend to collect benchmark results from the TEST ENVIRONMENT on our website, towards a complete comparison of solvers.

The tested solvers in alphabetical order are: BARON 8.1.5 [26, 30] (global solver), COCOS [16] (global), COIN with Ipopt 3.6/Bonmin 1.0 [19] (local solver), CONOPT 3 [8, 9] (local), KNITRO 5.1.2 [2] (local), Lindoglobal 6.0 [27] (global), MINOS 5.51 [24] (local), Pathnlp 4.7 [10] (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 performed best. Lindoglobal had the most correctly claimed global solutions, however, it made also the most mistakes claiming a global solution. More details can be found in Section 5.

2 Formulating optimization problems

We consider optimization problems that can be formulated as follows:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in \mathbf{x}, \\ & F(x) \in \mathbf{F}, \\ & x_i \in \mathbb{Z} \text{ for } i \in I, \end{aligned} \tag{2.1}$$

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 **boundconstrained** if $\mathbf{F} = \mathbb{R}^m$ 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 (2.1), differentiated by the properties of f, F, and I. If $f = \text{const}$, problem (2.1) 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 (2.1) 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 (2.1) 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 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**

$$\mathrm{d}_{\mathrm{feas,p}}(x_s, f_s) \leq \alpha \tag{2.2}$$

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 interval

$$\mathbf{x}_s = [x_s - \varepsilon \max(\|x_s\|_\infty, \kappa), x_s + \varepsilon \max(\|x_s\|_\infty, \kappa)] \tag{2.3}$$

with the parameter ε (set to 10^{-6} by default), the parameter κ (set to 1 by default). Then we compute the **objective violation**

$$v_o(x_s, f_s) := \langle f(x_s) + f'(\mathbf{x}_s)(\mathbf{x}_s - x_s) - f_s \rangle, \tag{2.4}$$

the **box constraint violations**

$$v_b(x_s, f_s) := \langle \mathbf{x}_s - \mathbf{x} \rangle, \tag{2.5}$$

and the **constraint violations**

$$v_c(x_s, f_s) := \langle F(x_s) + F'(\mathbf{x}_s)(\mathbf{x}_s - x_s) - \mathbf{F} \rangle, \tag{2.6}$$

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

$$\langle \mathbf{x}^i \rangle := \begin{cases} \min(|\underline{x^i}|, |\overline{x^i}|) & \text{if } 0 \notin [\underline{x^i}, \overline{x^i}], \\ 0 & \text{otherwise}, \end{cases} \tag{2.7}$$

i.e., the smallest absolute value within the interval $[\underline{x^i}, \overline{x^i}]$ for the component of x with index i. The gradients reflect the sensitivity to scaling in the different constraint violations. The complete componentwise violation is given by $v(x_s, f_s) = (v_o(x_s, f_s), v_b(x_s, f_s), v_c(x_s, f_s))^T$. We define the **feasibility distance** $\mathrm{d_{feas,p}} : \mathbb{R}^{n+1} \to \mathbb{R}$ as

$$\mathrm{d_{feas,p}}(x_s, f_s) := \|v(x_s, f_s)\|_p. \tag{2.8}$$

Since $\mathrm{d_{feas,\infty}} \leq \mathrm{d_{feas,p}}$ for all p with $1 \leq p \leq \infty$ we decided to choose $p = \infty$ in the Test Environment to check for feasibility via (2.2). In the remainder of the paper the feasibility check is also referred to as the **solution check** (see also Section 4.5).

Let $J = \{x_1, \ldots, x_N\}$ be the set of all solver results that have passed the solution check. We define the **global numerical solution** x_{opt} as the best numerically feasible solution found by all solvers used, i.e., $x_{\mathrm{opt}} \in J$ and $f(x_{\mathrm{opt}}) \leq f(x_j)$ for all $x_j \in J$. Another feasible solution $\widetilde{x} \in J$ is also considered global if $f(\widetilde{x})$ is sufficiently close to $f(x_{\mathrm{opt}})$, i.e.,

$$\delta \leq \beta, \tag{2.9}$$

with

$$\delta := \begin{cases} f(\widetilde{x}) - f(x_{\text{opt}}) & \text{if } |\min(f(\widetilde{x}), f(x_{\text{opt}}))| \leq \kappa, \\ \frac{f(\widetilde{x}) - f(x_{\text{opt}})}{|\min(f(\widetilde{x}), f(x_{\text{opt}}))|} & \text{otherwise }, \end{cases} \tag{2.10}$$

with κ as above and with the tolerance β which is set to 10^{-6} by default.

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

We define the **best point** found as

$$x_{\text{best}} = \begin{cases} x_{\text{opt}} & \text{if } J \neq \emptyset, \\ \arg\min_{x \in J} \mathrm{d}_{\text{feas,p}}(x) & \text{if } J = \emptyset, \end{cases} \tag{2.11}$$

i.e., the global 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 **hitlist**, cf. Section 4.6.

To assess the location of the global 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 default 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 (2.1). Prominent successful modeling languages are AMPL [11] and GAMS [1], but there are many more such as AIMMS, LINGO, LPL, MPL, see [18].

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

3 Basic functionality

This section guides the reader through the installation and the basic functionality of the Test Environment illustrated with simple examples how to add test problems, how to configure a solver, and how to run the Test Environment on the given problems.

3.1 Installation

The installation of the Test Environment is straightforward:

Download. The Test Environment is available on-line at
`http://www.mat.univie.ac.at/~dferi/testenv_download.html`.

Install. In Windows run the installer and follow the instructions. In Linux unzip the `.zip` file. Afterwards you can start the Test Environment at the unzip location via `java -jar TestEnvironment.jar`.

Requirements. The graphical user interface (GUI) of the 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.

Note that a folder that contains user specific files is created: for Windows the folder `TestEnvironment` in the application data subdirectory of your home directory; for Linux the folder `.testenvironment` in your home directory. We refer to this directory as the working directory **pwd**. All subdirectories of the pwd are set as default paths in the Test Environment configuration which can be modified by the user, cf. Section 4.1.2. The Test Environment configuration file (`TestEnvironment.cfg`), however, remains in the pwd.

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

3.2 Adding a new test library

After starting the 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 [16]. In case you do not have your problems given as `.dag` files, but as AMPL code, you need to convert your AMPL model to `.dag` files first which is possible via the Coconut Environment or easily via a converter script separately available on the Test Environment website [6]. Also you can find a huge collection of test problem sets from the Coconut Benchmark [28] given as `.dag` files on the Test Environment website.

Adding a new test library can be done in two ways: Either directly copy the `.dag` files to the directory of your choice within the `pwd/libs` directory before starting the Test Environment. Or click the **New** button in the GUI as shown in Figure 3.1. This creates a new directory in `pwd/libs` with the entered name. Then copy the `.dag` files into the directory created.

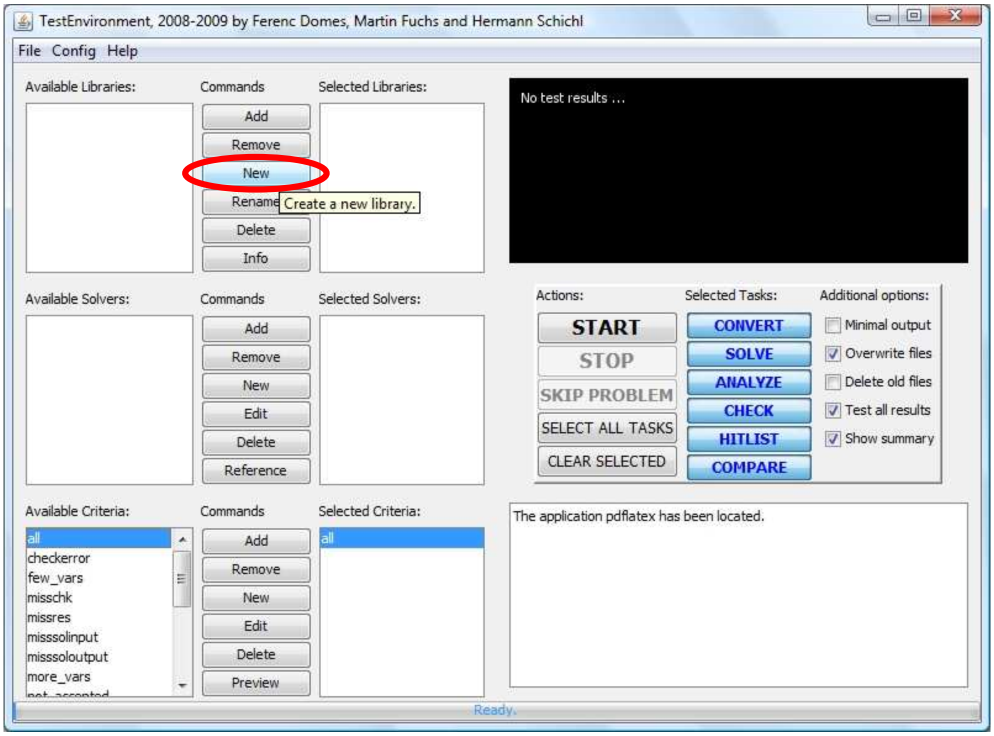

Figure 3.1: Click the **New** button to set up a new test problem library.

Let us create a new library **newlib**. We click the **New** button and enter 'newlib', then we copy 3 simple test problems to `pwd/libs/newlib`:

3.2.1 Simple example test library

t1.dag: The intersection of two unit circles around $x = (\pm 0.5, 0)^T$, cf. Figure 3.2. The problem formulation is as follows:

$$\begin{aligned} \min_x \;\; & x_2 \\ \text{s.t.} \;\; & (x_1 - 0.5)^2 + x_2^2 = 1, \\ & (x_1 + 0.5)^2 + x_2^2 = 1, \\ & x_1 \in [-3, 3], x_2 \in [-3, 3]. \end{aligned} \tag{3.1}$$

The feasible points of (3.1) are $x = (0, \pm\sqrt{3}/2)^T$. Minimizing x_2 results in the optimal solution $\widehat{x} = (0, -\sqrt{3}/2)^T \approx (0, -0.8660)^T$.

t2.dag: Two touching unit circles around $x = (\pm 1, 0)^T$, cf. Figure 3.3. The problem formulation is as follows:

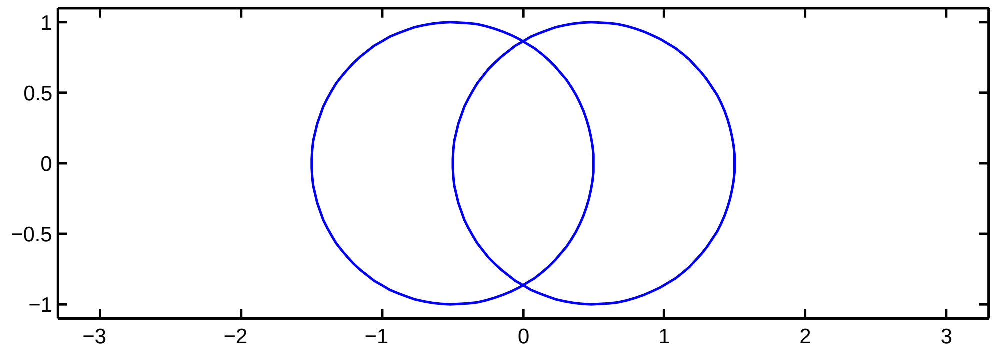

Figure 3.2: The feasible domain of (3.1) consists of the intersection points of two unit circles around $x = (\pm 0.5, 0)^T$.

$$\begin{aligned} \min_x \quad & 1 \\ \text{s.t.} \quad & (x_1 - 1)^2 + x_2^2 = 1, \\ & (x_1 + 1)^2 + x_2^2 = 1, \\ & x_1 \in [-3, 3], x_2 \in [-3, 3]. \end{aligned} \tag{3.2}$$

The only feasible point of (3.2) is $x = (0, 0)^T$.

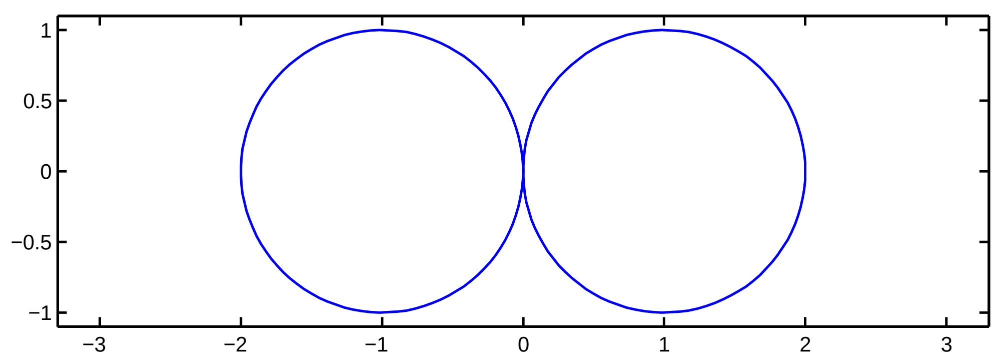

Figure 3.3: The only feasible point of (3.2) is at $x = (0, 0)^T$.

t3.dag: Two disjoint unit circles around $x = (\pm 2, 0)^T$, cf. Figure 3.4. The problem formulation is as follows:

$$\begin{aligned} \min_x \quad & 1 \\ \text{s.t.} \quad & (x_1 - 2)^2 + x_2^2 = 1, \\ & (x_1 + 2)^2 + x_2^2 = 1, \\ & x_1 \in [-3, 3], x_2 \in [-3, 3]. \end{aligned} \tag{3.3}$$

There is no feasible solution for (3.3).

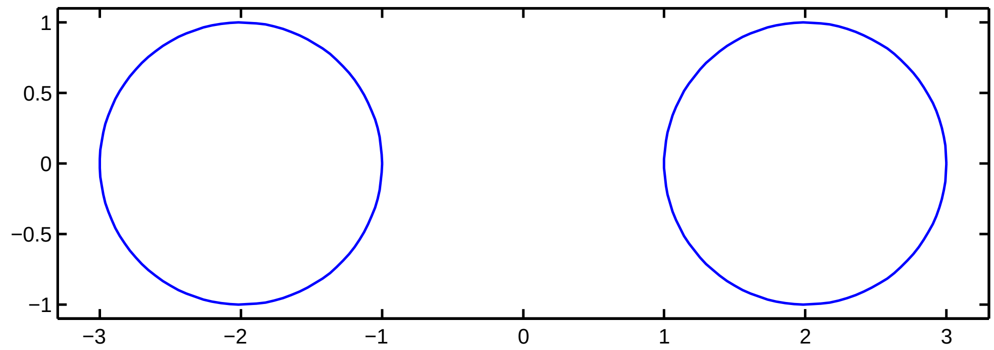

Figure 3.4: Obviously there is no feasible point for (3.3).

See Section 3.2.2 for the AMPL code of (3.1), (3.2), and (3.3), respectively. The AMPL code is converted to `.dag` files, with a command line conversion as shown in Figure 3.6. Also see Section 4.3 for more details.

In our examples we actually know the results of the three problems. In this case the user can optionally provide reference solutions as described in Section 4.2.

The files `t1.dag`, `t2.dag`, and `t3.dag` have to be copied to `pwd/libs/newlib`. To finish the creation the 'newlib' library click **Info**, then **Generate Dag Infos**. The generation of dag infos is necessary only if the dag infos do not exist yet, the **Info** window shows the number of correct dag info files. Afterwards select the 'newlib' entry among the 'Available Test Libraries' in the TEST ENVIRONMENT and click **Add** to select 'newlib' as one of the libraries that will be processed by the tasks on the right of the GUI. Next we learn how to add a solver that solves our three problems from 'newlib'.

3.2.2 AMPL code of the example problems

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;
```

3.3 Adding a solver

To add a solver to the 'Available Solvers' list, simply click the **New** button, cf. Figure 3.5.

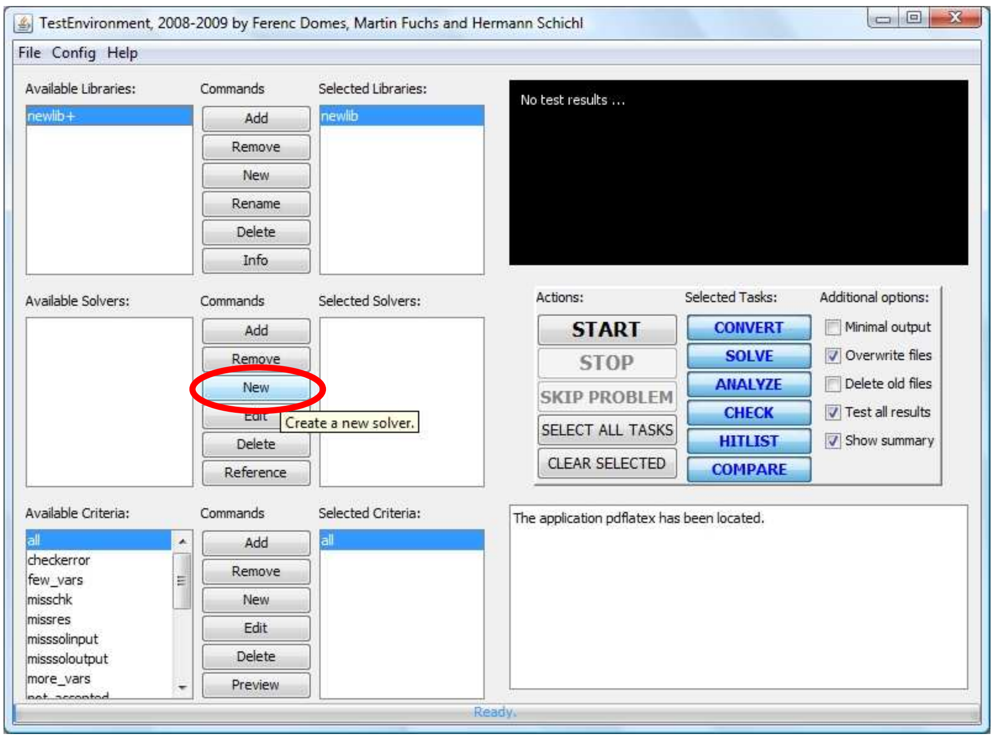

Figure 3.5: Add a solver by clicking **New**.

The 'Solver Configuration' window pops up, cf. Figure 3.6.

First enter a name. 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), so we enter 'KNITRO'. Afterwards we enter as 'Input File Extensions' first 'mod', then 'lst'.

We use the predefined solver configuration for KNITRO from the Test Environment website [6] which provides the solver configurations and installation guides for many well-known solvers, also see Section 4.3. We download the configuration archive for KNITRO/AMPL and extract it to the `pwd/solvers/KNITRO` directory. We only need to modify

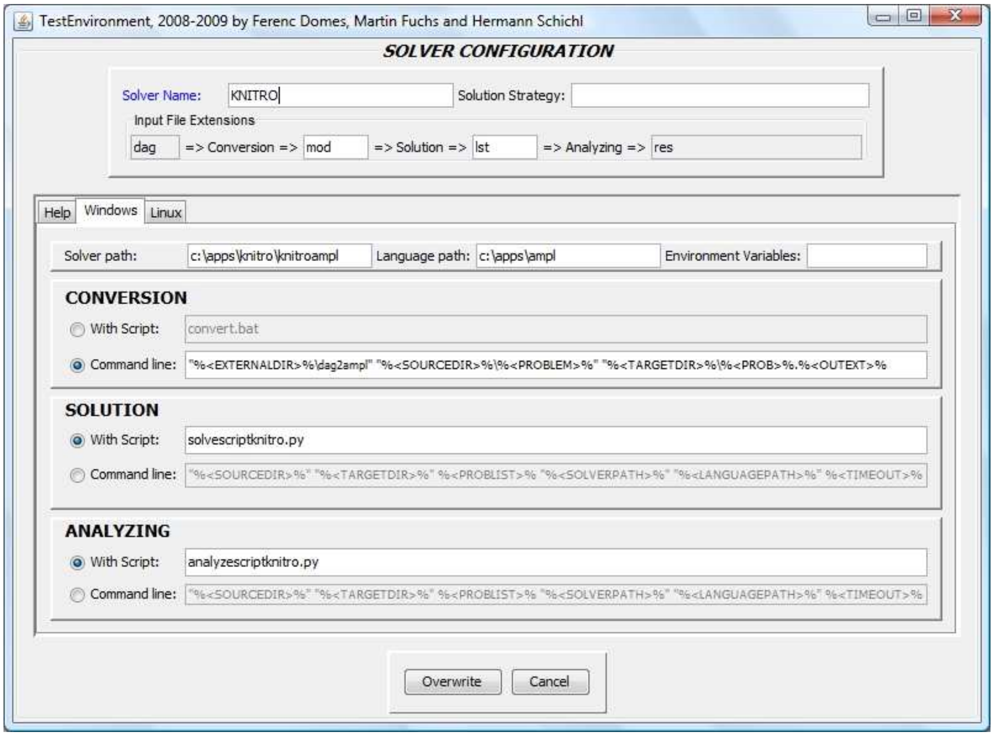

Figure 3.6: The solver configuration window

the path names for the AMPL and KNITRO path, i.e., edit the fields 'Language path' and 'Solver path', respectively, in the solver configuration window.

Click **Overwrite** and use **Add** to select the KNITRO solver. From the 'Available Criteria' add 'all', and we are ready to solve and analyze the 'newlib' test problems.

3.4 Solve the test problems

Just hit the **Select All Tasks** button and press **Start**, cf. Figure 3.7.

The selected problems are solved by the selected solvers, the results are analyzed by the Test Environment, the output files (in LaTeX and pdf) are written to the `pwd/results` folder, 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 selected 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.

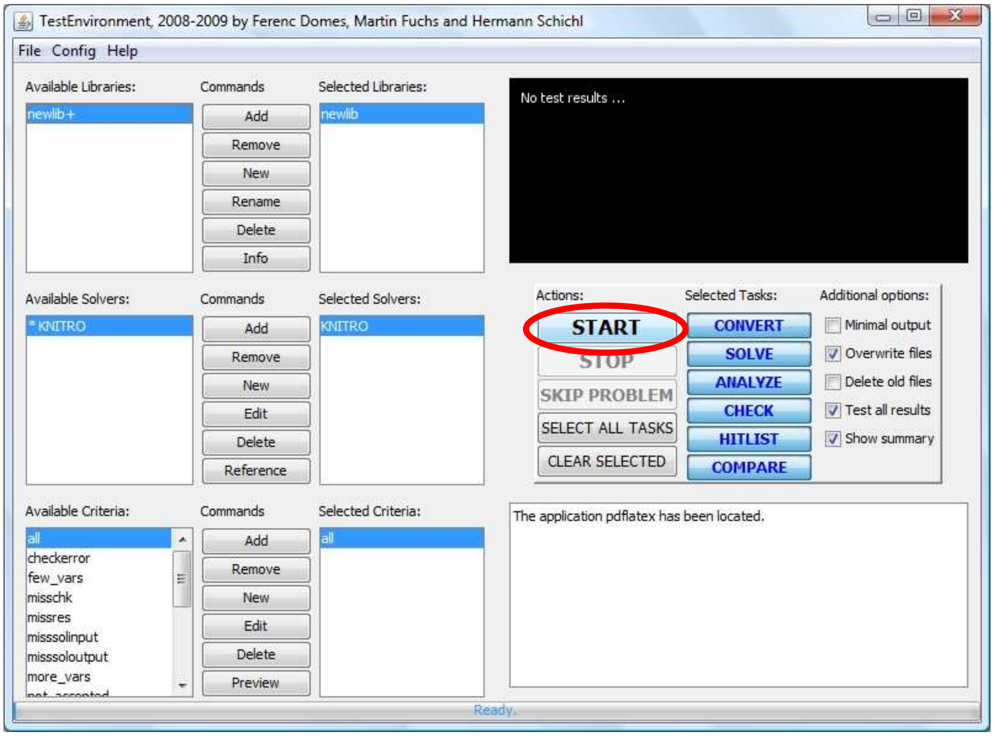

Figure 3.7: **Select All Tasks** and **Start**.

4 Advanced functionality

Now we know the basic procedure of solving a set of optimization problems within the Test Environment. Of course, there are plenty of things left that can be explored by the advanced user as will be introduced in this section.

4.1 Configuration

To access the configuration of the Test Environment you click on **Config** and choose the option you would like to configure.

4.1.1 Debug mode

Tick the Debug mode to run the Test Environment for debugging purposes increasing the amount of text output in the text window.

4.1.2 Default paths and other variables

To change the default path location of your libraries, result files etc. click on **Config** → **Variables**. The menu opened enables the user to add variables and to modify variables, cf. Figure 4.1.

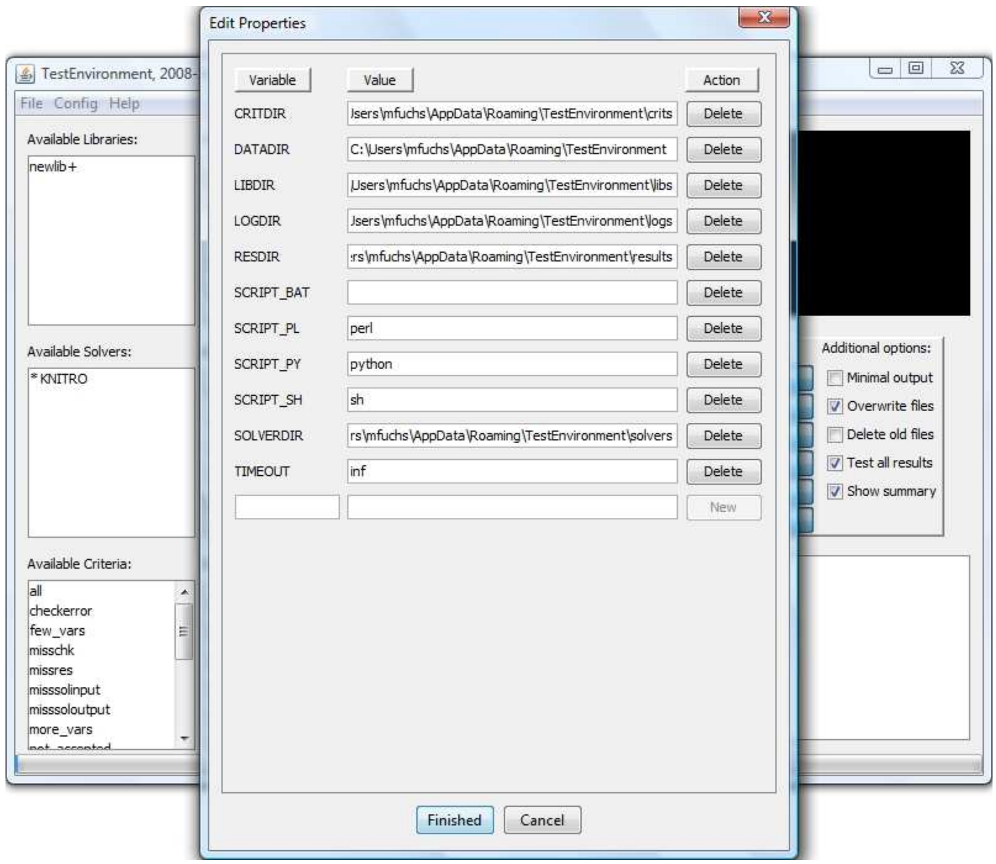

Figure 4.1: Here you can change the default paths or add or modify other variables.

Among these variables are the default paths for the criteria selection (CRITDIR), the test libraries (LIBDIR), the log files (LOGDIR), the result files (RESDIR), the data directory (DATADIR, typically the parent directory of the directories before), and the solver configurations (SOLVEDIR).

Also a general timeout can be set as TIMEOUT (for more details on timeouts see Section 4.4).

The variables 'SCRIPT_<ext>' define how scripts with the extension <ext> should be called.

4.1.3 Text output options

To configure the automatically generated LaTeX and pdf output click on **Config→Text Output**. For choosing what should be shown in the output just tick the associated box.

4.1.4 Shekel test

Click **Config→Rerun Shekel Test** to rerun the Shekel test on your machine, i.e., to determine the time it takes to evaluate the Shekel 5 function which could be used optionally for some performance measures.

4.2 .res files and reference solutions

As mentioned in Section 3.2.1, 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 `.res` file which has to be put into the `pwd/libs/newlib_sol` folder. This folder is automatically generated upon creation of the 'newlib' entry.

The generic format of `.res` files within the 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
```

which we write to `t1.res`. There are several fields that can be entered in `.res` files. Table 4.1 gives a survey of all possible entries.

Possible values for the model status in the `.res` file are shown in Table 4.2. The variables `x(1)`, ..., `x(n)` correspond to the variables in the problem (`.dag`) file, enumerated in the same sequence, but 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 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 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

Table 4.1: Entries in `.res` files.

`modelstatus`	solver model status, see Table 4.2
`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

```
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 Test Environment we intend to use this information to compare rigorous solvers.

Table 4.2: Modelstatus values.

0	Global 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

For providing a reference solution, the entries time and splits are optional. For problem `t2.dag` we provide a **wrong** solution in `t2.res`:

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

For problem `t3.dag` we do not provide any `.res` file.

To finish the setup of the 'newlib' test library with the given reference solutions click **Info**, and finally **Check Solutions**, cf. Figure 4.2. Note that the solution check for the reference solution of `t2.dag` has failed as we provided a wrong solution.

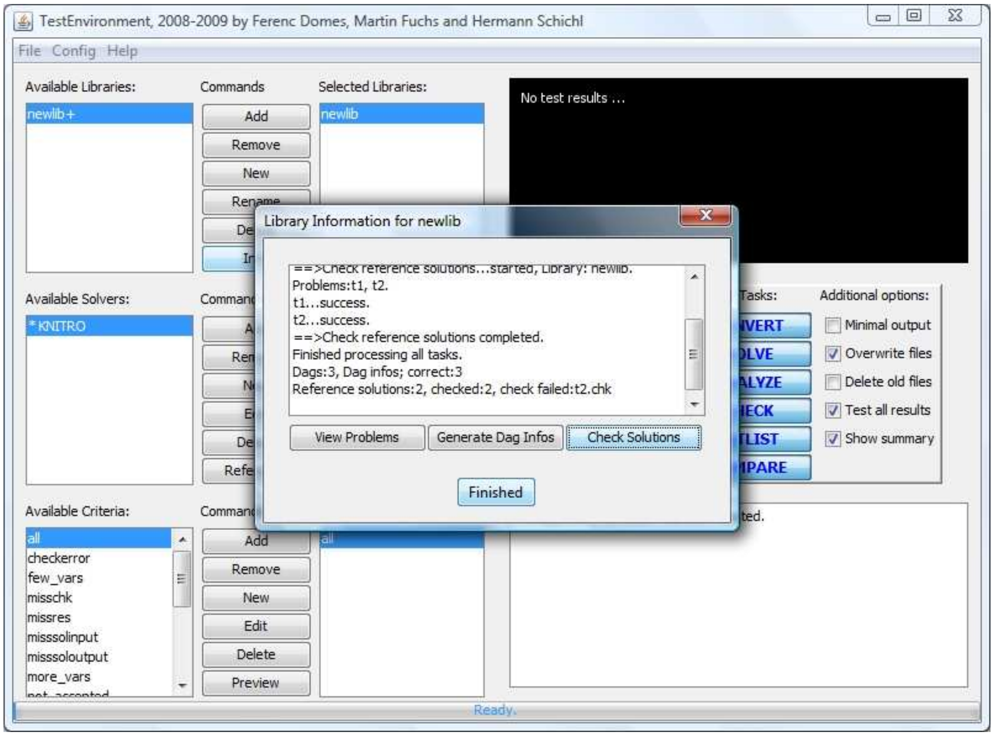

Figure 4.2: Test library information – DAG information and a check of the reference solutions

A given reference solution of a problem is considered in the same way as a solver result when computing J, x_{opt}, and x_{best}, cf. Section 2.

4.3 Solver setup

The Test Environment website [6] offers a collection of predefined solver configurations and instructions how to use them, such as the configuration we used to set up the KNITRO solver with AMPL interface in Section 3.3.

In our example we use one command line call and two scripts. Note that for the Linux version the quotation marks have to be omitted and \ has to be replaced by `/`. The first command line call is in charge of the conversion of the problem `.dag` files to the AMPL modeling language:

```
"%<EXTERNALDIR>%\dag2ampl" "%<SOURCEDIR>%\%<PROBLEM>%"...
        ..."%<TARGETDIR>%\%<PROB>%.%<OUTEXT>%
```

The `solvescriptknitro.py` script calls the solver. The Test Environment calls this script as

```
%<SCRIPT_PY>% "%<SOURCEDIR>%\solvescriptknitro.py" "%<SOURCEDIR>%" "%<TARGETDIR>%"...
        ...%<PROBLIST>% "%<SOLVERPATH>%" "%<LANGUAGEPATH>%" %<TIMEOUT>%
```

which would be a command line call equivalent to entering only the script name in the solver configuration (i.e., `solvescriptknitro.py`).

A solver is considered to be **compatible** with the Test Environment if it enables the user to set the maximal allowed time for the problem solution process, which is used by the Test Environment to compare different solvers by setting the same value `%<TIMEOUT>%` for each of them. For a complete list of Test Environment compatible solvers see the website [6].

To be able to call a solver it is necessary to know the solver location, i.e., the `%<SOLVERPATH>%` and/or the location of an associated modeling language, i.e., the `%<LANGUAGEPATH>%`. Hence we pass all these variables together with the problem list `%<PROBLIST>%` to the solve script.

All variables, e.g., `%<PROBLIST>%`, are also explained in detail in the Help tab of the solver configuration.

After calling the solver, the script `analyzescriptknitro.py` is used to generate the `.res` files from the solver output. Help on the required fields in a `.res` file can be found by pressing F1 or in Section 4.2. Analyze scripts (and convert scripts analogously) are called by the Test Environment in the same way as solve scripts, i.e.,

```
%<SCRIPT_PY>% "%<SOURCEDIR>%\analyzescriptknitro.py" "%<SOURCEDIR>%" "%<TARGETDIR>%"...
        ...%<PROBLIST>% "%<SOLVERPATH>%" "%<LANGUAGEPATH>%" %<TIMEOUT>%
```

Note that complete `.res` files (cf. Section 4.2) **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 Test Environment website requires basic knowledge about scripting string manipulations which is an easy task for advanced users and especially for solver developers.

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

Some solvers can be called setting a command line flag in order to generate `.res` files directly without the need of an analyze script.

4.4 Selection of criteria

The criteria selection is another core feature of the TEST ENVIRONMENT. The criteria editor can be accessed via the **New** button to create a custom criterion. The criteria are used to specify subsets of test libraries. As shown in Figure 4.3, the user has a lot of 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 4.3. The creation of a criterion in the TEST ENVIRONMENT GUI is straightforward using the 'Condition' editor together with the 'Logicals' window of the 'Criteria editor', cf. Figure 4.3.

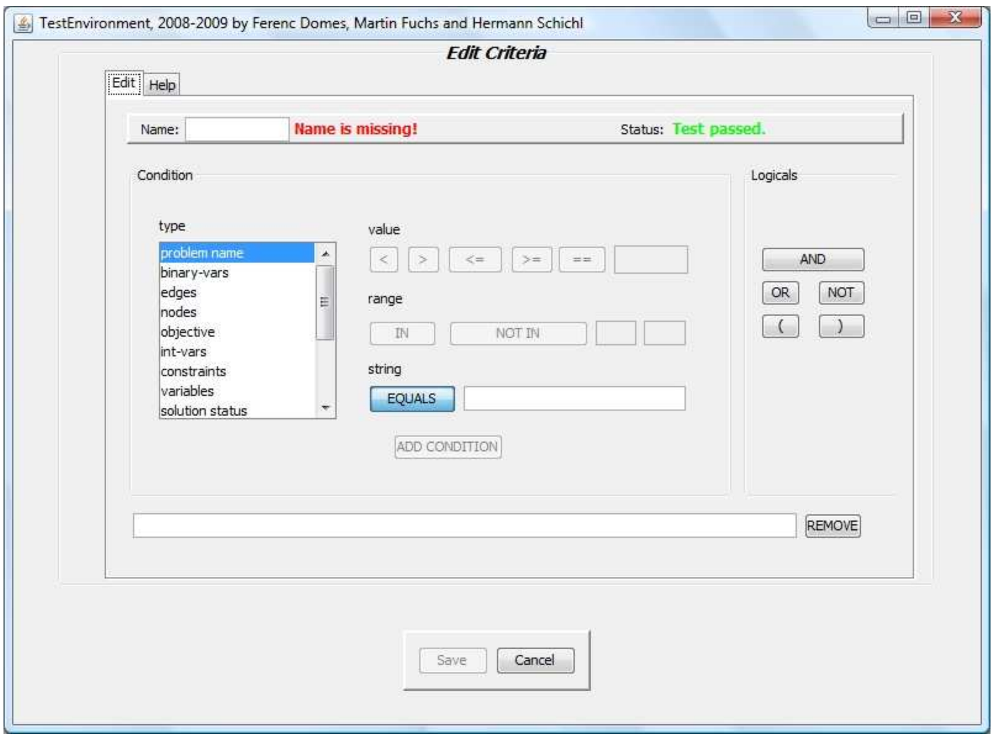

Figure 4.3: Criteria Editor

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

The 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.

One important feature of criteria is their use in setting **timeouts** in connection with further

Table 4.3: 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 4.2
`check error`	$\mathrm{d}_{\mathrm{feas,p}}$ 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

conditions like problem size. For example, to set up the timeout as in Table 5.3, 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]`

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.

4.5 Solution check: .chk files

The internal solution check of the Test Environment (cf. Section 2) is given by the program `solcheck` which produces `.chk` files. Every `.chk` file contains 6 entries: the `.dag` problem file name, an objective function upper and lower bound, the feasibility distance of the solver solution $\mathrm{d}_{\mathrm{feas,p}}(x_s, f_s)$ (where p is user-defined, default $p = \infty$), the feasibility distance with $p = \infty$, and finally the name of the constraint that determines the ∞-norm in $\mathrm{d}_{\mathrm{feas},\infty}(x_s, f_s)$.

4.6 Task buttons

There are 6 task buttons that can be found on the right side of the Test Environment GUI: **Convert**, **Solve**, **Analyze**, **Check**, **Hitlist**, **Compare**. In our example we executed all 6 tasks in a single run. By way of selecting or deselecting single tasks one can also run them separately. **Convert**, **Solve**, and **Analyze** correspond to the tasks configured in the solver setup described in Section 4.3. **Check** performs a solution check (cf. Section 2).

Hitlist generates a plain text file in the `pwd/results/hitlist` folder containing the best points x_{best} for each problem solved (also see Section 2).

Compare identifies easy and hard locations using the local solver that is highlighted by the `*` symbol in the 'Available solvers' window. Afterwards the result tables are generated. If for one problem the left-hand side expression in 2.9 is strictly negative the hitlist is not up to date and we throw a warning to recommend regeneration of the hitlist. This may happen if some results have been downloaded and compared to previous results before the hitlist was updated.

4.7 Action buttons

There are 5 action buttons on the right side of the Test Environment GUI: **Start**, **Stop**, **Skip Problem**, **Select All Tasks**, **Clear Results**. **Start** and **Stop** concern the tasks selected by the task buttons. **Skip Problem** skips the current problem viewed in the progress bar. **Select All Tasks** eases the selection and deselection of all task buttons. **Clear Results** deletes all files associated with the tasks selected by the task buttons.

4.8 Additional options

The 5 checkboxes on the right side of the Test Environment GUI are pretty much self explanatory. If **Delete old files** is enabled and a task selected by the task buttons is run, then all former results of the selected problems for all tasks below the selected task are considered obsolete and deleted. The **Test all results** checkbox concerns the case that a library was solved on several different computers. In our example the directory `pwd/solvers/KNITRO/res/newlib_<name>` contains all res files produced on the computer named `<name>`. To compare results from different computers, just copy these directories on one computer to `pwd/solvers/KNITRO/res` and click **Check**, **Hitlist**, and **Test**. The **Show summary** will open the summary pdf (cf. Section 3.4) after the **Test** task if enabled.

5 Numerical results

In this section we present two cases of test results produced with the 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 Test Environment in benchmarking solvers.

In the generated LaTeX tables we use the legend given in Table 5.1.

Table 5.1: 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 W, G?, I?, L?, see below)
easy	problems which have been classified as easy
hard	problems which have been classified as hard
n	number of variables
m	number of constraints
fbest	best objective function value found
obj	objective function value
Solution status codes (st):	
G	the result claimed to be a global optimizer
L	local solution found
I	Solver found the problem infeasible
TL	timeout reached and a local solution was found
U	unresolved (no solution found or error message)
X	model not accepted by the solver
TestEnvironment **status codes** (tst):	
G+	the global numerical solution has been found
G-	solution is not global
F+	a feasible solution was found
F-	no feasible solution was found
W	solution is wrong, i.e., [F+ and solution check failed and (G or L or TL)]
G!	correctly claimed global solution, i.e., [G and G+]
G?	wrongly claimed global solution, i.e., [G and G- and not W]
L?	wrongly claimed local solution, i.e., [L and F-]
I!	correctly claimed infeasibility, i.e., [I and F-]
I?	wrongly claimed infeasibility, i.e., [I and F+]

5.1 Results for newlib

The results for the example test library 'newlib' are shown in Table 5.2. We see that KNITRO has found correct solutions for t1 and t2 and could not resolve the infeasible problem t3. The TEST ENVIRONMENT treated the two feasible solutions for t1 and t2 as global solutions since no better solution was found by any other solver. The problem t3 was treated as infeasible since no feasible solution was found among all solvers. As KNITRO is a local solver it could not claim any solution as global or infeasible. The global solutions identified by the TEST ENVIRONMENT are classified as easy locations as they were found by the local reference solver.

Table 5.2: Results for 'newlib'.

knitro on Newlib								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	3	0	2	0	0	0	2	1
L	2	0	2	0	0	0	2	0
U	1	0	0	0	0	0	0	1

knitro summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
newlib	3	3	0	2	0	0	0	0	0	0
total	3	3	0	2	0	0	0	0	0	0

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	-

5.2 Solver benchmark

We have run the benchmark on the libraries LIB1, LIB2, and LIB3 of the COCONUT Environment benchmark [29], 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 [26, 30] (global solver), COCOS [16] (global), COIN with Ipopt 3.6/Bonmin 1.0 [19] (local solver), CONOPT 3 [8, 9] (local), KNITRO 5.1.2 [2] (local), Lindoglobal 6.0 [27] (global), MINOS 5.51 [24] (local), Pathnlp 4.7 [10] (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 5.3.

Table 5.3: 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

Additionally for solvers inside GAMS we used the GAMS options

```
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 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 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., [17]).

The results are shown in Tables 5.4 to 5.14, automatically generated by the TEST ENVIRONMENT. The performance of every solver on each of the three test libraries is given in Table 5.4 and Table 5.5 for LIB1, in Table 5.6 and Table 5.7 for LIB2, and in Table 5.8 and Table 5.9 for LIB3, respectively. Table 5.10 and Table 5.11 summarize the performance of every solver on the test libraries. The tables can be found at the end of the paper.

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 solutions. Looking at the tables 5.4 to 5.9 the global solvers are obviously superior in this respect, especially on hard locations.

Table 5.12 provides the reliability statistics of the solvers, showing the ratio of global solutions found over the number of accepted problems, the ratio of correctly claimed global solutions over the number of global 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 solutions among the global solutions found. Cocos is second with 76%, and Baron is third with 69%. But it should be remarked that Lindoglobal made 15% wrong solution claims as opposed to Baron with 8%. Not surprisingly, the local solvers had only very bad scores in claiming global 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 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 5.13 and Table 5.14. It shows the solver status and Test Environment status for each solver on each problem of size 1 from Lib1. One can see, e.g., that for the first problem 'chance.dag' all solvers have found the global numerical solution ecept for Cocos and Pathnlp which did not resolve the problem. BARON and Lindoglobal correctly claimed the global solution. We see how easily the 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 criteria selection, cf. Section 4.4.

Baron has found the most global solutions among all accepted problems. The local solver Coin also performed very well in this respect, at the same level as the global solver Lindoglobal. Hence Coin 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 47%. 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). New results with updated versions are continuously uploaded to the Test Environment website [6].

Acknowledgments

Partial funding of the project by the Research Grant FSP 506/003 of the University of Vienna is gratefully appreciated. 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. We also highly appreciate Arnold Neumaier's ideas for improving the TEST ENVIRONMENT, and the comments by Yahia Lebbah.

References

[1] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide.* The Scientific Press, 1988.

[2] R.H. Byrd, J. Nocedal, and R.A. Waltz. *Large-Scale Nonlinear Optimization*, chapter KNITRO: An Integrated Package for Nonlinear Optimization, pp. 35–59. Springer, 2006.

[3] S.E. Cox, R.T. Haftka, C.A. Baker, B. Grossman, W.H. Mason, and L.T. 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.

[4] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[5] E.D. Dolan, J.J. Moré, and T.S. Munson. Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-273, Mathematics and Computer Science Division, Argonne National Laboratory, 2004.

[6] F. Domes. TEST ENVIRONMENT website, `http://www.mat.univie.ac.at/~dferi/testenv.html`, 2009.

[7] 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.

[8] A.S. Drud. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming*, 31(2):153–191, 1985.

[9] A.S. Drud. CONOPT – a large-scale GRG code. *ORSA Journal on Computing*, 6(2): 207–216, 1994.

[10] M.C. Ferris and T.S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12(1):207–227, 1999.

[11] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press/Brooks/Cole Publishing Company, 2002.

[12] K.R. Fowler, J.P. Reese, C.E. Kees, J.E. Dennis, C.T. Kelley, C.T. Miller, C. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray, and T.G. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, 2008.

[13] Gamsworld. Performance tools, `http://gamsworld.org/performance/tools.htm`, 2009.

[14] J.C. 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.

[15] N.I.M. Gould, D. Orban, and P.L. Toint. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.

[16] H. Schichl et al. The COCONUT Environment, 2000–2010. Software. URL `http://www.mat.univie.ac.at/coconut-environment`.

[17] D.S. Johnson. A theoreticians guide to the experimental analysis of algorithms. *American Mathematical Society*, 220(5-6):215–250, 2002.

[18] J. Kallrath. *Modeling languages in mathematical optimization*. Kluwer Academic Publishers, 2004.

[19] R. Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[20] H. Mittelmann. Benchmarks, `http://plato.asu.edu/sub/benchm.html`, 2009.

[21] C.G. Moles, P. Mendes, and J.R. Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003.

[22] M. Mongeau, H. Karsenty, V. Rouze, and J.B. Hiriart-Urruty. Comparison of public-domain software for black box global optimization. *Optimization Methods and Software*, 13(3):203–226, 2000.

[23] J.J. Moré and S.M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[24] B.A. Murtagh and M.A. 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`.

[25] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinko. A comparison of complete global optimization solvers. *Mathematical programming*, 103(2):335–356, 2005.

[26] N.V. 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`.

[27] L. Schrage. *Optimization Modeling with LINGO*. LINDO Systems, 2008.

[28] O. Shcherbina. COCONUT benchmark, `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`, 2009.

[29] O. Shcherbina, A. Neumaier, D. Sam-Haroud, X.H. Vu, and T.V. Nguyen. *Global Optimization and Constraint Satisfaction*, chapter Benchmarking global optimization and constraint satisfaction codes, pp. 211–222. Springer, 2003. URL `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`.

[30] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3): 563–591, 2004.

Table 5.4: Performance of each solver on Lib1.

baron on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	34	57	33	89	74	253	11
G	104	20	38	17	44	4	103	1
L	103	12	19	6	45	33	103	0
I	3	2	0	0	0	2	2	1
X	36	0	0	7	0	22	29	7
U	18	0	0	3	0	13	16	2

cocos on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	27	68	22	72	91	253	11
G	150	22	53	5	53	35	146	4
I	6	3	0	2	0	2	4	2
TL	32	2	10	0	19	3	32	0
U	76	0	5	15	0	51	71	5

coin on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	6	68	22	70	93	253	11
G	3	1	2	0	0	1	3	0
L	215	5	66	18	70	61	215	0
X	25	0	0	4	0	14	18	7
U	21	0	0	0	0	17	17	4

conopt on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	17	67	23	50	113	253	11
G	3	1	2	0	0	1	3	0
L	191	16	65	11	50	65	191	0
X	25	0	0	4	0	14	18	7
U	45	0	0	8	0	33	41	4

Table 5.5: Performance of each solver on Lib1 ctd.

knitro on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	43	90	0	0	163	253	11
L	214	43	88	0	0	124	212	2
X	4	0	0	0	0	4	4	0
TU	13	0	0	0	0	12	12	1
U	33	0	2	0	0	23	25	8

lindoglobal on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	27	67	23	80	83	253	11
G	114	9	49	7	45	13	114	0
L	62	4	14	1	33	14	62	0
I	18	14	0	5	0	12	17	1
X	25	0	0	4	0	14	18	7
U	45	0	4	6	2	30	42	3

minos on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	2	63	27	44	119	253	11
G	3	1	2	0	0	1	3	0
L	180	1	61	12	43	64	180	0
X	30	0	0	9	0	14	23	7
U	51	0	0	6	1	40	47	4

pathnlp on `Lib1`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	264	2	61	29	54	109	253	11
G	3	1	2	0	0	1	3	0
L	182	1	59	16	53	54	182	0
X	31	0	0	10	0	14	24	7
U	48	0	0	3	1	40	44	4

Table 5.6: Performance of each solver on Lib2.

baron on Lib2								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	23	228	109	186	170	693	22
G	256	8	134	8	98	11	251	5
L	286	12	94	39	79	73	285	1
I	5	3	0	2	0	1	3	2
X	80	0	0	32	0	42	74	6
U	88	0	0	28	9	43	80	8

cocos on Lib2								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	31	173	164	163	193	693	22
G	296	23	135	20	93	42	290	6
I	9	7	0	3	0	4	7	2
TL	105	1	29	6	54	15	104	1
U	305	0	9	135	16	132	292	13

coin on Lib2								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	35	281	56	158	198	693	22
G	28	3	15	1	5	7	28	0
L	575	32	264	39	151	115	569	6
X	41	0	0	9	0	25	34	7
U	71	0	2	7	2	51	62	9

conopt on Lib2								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	77	239	98	138	218	693	22
G	28	3	15	1	6	6	28	0
L	517	74	221	55	129	107	512	5
X	37	0	0	4	0	25	29	8
U	133	0	3	38	3	80	124	9

Table 5.7: Performance of each solver on Lib2 ctd.

knitro on `Lib2`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	171	337	0	0	356	693	22
L	632	169	331	0	0	293	624	8
X	12	0	0	0	0	12	12	0
TU	9	0	3	0	0	3	6	3
U	59	0	2	0	0	46	48	11

lindoglobal on `Lib2`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	55	223	114	163	193	693	22
G	337	20	173	20	101	39	333	4
L	157	22	42	8	48	58	156	1
I	23	13	0	12	0	7	19	4
X	29	0	0	3	0	20	23	6
U	169	0	8	71	14	69	162	7

minos on `Lib2`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	21	204	133	94	262	693	22
G	27	3	15	1	7	4	27	0
L	390	18	188	29	86	81	384	6
X	205	0	0	87	0	103	190	15
U	93	0	1	16	1	74	92	1

pathnlp on `Lib2`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	715	33	240	97	131	225	693	22
G	23	3	15	1	2	5	23	0
L	495	30	222	49	119	99	489	6
X	32	0	0	6	0	20	26	6
U	165	0	3	41	10	101	155	10

Table 5.8: Performance of each solver on LIB3.

baron on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	8	150	28	102	20	300	7
G	257	5	150	4	102	1	257	0
I	8	3	0	0	0	3	3	5
X	33	0	0	23	0	10	33	0
U	9	0	0	1	0	6	7	2

cocos on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	22	161	17	112	10	300	7
G	252	12	140	8	99	4	251	1
I	8	5	0	3	0	2	5	3
TL	36	5	19	2	12	3	36	0
U	11	0	2	4	1	1	8	3

coin on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	2	168	10	47	75	300	7
G	3	0	3	0	0	0	3	0
L	211	2	163	1	45	1	210	1
X	9	0	0	3	0	6	9	0
U	84	0	2	6	2	68	78	6

conopt on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	1	139	39	34	88	300	7
G	4	0	4	0	0	0	4	0
L	169	1	134	1	33	0	168	1
X	16	0	0	9	0	7	16	0
U	118	0	1	29	1	81	112	6

Table 5.9: Performance of each solver on LIB3 ctd.

knitro on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	42	178	0	0	122	300	7
L	222	42	178	0	0	42	220	2
X	1	0	0	0	0	0	0	1
TU	6	0	0	0	0	6	6	0
U	78	0	0	0	0	74	74	4

lindoglobal on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	7	172	6	101	21	300	7
G	278	4	172	1	101	3	277	1
I	7	3	0	0	0	3	3	4
X	9	0	0	3	0	6	9	0
U	13	0	0	2	0	9	11	2

minos on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	1	141	37	53	69	300	7
G	4	0	4	0	0	0	4	0
L	184	1	134	0	49	1	184	0
X	10	0	0	4	0	6	10	0
U	109	0	3	33	4	62	102	7

pathnlp on `Lib3`								
st	all	wr	easy		hard			
			G+	G-	G+	G-	F+	F-
all	307	0	152	26	49	73	300	7
G	4	0	4	0	0	0	4	0
L	195	0	147	0	48	0	195	0
X	10	0	0	4	0	6	10	0
U	98	0	1	22	1	67	91	7

Table 5.10: Performance summary of every solver on the test libraries.

baron summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	228	38	146	82	1	34	2	0	2
Lib2	715	635	43	414	232	2	23	16	1	3
Lib3	307	274	11	252	252	5	8	0	0	3
total	1286	1137	92	812	566	8	65	18	1	8

cocos summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	264	53	140	106	2	27	22	0	4
Lib2	715	715	83	336	228	2	31	45	0	7
Lib3	307	307	28	273	239	3	22	1	0	5
total	1286	1286	164	749	573	7	80	68	0	16

coin summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	239	6	138	2	0	6	0	0	0
Lib2	715	674	46	439	20	0	35	5	6	0
Lib3	307	298	3	215	3	0	2	0	1	0
total	1286	1211	55	792	25	0	43	5	7	0

conopt summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	239	17	117	2	0	17	0	0	0
Lib2	715	678	86	377	21	0	77	4	5	0
Lib3	307	291	2	173	4	0	1	0	1	0
total	1286	1208	105	667	27	0	95	4	6	0

Table 5.11: Performance summary of every solver on the test libraries ctd.

knitro summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	260	45	90	0	0	43	0	2	0
Lib2	715	703	179	337	0	0	171	0	8	0
Lib3	307	306	44	178	0	0	42	0	2	0
total	1286	1269	268	605	0	0	256	0	12	0

lindoglobal summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	239	55	147	94	1	27	11	0	17
Lib2	715	686	118	386	274	4	55	43	1	19
Lib3	307	298	11	273	273	4	7	1	0	3
total	1286	1223	184	806	641	9	89	55	1	39

minos summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	234	2	107	2	0	2	0	0	0
Lib2	715	510	29	298	22	0	21	2	6	0
Lib3	307	297	1	194	4	0	1	0	0	0
total	1286	1041	32	599	28	0	24	2	6	0

pathnlp summary statistics										
library	all	acc	wr	G+	G!	I!	W	G?	L?	I?
Lib1	264	233	2	115	2	0	2	0	0	0
Lib2	715	683	42	371	17	0	33	3	6	0
Lib3	307	297	0	201	4	0	0	0	0	0
total	1286	1213	44	687	23	0	35	3	6	0

Table 5.12: Reliability analysis. Percentage of global solutions found/number of accepted problems (G+/acc), percentage of correctly claimed global solutions/number of global solutions found (G!/G+), percentage of wrong solutions/number of accepted problems (wr/acc).

solver	G+/acc	G!/G+	wr/acc
baron	71%	69%	8%
cocos	58%	76%	12%
coin	65%	3%	4%
conopt	55%	4%	8%
knitro	47%	0%	21%
lindoglobal	65%	79%	15%
minos	57%	4%	3%
pathnlp	56%	3%	3%

Table 5.13: Status of each solver on problems of size 1 of Lib1.

name	n	m	fbest	baron		cocos		coin		conopt		knitro		lindoglobal		minos		pathnlp	
				st	tst	st	tst	st	tst	st	tst	st	tst	st	tst	st	tst	st	tst
chance	4	3	2.989e+01	G	G!	U	-	L	G+	L	G+	L	G+	G	G!	L	G+	U	-
circle	3	10	4.574e+00	G	G!	G	G!	L	G+	U	-	L	W	I	I?	U	-	U	-
dispatch	4	2	3.155e+03	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-1-1	3	4	-1.400e-07	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-1-2	6	9	-9.920e-09	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-1-3	3	4	-9.964e-09	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	F+
ex14-1-4	3	4	-9.987e-09	X	-	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-1-5	6	6	-9.982e-09	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-1-6	9	15	-9.870e-09	G	G!	G	G!	L	F+	L	F+	L	F+	G	G!	L	F+	L	F+
ex14-1-8	3	4	0	G	G!	G	G?	L	F+	L	F+	L	F+	G	G!	L	F+	L	F+
ex14-1-9	2	2	-9.965e-09	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-2-1	5	7	-1.000e-08	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-2-2	4	5	-9.994e-09	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-2-3	6	9	-9.998e-09	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-2-4	5	7	-9.999e-09	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-2-5	4	5	-1.000e-08	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-2-6	5	7	-1.000e-08	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex14-2-7	6	9	-1.000e-08	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-2-8	4	5	-1.000e-08	G	G!	G	G!	L	G+	L	G+	L	F+	G	G!	L	G+	L	G+
ex14-2-9	4	5	-9.999e-09	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex2-1-1	5	1	-1.700e+01	G	G!	G	G!	L	F+	L	F+	L	F+	G	G!	L	F+	L	F+
ex2-1-2	6	2	-2.130e+02	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex2-1-4	6	4	-1.100e+01	L	W	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex3-1-1	8	6	7.049e+03	G	W	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex3-1-2	5	6	-3.067e+04	G	W	G	G!	L	F+	L	W	L	G+	G	G?	L	F+	L	F+
ex3-1-3	6	5	-310	G	G!	G	W	L	F+	L	F+	L	F+	G	G!	L	F+	L	F+
ex3-1-4	3	3	-4.000e+00	G	W	G	G!	L	G+	L	G+	L	G+	I	I?	L	G+	L	G+
ex4-1-1	1	0	-7.487e+00	G	W	G	G!	L	F+	L	W	L	F+	G	G!	L	F+	L	F+
ex4-1-2	1	0	-6.635e+02	G	W	G	G!	L	G+	L	W	L	G+	G	G!	L	G+	L	G+
ex4-1-3	1	0	-4.437e+02	G	W	G	G!	L	G+	L	F+	L	G+	I	I?	L	F+	L	F+
ex4-1-4	1	0	0	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex4-1-5	2	0	0	L	G+	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex4-1-6	1	0	7	G	G!	G	G!	L	F+	L	F+	L	F+	I	I?	L	F+	L	F+
ex4-1-7	1	0	-7.500e+00	G	G!	I	I?	L	G+	L	G+	L	G+	G	G!	L	G+	L	F+
ex4-1-8	2	1	-1.674e+01	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex4-1-9	2	2	-5.508e+00	G	G!	G	G!	L	F+	L	F+	L	F+	I	I?	L	F+	L	G+
ex5-2-2-case1	9	6	-4.000e+02	G	G!	G	G!	L	F+	L	F+	L	G+	G	G!	L	F+	L	F+
ex5-2-2-case2	9	6	-600	G	G!	U	-	L	F+	L	F+	L	F+	G	G!	L	F+	L	G+
ex5-2-2-case3	9	6	-7.500e+02	G	G!	G	G!	L	F+	L	F+	L	G+	G	G!	L	F+	L	F+
ex5-2-4	7	6	-4.500e+02	G	G!	G	G?	L	F+	L	F+	L	F+	G	G?	L	F+	L	F+
ex5-4-2	8	6	7.512e+03	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex6-1-1	8	6	-2.020e-02	L	W	G	G!	L	F+	L	F+	L	F+	L	G+	U	-	L	F+
ex6-1-2	4	3	-3.246e-02	L	G+	G	G!	L	G+	L	G+	L	F+	G	G!	L	F+	L	G+
ex6-1-4	6	4	-2.945e-01	L	G+	U	-	L	F+	L	F+	L	F+	G	G?	L	F+	L	F+
ex6-2-10	6	3	-3.052e+00	L	G+	G	G!	L	F+	L	G+	L	F+	L	G+	L	G+	L	F+
ex6-2-11	3	1	-2.670e-06	L	G+	I	I?	L	F+	L	F+	L	F+	L	G+	L	F+	L	F+
ex6-2-12	4	2	2.892e-01	G	G!	G	G!	L	F+	L	F+	L	F+	L	G+	L	F+	L	G+
ex6-2-13	6	3	-2.162e-01	L	G+	U	-	L	G+	L	G+	L	G+	L	G+	L	G+	L	G+
ex6-2-14	4	2	-6.954e-01	L	G+	G	G!	L	F+	L	F+	L	F+	I	I?	L	G+	L	G+
ex6-2-5	9	3	-7.075e+01	L	G+	G	G?	L	F+	L	F+	L	F+	L	G+	L	G+	L	F+
ex6-2-6	3	1	-2.600e-06	L	G+	U	-	L	F+	L	G+	L	F+	L	F+	L	F+	L	F+

Table 5.14: Status of each solver on problems of size 1 of LIB1 ctd.

name	n	m	fbest	baron		cocos		coin		conopt		knitro		lindoglobal		minos		pathnlp	
				st	tst	st	tst	st	tst	st	tst	st	tst	st	tst	st	tst	st	tst
ex6-2-7	9	3	-1.608e-01	L	G+	TL	G+	L	F+	L	F+	L	F+	L	G+	L	G+	L	F+
ex6-2-8	3	1	-2.701e-02	L	G+	TL	G+	L	F+	L	W	L	F+	L	G+	L	G+	L	F+
ex6-2-9	4	2	-3.407e-02	L	G+	G	G!	L	F+	L	F+	L	F+	L	G+	L	F+	L	F+
ex7-2-1	7	14	1.227e+03	G	W	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex7-2-2	6	5	-3.888e-01	L	W	G	W	L	G+	L	G+	L	G+	G	G!	L	G+	L	F+
ex7-2-3	8	6	7.049e+03	L	W	G	G?	L	F+	L	F+	L	F+	L	F+	L	G+	L	F+
ex7-2-4	8	4	3.918e+00	G	W	G	G!	L	G+	L	G+	L	G+	L	G+	L	F+	L	F+
ex7-2-5	5	6	1.012e+04	G	W	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex7-2-6	3	1	-8.325e+01	G	W	G	G!	L	G+	L	G+	L	G+	L	G+	L	G+	L	G+
ex7-2-7	4	2	-5.740e+00	G	W	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex7-2-8	8	4	-6.082e+00	G	W	G	G!	L	F+	L	F+	L	G+	L	G+	L	F+	L	F+
ex7-3-1	4	7	3.417e-01	L	G+	G	G!	L	G+	L	G+	L	F+	G	G!	L	F+	L	G+
ex7-3-2	4	7	1.090e+00	G	G!	U	-	L	G+	L	G+	L	F+	I	I?	L	G+	L	G+
ex7-3-3	5	8	8.175e-01	G	G!	G	G!	L	F+	L	F+	L	W	I	I?	L	F+	L	F+
ex7-3-6	1	2	CSP	G	G?	I	I!	U	-	U	-	U	-	I	I!	U	-	U	-
ex8-1-1	2	0	-2.022e+00	X	-	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	F+
ex8-1-2	1	0	-1.071e+00	X	-	G	G!	L	F+	L	F+	L	F+	G	G!	L	F+	L	F+
ex8-1-3	2	0	3	L	G+	G	G?	L	F+	L	F+	L	F+	L	G+	L	F+	L	F+
ex8-1-4	2	0	0	L	G+	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
ex8-1-5	2	0	-1.032e+00	L	W	G	G!	L	F+	L	F+	L	F+	G	G!	L	F+	L	F+
ex8-1-6	2	0	-1.009e+01	L	W	U	-	L	F+	L	W	L	F+	G	G!	L	F+	L	F+
ex8-1-7	5	5	2.931e-02	L	G+	G	G!	L	G+	U	-	L	G+	I	I?	L	G+	L	G+
ex8-1-8	6	5	-3.888e-01	L	W	G	W	L	G+	L	G+	L	G+	G	G!	L	G+	L	F+
ex8-5-1	6	5	-4.075e-07	X	-	TL	G+	X	-	X	-	L	F+	X	-	X	-	X	-
ex8-5-2	6	4	-6.129e-06	X	-	TL	G+	X	-	X	-	L	F+	X	-	X	-	X	-
ex8-5-3	5	5	-4.135e-03	X	-	G	G!	X	-	X	-	U	-	X	-	X	-	X	-
ex8-5-4	5	4	-4.251e-04	X	-	TL	G+	X	-	X	-	U	-	X	-	X	-	X	-
ex8-5-5	5	4	-5.256e-03	X	-	TL	G+	X	-	X	-	U	-	X	-	X	-	X	-
ex8-5-6	6	4	1.000e+30	X	-	TL	G+	X	-	X	-	U	-	X	-	X	-	X	-
ex9-2-4	8	7	5.000e-01	G	G!	G	G!	L	G+	L	G+	L	W	G	G!	L	G+	L	G+
ex9-2-5	8	7	5.000e+00	G	G!	G	G!	U	-	L	F+	L	W	G	G!	L	G+	L	F+
ex9-2-8	3	2	1.500e+00	G	G!	G	G!	G	G!	G	G!	L	G+	G	G!	G	G!	G	G!
himmel11	9	3	-3.067e+04	G	W	G	G!	L	F+	L	W	L	G+	G	G?	L	F+	L	F+
house	8	8	-4.500e+03	G	G!	G	G!	L	G+	L	G+	L	W	L	W	L	G+	L	G+
least	3	0	2.306e+04	L	W	G	G!	U	-	L	F+	U	-	L	F+	L	W	L	W
like	9	3	1.138e+03	X	-	TL	G+	X	-	X	-	U	-	X	-	X	-	X	-
meanvar	7	2	5.243e+00	G	G!	U	-	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
mhw4d	5	3	2.931e-02	L	W	G	G!	L	G+	L	G+	L	G+	L	G+	L	G+	L	G+
nemhaus	5	0	CSP	G	W	G	G!	G	W	G	W	X	-	G	W	G	W	G	W
rbrock	2	0	0	G	G!	G	G!	L	G+	L	G+	L	G+	G	G!	L	G+	L	G+
sample	4	2	7.267e+02	G	W	G	G?	L	F+	L	F+	L	F+	G	G?	L	G+	L	F+
wall	6	6	-1.000e+00	X	-	G	G!	X	-	X	-	U	-	X	-	X	-	X	-