Proceedings of TOGO 2010, pp. 1 – 4.

The Optimization Test Environment

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

[1]*University of Vienna, Faculty of Mathematics, Vienna, Austria*

[2]*CERFACS, Parallel Algorithms Team, Toulouse, France*

Abstract Testing is a crucial part of software development in general, and hence also in mathematical programming. 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.

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

The testing procedure typically consists of three basic tasks: a) organize test problem sets, also called test libraries; b) solve selected test problems with selected solvers; c) analyze, check and compare the results. The Test Environment is a graphical user interface (GUI) that enables to manage the tasks a) and b) interactively, and task c) 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. evaluate solvers on suitable test sets.

The first point considers a situation in which the user wants to improve parameters of a particular solver manually, see, e.g., [5]. 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 [10] (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., [17] for global optimization, or the large online collection [16], mainly for local optimization. Since in many 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], [9], 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 [11], 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.

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 [13] 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 [12], available for Unix/Linux, but not tailored 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 [19], in which the Coconut Environment benchmark [22] 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 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 is available on the Test Environment website [4]. 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 [14]. 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 designed to be independent from any choice of a modeling language. Nevertheless benchmark problem collections, e.g., given in AMPL such as COPS [3], can be easily converted to DAGs.

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.

Using the Test Environment we have performed a benchmark of eight solvers on 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 [22]. 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.

Benchmark test results

The tested solvers in alphabetical order are: BARON 8.1.5 [20] (global solver), Cocos [14] (global), COIN with Ipopt 3.6/Bonmin 1.0 [15] (local solver), CONOPT 3 [7] (local), KNITRO 5.1.2 [2] (local), Lindoglobal 6.0 [21] (global), MINOS 5.51 [18] (local), Pathnlp 4.7 [8] (local). 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.

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.

Baron has found the most global solutions among all accepted problems (71%). The local solver Coin also performed very well in this respect (65%), at the same level as the global solver Lindoglobal. 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). New results with updated versions are continuously uploaded to the Test Environment website [4]. For more details the interested reader is referred to [6].

Acknowledgments

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).

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, pages 35–59. Springer, 2006.

[3] 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.

[4] F. Domes. Test Environment website, http://www.mat.univie.ac.at/~dferi/testenv.html, 2009.

[5] 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.

[6] F. Domes, M. Fuchs, and H. Schichl. The optimization test environment. Submitted, 2010. Preprint available on-line at: http://www.mat.univie.ac.at/~dferi/testenv.html.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] H. Schichl et al. The COCONUT Environment, 2000–2010. Software.

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

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

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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, pages 211–222. Springer, 2003.