Acta
Technica
Jaurinensis

Vol.7., No.2., pp. 156–171, 2014
DOI: 10.14513/actatechjaur.v7.n2.272
Available online at acta.sze.hu

Benchmarking the LGO Solver Suite within the COCO Test Environment

M. F. Hatwágner[1], J. D. Pintér[2]

[1]**Széchenyi István University**
Dept. of Information Technology, Győr, Hungary
[2]**Pintér Consulting Services, Inc., Halifax, Nova Scotia, Canada**

Abstract: Optimization problems often arise in the context of scientific-engineering research and practice. In many situations that require optimization, there is no need to develop new, highly customized software for a new problem, because there are readily usable optimization packages to choose from. Benchmarking and testing software environments can greatly assist the process of choosing an appropriate optimization tool. The benchmarking environments typically include a substantial collection of well-known and widely used test functions; they also offer a properly defined methodology to compare the solvers considered. One of these benchmarking environments is called COCO (an abbreviation that stands for "COmparing Continuous Optimizers"). COCO has been used at the annual BBOB (Black-Box Optimization Benchmarking) workshops. COCO assesses the capabilities of optimization solvers based on a given collection of test problems and evaluation criteria, under identical and fully reproducible circumstances. The goal of our present study is to benchmark the LGO (Lipschitz Global Optimizer) solver suite and to compare it to several other solvers using COCO.

Keywords: Benchmarking optimization Software, LGO Solver Suite for Nonlinear Optimization, COCO Test Environment, Black-Box optimization Benchmarking.

1. Introduction

In this article the performance of the LGO solver suite is examined and compared to several other solvers using the COCO benchmarking environment. First, the main features of COCO are reviewed. This is followed by a concise description of the key features of LGO. Next, we describe the process of solver parameterization, and conclude with presenting the results based on the COCO test problem suite and its benchmarking process.

2. A Brief Description of the COCO Benchmarking Environment

GECCO, the GEnetic and Evolutionary Computation Conference [11]) has been held since 2009 in every year: the BBOB workshop [3] has been part of these conferences (with the exception of 2011). During the BBOB workshop the COCO [5] test environment has been used to compare various global optimization methods, and corresponding software. The yearly workshops always have a specific aim and emphasis, but there has not been a substantial change in the test environment or the overall objectives.

The comparison of the solvers is done applying a quantitative and completely reproducible process. The COCO test environment uses 24 real-valued test functions selected by a group of researchers on the basis of their mathematical properties: subsequently, these functions have been used to construct both the noiseless and noisy test model suites. The detailed description of these functions can be found in [8, 9]. Every model in the test suite has only a single objective: therefore the testing of multi-objective optimization capabilities and performance is currently not possible. The COCO environment allows testing of solvers written in various programming languages including C/C++, Java, Matlab, Python and R. The current version of COCO can be downloaded from the project website [6].

After linking a given solver to the test environment, the solver can be examined using two criteria set by COCO. The first criterion is the number of objective function evaluations required to solve each test problem. The second measure used is the runtime required to solve the same set of test problems. Based on practical reasons (with a view towards real-world applications), the first of these criteria is given a stronger emphasis.

The benchmarking results are generated automatically, and then stored in formatted output files: the latter can be downloaded from the annual BBOB websites (e.g. [28, 29]) for subsequent usage and comparisons. Python scripts are also included in the COCO test environment, to support the generation of figures and tables based on the output files. These objects can be then used to prepare topical reports and articles very easily, based on pre-assembled LaTeX templates.

For more detailed information on COCO and BBOB, see the BBOB set-up instructions [12] or the COCO user documentation [10].

3. A Brief Introduction to LGO

Consider the following general non-linear programming (NLP) problem:

$$\begin{gathered} \min f(x) \\ g(x) \leq 0 \\ x_l \leq x \leq x_u \end{gathered}$$

Here x is the real n-vector of the model function arguments; x_l and x_u are correspondingly defined (given) finite lower and upper bound n-vectors; $g(x)$ is the m-component vector of the general model constraints, and f is the objective function. The vector inequalities shown in the model formulation are (evidently) interpreted component-wise. The feasible set of the NLP problem will be denoted by D, is a proper subset of R^n.

If D is non-empty and all model functions f, g are continuous then – by the classical Bolzano-Weierstrass theorem – the NLP model has a non-empty set of (globally) optimal solutions. Notice at the same time that the NLP model may well be multi-modal: that is, initiating a suitable local optimization procedure from given starting points in D can lead to results of different quality. The objective of global optimization is to find the best possible solution - which gives the global optimum – under such circumstances.

The LGO solver suite has been designed to solve(numerically) instances of the NLP model stated above. LGO seamlessly integrates a suite of global and local scope algorithms (solvers) to handle the NLP in a robust and efficient manner. Let us point out that robustness and efficiency are partially conflicting criteria: LGO's overall development strategy is admittedly biased towards robustness, i.e. the ability to handle (also) possibly very tough optimization problems. NLP problems are tackled by LGO applying an integrated sequence of component solvers which, as a rule, find solutions to difficult optimization challenges – given a sufficient number of function evaluations and program execution time. All LGO solver components apply a direct (derivative-free) approach and proceed based only on model function evaluations. This feature may be advantageous in numerous "black box" type applications in which model function values may result from a numerical procedure such as simulation, or the solution of a system of differential equations and so on. Here we assume that the numerical procedure in question depends on (i.e., the corresponding model function is parametrized by) the decision variables x of the NLP.

We also note that LGO can handle general constrained nonlinear optimization problems – as opposed to merely box-constrained ones in which the general model constraints $g(x)$ are absent. Obviously, this feature adds complexity to LGO's overall design and it also influences its efficiency when the latter is measured by the number of function evaluations or program execution speed, without the need for measuring constraint satisfaction quality. As an example, consider the solution of a highly nonlinear system of equations, say in tens of variables – a tough challenge for many population-based heuristic methods, while being a routine task for LGO and similar scope global constrained optimization engines.

The in-depth technical discussion of LGO is outside of the scope of the present article. The theoretical foundations leading to LGO and some key implementation aspects are summarized in [18]. Platform-specific software implementation versions, numerous illustrative applications and case studies are discussed e.g. in [18–23].

The core LGO implementation is available for Windows / Linux / Mac / Unix OS based computer platforms, and it can be used to solve optimization problems formulated using

the Fortran, C and C# programming languages. LGO is also available as a solver option in different optimization modelling and technical computing environments [22, 23].

Since LGO and COCO both support the use of models developed using the C language, this version was selected for the tests described here. Fig. 1 summarizes the basic structure of LGO and its key components.

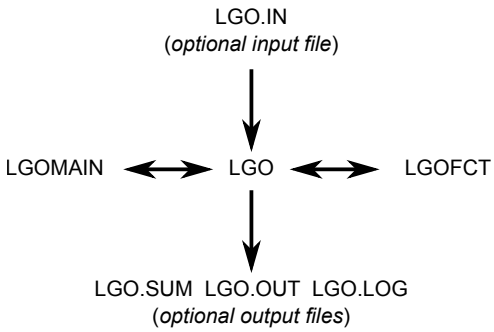

Figure 1. Structure of the LGO program.

The core LGO solver suite can be configured to use a specific input file and to generate result files, but these files are not needed in the COCO benchmarking context. The COCO environment itself generates randomized test data and also collects the results of the solver engine computations.

In the figure LGOMAIN is the main program (driver) file that calls LGO. After setting all static (i.e. once defined, then unchanging) optimization model information and a list of key solver parameters, LGOMAIN launches LGO. In the core implementation, LGO then iteratively calls LGOFCT: the latter serves to define all model functions in the NLP formulation.

Due to the setup of the COCO benchmarking framework, LGOMAIN's content has been embedded in the *MY_OPTIMIZER.c* file (the latter being a component of COCO). Furthermore, although in the core LGO implementation LGOFCT defines the model functions, in the COCO framework it is only a "placeholder" to redirect the objective function calls of the test environment.

Some of the key LGO solver settings were changed or tuned during the tests described here, therefore they are of particular interest.

One of these parameters is *opmode*, the operation mode selector. It has four possible settings:

0: Local Search (LS) only;

1: Global Branch-and-Bound (BB), followed by LS;

2: Global Adaptive Randomized Search (GARS), followed by LS;

3: Global Multi-start Randomized Search (MS), here each global search cycle is followed by LS.

For practical reasons, LGO starts with a LS phase which precedes all global search options. This is followed first by a regularly spaced sampling (RSS) presolver mode and another quick global presolver: each of these searches is followed by a LS phase. The RSS solver component is described in [24]. If further search is implied by *opmode* and it is supported by LGO's calling parameters and its internal resource allocation strategy, then one of BB + LS, GARS + LS, and MS + a set of LS phases follow.

For further details, we refer again to [18–20, 22–24].

4. Tuning of LGO's Solver Parameters

After completing the linking of LGO to the COCO test environment, some systematic tuning of the solver parameters was conducted to analyze the resulting benchmark results. All test functions were solved with the same preset parameter values during the benchmarking stages.

Let us note here that – according to the rules of COCO/BBOB studies – the modification of settings during runtime are forbidden: "On all functions the very same parameter setting must be used (which might well depend on the dimensionality, see Section Input to the Algorithm and Initialization). That means, a priori use of function-dependent parameter settings is prohibited (since 2012). In other words, the function ID or any function characteristics (like separability, multi-modality, ...) cannot be considered as input parameter to the algorithm. Instead, we encourage benchmarking different parameter settings as 'different algorithms' on the entire testbed." [17]

The above cited "rule" serves to simulate most real-life optimization situations in which in-depth problem-based insight typically cannot be used to effect solver parametrization.

4.1. The Effect of Different LGO Operational Modes

First of all, the impact of changing the value of *opmode* was examined. The recommended default setting is 3 corresponding to MS + LS [23]. We only advise the setting *opmode* = 0, if we want only a typically quick LS started from a given initial point.

After each of the four possible values were tested, it was concluded that there is no significant difference between the different operation mode based results, when solving the COCO test models . This finding is in line with LGO's overall development philospophy;

it is also supported by the fact that LGO does substantial search before switching to one of its BB + LS, GARS + LS or MS + LS modes. As a rule, the most resource-intensive global multi-start search and local search (MS + LS) mode produced the best results almost in all cases considering the proportion of the solved test function instances. The results are shown by Fig. 2.

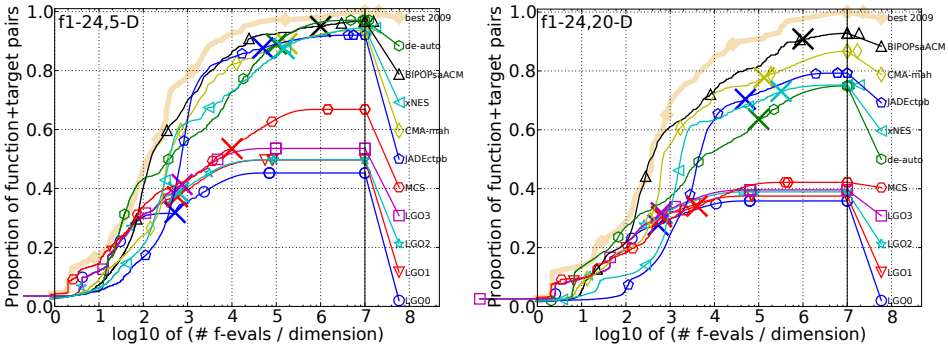

Figure 2. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 5-D and 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. The number after "LGO" represents the selected operation mode.

The performance measure used in the figure is ERT (the Expected value of Running Time based on randomized starting points). Directly citing the COCO developers' statement: ERT is quantitative, well-interpretable, relevant with respect to the "real world" and as simple as possible. The details of it [27] and the advantage of using it [1] are documented in the corresponding papers. The ERT computes a single measurement from a data sample set. Bootstrapping [7] can provide a dispersion measure for this aggregated measurement. COCO computes the running time of the single sample as the sum of function evaluations in the drawn trials (for the last trial up to where the target function value was reached) [1, 2].

The results of LGO were compared with some other prominent solvers in Fig. 2. The selection criteria for the solvers included were the following:

- MCS [14] has been included, because it can achieve similarly good performance to LGO (based on the in-depth performance study of [30] in which LGO and MCS were among the best derivative-free solvers among two dozen solvers when solving more than 500 test problems.
- The other solvers have been selected as the most successful ones in the BBOB 2012 competition, based on solving the COCO set of test models.

This way, 1+5 solvers were selected and indicated in the figure in addition to LGO (although it is sometimes not easy to differentiate among the plots belonging to the different solvers).

4.2. Experimenting with the Number of Objective Function Evaluations

As the figures indicate, at the beginning of the optimization process LGO produces nearly the same results as MCS, and achieves better results than many competitors such as BIPOPsaACM, CMA-mah or JADEctpb. However, in the later search phase – in the given resource / time frame and on the given test function set, with a given definition of required presicion for successful solution – MCS and most of the selected competitors surpass LGO.

Let us point out here again that the BBOB solvers have been specifically developed to solve merely box-constrained problems which make up the COCO test suite. More importantly – unlike many popular heuristic algorithms – LGO has a proper theoretical convergence guarantee for each of its global scope search methods: however, in theory the number of model function evaluations can (even should) go to infinity...

From this time on, the operation mode of LGO was fixed to multi-start random search and local search (MS + LS), and the other parameters were simply selected (pre-tuned). Specifically, we studied settings for the following LGO solver parameters: *g_maxfct* and *l_maxfct* define the maximal, total number of function evaluations in the selected global search phase and each local search phase, respectively. Moreover, *max_nosuc* determines the maximal number of function evaluations in the selected global search phase without improvement, before switching to the subsequent local search phase.

Several benchmarking trials were conducted using the same values for *g_maxfct, l_maxfct*, and *max_nosuc*. The default values of the parameter settings are actually dependent on the model size. We then heuristically applied the multiplier values 200, 400, 800, and 1600 to a simple estimate of model complexity, [23].

The increasing values of the parameters aided LGO to raise the proportion of the solved test functions. In 5D, it increased the proportion with approx. 5%, in 20D with approx. 3-4%. The latter value is practically the same as the value of MCS. (See Fig. 3).

4.3. Different Number of Objective Function Evaluations Allowed in the Various Optimization Phases

To decrease the number of objective function evaluations and thereby make the solver more effective, the value of *l_maxfct* was modified to 800 and 400 while the values of *g_maxfct* and *max_nosuc* remained the same: 1600. However, this did not result in a better overall performance as measured within COCO, see Fig. 4.

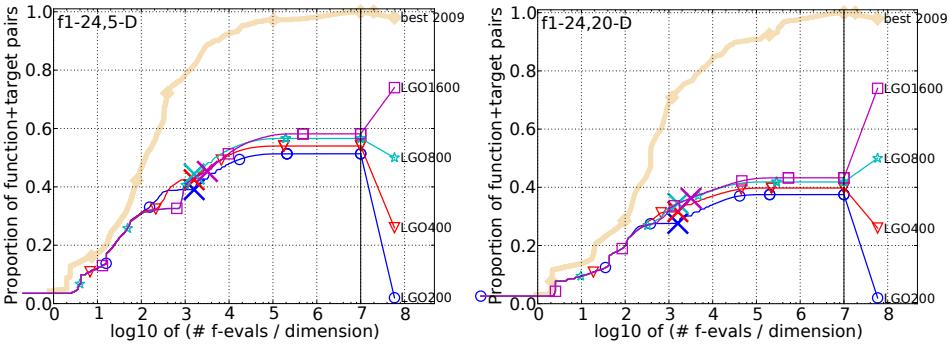

Figure 3. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension for 50 targets in $10^{[-8..2]}$ *for all functions and subgroups in 5-D and 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. The number after "LGO" represents the values of the examined solver parameters (*g_maxfct, l_maxfct, max_nosuc*).*

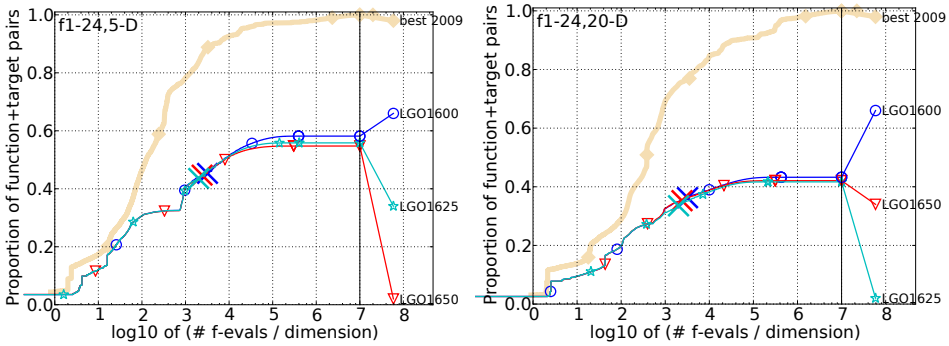

Figure 4. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension for 50 targets in $10^{[-8..2]}$ *for all functions and subgroups in 5-D and 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target. The value of* g_maxfct *and* max_nosuc *was 1600 in all cases. The values of* l_maxfct *was 1600 in plot "LGO1600", 800 in plot "LGO1650" and only 400 in plot "LGO1625".*

4.4. Comparing LGO with GLOBAL, MCS and NEWUOA

Next, the same value (1600) was used again for *g_maxfct, l_maxfct* and *max_nosuc*. In Fig. 5 and 6 the results of LGO are compared with the results of GLOBAL [4, 16], MCS [13, 14] and NEWUOA [26, 31].

GLOBAL is a multistart type stochastic method that involves a combination of sampling, clustering, and local search. It reduces the number of local search steps using clustering. The Multilevel Coordinate Search (MCS) algorithm was inspired by the DIRECT method [15]. MCS adaptively splits the search space into smaller boxes. The partitioning procedure is aimed at those boxes in which lower function values are expected to be found. By starting a local search from selected good points, an improved result is often obtained. Finally, NEWUOA is an iterative method that uses a quadratic model which is used in a trust region procedure for adjusting its variables and to approximate the value of the objective function.

If models with separable functions are to be solved, then MCS is the clear winner in 5D. The other three algorithms perform rather similarly: the difference in the proportion of the solved problems is less than 5%. In 20D, MCS is still in winning position, but the differences of the final success rates are less than about 6%.

LGO solved approx. 73% of the moderate test function instances (within the given time frame), but in this case the competitors had better performance results. The efficiency of the other three algorithms was very similar. In case of 20D NEWUOA is the winner, while LGO is the second best with a success rate of approx. 65% (again and again, within the given time frame. . .).

One can notice that in the initial phase of optimizing the ill-conditioned COCO test functions in 5D LGO shows strong performance, but later its convergence speed decreased and it achieved only the third place. MCS was the loser of this kind of benchmark in 5D and 20D as well.

LGO performed best if the goal was to solve multi-modal functions. In 5D, the differences between the results of LGO, MCS and GLOBAL are minimal, while NEWUOA – due to its admittedly more local search scope – seems to have a serious drawback. LGO is the winner in 20D, too. Let us recall here the important fact global optimization is the key intended target application area for using LGO.

If the functions are weakly structured multi-modal, LGO is only the third, and the convergence is slow in 5D. It is very suprising, that the winner in 5D (MCS) is only the third in 20D. In the later case, NEWUOA is the best and LGO is the last (approx. 34% of instances solved).

In general, MCS proved to be the best algorithm in 5D cases. LGO was the last and it was able to solve about 58% of the function instances. It is a bit strange, but MCS was the

worst algorithm in 20D. NEWUOA was the best algorithm, and LGO placed third. LGO solved approx. 42% of the test function instances.

It can be seen, that LGO performs well if the goal is to determine the optima of multi-modal functions, while NEWUOA and GLOBAL are able to solve more of the COCO tests using the same number of function evaluations. It should be pointed out again that this finding essentially depends on the COCO test model suite, since the far more detailed study [30] places each of LGO, MCS, and NEWUOA among its "winning solvers". We will cite the summary "verdict" of the latter study in our Conclusions below.

5. Conclusions

Benchmarking optimization software can be very helpful for users to select appropriate tools to handle their models – especially in the context of this article when the model class considered is extremely broad, covering from relatively simple problems to some hard global optimization challenges.

In the most general terms, the objective of the LGO software is to provide an overall robust and efficient suite of solver modules which help to obtain a high-quality feasible solution, across a significant range of general constrained nonlinear – global and local – optimization problems. Recall that this is done in a derivative-free setting: i.e. LGO does not require model function gradients or higher order information. This feature makes it applicable even to completely "black box" models – as long as these are defined by continuous functions.

Given this broadly defined scope mandate, it is not too surprising that on a small set of test models chosen (or perhaps even "fabricated") by other researchers LGO sometimes does better, but at other times does not, than other good quality solvers such as GLOBAL, MCS or NEWUOA. LGO, MCS and NEWUOA are among the most successful bound-constrained solvers, as per the recent in-depth numerical performance study of Rios and Sahinidis [30]. This study is based on a far more extensive set of test models than the COCO test model collection: from among more than 20 solvers LGO, MCS, and NEWUOA are the best performers with some other solver not discussed here. Below we cite the study of Rios and Sahinidis [30]:

"This paper addresses the solution of bound-constrained optimization problems using algorithms that require only the availability of objective function values but no derivative information... The paper presents a review of derivative-free algorithms, followed by a systematic comparison of 22 related implementations using a test set of 502 problems. The test bed includes convex and nonconvex problems, smooth as well as nonsmooth problems. The algorithms were tested under the same conditions and ranked under several criteria, including their ability to find near-global solutions to nonconvex problems, improve a given starting point, and refine a near-optimal solution. A total of 112,448 problem instances were solved. We find that the ability of all these solvers to obtain good solutions diminishes

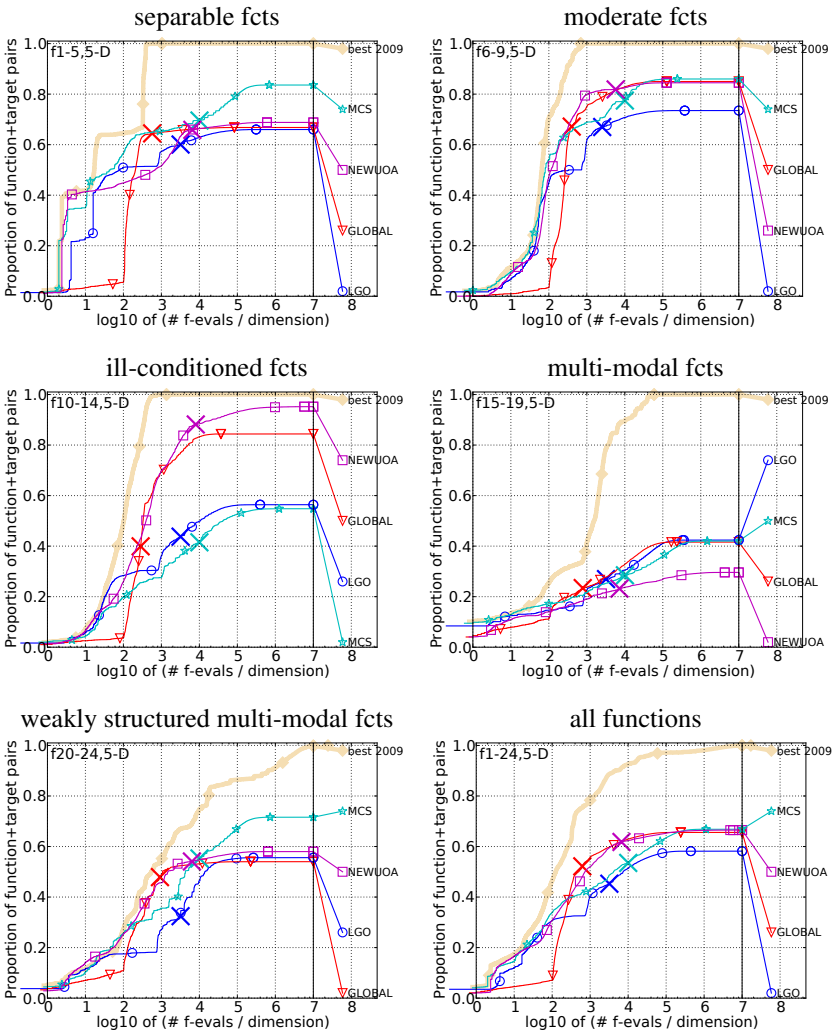

Figure 5. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 5-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target.

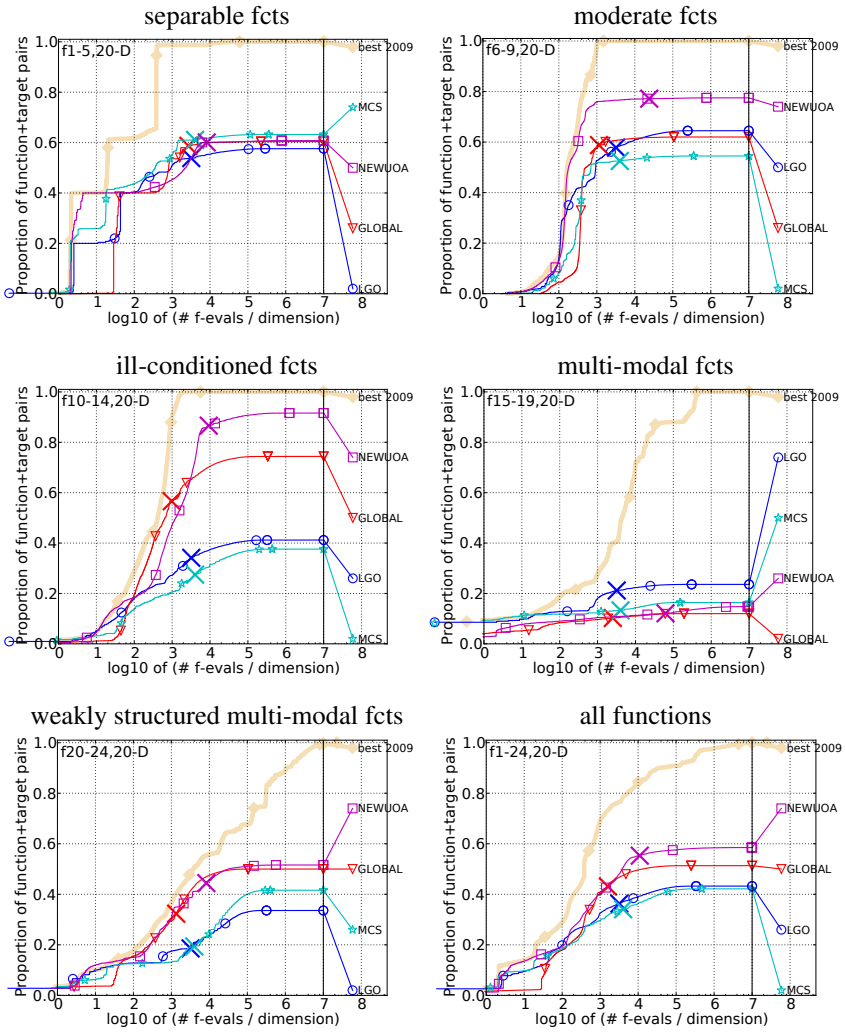

Figure 6. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 20-D. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each single target.

with increasing problem size. For the problems used in this study, TOMLAB/MULTIMIN, TOMLAB/GLCCLUSTER, MCS and TOMLAB/LGO are better, on average, than other derivative-free solvers in terms of solution quality within 2500 function evaluations, while TOMLAB/OQNLP, NEWUOA and TOMLAB/MULTIMIN show superior performance in terms of refining a near-optimal solution."

(The TOMLAB/LGO implementation mentioned above is based on a several year old core LGO solver.)

Cf. also the recent study by Pintér and Kampas [25] which is based on a markedly different – and again more diverse – collection of models than the COCO test suite.

To conclude, our present study summarizes the numerical experiments conducted in the COCO benchmarking environment, highlights their key findings. A set of heuristically tested ("optimized") LGO solver parameter values was chosen and used. We also compared LGO with several other prominent solvers on the COCO test function set. We found that if the goal is to optimize highly multi-modal functions then LGO performed well; in other (arguably, simpler) cases the "competitors" often outperformed LGO to some extent.

We plan to expand on our own benchmarking studies, and we will continue to report results for more comprehensive test problem suites. Among our collected test problems we have included also some truly difficult scalable model-classes that – in addition to their real-life background and relevance – pose a challenge to the best state-of-the art nonlinear solvers. We also include tests on general constrained (i.e., not only box-constrained) models which can greatly surpass in difficulty their box-constrained (embedded) sub-models. The results of these tests will appear in forthcoming work.

Acknowledgement

The Project leading to this study has been supported by the Hungarian Government and co-financed by the European Social Fund: TÁMOP-4.2.2.A-11/1/KONV-2012-0012: Basic research for the development of hybrid and electric vehicles.

References

[1] Auger, A., Hansen, N.: *Performance evaluation of an advanced local search evolutionary algorithm*, in Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), pp. 1777–1784, 2005

[2] Auger, A., Ros, R.: *Benchmarking the pure random search on the BBOB-2009 testbed*, in GECCO (Companion) ACM, Editor: Franz Rothlauf, pp. 2479–2484, 2009

[3] Black-Box Optimisation Benchmarking homepage, *http://coco.gforge.inria.fr/doku.php?id=bbob-2013*

[4] Csendes, T., Pál, L., Sendin, J. O. H., and Banga, J. R.: *The GLOBAL Optimization Method Revisited*, Optimization Letters, Vol. 2, pp. 445–454, 2008

[5] Documentation site of COCO (COmparing Continuous Optimisers), *http://coco.gforge.inria.fr/doku.php*

[6] Download page of COCO, *http://coco.lri.fr/downloads/download13.09/bboball13.09.tar.gz*

[7] Efron, B., Tibshirani, R.: *An introduction to the bootstrap*, Chapman & Hall/CRC, 1993

[8] Finck, S., Hansen, N., Ros, R., Auger, A.: *Real-Parameter Black-Box Optimisation Benchmarking 2010: Presentation of the Noiseless Functions*, Working Paper 2009/20, compiled April 13, 2013, *http://coco.lri.fr/downloads/download13.09/bbobdocfunctions.pdf*

[9] Finck, S., Hansen, N., Ros, R., Auger, A.: *Real-Parameter Black-Box Optimisation Benchmarking 2010: Presentation of the Noisy Functions*, Working Paper 2009/21, compiled April 13, 2013, *http://coco.lri.fr/downloads/download13.09/bbobdocnoisyfunctions.pdf*

[10] Finck, S., Ros, R.: *COCO (COmparing Continuous Optimisers) Software: User Documentation*, compiled April 13, 2013, *http://coco.lri.fr/downloads/download13.09/bbobdocsoftware.pdf*

[11] Genetic and Evolutionary Computation Conference homepage, *http://www.sigevo.org/gecco-2013/*

[12] Hansen, N., Auger, A., Finck, S., Ros, R.: *Real-Parameter Black-Box Optimisation Benchmarking: Experimental Setup*, compiled April 13, 2013, *http://coco.lri.fr/downloads/download13.09/bbobdocexperiment.pdf*

[13] Huyer, W., Neumaier, A.: *Benchmarking of MCS on the Noiseless Function Testbed*, Manuscript, 2009, *http://www.mat.univie.ac.at/ neum/ms/mcs_exact.pdf*

[14] Huyer, W., Neumaier, A.: *Global Optimization by Multilevel Coordinate Search*, Journal of Global Optimization, Vol. 14, pp. 331–355, 1999

[15] Jones, D.R., Perttunen, C.D., Stuckman, B.E.: *Lipschitzian optimization without the Lipschitz Constant*, Journal of Optimization Theory and Applications, Vol. 79, pp. 157–181, 1993

[16] Pál, L., Csendes, T., Markót, M. C., and Neumaier, A.: *Black-box optimization benchmarking of the GLOBAL method*, Evolutionary Computation, Vol. 20, pp. 609–639, 2012

[17] Parameter Setting and Tuning of Algorithms, part of BBOB 13.09 documentation, *http://coco.lri.fr/COCOdoc/bbo_experiment.html#sec-tuning*

[18] Pintér, J. D.: *Global Optimization in Action*, Kluwer Academic Publishers, Dordrecht, 1996

[19] Pintér, J. D.: *LGO: A program system for continuous and Lipschitz optimization*, in Developments in Global Optimization, Editors: Bomze, I.M., Csendes, T., Horst, R. and Pardalos, P.M., Kluwer Academic Publishers, Dordrecht, pp. 183–197, 1997

[20] Pintér, J. D.: *Global optimization: Software, test problems, and applications*, in Handbook of Global Optimization, Volume 2, Editors: Pardalos, P. M. and Romeijn, H. E., Kluwer Academic Publishers, Dordrecht, pp. 515–569, 2002

[21] Pintér, J. D., Editor: *Global Optimization – Scientific and Engineering Case Studies*, Springer Science + Business Media, New York, 2006

[22] Pintér, J. D.: *Software development for global optimization*, in Global Optimization: Methods and Applications, Fields Institute Communications Volume 55, Editors: Pardalos, P.M. and Coleman, T. F., American Mathematical Society, Providence, RI, pp. 183–204, 2009

[23] Pintér, J. D.: *LGO – A Model Development and Solver System for Global-Local Nonlinear Optimization*, User's Guide (Current version: June 2013), Pintér Consulting Services, Inc. Canada; www.pinterconsulting.com, 2013

[24] Pintér, J. D., Horváth, Z. *Integrated experimental design and nonlinear optimization to handle computationally expensive models under resource constraints*, Journal of Global Optimization, Vol. 57, pp. 191–215, 2013

[25] Pintér, J. D., Kampas, F. J.: *Benchmarking nonlinear optimization software in technical computing environments: Global optimization in Mathematica with MathOptimizer Professional*, TOP (An official journal of the Spanish Society of Statistics and Operations Research), Vol. 21, pp. 133–162, 2013

[26] Powell, M. J. D.: *The NEWUOA software for unconstrained optimization without derivatives*, in Large Scale Nonlinear Optimization, Editors: Di Pillo, G. and Roma, M., Springer Science + Business Media, New York, pp. 255–297, 2006

[27] Price, K.: *Differential evolution vs. the functions of the second ICEO*, in Proceedings of the IEEE International Congress on Evolutionary Computation, pp. 153–157, 1997

[28] Raw data files of BBOB 2012,
http://coco.lri.fr/BBOB2012/rawdata/

[29] Raw data files of BBOB 2013,
http://coco.lri.fr/BBOB2013/rawdata/

[30] Rios, L. M., Sahinidis, N. V.: *Derivative-free optimization: a review of algorithms and comparison of software implementations*, Journal of Global Optimization, Vol. 56, pp. 1247–1293, 2013, DOI: 10.1007/s10898-012-9951-y

[31] Ros, R.: *Benchmarking the NEWUOA on the BBOB-2009 function testbed*, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, ACM, pp. 2421–2428, 2009, ISBN: 978-1-60558-505-5, DOI: 10.1145/1570256.1570338