How Difficult is Nonlinear Optimization?
A Practical Solver Tuning Approach, with Illustrative Results

János D. Pintér
Pintér Consulting Services, Inc., Canada www.pinterconsulting.com janos.d.pinter@gmail.com

Submitted for publication: June 2014.

Abstract

Nonlinear optimization (NLO) *per definitionem* covers a vast range of problems, from trivial to practically intractable. For this reason, it is impossible to offer "guaranteed" advice to NLO software users. This fact becomes especially obvious, when facing unusually hard and/or previously unexplored NLO challenges.

In the present study we offer some related practical observations, propose a simple heuristic approach, and then suggest corresponding option settings for use with the Lipschitz Global Optimizer (LGO) solver suite. LGO serves for general – global and local – NLO. The LGO option settings proposed here are directly related to the "expectably sufficient" computational effort to handle a broad range of NLO problems. These option settings are then evaluated experimentally, by solving a collection of widely used NLO test problems which are based on various real-world optimization applications and academic challenges. We also include illustrative results for several well-known scalable optimization problems which are scientifically relevant and increasingly difficult as the size of the model-instances grows.

Based on our computational test results, it is possible to offer careful guidance to LGO users, and – arguably, *mutatis mutandis* – to users of other NLO software products with a similarly broad mandate.

Keywords

Nonlinear (global and local) optimization; LGO solver suite; LGO option settings; NLO software testing and benchmarking; Test model collections; Numerical test results.

1. Introduction

Nonlinear optimization (NLO) encompasses a tremendous range of models, from (almost) trivial to (practically) intractable. Therefore, strictly speaking, it is not possible to offer "universally valid" advice to NLO software users regarding the necessary solver effort to handle "all possible" problems. This fact becomes relevant, when facing unusually hard and/or previously unseen challenges – whether arising from some real-world application or from some popular academic exercise.

To give a set of examples from diverse application areas – based on our own experience – we mention computational physics and chemistry models (Pintér, 2001; Stortelder, de Swart and Pintér, 2001); laser design (Isenor, Pintér and Cada, 2003); radiotherapy (Tervo, Kolmonen, Lyyra-Laitinen, Pintér and Lahtinen, 2003); object packings and related logistics (Kampas and Pintér, 2006; Pintér and Kampas, 2006; Castillo, Kampas, and Pintér, 2008; Pintér and Fasano, 2015); automotive component design (Goossens, McPhee, Pintér and Schmitke, 2007); surveillance network design (Gammon, Pintér and Schwartz, 2007); the operation of integrated oil and gas production systems (Mason, Emelle, van Berkel, Bagirov, Kampas and Pintér, 2007); fuel processing technology

analysis (Pantoleontos, Basinas, Skodras, Grammelis, Pintér, Topis and Sakellaropoulos, 2009); the calibration of artificial neural networks (Pintér, 2012); experimental design combined with optimization with a view towards multi-disciplinary engineering projects (Pintér and Horváth, 2013); financial modeling and optimization (Čağlayan and Pintér, 2013); environmental engineering (Deschaine, Lillys, and Pintér, 2013; Pintér and Satish, 2014); space engineering (Fasano and Pintér, 2013), and many other areas. For a broad range of further NLO applications – including in-depth case studies – consult e.g. (Pintér 1996, 2002, 2006).

In spite of the understandable difficulty of providing "always valid" guidance, users of specific software products rightly expect – and can greatly benefit from – practical advice related to solver option settings (tuning). This fact serves as a key motivation for in-depth NLO software testing and benchmarking studies.

In the present study we offer simple, partially heuristic observations and corresponding solver option settings for a specific NLO software product. These observations and the resulting option settings are then validated experimentally, by solving a collection of widely used test problems. Our current model collection is mostly based on a range of typical real-world optimization applications retrieved from test suites and/or received from our software users. We also include some scalable optimization challenges which are scientifically relevant and – as a rule – become increasingly difficult to solve as the size of the model-instance considered increases. The test problem collection has been solved by the current implementation of the Lipschitz Global Optimizer (LGO) solver suite for global and local NLO (Pintér, 1996, 1997, 2005, 2009, 2014).

Based on our in-depth test results, it is possible to offer careful guidance to LGO users. Our conclusions are related to the "expectably sufficient" computational effort to handle a range of NLO problems. Arguably, *mutatis mutandis*, similar advice can be put to good use regarding other NLO software products with a similarly broad mandate and conceptually similar (global scope) solver strategies.

Following this Introduction, Section 2 presents a concise general NLO model form, with a few pertinent technical notes. Section 3 highlights the topic of NLO software testing and benchmarking. Section 4 briefly reviews the key algorithmic features of the LGO solver suite. A simple-to-use heuristic approach to set the model-dependent search effort in LGO is described in Section 5. Next, in Section 6 a summary of results is presented on the basis of solving a set of box-constrained test problems. Section 7 summarizes our results for general constrained NLO problems; here we also include a small set of illustrative results related to a class of scalable models. The conclusions presented in Section 8 are followed by acknowledgements and a list of key references. Our numerical results obtained by solving the complete current test model collection with a set of suitable LGO option settings are summarized in Appendices 1 and 2.

2. The Continuous Nonlinear Optimization Model

In order to introduce a concise "generic" NLO model form, we shall use the following symbols and assumptions:

$x \in \boldsymbol{R}^n$ n-dimensional real-valued vector of decision variables;

$f : \boldsymbol{R}^n \to \boldsymbol{R}$ continuous (scalar-valued) objective function;

$D \subset \boldsymbol{R}^n$ set of feasible solutions, a proper subset of $\boldsymbol{R}^n$.

The feasible set D is assumed to be non-empty, closed and bounded: D is defined by

$l \in \boldsymbol{R}^n$, $u \in \mathbf{R}^n$ finite (component-wise) lower and upper bounds of x; and

$g : \boldsymbol{R}^n \to \boldsymbol{R}^m$ (optionally defined or absent) m-vector of continuous constraint functions.

Applying these notations, we consider the following nonlinear optimization model:

(1) $\min f(x)$
$x \in D.$

Here the feasible set is defined by

(2) $D := \{x: l_i \le x_i \le u_i \text{ for } i=1,\ldots,n;\ g_j(x) \le 0 \text{ for } j=1,...,m\}$.

Evidently, the concise model formulation (1)-(2) covers numerous special cases under additional structural specifications. To clearly address the NLO model-class, we will assume that in (1)-(2) f and/or some components of g are *nonlinear* functions.

Let us point out the absence of the frequently postulated model convexity assumptions: convexity would be implied (for instance) by requiring that f and all components of g are convex functions. In absence of such assumptions, certain instances of (1)-(2) may well be multi-modal, i.e. they could have multiple optima. In such cases, one is typically interested in finding the *globally optimal solution* (set) – as opposed to finding one of the (possibly numerous) *local optima*. It is worth emphasizing that *all* real-world application examples listed in Section 1 lead to non-trivial *global* optimization problems. To differentiate, we will use the abbreviation GO for global optimization and LO for local optimization (problems), both being sub-classes of NLO.

The analytical conditions preceding the NLO model statement guarantee that the set of global solutions to (1)-(2) is non-empty. The postulated model structure also supports the application of suitably designed *globally convergent* deterministic and/or stochastic optimization algorithms. For related technical details that are outside of the scope of the present discussion, consult e.g. Pintér (1996). For further comprehensive expositions regarding global optimization models and algorithms, consult also Horst and Pardalos (1995), Pardalos and Romeijn (2002).

In spite of the stated general theoretical (existence and global convergence) guarantees, the numerical solution of NLO – especially of GO – model instances can become a real challenge, even for small models. To illustrate this point, see Figure 1 that displays a set of GO model instances. Observe that the models visualized are very small (n=2), and are defined without considering general constraints (m=0). Without specific model-instance related knowledge or insight, all of these models – and, of course, great many others – can be seen as a possible NLO problem instance to solve. Increasing the model (variable) dimension n can make such models – in a well-defined theoretical sense – *exponentially* more difficult to handle. For related in-depth discussions, we refer again to Horst and Pardalos (1995), Pintér (1996), Pardalos and Romeijn (2002). This exponential "curse of dimensionality" spells trouble for solver strategies that are aimed to handle all possible model-instances with a deterministic guarantee of precision.

Clearly, the added consideration of general constraints (m) can make NLO problems even more difficult to solve. To illustrate this aspect, see Figure 2 which shows a feasible solution to an instance of the following packing problem: for a collection of c circles with arbitrarily given radii, find a non-overlapping configuration within an embedding circle. (Placing two or more of the c circles into each other is not allowed, of course.) The eventual objective in this model-class is to minimize the radius of the container circle. For c = 20 circles (see the instance shown by Figure 2), the number of decision variables is $n = 2c + 1 = 41$, and the number of general constraints is $m = c\,(1 + (c–1)/2) = 210$: consult (Pintér and Kampas, 2006; Castillo, Kampas and Pintér, 2008).

Notice that the number of *non-convex* constraints in this model-class grows as a quadratic function of the model size c. This is an example of a *scalable* GO model-class: its model instances become rapidly more challenging as c increases: we will use this example later on when reporting illustrative numerical results.

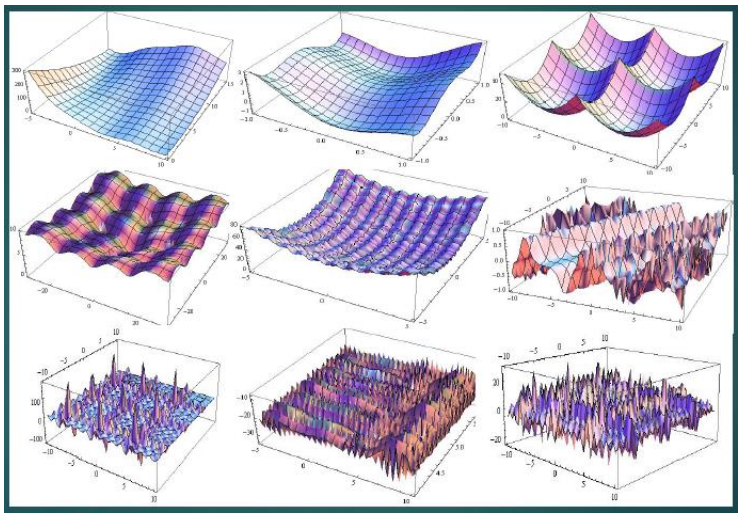

Figure 1 Box-constrained GO model instances (n=2, m=0)

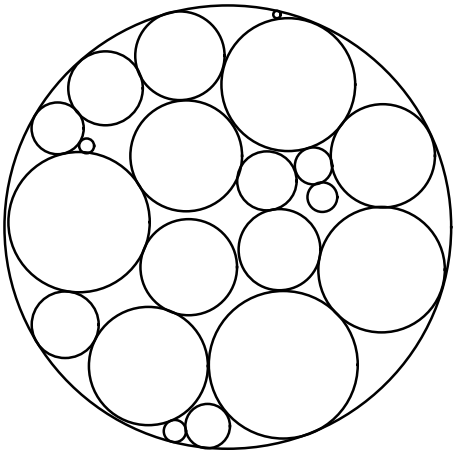

Figure 2 A general constrained GO model instance (n=41, m=210)

As the examples shown by Figures 1 and 2 illustrate, the solution of many NLO problems requires global – as opposed to local – scope search. Although in many important cases one can formulate and use models with a guaranteed "easy" (provably convex or uni-modal) structure, or one could have access to a "sufficiently close guess" of the global solution of a multi-modal problem, these "easy" situations are far from being the rule – in fact, in the real-world of NLO applications they may even be the exception. This is definitely the case to consider when the researcher (model developer and software user) faces a previously unsolved NLO problem type, especially so if the problem has a partially or completely "black box" structure. Such situations often arise in solving various engineering and scientific optimization problems as illustrated by several examples mentioned in Section 1.

3. Software Tests and Benchmarking

As noted earlier – given the vast range of possible NLO applications – it is impossible to offer guaranteed advice to NLO software users, along the lines of the optimistic statement shown below:

"Use software X with the following fixed option settings […options listed here…] and you will always succeed in solving your NLO problems".

To be a bit more quantitative regarding this "promise", success means that a sufficiently precise numerical solution of the NLO problem at hand can be found, applying pre-specified option settings and computational effort such as the maximal number of model function evaluations and/or a given runtime limit. Obviously, we are not talking here about "successful" methods based on completely exhaustive global search strategies: such methods do offer guarantees, but their computational costs can become prohibitive in the context of many real-world applications.

Notwithstanding this general *caveat*, NLO software users rightly demand – and can significantly benefit from – useful general guidance. Such guidance can (or sometimes should) be based on extensive numerical experience and tests. Tests and benchmarking studies provide an objective approach to evaluate the overall robustness and efficiency of optimization software products, by using these to handle a given collection of test problems in a realistic and objective (completely reproducible) manner.

For popular examples of test problems and benchmarking studies in the general NLO context, consult e.g. Hock, and Schittkowski (1981), Moré, Garbow and Hillström (1981), Schittkowski (1987), Averick and Moré (1991), Dolan, Moré, and Munson (2004), Schittkowski (2008), Moré and Wild (2009), with many further references therein to the original model sources. With respect to GO test models and benchmarking, cf. e.g. Floudas *et al.* (1999), Pintér (2002), Ali, Khompatraporn and Zabinsky (2005), Khompatraporn, Pintér and Zabinsky (2005), Neumaier, Shcherbina, Huyer and Vinkó (2005), Pintér and Kampas (2013), Rios and Sahinidis (2013), Hatwágner and Pintér (2014) – again, with numerous further topical references therein.

In the present study we have used models selected primarily from the following sources: Hock and Schittkowski (1981); Moré, Garbow and Hillström (1981); Schittkowski (1987); Dolan, Moré, and Munson (2004); Ali, Khompatraporn and Zabinsky (2005); Schittkowski (2008); Pintér and Kampas (2013). Most of these models have been widely used also by other researchers; and many of the models selected have a practical (engineering or natural science based) background. We also added several of our own tests, including scalable models and problems received from software users.

4. The LGO Solver Package for Global and Local Nonlinear Optimization

As illustrated by Figures 1 and 2, the proper solution of many NLO problems requires GO methodology. In addition to this fact, many of the actual NLO case studies listed in Section 1 also have an inherent "black box" feature. That is, some of the model functions f and g are evaluated by a numerical procedure that − in a step-wise model development and testing framework, as well as in many actual applications − may be subject to frequent changes and modifications. Similar situations often arise in the practice of NLO: their handling then requires the use of flexible solver technology, without the requirement for in-depth model related information that could be difficult or impossible to obtain. To give an example, software users may have to solve proprietary models, and may be unable to share more information with the solver (and its developer) than function values $f(x)$ and $g(x)$ returned by the "model box", for input arguments x that are sequentially generated by the solver engine.

Developed since the late 1980s, the Lipschitz Global Optimizer (LGO) solver suite has been designed with these requirements in mind, in order to support also the solution of "black box" problems, in addition to "standard" analytically defined models. The overall design of LGO is based on the flexible combination of several nonlinear optimization algorithms, each with corresponding theoretical global and local convergence properties.

It should be noted that the name LGO corresponds (only) to the original global solver embedded in the software: in fact, even this component uses only model function values, without requiring exact (typically unknown) Lipschitz-continuity information. Hence, LGO can indeed handle models with merely continuous structure, and its operations are based on directly sampled model function values.

Now we will briefly describe the overall structure of LGO. LGO includes a local solver (LS) which precedes all global search options. LS can be started either from an initial solution point x provided by the user, or from a default setting used by LGO. The LS mode can be also used without a preceding or subsequent global search phase. It is worth pointing out that in many test models – with literature based (typically good) initial solution guesses – the first LS phase of LGO already finds the solution. This is not the case, however, in many difficult GO models – such as the circle packing model introduced earlier – in which a suitable initial solution guess may not be available.

Following the LS phase, two relatively quick global pre-solvers are launched in a sequence: each of these is followed by LS from the current best point (if an improved solution has been found). The overall purpose of these solver components is to provide a relatively good quality solution with "minimal" global search effort.

Next, one of three "exhaustive" global search options is invoked: the methods to choose from are branch-and-bound (BB), single-start (partially) randomized search (RS), and multi-start (partially) randomized search (MS). Each of BB and RS is followed by a LS phase, while each major MS step is followed by a corresponding LS phase.

Based on the above summarized solver options, LGO – as a stand-alone solver suite – can be used for both global and local NLO. All LGO solver components are implemented in derivative-free form: therefore they can proceed based only on model function evaluations.

Without going into further details in the present article, we refer to Pintér (1996) for the discussion of the theory leading to the global search options BB, RS and MS. Further in-depth discussions can be found e.g. in Horst and Pardalos (1995), Pardalos and Romeijn (2002), or – to cite a more recent work – in Locatelli and Schoen (2013). The LS method is a generalized reduced gradient algorithm implementation: for background, consult e.g. Edgar, Himmelblau and Lasdon (2001). The relatively inexpensive first global pre-solver is described in Pintér and Horváth (2013); the second one is a recently added (unpublished) heuristic strategy.

In numerical practice – that is, in computational resource-limited runs – each of LGO's "exhaustive" global search options generates a global solution estimate(s) that is (are) refined by the seamlessly following local search mode(s). This way, the expected result of using LGO is a global and/or local search based high-quality feasible solution that meets at least the local optimality criteria. (To guarantee theoretical local optimality, standard local smoothness conditions need to apply at least when LS is invoked.) At the

same time, one should keep in mind that no global − or, in fact, any other − optimization software will "always" work satisfactorily, with default settings and under resource limitations related to model size, time, model function evaluation, or to other preset usage limits. With this cautionary remark in mind, extensive numerical tests and a growing range of practical applications demonstrate that LGO and its platform-specific implementations can find high-quality numerical solutions to complicated and sizeable GO problems. A set of real-world optimization challenges handled by various LGO implementations have been cited in Section 1.

LGO is available for use with a range of compiler platforms (C/C++/C#, Fortran 77/90/95), with seamless links to several optimization modeling languages (AIMMS, AMPL, GAMS, MPL), MS Excel spreadsheets, and to the leading high-level technical computing systems Maple, *Mathematica*, and MATLAB. For algorithmic and implementation details not discussed here, we refer to Pintér (1996, 1997, 2002, 2005, 2009, 2014), with further pointers to specific implementations.

The structure of the compiler-based core LGO implementation is shown by Figure 3.

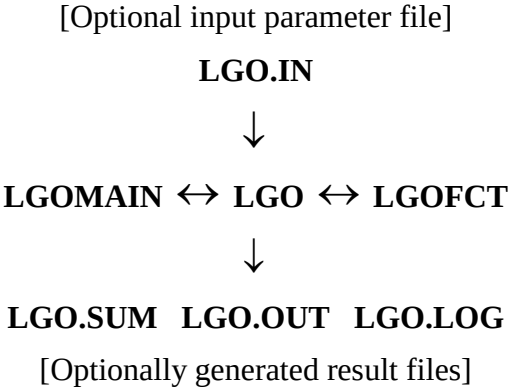

Figure 3 LGO application program structure

Here LGOMAIN is a driver program that defines or retrieves from an input file (see below) LGO's *static* calling parameters before activating LGO. Here the adjective static refers to model descriptor and solver option information that is defined or read only once and then remains unchanged during any given LGO run. LGOMAIN may also include additional user actions such as links to other program files and to external applications, to report generation and to the further optional use of LGO results.

LGOFCT serves to define the model objective f and constraint functions g that define the *dynamic* components of an optimization problem. Here dynamic means that this file will be called in every algorithmic iteration step of LGO, to evaluate its functions for a sequence of input variable arguments x. Again, this file may include calls to other application programs, in order to evaluate the model functions.

LGO.IN is an optionally used LGO input parameter (text) file that stores LGO's static calling parameters.

The source code files LGOMAIN and LGOFCT are to be compiled and linked to the LGO (object or dynamic link library) file. LGOMAIN calls LGO; the latter iteratively calls LGOFCT. LGO's operations can be (partially) controlled by the static input parameter file LGO.IN, or by changing LGOMAIN: this structure supports the repeated executions of LGO runs under various model specifications and/or solver option settings. Of course, LGOFCT can also be changed if necessary to test different model variants. LGOMAIN optionally reads LGO.IN when launched; in the opposite case all calling parameters are directly defined in LGOMAIN.

As indicated by Figure 3, LGO optionally generates result text files, on different levels of detail specified by the user. The first one of these files, called LGO.SUM, presents only a concise summary of the results obtained. The second file, called LGO.OUT, provides more detailed information related to the complete optimization process. The third file, called LGO.LOG, reports the entire sequence of all arguments x generated and the resulting function values f and g.

The key features of the LGO computational environment used in the present study are summarized below (noting that we use an average laptop PC as of 2014).

Hardware: 64-bit, x64 processor Intel Core i5-3337U CPU @ 1.80GHz, 16.00 GB RAM.
Operating system: 64-bit Windows 8.1.
Model development and (compilation / linking / run) test environment: Lahey/Fujitsu Fortran (LF95 release version 7.3) by Lahey Computer Systems (2013).
Optimization software: LGO solver suite (Pintér, 2014).

After selecting all LGO parameters, a single optimization run is executed in all cases. Repeated runs are not needed, since LGO produces identical runs with the same initializations and solver options on the same hardware and using the same compiler or modeling environment. Runs (with possibly somewhat changing numerical results) can be executed by changing the control options of LGO. These options make possible e.g. the selection of the global search mode applied (BB, RS, or MS), the use of different starting points in the local search mode, the use of different random seeds, and the allocation of computational resources and runtime.

For the purposes of this study, an integrated model function LGOFCT has been set up which supports the sequential solution of the test model collection(s) considered. In addition, two specific LGOMAIN driver programs were coded: these serve also to record the set of solutions and a summary of results, for each test problem collection. A summary of these results will be presented in Sections 6 and 7; more detailed illustrative results will be given in Appendices 1 and 2.

5. A Simple Quadratic Model Based Heuristic for Setting Solver Options

In this section we introduce and discuss a simple-to-use heuristic to estimate the expected numerical difficulty of NLO problems when we have no or limited prior knowledge. This

estimate will be used to advise users with respect to the maximal numerical effort allocated to the LGO solver suite.

Recalling (1)-(2), we shall consider an instance of the general NLO model:

$$\begin{aligned} &\min f(x) \\ &l \le x \le u \quad i = 1,\ldots,n \\ &g_j(x) \le 0 \quad j = 1,\ldots,m. \end{aligned}$$

If (1) is a nonlinear model indeed, then one can conjecture that to handle it is at least as difficult then to identify and then possibly solve a quadratic optimization problem (QOP), first without any consideration given to the (possibly present or absent) general constraints g. Hence, we will simply assume that $f(x)$ is modelled by a QOP.

An unconstrained QOP is defined by the n-vector x and the quadratic form

(3) $$f(x) = Q_n(x) = x^T A x + b^T x + c.$$

Since A is a symmetric matrix, in (3) the number of (presumably unknown) model parameters is

(4) $$EMC(n) = n(n+1)/2+n+1.$$

If we have to find these parameters by directly sampling f, then at least $EMC(n)$ function evaluations are necessary to achieve this. We skip the discussion of possible numerical difficulties, and simply assume that the sampling procedure would directly lead to finding all parameters of $Q_n(x)$. We also assume that in this case, we could find directly the unconstrained solution of min $Q_n(x)$.

Next, if instead of an unconstrained problem we have to deal with the NLO model (1)-(2) in which $m>0$, then – again, using a (perhaps rather massive) simplification – we can assume that each constraint g_j adds only as much difficulty as a new decision variable. This elementary argument leads to the following simple complexity estimate regarding the general model (1)-(2):

(5) $$EMC(n,m) = (n+m)(n+m+1)/2+(n+m)+1.$$

Again, if we had to find all parameters of this extended model – symbolically denoted by $Q_{n.m}(x)$ – then at least $EMC(n,m)$ function evaluations would be required (in the context of a properly defined extended model).

The above introduced estimates $EMC(n)$ and $EMC(n,m)$ can be then used to provide an approximation of the basic computational effort required to handle some NLO model, in the presence of mere box constraints or of more general constraints, respectively.

Specifically, if we know (or conjecture) that the model (1) is just an "easy" convex QOP indeed, then its perceived difficulty can be parameterized by a difficulty factor $df = 1$. For more complicated models, it would be advisable to use a factor $df > 1$, associating larger multipliers with more difficult models. Then $df \cdot EMC(n)$ and $df \cdot EMC(n,m)$ can be used to estimate the necessary global and/or local search effort within a solver like LGO. Informally, setting e.g. $df \cdot EMC(n)$ as the number of global search steps (function evaluations) is dictated by the user's belief that the model is not more difficult than *df* individual unconstrained QOPs which are put together in a single NLO model.

Notice that the proposed very simple heuristics is applicable even with rather minimal knowledge related to the NLO problem to solve. Users more often than not have some topical (application area related) experience and hence they can guess the expected difficulty of their model. If the model function evaluations are inexpensive, then there is no harm to set the factor *df* to large values; in the opposite case, it would be nice to guess well the "smallest possible" *df* that allows to solve most (by assumption, "randomly arising") NLO problem-instances.

Let us note at the same time that a theoretically correct (exponential complexity based) model would be practically useless – except for very small models – due to the "curse of dimensionality" aspect discussed briefly earlier. This remark hints at the possibility that in very difficult models that, in fact, have exponential complexity, the simple heuristic proposed above may not work with a hundred percent guarantee of success – while it still could support finding numerical solutions of reasonable quality. This point will be illustrated later on the circle packing model-class – a notoriously difficult scalable problem with instances which could well have an exponential number of locally optimal solutions.

To illustrate the suggested approach, recall Figure 1 which shows a collection of NLO (in fact, GO) models. If we would like to solve these models numerically, then a simple inspection indicates in the "first row" of the figure one could use e.g. *df* set between 1 and 10; in the "second row" *df* could be set between 10 and 100; and in the "third row" one could perhaps set *df* between 100 and 1000.

In the circle packing model-instance illustrated by Figure 2, n=41, m=210: hence, $EMC(n,m) = 255*128+256 = 32896$. Assuming that this is a rather difficult model-type, one can suggest e.g. df =100 or df = 1000, thereby implying a global search effort set to 3,289,600 to 32,896,000 model function evaluations. The factor *df* and $EMC(n,m)$ can be also used to define (automatically) the maximal number of unsuccessful global search steps, and the maximal number of local search steps: in fact, these are some of the key option settings in LGO. In our numerical experiments described here, we will simply set all three values equal to $df \cdot EMC(n)$ or $df \cdot EMC(n,m)$ depending on whether we deal with simple box-constrained or general constrained models.

In the next two sections, we present numerical results that are based on using the simple heuristics introduced above, solving a representative set of frequently used test problems.

6. Numerical Results for Box-Constrained Test Problems

At the time of writing this article, 70 test problems (TPs) have been selected from Moré, Garbow and Hillström (1981), Pintér (2001), Ali, Khompatraporn and Zabinsky (2005), Pintér and Kampas (2013); we also included a few of our own test models. In these tests – chosen by other researchers and by ourselves due to their specific features and challenging nature – the number of decision variables n varies between 2 and 50. Some of the test models are scalable: hence, in principle they can be used to generate infinitely many further test models as n can become arbitrarily large. Larger values of n, as a rule, lead to increasingly difficult scalable models.

A complete set of results for a single test run sequence is presented in Appendix 1. A summary of the results of several such test run sequences is presented in Table 1, followed by explanatory notes.

Table 1

LGO Results Using Various Solver Options and Difficulty Factors

70 Box-constrained NLO Test Problems

LGO Release 2014-05-06

	LS		BBLS		RSLS		MSLS	
DF	NSS	ARE	NSS	ARE	NSS	ARE	NSS	ARE
50	48	~250	57	0.080425	56	0.075448	57	0.070013
100	As above		64	0.019426	62	0.019655	64	0.022296
200	As above		63	0.030226	64	0.045406	62	0.023298
400	As above		66	0.006572	65	0.030884	66	0.001944
1000	As above		66	0.006449	65	0.030883	70	0.000001

Legend and notes to Table 1

LGO results using the LS, BBLS, RSLS and MSLS operational modes (recall Section 4).

DF The heuristic difficulty factor *df* set here between 50 and 1000 (recall Section 5).

RE Relative solution error calculated as $RE = |f^{LGO} - f^*| / (1+|f^*|)$; here f^* is the best known solution, and f^{LGO} is the numerical optimum estimate found by LGO.

ARE Average relative error, based on the relative error RE values for all TPs.

NSS Number of successful solutions: the solution found is accepted, if RE < 10^{-4}.

Let us remark that the RE of successful solutions is typically much smaller than 10^{-4}: consult Appendix 1 for numerical details.

Based on the results summarized by Table 1, the following conclusions can be drawn.

The local search (LS) option of LGO successfully solves 48 of the 70 TPs. Note that this arguably rather "optimistic" finding could strongly depend on the (typically rather good quality) starting points provided with the Moré-Garbow-Hillström models. As their article also suggests, the success rate could rapidly decrease with "less good" initial solution guesses. We can also see that increasing *df* has no effect on the LS results, since *df*=50 is already sufficient to find the solution of 48 models and the subsequent global search phases are never activated.

Let us also note here that in GO test models – as a rule – no starting point is provided, thereby expecting a global scope search. In such cases, by default LGO uses the mid-point of the interval defined by the box constraints as starting point for LS. In some academic GO models this point coincides with the solution: to avoid "instant solver gratification", we properly changed the initial solution in all such models. A similar approach has been followed also by Pintér and Kampas (2013): instead of pre-calibrated initial points interval mid-points or randomly generated initial solutions were used.

The global search options BBLS, RSLS and MSLS all give nearly monotonically improving results with increasing *df* values. Due to the rather complex overall design of LGO – including its internal resource allocation rules and the presence of stochastic sampling strategies – it is possible that more search effort could lead to somewhat less good results in certain cases and for certain option parameterizations. Note that this *numerical* finding is not in contradiction to the *theoretical* convergence guarantees which are all based on a properly defined infinite sampling procedure.

Both BSLS and RSLS seem to reach their (current) numerical limitations at solving 66 or 65 of the 70 TPs, even when *df* is increased. This finding serves as a hint towards improving the robustness of these solver options. Only the computationally most demanding MSLS solver option can successfully handle all 70 TPs when *df* is set to a rather high value of 1000. This is an apparent "overkill" for many problems (since LS already solves 48 of these); however, we want to provide LGO solver option settings without problem-specific tuning.

Let us remark finally that the runtime for the longest program run (solving all TPs by using MSLS with the setting *df* = 1000) is approximately 280 sec. In this case, the average normalized number of function evaluations (FE) is 1380 when each FE is scaled by *EMC*(*n*). Please see Appendix 1 for additional details.

7. Numerical Results for General Constrained Test Problems

For these tests, 60 models have been selected (at the time of this writing) from Hock and Schittkowski (1981), Floudas *et al.* (1999), Dolan, Moré and Munson (2004), Pintér and Kampas (2013); we included again a few of our own test models, and some other models received from LGO software users. In these TPs, the number of decision variables (*n*) varies between 2 and 98; the number of general constraints (*m*) varies between 1 and 55. Again, some of the test models chosen are scalable: hence, in principle they can be used to generate infinitely many further models in which *n* and *m* can become arbitrarily large. As a rule, this "up-scaling" leads to models of rapidly increasing difficulty.

A complete set of results for a single test run sequence is presented in Appendix 2. A summary of the results of several such test run sequences is presented in Table 2, followed by explanatory notes.

Table 2

LGO Results Using Various Solver Options and Difficulty Factors

60 General Constrained NLO Test Problems

LGO Release 2014-05-06

	LS		BBLS		RSLS		MSLS	
DF	NFS	NSS	NFS	NSS	NFS	NSS	NFS	NSS
50	54	36	58	45	57	46	58	51
100	55	37	59	47	58	48	59	54
200	As above		59	47	58	49	60	60
400	As above		58	46	58	49	60	59*
1000	As above		58	46	58	49	60	59*

Legend and notes to Table 2

LGO results using the LS, BBLS, RSLS and MSLS operational modes (recall Section 4).

DF The heuristic difficulty factor *df* set between 50 and 1000 (recall Section 5).

RE and ARE are defined and calculated as explained at Table 1. For brevity, these indicators and the corresponding values are not included here in Table 2. Please consult Appendix 2 for detailed numerical results.

NFS Number of feasible solutions: the solution found is accepted, if the maximal constraint violation error is less than 10^{-6}. By using an absolute number (namely, 10^{-6}) here, we tacitly assume that the models are properly (or rather, "not too badly") scaled. Model scaling issues are discussed elsewhere: consult e.g. the LGO manual (Pintér, 2014).

NSS Number of successful solutions: the solution found is accepted, if RE < 10^{-4}. Again, the RE of successful solutions is typically much smaller than 10^{-4}: see Appendix 2 for details.

* In these two cases (as indicated in Table 2), a solution with a fairly small error is found in a single rather difficult TP.

Based on the results summarized by Table 2, the following conclusions can be drawn.

All solver modes – including the fastest LS mode – find feasible solutions in 90% or more of all test runs, even when *df* is set only to 50. This is an important feature of LGO, since infeasible solutions may well be unacceptable in many NLO applications.

The local search (LS) option of LGO successfully solves 36 of the 60 TPs when *df* is set to 50, solving 37 TPs when *df*=100, but then it can't improve further its success rate (since the subsequent global search phases are not activated). Note again that this finding in general may be somewhat "optimistic" since it could strongly depend on the (typically rather good) starting points provided with the Hock-Schittkowski, as well as in some cases for the Floudas *et al.* (1999), and Dolan-Moré-Munson models. The success rate could become smaller with less suitable initial solution guesses.

The global search options BBLS, RSLS and MSLS give again nearly monotonically improving results – in terms of both NFS and NSS – with increasing *df*. (Please recall the note related to non-monotonic improvement at the discussion of Table 1.) Both BSLS and RSLS seem to reach their (current) numerical limitations at finding only 58 of 60 feasible solutions, and globally solving only 46 or 49 of the 60 TPs, when *df* =1000. Again, this finding points towards the need of improving the default robustness and efficiency of these solver options. Only the most demanding MSLS solver option can successfully handle (almost) all 60 TPs. (As noted above, MSLS finds a solution with a fairly small error in a single rather difficult TP.) The runtime for the longest program run (solving all TPs by using MSLS with *df* = 1000) is approximately 143 sec. In this case, the average normalized number of function evaluations (FE) is 1300 when each FE is scaled by *EMC*(*n*,*m*). Overall, this TP collection seems a bit easier than the bound-constrained set. Compared to the box-constrained results, we can also see that the model difficulty estimate *EMC*(*n*,*m*) seems to work fairly well – at least for the considered set of test models.

To illustrate the effect of scalable models on *default* LGO solver performance (without any added insight, modeling effort or solver tweaking), we cite next a set of illustrative results related to general (non-uniform size) circle packings. Recalling this problem-type

(cf. Figure 2), we will use the following specific model version inspired by a circle packing competition (Zimmermann, 2006).

Problem instance *(c)*: For a given (positive integer) c, find min $R(c)$ such that all circles with radii $r(i)=i$, $i=1,\ldots,c$ can be placed into a container circle with radius $R(c)$, in a pairwise non-overlapping arrangement. We can assume that $c>4$, since all smaller configurations have obvious optimal solutions.

Since the circle packing competition has been attended (also) by many computational optimization experts for approximately one year, we assume that the best numerical results published at the competition's website are rather close to the (theoretically unknown) global optima for $c=5,\ldots,50$.

For small configurations (for $c=5,\ldots,10$) LGO can solve Problem (c) with high precision, when using the factor $df = 100$. However, the quality $q(c)$ of this default (automatically derived) solution seems to deteriorate as c increases. To illustrate this issue, several examples are cited below:

$q(10)$~100%; $q(11)$~98.39%2; $q(15)$~98.29%; $q(20)$~97.26%; $q(25)$~96.72%,…

Of course, one could simply (and still heuristically) apply a sequence of factors $df = df(c)$ that will increase with increasing c. However, more substantially we think that these findings indicate that the numerical solution procedure would benefit from additional model-related insight – as opposed to just using LGO (or another NLO solver engine) in its default mode.

There exist many important, and similarly tough scalable problems of professional relevance. For example, we mention here the various potential energy models proposed by Coulomb, Fekete, Lennard and Jones, Morse, and Thomson: consult e.g. Pintér (2001), Stortelder, de Swart and Pintér (2001), Pintér and Kampas (2013) with topical references therein.

8. Conclusions

Nonlinear optimization applications originate from great many areas and they come in vastly different forms, including also very tough, unusual and/or brand new challenges. These facts of reality make it difficult – in fact, impossible – to give valid advice that will "always work" for all NLO software users and for all possible problems.

In the present study, we introduce some simple (general and partially heuristic) modeling observations and suggest corresponding key solver option settings applied to the LGO optimization software. The suggested option settings are then checked by solving a mid-size collection of widely used and (mostly) practically motivated test problems, including also several scalable optimization challenges. Based on our numerical results, it is possible to draw some careful conclusions and to offer advice to LGO users and – *mutatis mutandis* – to users of other NLO software products with a similarly broad mandate.

Using a fairly diverse set of both box-constrained and general constrained NLO test models, LGO seems to be able to solve these models, if its maximal global as well as maximal local search effort (i.e. the number of function evaluations) are set according to to $df \cdot EMC(n,m)$. Here $EMC(n,m) = (n+m)(n+m+1)/2+(n+m)+1$ (recalling also formulas (1) to (4)). The multiplier *df* can be chosen between 100 and 1000: lower values can be recommended for simpler problem-types, and higher for more difficult and instances. As seen and noted earlier, the higher *df* settings may be an "overkill" for simpler problems, but if overall robustness is preferred *vs.* efficiency, then even higher values can be used. Obviously, all similar advice should be verified and adapted to the actual models as much as possible. Model related insight will always remain very useful – whenever available.

Let us also note that – due to the relatively fast program execution for our currently used test problem sequences – it is possible to repeat these test runs in a few minutes whenever the LGO solver code (or *df* and other factors) are modified. This is regularly done indeed for quality assurance.

We plan to continue this line of research and plan its substantial extensions towards other model-classes. We also continue research work related to the subject of NLO under computational resource limitations.

9. Acknowledgements

The work summarized here is partially based on the findings of research conducted with numerous colleagues (approximately over the past 15 years): consult the list of related references cited in Section 1.

The author's research work on various NLO applications (as referred to in this article) has been partially supported by the following organizations:

Centre for Mathematics and Computer Science (CWI), Netherlands

Dalhousie University, Canada

Defence Research and Development, Canada

HydroGeoLogic, USA

Maplesoft, Canada

National Research Council, Canada

Özyeğin University, Turkey

Shell International Exploration and Production, Netherlands

Széchenyi István University, Hungary

University of Kuopio, Finland.

The preparation of this article and of a closely related invited talk presented at the DSOC Workshop (Széchenyi István University, May 2014) has been supported by the Hungarian Government and the European Social Fund. The related grant support has been received through the University's TÁMOP-4.2.2.A-11/1/KONV-2012-0012 Project.

Appendix 1

Both appendices are directly cited from the result files generated by LGO and its specific test program driver (LGOMAIN), with minimal editing for better readability.

LGO Solver Suite Results for a Collection of
Box-constrained Nonlinear Optimization Test Problems

LGO solver suite release version 2014-05-06

LGO run started at 2014- 5-30 18:30: 1

Legend

TM Test Model No.

MN Model Name

NV Number of variables

OV Known optimum value (or best known value)

OS Numerical optimum value returned by the solver

FE Number of model function evaluations

RT Solver runtime (complete execution time (including license verification at the first model run)

Results

TM	MN	NV	OV	OS	FE	RT
1	Ackley	3	0.0000000000	0.0000000000	14109	0.01
2	Becker-Lago	2	0.0000000000	0.0000000000	7236	0.00
3	Bohachevsky 1	2	0.0000000000	-0.0000000298	7574	0.00
4	Bohachevsky 2	2	0.0000000000	-0.0000000092	7461	0.00
5	Branin	2	0.3978870000	0.3978873577	6299	0.00
6	Cosine mixture	3	-3.0000000000	-2.9999999856	13100	0.01
7	Easom	2	-1.0000000000	-1.0000000000	6430	0.00
8	Exponential	3	-1.0000000000	-1.0000000000	12380	0.00
9	Gaussian	2	-1.2969500000	-1.2969540460	6432	0.00
10	Goldstein-Price	2	3.0000000000	3.0000000000	7474	0.00
11	Griewank 3d	3	0.0000000000	0.0000000000	13530	0.01
12	Hartman 3d	3	-3.8627820000	-3.8627822416	13718	0.01
13	Hartman 6d	6	-3.3223226000	-3.3223224245	33388	0.02

14	Hosaki	2	-2.3458000000	-2.3458115761	5947	0.00
15	KowalikOsborne	4	0.0001034770	0.0001034772	21314	0.01
16	Levy	3	0.0000000000	0.0000000000	13576	0.01
17	McCormick	3	-1.9132230000	-1.9132229550	13188	0.01
18	Meyer-Roth	3	0.0000435526	0.0000435526	19396	0.01
19	Michalewicz	2	-38.8502944800	-38.8502944674	1547	0.00
20	Miele-Cantrell	4	0.0000000000	0.0000000000	29537	0.02
21	Mod.Rosenbrock	2	0.0000000000	0.0000000000	8004	0.00
22	Neumaier 3	3	-7.0000000000	-7.0000000000	12589	0.00
23	Neumaier 4	4	0.0000000000	0.0000006864	30003	0.03
24	Paviani	10	-45.7784690857	-45.7784755294	57534	0.06
25	Pinter SinPrd 2d	2	-99.5005111694	-99.5005077849	5976	0.00
26	Powell BadScale	2	0.0000000000	0.0000000042	1881	0.00
27	Powell Quadratic	4	0.0000000000	0.0000000000	26321	0.01
28	Price	2	0.9000000000	0.8999999985	5682	0.00
29	Rastrigin 3d	3	0.0000000000	0.0000000000	13990	0.01
30	Rosenbrock 3d	3	0.0000000000	0.0000000016	19392	0.01
31	Saddle 2d	2	-0.3523860000	-0.3523860738	7071	0.00
32	Schaffer 1	2	0.0000000000	0.0000000000	7572	0.00
33	Schwefel 10d	10	-4189.8288700	-4189.8288727	27706	0.03
34	Shekel 5	4	-10.1532000000	-10.1531996791	23245	0.01
35	Shekel 7	4	-10.4029000000	-10.4029405668	22678	0.01
36	Shekel 10	4	-10.5364000000	-10.5364098167	21690	0.01
37	Shubert 2d	2	-186.730910000	-186.730908831	6633	0.01
38	Shubert 3	3	0.0000000000	0.0000000000	13164	0.01
39	6-hump Camel	2	-1.0316284535	-1.0316284535	7025	0.00
40	3-hump Camel	2	0.0000000000	0.0000000000	7478	0.00
41	Trefethen 4	2	-3.3068686475	-3.3068651647	7624	0.00
42	Wood	4	0.0000000000	0.0000000000	30009	0.01
43	Beale	2	0.0000000000	0.0000000000	8307	0.00
44	FreudensteinRoth	2	0.0000000000	0.0000000017	6569	0.00
45	BrownBadScale	2	0.0000000000	0.0000000000	4407	0.00
46	JennrichSampson	2	124.36218200	124.36218236	8236	0.01
47	Helical valley	3	0.0000000000	0.0000000000	16987	0.01
48	Bard	3	0.0082148770	0.0082148770	15419	0.01
49	Gaussian	3	0.0000000113	0.0000005797	10114	0.01
50	Box	3	0.0000000000	0.0000000000	15967	0.01

51	Brown-Dennis10	4	1.5590657373	1.5590657373	22917	0.03
52	Biggs exp 6	6	0.0000000000	0.0000000003	56003	0.19
53	Pinter ModCal3d	3	0.0000000000	0.0000004330	13330	0.02
54	Zangwill	3	0.0000000000	0.0000000000	12556	0.00
55	Engvall	3	0.0000000000	0.0000000061	16050	0.01
56	Cragg-Levy	4	0.0000000000	0.0000000000	28299	0.02
57	Chebyquad 9	9	0.0000000000	0.0000000000	95955	0.06
58	Osborne 1	5	0.0000546489	0.0000546938	42003	0.12
59	Osborne 2	11	0.0401377000	0.0401377457	67574	1.12
60	Meyer	3	87.945800000	87.948507677	5116	0.01
61	Watson 6	6	0.0022876705	0.0022876710	46601	0.03
62	Madsen	2	0.7731990565	0.7731990565	7036	0.00
63	Penalty1 10	10	0.0000708765	0.0000708765	92082	0.04
64	Lennard-Jones13	33	-44.32680138	-44.32680142	1190006	1.51
65	Logpotential25	47	-80.99751000	-80.99642624	2352002	100.12
66	CoulombPot25	47	243.81276030	243.81363316	2352023	81.36
67	PowersumPot25	47	-414.6246380	-414.6246239	2353200	80.94
68	CoulombPotv225	50	243.81276030	243.81276043	2652036	17.86
69	Spectroscopy	6	0.0000000000	0.0000000064	44060	0.87
70	Pulmonary Imp	5	0.2124598394	0.2124598393	36892	0.04

Summary of results for test cycle

Quadratic function model: estimated model complexity

modc=(nvars+ncons)*(nvars+ncons+1)/2+(nvars+ncons)+1

Multiplier df in search effort setting: 1000

Maximal global search effort set to df*modc

LGO operational mode: 3

Relative error tolerance for successful solution: 0.000100

Number of successful solutions: 70 of 70

Average relative error of solutions found: 0.000001

Average normalized no. of function evaluations (nfe): 1380

For each model, normalization is defined by nfe/modc

Total LGO solver runtime (seconds): 284.73

Further details are available upon request from the author.

Appendix 2

LGO Solver Suite Results for a Collection of
General Constrained Nonlinear Optimization Test Problems

LGO solver suite release version 2014-05-06

LGO run started at 2014- 5-31 9:20:46

Legend

TM --- Test Model No.

MN --- Model Name

NV --- Number of variables

NC --- Number of constraints

OV --- Known optimum value (or best known value)

OS --- Numerical optimum value returned by the solver

CV --- Maximal constraint violation at the solution found

FE --- Number of model function evaluations

RT --- Solver runtime (complete execution time (including license verification at the first model run)

Results

TM	MN	NV	NC	OV	OS	CV	FE	RT
1	Pressure Vessel	4	3	662.7790405	662.7790405	0.00000000	14403	0.00
2	Planar Truss	4	1	176659.214938	176659.214938	0.00000001	8403	0.00
3	Box Design	3	5	50.96507524	50.96507524	0.00000000	18002	0.01
4	Pinter Inf NLSE1	2	2	0.14807756	0.14807756	0.00000000	3090	0.00
5	Jussien	10	8	0.00000000	0.00000000	0.00000000	76003	0.03
6	Moennigmann	8	8	0.00000000	0.00000000	0.00000000	52174	0.03
7	Circuit Design	9	9	0.00000000	0.00000001	0.00000000	76002	0.08
8	Chem Process	9	6	-400.0000000	-400.0000000	0.00000001	54403	0.02
9	PHH ChemEq	5	5	0.00000000	0.00000000	0.00000000	26407	0.01
10	Pinter Inf NLSE2	5	5	0.03049235	0.03049235	0.00000000	26403	0.03
11	Stratified Sample	4	2	726.67935618	726.70448678	0.00000001	11204	0.00
12	Alkylation Proc.	15	16	-1.76499900	-1.76498384	0.00000000	165960	0.12
13	Engineering Dsn	4	4	-0.06578947	-0.06578947	0.00000000	18003	0.00
14	Hedar G1	13	9	-15.00000000	-15.00000000	0.00000000	110484	0.06
15	Hedar G4	5	6	-30665.53891	-30665.53891	0.00000001	22875	0.00
16	Pack10IdCircles	21	55	-0.26225892	-0.26225892	0.00000000	1206035	1.47
17	Pack10DiffCircl	21	55	22.00019301	22.00019301	0.00000000	1201202	1.53
18	KampasNoncx1	2	2	-1.41421356	-1.41421357	0.00000001	6003	0.02

19	LargestSmallP6	10	20	-0.67498142	-0.67498144	0.00000000	198402	0.28
20	Hock-Schittk62	3	2	-26272.51449	-26272.51451	0.00000000	8402	0.00
21	Hock-Schittk113	10	8	24.30620907	24.30620907	0.00000001	76020	0.03
22	HangingChain50	98	50	-0.45553991	-0.45553991	0.00000001	447014	317.49
23	MinPythTriplet	3	6	1.00000000	1.00000000	0.00000000	14410	0.00
24	NoncxQuad	6	8	1.80927500	1.80927508	0.00000000	48003	0.03
25	ConstrTrefethen4	2	2	-3.29229011	-3.29229011	0.00000000	6003	0.00
26	Hock-Schittk93	6	2	135.0759605	135.0759605	0.00000001	14371	0.00
27	HansenChMix	18	13	2.85246519	2.85246519	0.00000000	211468	0.14
28	Hock-Schittk37	3	2	-3456.00000	-3456.00000	0.00000000	8403	0.02
29	Hock-Schittk32	3	2	1.00000000	1.00000000	0.00000000	5729	0.00
30	Hock-Schittk33	3	2	-4.58578644	-4.58578644	0.00000000	8405	0.00
31	Hock-Schittk39	4	2	-1.00000000	-1.00000001	0.00000001	6127	0.00
32	Hock-Schittk40	4	3	-0.25000000	-0.24999770	0.00000000	14403	0.00
33	Hock-Schittk47	5	3	0.00000000	0.00000000	0.00000000	18002	0.01
34	Hock-Schittk56	7	4	0.00000000	0.00000000	0.00000000	31295	0.02
35	Hock-Schittk63	3	2	961.715172	961.715176	0.00000000	8404	0.00
36	Hock-Schittk64	3	1	6299.842412	6299.842412	0.00000001	6004	0.00
37	Hock-Schittk66	3	2	0.51816327	0.51816327	0.00000001	8403	0.02
38	Hock-Schittk71	4	2	16.8492172	16.8492058	0.00000000	11208	0.00
39	Hock-Schittk77	5	2	0.24150513	0.24150513	0.00000000	14411	0.00
40	Hock-Schittk78	5	3	-2.91967811	-2.91967786	0.00000000	18008	0.01
41	Hock-Schittk81	5	3	0.05394985	0.05395189	0.00000000	18004	0.00
42	Hock-Schittk104	8	6	3.95116343	3.95116343	0.00000001	44067	0.06
43	Hock-Schittk106	8	6	7049.330923	7049.720652	0.00000001	48013	0.01
44	Hock-Schittk108	9	13	-0.86602544	-0.86602540	0.00000000	110410	0.06
45	Hock-Schittk112	10	6	-47.76109086	-47.76109086	0.00000000	61355	0.06
46	Hock-Schittk18	2	2	5.00000000	5.00000000	0.00000000	3255	0.00
47	Hock-Schittk57	2	1	0.02845967	0.02845967	0.00000000	2004	0.00
48	Hock-Schittk100	7	4	680.6300573	680.6300574	0.00000001	31203	0.02
49	Hock-Schittk105	8	1	1138.394014	1138.394014	0.00000000	11004	0.45
50	Hock-Schittk111	10	3	-47.76109100	-47.76109101	0.00000001	42016	0.06
51	Stability1	4	7	0.34173955	0.34173955	0.00000000	31203	0.01
52	Stability2	5	8	0.81752905	0.81752905	0.00000000	42003	0.02
53	CxObjLinCon	50	50	0.00471018	0.00471018	0.00000000	2060403	5.33
54	KampasNoncx2	2	2	-1.41421356	-1.41421351	0.00000000	6003	0.00
55	SASSimpleNLP	3	2	1.00000000	1.00000000	0.00000000	5746	0.00

56	SASScalableNLP	25	11	-6.49876543	-6.49876543	0.00000000	281233	0.19
57	HilbertLinEqs	3	3	0.00000000	0.00000000	0.00000000	11206	0.00
58	KampasNoncx3	2	2	3.00000000	2.99999996	0.00000001	6003	0.00
59	KampasNoncx4	2	1	-24.94430978	-24.94430978	0.00000000	4002	0.00
60	MichalewiczExt	2	2	-21.64417091	-21.64417091	0.00000000	6002	0.02

Further details are available upon request from the author.

References

Ali, M.M., Khompatraporn, C. and Zabinsky, Z.B. (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, 31, 635–672.

Averick B.M. and Moré, J.J. (1991) The Minpack-2 test problem collection. *Technical report ANL/MCS-TM-157*, Argonne National Laboratory, Argonne, IL.

Çağlayan, M.O. and Pintér, J.D. (2013) Development and calibration of currency market strategies by global optimization. *Journal of Global Optimization* 56, 353-371.

Castillo, I., Kampas, F.J., and Pintér, J.D. (2008) Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research*, 191, 786–802.

Deschaine, L.M., Lillys, T.P., and Pintér, J.D. (2013) Groundwater remediation design using physics-based flow, transport, and optimization technologies. *Environmental Systems Research* 2, 2:6. Available for download at http://www.environmentalsystemsresearch.com/content/2/1/6.

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

Edgar, T.F., Himmelblau, D.M., Lasdon, L.S. (2001) *Optimization of Chemical Processes*. (2nd Edn) McGraw-Hill, New York.

Fasano, G. and Pintér, J.D., Eds. (2013) *Modeling and Optimization in Space Engineering.* Springer Science + Business Media, New York.

Floudas, C.A., Pardalos, P.M., Adjiman C.S., Esposito, W.R., Gümüş, Z.H., Harding, S.T., Klepeis, J.L., Meyer, C.A. and Schweiger, C.A. (1999) *Handbook of Test Problems in Local and Global Optimization.* Kluwer Academic Publishers, Dordrecht.

Hatwágner, M.F. and Pintér, J.D. (2014) Benchmarking the LGO solver suite within the COCO test environment. *Acta Technica Jaurinensis* 7 (2), 156–171.

Hock, W. and Schittkowski, K. (1981) *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems, No. 187. Springer, Heidelberg.

Gammon, M.A., Pintér, J.D., and Schwartz, B. (2007) Enhancement of optimization capability in TacTool using the Lipschitz Global Optimizer (LGO) program. *DRDC Atlantic TM 2006-167*, Dartmouth, NS, Canada.

Goossens, P., McPhee, J., Pintér, J.D., and Schmitke, C. (2007) Driving innovation: how mathematical modeling and optimization increase efficiency and productivity in vehicle design. *Technical Memorandum*, Maplesoft, Waterloo, ON.

Horst, R. and Pardalos, P.M., Eds. (1995) *Handbook of Global Optimization*, Volume 1. Kluwer Academic Publishers, Dordrecht.

Isenor, G., Pintér, J.D. and Cada, M. (2003) A global optimization approach to laser design. *Optimization and Engineering*, 4, 177-196.

Kampas, F.J. and Pintér, J.D. (2006) Configuration analysis and design by using optimization tools in *Mathematica*. *The Mathematica Journal*, 10 (1), 128-154.

Khompatraporn, Ch., Pintér, J.D. and Zabinsky, Z.B. (2005) Comparative assessment of algorithms and software for global optimization. *Journal of Global Optimization*, 31, 613–633.

Lahey Computer Systems (2013) Lahey/Fujitsu Fortran 95 (Release 7.3). Incline Village, NE.

Locatelli, M. and Schoen, F. (2013) *Global Optimization: Theory, Algorithms, and Applications*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Mason, T.L., Emelle, C., van Berkel, J., Bagirov A.M., Kampas, F.J. and Pintér, J.D. (2007) Integrated production system optimization using the Lipschitz Global Optimizer and the Discrete Gradient Method. *Journal of Industrial and Management Optimization*, 3 (2), 257-277.

Moré, J.J., Garbow, B.S. and Hillström, K.E. (1981) Testing unconstrained optimization software. *ACM Transaction on Mathematical Software* 7, 17–41.

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

Neumaier, A, Shcherbina, O., Huyer, W. and Vinkó, T (2005) A comparison of complete global optimization solvers. *Mathematical Programming, Series B* 103, 335–356.

Pantoleontos, G., Basinas, P., Skodras, G., Grammelis, P., Pintér, J.D., Topis, S., and Sakellaropoulos, G.P. (2009) A global optimization study on the devolatilisation kinetics of coal, biomass and waste fuels. *Fuel Processing Technology*, 90, 762–769.

Pardalos, P.M. and Romeijn, H.E., Eds. (2002) *Handbook of Global Optimization*, Volume 2. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (1996) *Global Optimization in Action*. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (1997) LGO − A program system for continuous and Lipschitz optimization. In: Bomze, I.M., Csendes, T., Horst, R. and Pardalos, P.M., Eds. *Developments in Global Optimization*, pp. 183-197. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (2001) Globally optimized spherical point arrangements: model variants and illustrative results. *Annals of Operations Research* 104, 213–230.

Pintér, J.D. (2002) Global optimization: software, test problems, and applications. In: Pardalos, P.M. and Romeijn, H.E., Eds. *Handbook of Global Optimization*, Volume 2, pp. 515-569. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (2005) Nonlinear optimization in modeling environments: software implementations for compilers, spreadsheets, modeling languages, and integrated computing systems. In: Jeyakumar, V. and Rubinov, A.M., Eds., *Continuous Optimization: Current Trends and Modern Applications,* pp. 147-173. Springer Science + Business Media, New York.

Pintér, J.D., Ed. (2006) *Global Optimization: Scientific and Engineering Case Studies.* Springer Science + Business Media, New York.

Pintér, J.D. (2009) Software development for global optimization. In: Pardalos, P.M. and T. F. Coleman, Eds., *Global Optimization: Methods and Applications*, pp. 183-204. *Fields Institute Communications Volume 55*. Published by the American Mathematical Society, Providence, RI.

Pintér, J.D. (2012) Calibrating artificial neural networks by global optimization. *Expert Systems with Applications* 39, 25–32.

Pintér, J.D. (2014) *LGO ─ A Model Development and Solver System for Nonlinear (Global and Local) Optimization. User's Guide.* (Current version) Distributed by Pintér Consulting Services, Inc., Canada; www.pinterconsulting.com.

Pintér, J.D. and Fasano, G., Eds. (2015) *Optimized Packings and Their Applications.* Springer Science + Business Media, New York. (To appear)

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

Pintér, J.D. and Kampas, F.J. (2006) *MathOptimizer Professional*: Key features and illustrative applications. In: Liberti, L., and Maculan, N., Eds. *Global Optimization: From Theory to Implementation*, pp. 263-279. Springer Science + Business Media, New York.

Pintér, J.D. and Kampas, F.J. (2013) 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) 21, 133-162.

Pintér, J.D. and Satish, M.G. (2014) Global vs. local optimization for model calibration in engineering studies. (Submitted for publication)

Rios, L.M., and Sahinidis, N.V. (2013) Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization* 56, 1247–1293.

Schittkowski, K. (1987) More test examples for nonlinear programming. *Lecture Notes in Economics and Mathematical Systems 182*. Springer, Berlin.

Schittkowski, K. (2008) An updated set of 306 test problems for nonlinear programming with validated optimal solutions – user's guide. Research Report, Department of Computer Science, University of Bayreuth.

Stortelder, W.J.H., de Swart, J.J.B. and Pintér, J.D. (2001) Finding elliptic Fekete points sets: two numerical solution approaches. *Journal of Computational and Applied Mathematics* 130, 205–216.

Tervo, J., Kolmonen, P., Lyyra-Laitinen, T., Pintér, J.D. and Lahtinen, T. (2003) An optimization-based approach to the multiple static delivery technique in radiation therapy. *Annals of Operations Research* 119, 205-227.

Zimmermann, A. (2006) Al Zimmermann's Programming Contests. Circle Packing. http://recmath.com/contest/CirclePacking/index.php.