Journal of Global Optimization (2005) 31: 613–633
DOI 10.1007/s10898-004-9971-3


# Comparative Assessment of Algorithms and Software for Global Optimization

CHAROENCHAI KHOMPATRAPORN[1], JÁNOS D. PINTÉR[2] and ZELDA B. ZABINSKY[1]
[1]*Industrial Engineering, Box 352650, University of Washington, Seattle, Washington, 98195-2650, USA (e-mails: ckhom@u.washington.edu, zelda@washington.edu)*
[2]*Pinter Consulting Services Inc., and Dalhousie University PCS, 129 Glenforest Drive, Halifax, NS, Canada B3M 1J2 (e-mail: jdpinter@is.dal.ca)*

**Abstract.** The thorough evaluation of optimization algorithms and software demands devotion, time, (code development and hardware) resources, in addition to professional objectivity. This general remark is particularly valid with respect to global optimization (GO) software since GO literally encompasses "all" mathematical programming models. It is easy not only to fabricate very challenging test problems, but also to find realistic GO problems that pose a formidable task for any algorithm of today and of tomorrow.
A report on computational experiments should ideally cover a large number of aspects: a detailed description and practical background of the models; earlier related work; solution approaches; algorithm implementations and their parameterization; hardware platforms, operating systems, and software environments; an exact description of all performance measures; report of successes and failures; analysis of solver parameterization effects; statistical characteristics for randomized problem-classes; and a summary of results (in text, tabular and/or graphical forms).
An extensive inventory of classical NLP and GO test problems, as well as more recent (and often much harder) test suites have been suggested. This paper reviews several prominent test collections, discusses comparison issues, and presents illustrative numerical results. A second paper will perform a comparative study using ideas presented here, drawing also on discussions at the Stochastic Global Optimization Workshop (held in New Zealand, June 2001).

**Key words:** Analysis and comparison of algorithms and software, Global optimization, Test problems and applications

## 1. Introduction

One of the very first questions to consider in formulating and solving a quantitative decision model is: what optimization approach and – eventually – software should be used? Ideally, we want to choose the method that would be most *suitable* for solving the problem at hand. The word "suitable" can reflect and imply different interests for different users. For instance, one practitioner may emphasize the speed of the algorithm to find a "good" solution, even though the solution found is not guaranteed to be the

optimum. Another expert may give more attention to rigorous guarantees and to the accuracy of the solution obtained by the algorithm, and so on.

Numerical experiments to compare algorithms and software is an important area complementing the theoretical analysis of optimization methods, since there is not a single universal algorithm that performs "best" for all possible categories of optimization problems and instances. In this paper, we will discuss computational testing issues related to comparing global optimization (GO) strategies and solvers.

Modern optimization algorithms typically utilize iterative techniques. These algorithms, in combination with recent advances in computer technology, allow practitioners to address large-scale and/or complex problems that could not have been solved otherwise.

It may be beneficial to first reflect on a definition of what constitutes an algorithm. Kronsjö [23] defines an algorithm as "a procedure consisting of a finite set of unambiguous rules which specify a finite sequence of operations that provides the solution to a problem, or to a specific class of problems." This definition reveals several key attributes that are worthwhile emphasizing.

- An algorithm must be rigorously and precisely defined, to eliminate ambiguity.
- An algorithm must provide a solution (even if not the best possible) to the problem, after a finite number of computational steps such as model function evaluations, and/or after a reasonable amount of time specified by the user. If there is simply no acceptable solution, then such cases need to be handled properly.
- An algorithm must be applicable to entire classes of problems, rather than just to a particular problem instance.

This list of main points is in general agreement with the suggestions by Arora [2, 3] and Bazaraa et al. [4] on the attributes of a good numerical optimization method. More specifically, they propose that a good algorithm has the following attributes:

- *Generality*. The algorithm should be insensitive to secondary details of the problem structure; that is, the algorithm should converge to a solution without any further restrictions on the structure of the problem instance (assuming, of course, that the given problem belongs to the scope of the algorithm).
- *Efficiency*. The algorithm should (i) keep the amount of calculations as small as possible so that the actual computation time is "minimal", or at least acceptable; (ii) require relatively few iterations and thus converge quickly; and (iii) be relatively insensitive to the initial starting point and other specifications.

- *Reliability*. The algorithm should be reliable and solve the given problem to a reasonable degree of accuracy specified by the user.
- *Ease of Use*. The application of the algorithm – at least with its default settings – should be relatively easy to understand, even by novice or less experienced users. The algorithm should have as few parameters that require explicit "tuning" as possible.

The attributes listed above are interrelated. For example, algorithms that are very sensitive to the selection of the tuning parameters are often problem dependent, and hence not sufficiently general. Furthermore, tradeoffs between these attributes are usually inevitable. When robustness and ease of use increase, efficiency typically decreases to some extent, and vice versa. That is, often a computational "price" (such as CPU time increase) has to be paid in order to obtain a robust and easy to use algorithm [50]. At the same time, an algorithm tailored to a specific class of problems with common structure is frequently more efficient for that particular problem class than an algorithm that could be applied to a more general class of problems, but a lot less efficient – or even unusable – when applied to problems outside that class. Thus, there certainly are tradeoffs between generality, efficiency, trustworthiness, and ease of use. For additional related discussions in the GO context, please consult Neumaier [34] and Pintér [39].

## 2. A Review of Comparative Characteristics

Once an algorithm is developed, the next phase is to demonstrate that it can be implemented in a practically usable form. This section discusses how researchers have been appraising the "goodness" of computational optimization algorithms.

In a fair comparison of optimization algorithms, there are many aspects that need to be addressed. The first and probably the most important issue is the selection of objective, empirical, and reproducible measures of merit.

### 2.1. MEASURES OF MERIT

Based on the key attributes of a good optimization algorithm as discussed above, below we propose guidelines for merit measures reflected by the following points.

#### 2.1.1. *Generality*

*Model size*. The complexity of algorithms is usually interpreted in relation to the size of the problems considered: the size is determined (as a first approximation) by the number of model variables and constraints. In general, as the model becomes larger, algorithms take a longer time to solve them; furthermore, it often is the case that once the model dimension or

the number of constraints reaches a certain algorithm-specific level, in practice the algorithm can no longer solve them on the given hardware platform [23]. A simple explanation of this experience is that the computational burden and the memory storage requirements, especially for GO problems, can dramatically increase with growing model size [52]. Although global optimization problems are known to be NP-complete [53], the hope is that practitioners will be able to address even sizeable GO problems in some efficient manner.

In order to acquire practical solutions to large dimensional global optimization problems – in addition to rigorous deterministic approaches – heuristic and stochastic methods are also considered. Such general approaches that have been successfully applied to other difficult problem types, such as combinatorial optimization, can often be adapted to continuous GO models. Measures such as the absolute or relative deviation from the optimum value (known in many test examples) that proved to be useful in evaluating integer programming methods can also be applied in the GO context.

*Configurations of local minima.* The number, location, region of attraction, and value of local solutions can all significantly affect the performance of GO algorithms. Problems with relatively few local minima are expected to be easier to solve than problems with many local minima. The spatial distribution of local solutions may also influence algorithm performance. If the local minima are concentrated in only a few areas on the feasible region, then the detection of one local minimum may easily lead to finding other local minima, whereas if the local minima are all scattered throughout the feasible region, then it can be far more difficult to find the global one. Furthermore, if the global optimum has a relatively small region of attraction, and/or there are several nearly identical (good quality) solutions, then these circumstances can make the solution procedure far more difficult.

Törn et al. [51] discuss the impact of distribution of minima on algorithm performance. The phrase *embedded global minimum* is used to describe the situation when the global minimum is close to some other local minimizers. Detecting these local minima often leads the algorithm to also detect the global minimum. The opposite situation is termed *isolated global minimum*: purely local search techniques can be rightfully expected to perform more poorly in the latter case.

#### 2.1.2. *Efficiency*

Computational time, normally reported as net program execution time or CPU time, is an important indicator that shows how an algorithm performs on a specific problem. However, as Kronsjö [23] and numerous others caution, the execution time of an algorithm can be greatly affected both by programming/implementation skills and by the hardware used. Skillful coding may not basically alter the underlying core algorithm, but it can

significantly influence the speed of program execution. Moreover, execution times of any two algorithms are not directly comparable when done on machines with different specifications.

With this in mind, it is often suggested that instead of reporting execution time one should report the number of model function evaluations needed by the algorithm to numerically converge to the optimum within a given degree of accuracy. Note however, that the number of the objective function evaluations also could depend on programming skills. Some algorithms may also have other computational requirements (such as finding the inverse of the Hessian matrix) that are not captured by solely counting direct function evaluations. Therefore, a joint report combining both execution time (relative to some widely accepted benchmark operation) and the number of function evaluations is perhaps a better approach.

So far we have discussed the *time complexity* of an algorithm, i.e. the time required to execute the algorithm. There are two other types of complexity that should also be considered, namely *space complexity* and *computational complexity*. Space complexity is the memory required by an algorithm to complete the entire execution. A smaller memory requirement is obviously preferred to a large memory requirement. It is possible that certain methods applied to high-dimensional problems imply huge memory requirements that exceed the limitations of most, if not all, computers. Therefore space complexity should be considered as a part of algorithm performance.

Computational complexity in practical terms refers to the number of arithmetic or logical operations that an algorithm requires to solve a given problem to a given precision. In numerical computations, *algebraic* and *analytic complexity* (two branches of computational complexity) should be distinguished. Algebraic complexity indicates a known bound on the number of arithmetic operations required by the algorithm to achieve the solution. This is not a practically usable measure in the case of GO problems, because theoretical convergence typically needs an infinite number of search points and function evaluations. By contrast, analytic complexity focuses on how much computational effort is needed to yield a solution within a certain degree of accuracy. Thus, in terms of computational complexity, a better algorithm is the one requiring fewer arithmetic operations to achieve a solution with a predetermined degree of accuracy.

There is also another type of complexity called *theoretical complexity*. Commonly, theoretical complexity is categorized by theoretical properties of convergence. For an algorithm to be rigorous, it must guarantee convergence by generating a sequence of points that converges to the optimum. Deterministic algorithms may have an absolute convergence property if they sample a dense set: the sequence of solutions converges to the optimum for arbitrary starting points [49]. Stochastic algorithms do not have absolute convergence; instead, they converge in probability to the global solution set.

The next issue to examine is theoretical convergence rate. It is usual to consider the order or rate of convergence as the iteration index tends to infinity, see Bazaraa et al. [4] or Brent [6]. This convergence concept reflects the behavior of the tail of the sequence and hence it is a local property. A higher order of convergence implies, at least in theory, greater speed of convergence. In global optimization, however, we must consider the entire convergence history, since algorithms may converge quickly at the tail, but slowly at the beginning.

Deterministic algorithms may use time until reaching the target set as a measure of speed of the overall algorithm, while stochastic algorithms may use the expected number of iterations until first reaching the target set as a similar analytical measure. Examples of analysis of stochastic algorithms that use the expected number of iterations until first reaching the target set are Pure Adaptive Search [55] and Hesitant Adaptive Search [7]. In contrast to the order of convergence (a local measure) the number of iterations to reach a target set may be considered as a global measure of overall convergence for the algorithm studied.

#### 2.1.3. *Reliability*

The reliability or "trustworthiness" of an algorithm is often characterized using both theoretical analysis and computational results. Exact deterministic GO methods [22, 33, 40], as a rule, provide guaranteed accuracy after a finite number of iterations, but require a theoretically infinite number of iterations for absolute convergence.

As mentioned earlier, stochastic global optimization algorithms guarantee the convergence to the solution only in probability. This implies that the results obtained in a finite number of iterations or a given time may not be guaranteed to be near to the optimum; furthermore, such methods can provide (at best) *statistical* bounds on the optimal value. Of course, one needs to balance speed, e.g. execution time or number of function evaluations, against the degree of assurance regarding the results. In certain cases and applications, strict solution guarantees are indispensable. In many other situations, however, practicality dictates the acceptance of good quality solutions obtained by a limited computational effort.

Törn and Žilinskas [52] suggested a comparison by using the success ratio $s/m$, where $m$ is the total number of times that the algorithm, started from random initial points, is applied to a GO problem, and $s$ is the number of cases in which the algorithm successfully finds the global optimum. Based on this concept, careful numerical experiments can be conducted and a quantitative comparison can be drawn among probabilistic algorithms.

Note that solution accuracy is also an important numerical issue, particularly for continuous optimization problems. Algorithms should be robust

against rounding errors which limit the accuracy of the solution. Examples illustrating the effect of rounding errors can be found in Brent [6].

#### 2.1.4. *Ease of Use*

The effort of preparing the input data and model formulation in order to use a given algorithm should also be taken into consideration [4]. An algorithm that needs extensive input data preparation, such as data sorting or complicated data format conversion, is less desirable because such preparation can be a time consuming process. In general, users want optimization software to be as simple as possible. Algorithms that are straightforward to use are more attractive than those that are difficult to comprehend at the developer or user level. Ease of understanding also leads to easier and proper implementation which enhances result reproducibility. Hence, a very important goal in algorithm and software development is that an average expert or a non-expert alike should be able to grasp the ideas underlying the algorithm without much difficulty.

From the practitioner's point of view, the ideal algorithm is basically a *black box* that will output the final solution without too much preparation required from the user. Therefore algorithms that depend upon a large number of externally set option parameters are – as a rule – less user friendly. Observe also that the introduction of too many algorithm parameters (which may require pre-calibration or "tuning" for new problems) indicates strong problem-dependency, and hence renders the algorithm less useful. Again, this point implies nontrivial compromises between user-friendliness and sophisticated implementations. A good compromise is often to provide preset or default parameterizations which will work acceptably well even for the inexperienced user, while advanced users can have the option of overriding these settings.

### 2.2. GLOBAL OPTIMIZATION MODEL TYPES AND CLASSIFICATIONS

Global optimization problems are truly heterogeneous. As noted previously, GO models encompass all of the usual categories in mathematical programming, including linear models as well as the broad nonlinear category.

There is no universally accepted categorization for nonlinear problems. As an example, Törn and Žilinskas [52] classified global optimization problems as follows:

*Type A*: Unconstrained global optimization problems
(A1) Solvable problems
(A2) General unconstrained problems

*Type B*: Constrained global optimization problems
(B1) Special form problems
(B2) General constrained problems.

This tentative classification is obviously rather simplified, and it can be refined. A far more detailed GO problem classification can be found in [20]. The electronically available articles [37, 38] also provide a concise, but fairly detailed nomenclature of model-types, as well as a review of the most frequently used solution approaches and software.

Törn et al. [51] attempted to categorize problems from a pragmatic solution point of view regarding the number of modes and the ease or difficulty in finding these modes. They use the following categories:

(a) unimodal
(b) easy multimodal
(c) moderately difficult multimodal
(d) difficult multimodal problems.

The degree of difficulty is determined by the *embeddedness* of the global minimum relative to the minimizers, and the probability of missing the region of attraction of the global minimum in a stochastic search based approach. Problems with isolated global solutions and with a smaller chance of finding the region of attraction to the global minimum are rightfully considered more difficult.

In the past decade there has been a very significant progress in GO methodology, but there are and will always remain models of extreme complexity. As the tutorial [39] emphasizes, it is not difficult to construct GO test problems which pose a tremendous challenge to any particular GO method, whether today or tomorrow. Although many practical GO problems are less "intimidating" than such purely mathematical constructions, practically motivated models can also be very difficult. For instance, a few years ago Ratschek and Rokne [41] analyzed a circuit design model described by a system of (seemingly not too complicated) nonlinear equations in just nine variables. The verified solution of this model to a specific significant accuracy took a collective work power of tens of workstations and several months of total runtime. Recent advances in global optimization techniques and the increase of computer power made it possible to solve the circuit design problem – as well as a large variety of complex engineering design problems – in a reasonable amount of time. Examples can be found in [1, 9, 16–18, 28, 35, 39, 42, 54].

### 2.3. TEST PROBLEMS

Perhaps the most comprehensive printed source of global optimization test problems is Floudas et al. [16] (a significantly expanded version of [17]).

Floudas and Pardalos [15] classify GO test problems into four main categories:

(i) quadratic programming problems
(ii) quadratically constrained problems
(iii) general nonlinear programming problems
(iv) real-world application problems.

Note that the problem-class (i) is contained by (ii); the latter is contained by (iii); while (iv) may contain elements from any of the classes above. The volume [16] classifies test problems into 14 (again, partly overlapping) categories all of which are mathematically related. Examples of prominent model categories in [16] are quadratically constrained problems, mixed-integer nonlinear models, semidefinite programming, and dynamic programming problems.

For several good GO test problem collections available on-line, see Mittelmann [29] and Neumaier [34]. New classes of test problems are also being proposed regularly: these include, for instance, test models by Schoen [48], Mathar and Žilinskas [27], and several randomized problem-classes discussed in Pintér [36].

Let us point out here the close theoretical connection between integer programming (IP) and continuous GO models. As it is known, IP models can be directly transformed into GO equivalents, since each disjunctive binary relation can be represented by a reverse convex continuous constraint (consult, for instance [21, 36]). This fact implies that, at least in theory, IP models may also serve to test GO strategies. For instance, interested readers may consult [25] as a good collection of problems closely related to the traveling salesperson model.

There is another significant issue to consider here: namely, the use of standard academic tests vs. real-world problems. The eventual goal of global optimization should include applicability to real-world problems which can be massively nonlinear, complex, and high-dimensional, and/or have an unusual structure. Hence, reporting results on a particular real-world problem or a small class of such problems may not be too useful for the purpose of comparison, unless the problem is of true significance, and the test conditions and results can be directly reproduced. At the same time, purely academic test problems are sometimes either a bit too simplistic or have a rather ''fabricated'' structure, although some of these models may well represent (possibly simplified) real problems.

To conclude this section, note that the selection of test problems in itself poses a difficult philosophical question: which problem suites can be chosen as the *true benchmark* that all GO algorithms should be tested against? For related discussions, see e.g. [30, 34, 39].

### 2.4. REPORTING TEST RESULTS

Most GO test related works (available in the published literature) do not report all types of merit measures mentioned earlier, but only a

combination of some of these measures. This is probably due to the very serious resource demands of properly detailed testing. Another natural issue is scientific objectivity. Törn et al. [51] mention that comparisons of algorithms in the literature are often not completely fair, due to some (almost unavoidable) subjectivity and parameter tuning to the test suite used. A frequently observed example is the selection of stopping criteria that are chosen in light of the known or pre-set solution. Such a choice would typically shed too favorable a light on the algorithm in question.

Table 1 lists criteria and other details that are often reported in the literature to indicate the performance of global optimization algorithms on test problems. Typically, there exist some pointers available regarding the motivation to select a particular test problem, or a class of problems. Test problems are often chosen because they are pertinent to some real-world applications. In other cases, they are used because comparable benchmark results exist from other algorithms. The model functional form, dimensionality, and feasible region are commonly reported, whereas the numbers of local and global minima may be unknown, especially in many practical problems and in complex test problems, or simply omitted from the report.

Clearly, the best function value found is a very important measure. There are many difficult problems, such as traveling salesperson problem instances, various molecular architecture models, and over-determined systems of equations, in which the optimal solution could be unknown: in such cases, the best function value found so far is of primary interest.

*Table 1.* Performance comparison aspects of GO algorithms

| Aspects | notes |
|---|---|
| 1. Test problem formulation | Practical motivation, or original reference |
| 2. No. of variables/constraints | Smaller, simpler models are often (much) easier |
| 3. Feasible region | Smaller region is often (but not always) easier |
| 4. No. of local minima | Fewer minima often make models easier, The case of embedded global optimum is easier |
| 5. No. of global minima | Easier, if only one of these needs to be found; more difficult, if several or all have to be found |
| 6. Best function value found | Closer to optimality is better |
| 7. CPU time | Faster is better |
| 8. No. of function evaluations | Fewer is better |
| 9. Accuracy | Higher is better |
| 10. Average no. of iterations required per replication (in multiple replications) | Fewer is better |
| 11. No. of replications | More is better |
| 12. Success rate | Higher the better |
| 13. Tuning parameters | Fewer is better, less sensitive is better |
| 14. Stopping criteria | May vary |
| 15. Platform | May vary |
| 16. Additional comments | Summary, recommendations can be included |

The CPU time and the actual number of model function evaluations seem to be the other most frequently reported measures in the literature. For a given hardware platform, one can compare the efficiency of various algorithms by comparing CPU times used by these algorithms. Direct CPU time comparisons are not recommended, when different platforms are used: in such cases, one may use scaled time units, based upon certain benchmark evaluations, but this may lead to (unavoidable) biases.

The accuracy of an algorithm is usually pre-set internally, or set by its user, via a list of options. A higher level of accuracy is generally preferable, but there has not yet been a common agreement on the ideal level of accuracy that should be used as the benchmark. In practice, the level of accuracy is dictated by the actual model and data.

The average number of major iterations is less often reported. As discussed earlier, one of the reasons is that this measure is dependent on the programming skills of the algorithm developer. Some algorithms are designed to use a large number of iterations, but during each iteration the objective function is evaluated only once, while other algorithms may exploit many objective function evaluations per major iteration, but need a lot fewer of such iterations. Practical applications often involve computationally expensive objective function calculations, which may dominate the computational overhead in a larger number of iterations. Thus, the total number of function evaluations is a widely accepted and relevant measure.

Note that given implementations of deterministic global optimization methods are expected to produce identical (reproducible) results. However, the success rate for deterministic methods is relevant, when some test settings such as the starting point could affect the quality of the solution.

The number of replications and success rate pertain more directly to stochastic global optimization (SGO) algorithms. Since SGO methods guarantee finding the global optimum only in probability, their robustness needs to be tested via properly chosen statistical tools. Note, however, that these methods can also be equipped with fixed or variable random seed mechanisms that enable (as an option) the generation of identical results. This feature is implemented in many professional solver systems.

Most numerical optimization algorithms have a few parameters that need to be ''tuned'' heuristically. Examples include the *mutation rate* in genetic algorithms, or the *cooling schedule* in simulated annealing algorithms. The stopping criteria used in one algorithm often differ from those of another algorithm. For example, one can mention a pre-assigned constraint satisfaction or Karush–Kuhn–Tucker condition satisfaction accuracy. Other pragmatic numerical criteria may include the maximum number of model function evaluations, the number of evaluations without ''noticeable'' improvement in objective function values, and/or a preset maximum program execution time. These parameters are usually decided

somewhat arbitrarily, making a fair comparison of algorithms more difficult.

A practical issue that adds to the complication of evaluating algorithms is the variety of available computer hardware and software platforms. As previously mentioned, program execution times are basically incomparable, unless the algorithms are implemented on the same hardware platform with the same configurations. This issue becomes even more complicated when algorithms can utilize parallel processing on several platforms. Further differences may appear even between identical and/or similar algorithm implementations when using different programming languages or compilers, not to mention the added burden of user-friendly but resource-intensive program interface features.

In many cases, the performance of algorithms is depicted graphically to illustrate their progress or convergence rate as a function of iterations for given test problems. For the SGO approach, the average and standard deviation characteristics as well as the best and worst cases encountered are also frequently reported when solving the same problem repeatedly. Such information can be useful in the statistical analysis of SGO methods. A more versatile procedure to directly compare algorithms was recently proposed by Dolan and Moré [12]: their approach is applicable not only to execution time, but also to other comparative measures discussed above.

Without going into details beyond the scope of this paper, observe that the criteria listed are often (at least partially) conflicting. Consider, for instance, the tradeoff between accuracy required and a pre-set maximum number of objective function evaluations. In such cases, concepts and techniques used in multi-objective optimization (specifically including the selection of non-dominated, Pareto optimal algorithms for a given set of test problems) can be brought to the subject. Such analysis can lead to insights related to the strengths and weaknesses of optimization algorithms. Additional references on reporting test results can be found on the web sites [29, 34].

## 3. Algorithm Implementations and Comparative Numerical Experiments

It is important to recognize the difference between an algorithm and its actual software implementation. Although the latter is basically a machine-readable form of the underlying algorithm, the software itself is not the algorithm. The effectiveness of a software implementation may not directly reflect the effectiveness of the corresponding algorithm, because the coding skills of the developer can greatly affect the performance of the software. However, in order to compare algorithms for a real-world setting, one needs to compare both the algorithms and their implementations, as opposed to the algorithms alone. Several important aspects of algorithm comparison have been discussed in the previous section. Here we will focus

on the issues arising from algorithm comparison through software. Part of this section is based on the work by Reklaitis et al. [42].

### 3.1. A BRIEF HISTORY OF COMPARATIVE ASSESSMENTS IN CONSTRAINED OPTIMIZATION

In 1968, Colville [8] made a pioneering attempt at comparing the performance of algorithms, by sending out eight test problems to developers of 30 nonlinear optimization codes. The participants were required to submit the result of the ''best effort'' on each problem and the corresponding program execution time. Characteristics of Colville's set of test problems are summarized in Table 2.

This work was a significant step towards establishing objective evaluation and comparison criteria. Eason [13] observed, however, that Colville's study contains three major flaws. First, the execution times collected in the study are not comparable because the effect caused by the difference in the compilers and platforms running the algorithms was not removed. Second, the participants could apply their codes as many times as they liked and only the best results were reported. Hence, if an independent investigator would like to apply the same code to the same problem, he or she may not be able to reproduce the reported results. Third, no two participants reported the same accuracy of their results since this aspect was not prespecified.

Eason and Fenton [14] later performed a comparative study of 20 optimization codes using 13 test problems, which are summarized in Table 3. Their study primarily focused on penalty-type methods and all of the computations were performed on the same computer. The major inadequacies of this study were due to (i) failure to include other powerful methods available at the time and (ii) shortcomings in the difficulty of the problems.

*Table 2.* Colville [8] problem set (source: [42])

| Problem name and/or source | Number of variables | Number of inequality constraints ($I$) | Number of equality constraints ($E$) | Total number of constraints ($I+E$) | Total number of bounds on variables |
|---|---|---|---|---|---|
| 1. Shell | 5 | 10 | 0 | 10 | 5 |
| 2. Shell | 15 | 5 | 0 | 5 | 10 |
| 3. Mylander/Res. Analysis Corp. | 5 | 6 | 0 | 6 | 10 |
| 4. Wood/Westinghouse | 4 | 0 | 0 | 0 | 8 |
| 5. Efroymson/Esso | 6 | 4 | 0 | 4 | 0 |
| 6. Huard/Electricite de France | 6 | 0 | 4 | 4 | 12 |
| 7. Gauthier/IBM France | 16 | 0 | 8 | 8 | 32 |
| 8. Colville/IBM | 3 | 14 | 0 | 14 | 6 |

*Table 3.* Eason and Fenton problem set (source: [42])

| Problem name and/or source | Number of variables | Number of inequality constraints ($I$) | Number of equality constraints ($E$) | Total number of constraints ($I+E$) | Total number of bounds on variables |
|---|---|---|---|---|---|
| 1. Colville #1 | 5 | 10 | 0 | 10 | 5 |
| 2. Post office parcel problem | 3 | 2 | 0 | 2 | 6 |
| 3. Colville #3 | 5 | 6 | 0 | 6 | 10 |
| 4. Colville #4 | 4 | 0 | 0 | 0 | 8 |
| 5. Rosenbrock | 2 | 0 | 0 | 0 | 4 |
| 6. Colville #5 | 6 | 0 | 4 | 4 | 12 |
| 7. Beightler/Journal bearing | 2 | 1 | 0 | 1 | 4 |
| 8. Siddall/Flywheel | 3 | 2 | 0 | 2 | 6 |
| 9. Siddall/Chemical reactor | 3 | 9 | 0 | 9 | 4 |
| 10. Mischke / Gear train | 2 | 0 | 0 | 0 | 4 |
| 11. Mischke/CAM design | 2 | 2 | 0 | 2 | 4 |
| 12. Eason/Mechanism synthesis | 4 | 0 | 0 | 0 | 8 |
| 13. Eason/Gear train | 5 | 4 | 0 | 4 | 3 |

Another major comparative study of nonlinear programming (NLP) methods was implemented by Sandgren [45, 46]. The experimental procedure employed in the study consisted of the following steps:

1. Assembly of solver codes and test problems.
2. Elimination of some (seemingly inferior) codes using 14 preliminary test problems.
3. Application of the remaining codes to the full suite of test problems.
4. Removal of the test problems on which fewer than 5 codes were successful.
5. Aggregation of the test results.
6. Preparation of individual and composite utility curves.

The purpose of step 2 in the experimental procedure was to avoid the possibility of wasted effort for codes that did not have the potential to solve the full suite of the test problems. (Solution times were far more significant than today, for the test models selected.) Sandgren was thus prioritizing reliability of the method over other measures of performance. Sandgren's test problem set included problems 2, 7, and 8 of the Colville problem set, all 13 problems from the Eason and Fenton problem set, eight problems from the Dembo problem set which is summarized in Table 4, a welded beam problem, and six industrial design application problems. Dembo's test suite [11] contains geometric programming problems which are a rather difficult class to solve by general NLP methods. After the removal of several test problems (in step 4 of the procedure), there were

*Table 4.* Dembo problem set (source: [42])

| Problem name and/or source | Number of variables | Number of inequality constraints ($I$) | Number of equality constraints ($E$) | Total number of constraints ($I+E$) | Total number of bounds on variables |
|---|---|---|---|---|---|
| 1. Gibbs free energy | 12 | 3 | 0 | 3 | 24 |
| 2. Colville #3 | 5 | 6 | 0 | 6 | 10 |
| 3. Alkylation process model (Bracken and McCormick) | 7 | 14 | 0 | 14 | 14 |
| 4. Practor design (Rijckaert) | 8 | 4 | 0 | 4 | 16 |
| 5. Heat Exchanger (Avriel) | 8 | 6 | 0 | 6 | 16 |
| 6. Membrane Separation (Dembo) | 13 | 13 | 0 | 13 | 26 |
| 7. Membrane Separation (Dembo) | 16 | 19 | 0 | 19 | 32 |
| 8. Beck and Ecker | 7 | 4 | 0 | 4 | 14 |

only 23 problems left. A summary of the codes used in Sandgren's study can be found in [42, 45, 46]. A thoughtful step in the study was that all print operations were removed from the basic iterative loop so that accurate execution time of the algorithm itself can be obtained.

The program execution time was the main performance measure used in Sandgren's study. He ranked the codes based on the relative number of problems solved within a series of specified time limits. The limits are based on a fraction of the average time for all codes on each problem. Moreover, the execution time to find the solution within a pre-specified accuracy for a problem was normalized by dividing it by the average execution time on that problem. This normalization allows direct comparison among algorithms. A generic example of Sandgren's ranking is shown in Table 5.

To read Table 5 , the second column from the left indicates that Code A could solve 7 problems in 25% of the average execution time, then 13 problems in 50% of the average execution time in the next column, and so on. Using the number of problems solved within a certain ratio of the average execution time as the comparison basis, fast codes can be easily identified. From the table, Codes A and B dominate by being consistently faster than Codes C and D. However, Code A is faster than Code B at the 50% of the average execution time level, while after the 75% level, Code B worked faster than Code A. Similar comparisons can be performed between other codes.

A major computational study of NLP codes summarized here was performed by Schittkowski [47]. His study includes 20 codes on 180 randomly generated problems with predetermined characteristics and multiple starting points. An important difference between the Schittkowski and Sandgren studies is that quadratic programming methods were also included in Schittkowski's work [42].

*Table 5.* Number of problems solved at accuracy level $\varepsilon = 10^{-4}$ within a percentage of normalized execution time

| Codes | Fraction of normalized execution time | | | | | |
|---|---|---|---|---|---|---|
| | 0.25[a] | 0.50 | 0.75 | 1.00 | 1.50 | 2.50 |
| Code A | 7 | 13 | 14 | 16 | 16 | 16 |
| Code B | 0 | 9 | 14 | 17 | 19 | 20 |
| Code C | 0 | 1 | 2 | 3 | 4 | 6 |
| Code D | 0 | 0 | 0 | 0 | 3 | 9 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

[a]Times the average execution time.

The codes were evaluated based upon the following nine criteria: efficiency, reliability, global convergence, ability to solve degenerate problems, ability to solve ill-posed problems, ability to solve indefinite problems, sensitivity to starting points, sensitivity to problem variations, and ease of use. Each code was applied to all 180 test problems. The data were collected in the same manner as in the Sandgren's study. Schittkowski then weighed the nine criteria according to Saaty's priority theory [43, 44] as outlined by Lootsma [26]. Schittkowski's weighting scheme is listed in Table 6. Using this weighting scheme as the measures he arrived at a ranking of the codes analyzed.

Several criteria in Schittkowski's weighting scheme have been mentioned in the beginning of this paper. Others may adopt different weighting factors or schemes, but Schittkowski's study does imply the need to consider several measures of performance in a detailed comparative study. An interesting observation from this work is that it supports the theory that the reliability of a code may reflect coding skills more than the quality of the algorithm itself [42].

Future computational comparison studies for global optimization algorithms can use the aforementioned and similar experiences. Specifically, the studies should consider multiple measures of performance, a comprehensive choice of test problems, and a well-defined procedure to conduct the evalu-

*Table 6.* Schittkowski's weighting scheme (source: [42])

| Criteria | Weights |
|---|---|
| 1. Efficiency | 0.32 |
| 2. Reliability | 0.23 |
| 3. Global Convergence | 0.08 |
| 4. Ability to solve degenerate problems | 0.05 |
| 5. Ability to solve ill-posted problems | 0.05 |
| 6. Ability to solve indefinite problems | 0.03 |
| 7. Sensitivity to slight problem variations | 0.03 |
| 8. Sensitivity to starting points | 0.07 |
| 9. Ease of use | 0.14 |

ations. Note in this context that a large number of further test problems, such as randomized systems of nonlinear equations, randomly generated clustering problem-instances, as well as detailed case studies, are discussed in [36]. For an extensive collection of other practically motivated tests and case studies, the readers may consult, for instance [1, 5, 9, 10, 17–19, 24, 28, 31, 32].

## 4. An Illustrative Comparison

In the companion paper [1], numerical results are presented comparing several different algorithms. In this paper, we present sample results to highlight the difficulty in comparing performance.

Consider three algorithm variations that have been tested on a specific global maximization problem. Figure 1 illustrates the best objective function value found by the $k$th iteration, averaged over five replications. By assumption, each replication started with a different random starting point and random seed (for the pseudo-random number generator), and the five replications were all considered successful, meaning that they achieved a point within the specified target set within 1000 function evaluations.

From Figure 1, variation A makes the best improvement early in the run, until approximately the 190th iteration when B starts to outperform A. After this, the progress of variation A slows down and eventually it performs worst (as shown up to 1000 iteration steps). Line 1 indicates where variation B passes variation A and similarly line 2 indicates where C passes A. As depicted in the figure, after about the 450th iteration, variations B and C converge at about the same rate, so it may be concluded that B dominates C.

Figure 1 reveals two important aspects in numerical comparison of algorithms. First, it emphasizes the importance of the stopping criteria. Often times algorithms are set to stop after a certain number of iterations. This number of iterations is usually arbitrarily decided by the user. In this

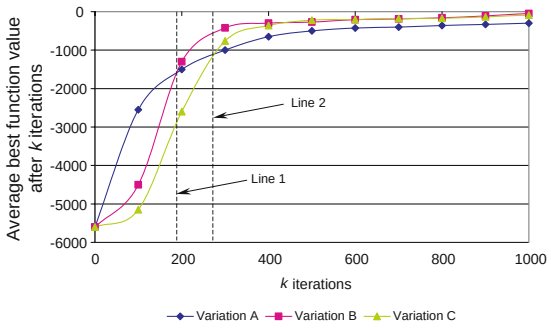


*Figure 1.* Illustrative performance comparison of three SGO algorithm variations.

example, if the stopping criterion was 100 iterations, we would conclude that variation A performed the best; however, if it was 400, then we would favor variation B, and at 1000, it is difficult to distinguish the performance of variations B and C.

Second, the figure stresses the consequence of the selection of the target level set. If the target set in the above comparison must yield an average objective function value of −2000 or better, then variation A would perform the best because it makes the fastest progress in the beginning. However if it was −100 or better, then variation A would be considered a failure because it never achieves −100 during the (again, arbitrary number of) 1000 iterations. Thus it is important to report the whole graph to allow readers to make their own conclusions about the performance of the algorithms being compared.

## 5. Conclusions

Measures of computational performance for global optimization algorithms and software have been discussed. We propose the use of several such measures in comparing different methods and address some of the difficulties of conducting a fair comparison. Highlights of earlier studies restricted to constrained nonlinear programming have been summarized, which may be used to guide future studies on GO algorithms and their software implementations.

In the forthcoming second part of this paper we will conduct a systematic comparative study of several global optimization methods, based on a collection of test functions.

## Acknowledgements

The authors thank Graham R. Wood and his team for organizing the Stochastic Global Optimization Workshop (Hanmer Springs, New Zealand, 2001), which inspired this work. We are grateful for insightful comments provided by Arnold Neumaier. We would also like to thank Yanfang Shen and Eva H. Dereksdottir for their assistance on the manuscript.

The work of Zelda B. Zabinsky and Charoenchai Khompatraporn has been partially supported by NSF Grant No. DMI-9820878 and the Marsden Fund administered by the Royal Society of New Zealand. The work of János D. Pintér has been partially supported by grants received from the National Research Council of Canada (NRC IRAP Project No. 362093) and from the Hungarian Scientific Research Fund (OTKA Grant No. T 034350).

## References


1. 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.
2. Argonne National Laboratories (1993), *MINPACK-2 Test Problem Collection*. See also the accompanying notes titled Large-scale optimization: Model problems, by B.M. Averick and J.J. Moré (see http://www-c.mcs.anl.gov/home/more/tprobs/html).
3. Arora, J.S. (ed.) (1997), *Guide to Structural Optimization*, ASCE Manuals and Reports on Engineering Practice No. 90, American Society of Civil Engineers, New York.
4. Arora, J.S. (1990), Computational design optimization: a review and future directions. *Structural Safety* 7, 131–148.
5. Bazaraa, M.S., Sherali, H.D. and Shetty, C.M. (1993), *Nonlinear Programming: Theory and Algorithms* 2nd ed., John Wiley and Sons, New York.
6. Bomze, I.M., Csendes, T., Horst, R. and Pardalos, P.M. (eds.) (1997), *Developments in Global Optimization*, Kluwer Academic Publishers, Dordrecht.
7. Brent, R.P. (1973), *Algorithms for Minimization without Derivatives*, Prentice-Hall, New Jersey.
8. Bulger, D.W. and Wood, G.R. (1998), Hesitant adaptive search for global optimisation. *Mathematical Programming* 81, 89–102.
9. Colville, A.R. (1968), A comparative study of nonlinear programming codes, *Technical Report. No. 320-2949*, IBM Scientific Center, New York.
10. Corliss, G.F. and Kearfott, R.B. (1999), Rigorous Global Search: Industrial Applications. In: Csendes, T. (ed.), *Developments in Reliable Computing*, 1-16. Kluwer Academic Publishers, Dordrecht.
11. De Leone, Murli, A., Pardalos, P.M. and Toraldo, G. (eds.) (1998), *High Performance Software for Nonlinear Optimization: Status and Perspectives*, Kluwer Academic Publishers, Dordrecht.
12. Dembo, R.S. (1976), A set of geometric programming test problems and their solutions. *Mathematical Programming* 10, 192–213.
13. Dolan, E.D. and Moré, J.J. (2002), Benchmarking optimization software with performance profiles. *Mathematical Programming* 91, 201–213.
14. Eason, E.D. (1977), Validity of Colville's time standardization for comparing optimization codes. *ASME Des. Eng. Tech. Conf.*, *Paper No. 77-DET-116*, Chicago.
15. Eason, E.D. and Fenton, R.G. (1974), A Comparison of numerical optimization methods for engineering design. *ASME J. Eng. Ind. Ser. B* 96(1), 196–200.
16. Floudas, C.A. and Pardalos, P.M. (1990), A Collection of Test Problems for Constrained Global Optimization Algorithms, *Lecture Notes in Computer Science 455*, Springer-Verlag, Berlin.
17. Floudas, C.A., Pardalos, P.M., Adjiman, C.S., Esposito, W.R., Gümüs, 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.
18. Grossmann, I.E. (ed.) (1996), *Global Optimization in Engineering Design*, Kluwer Academic Publishers, Dordrecht.
19. Hendrix, E.M.T. (1998), *Global optimization at work*. Ph.D. dissertation, University of Wageningen, the Netherlands.
20. Hock, W. and Schittkowski, K. (1987), Test Examples for Nonlinear Programming Codes, *Lecture Notes in Economics and Mathematical Systems 187*, Springer-Verlag, Berlin.
21. Horst, R. and Pardalos, P.M. (eds.) (1995), *Handbook of Global Optimization*, Kluwer Academic Publishers, Dordrecht.

22. Horst, R. and Tuy, H. (1996), *Global Optimization – Deterministic Approaches*, 3rd ed., Springer-Verlag, Berlin.
23. Kearfott, R.B. (1996), *Rigorous Global Search: Continuous Problems*, Kluwer Academic Publishers, Dordrecht.
24. Kronsjö, L. (1987), *Algorithms*: *Their Complexity and Efficiency*, 2nd ed., John Wiley and Sons, New York.
25. Laguna, M. and González-Velarde, J-L. (eds.) (2000), *Computing Tools for Modeling, Optimization and Simulation*. Kluwer Academic Publishers, Boston.
26. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (eds.) (1985), *The Traveling Salesman Problem*, John Wiley and Sons, New York.
27. Lootsma, F.A. (1980), Ranking of Nonlinear Optimization Codes According to Efficiency and Robustness. In: Collatz L., Meinardus, G., and Wetterling, W., Birkhäuser, (eds.), *Konstruktive Methoden der Finite Nichtlineare Optimierung*, Basel, Switzerland, 157–168.
28. Mathar, R. and Žilinskas, A. (1994), A class of test functions for global optimization. *Journal of Global Optimization* 5, 195–199.
29. Mistakidis, E.S. and Stavroulakis, G.E. (1997), *Nonconvex Optimization. Algorithms, Heuristics and Engineering Applications of the F.E.M.*, Kluwer Academic Publishers, Dordrecht.
30. Mittelmann, H.D. (2001), *Benchmarks for Optimization Software* (see http://plato.la.asu.edu/bench.html).
31. Mittelmann, H.D., and Spellucci, P. (2001), *Decision Tree for Optimization Software* (see http://plato.la.asu.edu/guide.html).
32. Mockus, J., Eddy, W., Mockus, A., Mockus, L. and Reklaitis, G. (1996), *Bayesian Heuristic Approach to Discrete and Global Optimization*. Kluwer Academic Publishers, Dordrecht.
33. Moré, J.J., Garbow, B.S. and Hillström, K.E. (1981), Testing unconstrained optimization Software. *ACM Transactions on Mathematical Software* 7, 17–41.
34. Neumaier, A. (1990), *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge.
35. Neumaier, A. (2002), *Global Optimization* (see http://www.mat.univie.ac.at/~neum/glopt.html and the links therein).
36. Pardalos, P.M. and Rosen, J.B. (1987), Constrained global optimization: algorithms and applications. *Lecture Notes in Computer Science 268*, Springer-Verlag, Berlin.
37. Pintér, J.D. (1996), *Global Optimization in Action*, Kluwer Academic Publishers, Dordrecht.
38. Pintér, J.D. (1996), Continuous global optimization software: a brief review, *Optima 52*, 1–8 (see http://plato.la.asu.edu/gom.html).
39. Pintér, J.D. (1999), Continuous global optimization: an introduction to models, solution approaches, tests and applications. *Interactive Transactions in Operations Research and Management Science* 2(2) (see http://catt.bus.okstate.edu/itorms/pinter/).
40. Pintér, J.D. (2001), *Computational global optimization in nonlinear systems – An interactive tutorial*. Lionheart Publishing, Atlanta, GA (see http://www.lionhrtpub.com/books/globaloptimization.html).
41. Ratschek, H. and Rokne, J. (1988), *New Computer Methods for Global Optimization*, Ellis Horwood, Chichester.
42. Ratschek, H. and Rokne, J. (1993), Experiments using interval analysis for solving a circuit design problem. *Journal of Global Optimization* 3, 501–518.
43. Reklaitis, G.V. Ravindran, A. and Ragsdell, K.M. (1983), *Engineering Optimization, Methods and Applications*, John Wiley and Sons, New York.
44. Saaty, T.L. (1977), A scaling method for priorities in hierarchical structures. *J. Math. Psych.* 15(3), 234–281.

45. Saaty, T.L. (1988) *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, 2nd ed., Eta Services Ltd., Suffolk, UK.
46. Sandgren, E. (1977), The utility of nonlinear programming algorithms. Ph.D. Dissertation. Purdue University, Ann Arbor, MI, Document No. 7813115.
47. Sandgren, E. and Ragsdell, K.M. (1980), The utility of nonlinear programming algorithms: a comparative study – Part 1 and 2. *ASME J. Mech. Des.* 102(3), 540–551.
48. Schittkowski, K. (1980), Nonlinear Programming Codes: Information, Tests, Performance. *Lecture Notes in Economics and Mathematical Systems 183*, Springer-Verlag, Berlin.
49. Schoen, F. (1993), A wide class of test functions for global optimization. *Journal of Global Optimization* 3, 133–138.
50. Stephens, C.P. and Baritompa, W. (1998), Global optimization requires global information. *Journal of Optimization Theory and Applications* 96(3), 575–588.
51. Thanedar, P.B., Arora, J.S., Li, G.Y. and Lin, T.C. (1990), Robustness, generality and efficiency of optimization algorithms for practical applications. *Structural Optimization* 2, 203–212.
52. Törn, A., Ali, M.M. and Viitanen, S. (1999), Stochastic global optimization: problem classes and solution techniques. *Journal of Global Optimization* 14, 437–447.
53. Törn, A. and Žilinskas, A. (1989), Global Optimization. *Lecture Notes in Computer Science 350*, Springer-Verlag, Berlin.
54. Vavasis, S.A. (1995), Complexity issues in global optimization. In: Horst, R. and Pardalos, P.M. (eds.), *Handbook of Global Optimization,* pp. 27–41, Kluwer Academic Publishers, Dordrecht.
55. Zabinsky, Z.B. (1998), Stochastic methods for practical global optimization. *Journal of Global Optimization* 13, 433–444.
56. Zabinsky, Z.B., Smith, R.L. (1992), Pure adaptive search in global optimization. *Mathematical Programming* 53, 323–338.