Noname manuscript No.
(will be inserted by the editor)

Tuning BARON using derivative-free optimization algorithms

Jianfeng Liu · Nikolaos Ploskas · Nikolaos V. Sahinidis

March 5, 2018

Abstract Optimization solvers include many options that allow users to control algorithmic aspects that may have a considerable impact on solver performance. Tuning solver options is often necessary to reduce execution time and improve solution quality. Previous studies of solver tuning techniques have focused on mixed-integer linear programming and local nonlinear programming solvers. In this paper, we investigate the potential of tuning a global optimization solver for nonlinear and mixed-integer nonlinear programming problems. In particular, derivative-free optimization (DFO) algorithms are used to find optimal values for options of the global optimization solver BARON. A set of 126 problems from the GLOBALLib and MINLPLib collections are utilized in a computational study from which we conclude that tuning options can improve the default performance of BARON for individual problems and an entire library. Additionally, we present a systematic comparison of 27 DFO solvers in terms of their ability to improve the performance of the global solver. We find that several DFO implementations are much better than others in terms of finding optimal tuning parameters.

Keywords Solver tuning · Derivative-free optimization · Global optimization

1 Introduction

Optimization solvers provide many options that give users the flexibility to control solver performance on difficult problems. These options can significantly impact a solver's performance. Therefore, tuning options is often necessary in order to identify the best option values for a specific problem and improve solver performance in terms of execution time and solution quality. However, it is typically impossible to enumerate all possible option settings because there is a very large number of possible combinations. As a result, software developers tend to set default option values based on limited computational experience over a set of test problems and option values. Nonetheless, the default values do not guarantee an optimal solver configuration and may lead to poor performance in specific large problem instances. As a result, the development of a systematic method for tuning optimization solver options has received considerable attention in the literature [2,5,6,8,17,18,19,40].

Jianfeng Liu
Department of Chemical Engineering, Carnegie Mellon University.

Nikolaos Ploskas
Department of Chemical Engineering, Carnegie Mellon University.

Nikolaos V. Sahinidis
Department of Chemical Engineering, Carnegie Mellon University, e-mail: sahinidis@cmu.edu. Address all correspondence to this author.

This paper addresses the problem of identifying option settings that result in the best solver performance in terms of execution time and/or solution quality. Tuning solver options can be regarded as an optimization problem. This optimization problem is hard to solve for two reasons. First, since the relationship between options and solver performance is implicit, there exists no algebraic form to describe the performance function. Therefore, the optimization solver must be treated as a black-box system, whose inputs are values for the different options and output is a performance metric, such as the execution time and/or solution quality for a given problem. Second, since some of the options may take discrete values, the surface of the objective function is complex and nonsmooth, meaning that derivative information by local approximation may be inaccurate. Consequently, algorithms requiring explicit functional representations of the objective function or accurate gradients cannot be used to handle this black-box optimization problem. However, derivative-free optimization (DFO) algorithms may be well suited for this tuning problem, since they do not require explicit functional representations of the objective function or gradients [32].

Most of the solver tuning strategies proposed in the literature involve DFO algorithms. The MADS algorithm for DFO was utilized by Audet and Orban [2], who proposed a general DFO approach to search for locally optimal settings and applied it to tune a standard trust-region method. Baz et al. [5] proposed the Selection Tool for Optimization Parameters (STOP) method, which is an implementation of a two-step heuristic method including several selection schemes. The computational time required to solve an optimization problem was used as the performance criterion of the algorithm. The approach was later extended by Baz et al. [6] to tune parameters using the optimality gap and the best solution within a given runtime as the performance criterion. STOP has been used to tune the MILP solvers CBC, CPLEX, and GLPK in [5] and the MILP solver CPLEX in [6]. Hutter et al. [16] introduced ParamILS, a stochastic local search approach for automated algorithm configuration. Hutter et al. [19] presented two instantiations of ParamILS, BasicILS and FocusedILS, and used them to tune the MILP solver CPLEX. Hutter et al. [17] used FocusedILS to tune the MILP solvers CPLEX, GUROBI, and LPSOLVE. Their algorithm outperformed the CPLEX special-purpose automated tuning tool. Furthermore, Hutter et al. [18] extended their previous work to general algorithm configuration problems, allowing many categorical parameters and optimization for sets of instances and yielded further improvements for the configuration of CPLEX. Stewart [40] developed a model-based response surface algorithm, which led to slightly poorer performance than STOP, but required substantially less computational effort for tuning. A random sampling-based method was introduced by Chen et al. [8] to tune the NLP solver IPOPT, showing that NLP solvers can be improved remarkably.

Most of the methods that were proposed to tune options for optimization solvers improved solver performance. However, all previously proposed tuning methods have focused on MILP and local NLP solvers. As no prior work has been done to investigate whether similar benefits can be achieved by tuning global optimization solvers, the primary objective of this paper is to investigate the potential of tuning a global optimization solver for NLP and MINLP problems. More specifically, the systematic tuning of the global solver BARON [37] is addressed in this paper. As with prior tuning approaches, we will also rely on DFO solvers. However, we should note that all tuning approaches proposed in the literature involve a few DFO solvers, typically, two or three of them. Although extensive comparisons of various DFO methods, as well as their corresponding software implementations, have been made by Rios and Sahinidis [32], these comparisons were performed on algebraically available problems, i.e., not on true true black-box systems such as the solver tuning problem. Thus, the second objective of this paper is to present a systematic comparison of different DFO solvers in solving tuning problems, with the aim to increase our understanding of the capabilities of DFO solvers to improve black-box systems.

The remainder of this paper is organized as follows. Section 2 provides a brief introduction to BARON and its options. Section 3 includes a brief description of DFO algorithms, while Section 3.1 introduces the corresponding software implementations that are used in this paper for tuning BARON. In Section 4, numerical results based on a set of 126 problems from GLOBALLib [13] and MINLPLib [7] collections are presented to show the effect of option tuning on BARON's performance. Section 5 presents the

systematic comparison of 27 state-of-the-art DFO solvers in terms of their ability to improve BARON's performance through option tuning. Finally, conclusions are provided in Section 6.

2 BARON and BARON options

The Branch-And-Reduce Optimization Navigator (BARON) [44] is a computational system for the global solution of algebraic NLP and MINLP problems in the form of:

$$\begin{array}{lll} \text{(P)} & \min & f(x) \\ & \text{s.t.} & g(x) \leq 0 \\ & & x_i \in \mathbb{R}, \quad i = 1, \ldots, n_{\mathrm{d}} \\ & & x_i \in \mathbb{Z}, \quad i = n_{\mathrm{d}} + 1, \ldots, n \end{array}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R}^n \mapsto \mathbb{R}^m$ are assumed to be factorable functions, i.e., recursive compositions of sums and products of univariate functions. Furthermore, f and g are assumed to be available in explicit algebraic form. As an extension of the traditional branch-and-bound approach, branch-and-reduce incorporates several constraint propagation and duality techniques to significantly expedite convergence [24, 33, 34, 38, 30, 31]. At its inception, BARON was a branch-and-bound system. Built around a branch-and-bound routine, the software allowed the implementation of different branch-and-bound algorithms via suitable callbacks. As a result, the software facilitated the development of specialized implementations of several classes of algorithms for various problem classes, including separable concave minimization and indefinite quadratic programming [35]. The current version of the software exclusively addresses the solution of general, factorable NLP and MINLP problems [36]. BARON's algorithms now incorporate polyhedral outer-approximation [43] and other advanced convexification techniques [3, 41, 42, 48] to compute bounds in the course of the algorithm.

A primary objective of the BARON project is to provide its developers with a platform for experimentation with various branch-and-bound algorithms. As such, the software offers a large number of algorithmic options. Before discussing these options, an overview of the algorithm is provided. In its preprocessing step, BARON reduces the domains of the problem variables as much as possible with feasibility- and optimality-based range reduction techniques. Other optimization solvers may be invoked in this stage to optimize individual problem variables over relaxations of the original problem. Additionally, local search techniques are utilized in an effort to identify good feasible solutions. While local search and range reduction are used in subsequent nodes of the branch-and-bound tree, these techniques are used to a greater degree in the preprocessing stage. Following preprocessing, the iterative part of the algorithm proceeds as follows:

1. a rectangular subdomain of the feasible space is selected for processing
2. feasibility-based range reduction is applied to reduce ranges of variables as much as possible
3. lower bounding is applied via polyhedral outer approximations that are generated in a number of rounds aiming to strengthen an initially loose linear programming (LP) relaxation in the vicinity of the relaxation solution. Nonlinear and mixed-integer linear programming relaxations are subsequently utilized if deemed suitable for the problem at hand.
4. optimality arguments are used for further range reduction
5. local search is applied if it seems promising
6. processing of the subdomain is repeated if it seems compelling
7. unless fathomed by infeasibility or inferiority, the subdomain is split into two rectangular subdomains by subdividing a single problem variable into two subintervals

BARON [37] offers more than 50 algorithmic options. Many of these options impact the performance of the algorithm at various levels. BARON's options are separated into various groups by functionality, such as termination options, relaxation options, range reduction options, tree management options, local search options, output options, subsolver options, licensing options, and other options. In this paper, we

focus on 11 options of BARON that we deem important for an initial tuning experimentation, including eight discrete and three continuous options. We selected these variables after an initial computational study that showed that changes in the values of these options can affect BARON's performance on various problems. These options are listed in Table 1 and perform the following functions:

1. BasFra: While solving LP problems in different parts of the tree, information about the basis is stored for warm starting nodes. More specifically, the current LP working basis will not be modified if at least BasFra fraction of the problem basic variables are also basic in the saved basis for the node that is about to be solved. BasFra is a continuous option that can take any value between 0.05 and 0.95. By default, BARON sets BasFra equal to 0.7.
2. BrPtStra: Once the branching variable has been chosen, the option BrPtStra is used to determine the point where branching will take place. It can take the value of 0 for BARON's dynamic selection rule, 1 for using the relaxation solution for branching (ω-branching), 2 for using bisection-branching, or 3 for using a convex combination of ω- and bisection-branching. By default, BARON sets BrPtStra equal to 0.
3. BrVarStra: The option BrVarStra determines BARON's branching variable strategy. It can take the value of 0 for using BARON's dynamic selection rule, 1 for using the branching variable that corresponds to the largest deviation between nonconvex functions and the current relaxation, or 2 for using the longest edge variable. By default, BARON sets BrVarStra equal to 0.
4. ConvexRatio: The option ConvexRatio is a continuous option that can take values between 0.05 and 0.95. It corresponds to the factor used when BrPtStra is equal to 3 in order to construct the convex combination between the relaxation solution and the mid point of the range of the branching variable. By default, BARON sets ConvexRatio equal to 0.7.
5. ModBrpt: The option ModBrpt is a binary option that defines whether or not the branching point should be set to the value of the incumbent, if the latter is in the current bounding box. It can take the value of 0 for not setting the branching point to the value of the incumbent, or 1 for setting the branching point to the value of the incumbent. By default, BARON sets ModBrpt equal to 1.
6. NOuter1: In the relaxation constructor, BARON outer approximates convex univariate functions by supporting hyperplanes. The option NOuter1 determines the number of outer approximators of convex univariate functions. This is a discrete option that must be a nonnegative integer. By default, BARON uses 4 supporting hyperplanes.
7. NOutIter: After solving an initial outer approximating LP relaxation, BARON adds cutting planes at the LP solution, solves the newly constructed LP, and repeats the cutting plane generation a number of times. The option NOutIter determines the number of rounds of this cutting plane generation process. This option is also discrete and must be a nonnegative integer. By default, BARON uses 4 rounds of cutting planes.
8. PDo: Most constraint propagation tests (feasibility-based range reduction tests) are inexpensive making it viable to execute them as often as possible. However, solving optimization problems to reduce ranges of variables can be expensive if not done in moderation. In BARON, this process is referred to as probing. The option PDo determines the number of probing problems solved at a node of the branch-and-bound tree. This option is also discrete and can take any discrete value between -2 and n. A value of PDo equal to -2 lets BARON automatically decide the number of probes allowed. A value equal to -1 means that all possible probing problems will be solved (two for each problem variable). Finally, a value of $k > 0$ specifies that k probing subproblems are to be solved. By default, BARON automatically decides the number of probing problems allowed.
9. PFreq: Since probing can be expensive, the option PFreq can be used to specify the level-frequency of probing applications. A value of PFreq equal to k means that probing is done every k levels of the branch-and-bound tree. Again, this is a discrete option that can take any positive value. By default, BARON sets PFreq equal to 3.
10. ProFra: BARON utilizes two types of probing problems. In one type, referred to as forced probing, probing variables are fixed at a point and the relaxation is solved at that point to provide information needed for optimality-based range reduction. The option ProFra specifies the fraction of probe to

Table 1 BARON's options to be tuned

Index	Name	Type	Range	Default value
1	BasFra	continuous	0.05:0.95	0.7
2	BrPtStra	discrete	0:3	0
3	BrVarStra	discrete	0:2	0
4	ConvexRatio	continuous	0.05:0.95	0.7
5	ModBrpt	discrete	0:1	1
6	NOuter1	discrete	1:10	4
7	NOutIter	discrete	1:10	4
8	PDo	discrete	$-2:10$	-2
9	PFreq	discrete	1:10	3
10	ProFra	continuous	0.05:0.95	0.67
11	MaxNodePass	discrete	1:10	5

bound distance from the relaxation solution when forced probing is done. This is a continuous option that can take any value between 0.05 and 0.95. Its default value in BARON is 0.67.

11. MaxNodePass: The integer option MaxNodePass determines the maximum number of passes allowed through a node. If postprocessing improves the node's lower bound in a way that satisfies certain absolute or relative tolerances, the process of lower bounding followed by postprocessing is repeated up to MaxNodePass times. This option can take an integer value between 1 and 10. By default, BARON sets MaxNodePass equal to 5.

Table 1 lists the selected options along with their default values in BARON and a range of values that will be used in the computational study of this paper for each one of these options. Default values were obtained after years of extensive computational experimentation with a large set of benchmark instances. For simplicity, an upper bound of 10 is imposed for integer options that can take any positive value. For instance, only up to 10 outer approximators per univariate function are allowed (option NOuter1). Since some of the options considered are continuous, the 11 options that were selected give rise to infinite configurations. Even if we only consider the discrete options in Table 1, the total number of possible combinations still exceeds 3 million. Therefore, it is impractical to enumerate all possible option combinations to identify the optimal ones for a reasonably large collection of even moderately difficult problems. For this reason, DFO algorithms are utilized to identify optimal option combinations.

3 Derivative-free optimization algorithms

Option tuning is the problem of searching for better (in comparison to default) values of options in specific problem instances or groups of instances. This search may lead to an improvement of an optimization solver's performance, typically measured in terms of execution time and/or solution quality. Therefore, an option tuning problem can be regarded as an optimization problem in the following form:

$$\begin{aligned} \min \quad & f(C, P) \\ \text{s.t.} \quad & C \in R \end{aligned}$$

where f is the metric or the performance function to measure the performance of option settings; C represents the values of options; P indicates a problem instance or a set of instances; R is the feasible region of option combinations. All options C are independent of each other and can take any value between their respective lower and upper bounds. As a result, every possible combination of different parameters forms a feasible solution to the optimization problem.

It is apparent that optimal option settings can be obtained by solving the optimization problem mentioned above. However, several challenges make this problem difficult to solve. First, because the relationship between solver options and performance is often implicit and complex, it is impossible

to pose the objective function in an explicit algebraic form. Therefore, the tuning problem is often a black-box optimization problem. The input to this black-box system is the configuration of options and the output is the value of the performance function. Additionally, the performance function is always nonsmooth and nondifferentiable. Due to the absence of derivative information for the objective function, algorithms requiring explicit functional representations of the objective function cannot be used to solve this black-box optimization problem. However, DFO algorithms are well suited to handle the tuning problem.

DFO algorithms are designed to solve optimization problems, whose gradient information is unavailable or unreliable. The first DFO algorithm, a direct-search approach later called Nelder-Mead method, was developed by Spendley et al. [39] and Nelder and Mead [27] for handling multi-variable optimization problems. Prompted by the need to solve practical problems, DFO has become an attractive area of research with the number of studies pertaining to DFO algorithms growing fast. A large number of new algorithms based on various ideas have been developed, including generalized pattern search [45], trust-region methods [28], and radial basis functions [29]. By combining algorithms from different categories, several hybrid strategies have been generated showing competitive performance [11,22,46]. Most recently, DFO algorithms have been implemented in software packages. Currently, DFO solvers can handle a wide range of applications, including engineering design [4,12,15] and option tuning [2,17].

Comparing DFO implementations is an important task. However, most of the previous work only focused on a small number of test problems and applied few DFO solvers [1,9,10,12,14,20,21,25,26,47]. Recently, a systematic comparison of 22 implementations using a set of 502 problems was conducted by Rios and Sahinidis [32], making significant progress toward assessing the performance of different implementations. However, these comparisons were not performed on true black-box systems. Comparisons on black-box systems are still badly needed in the DFO area. Therefore, the second goal of this paper is to compare the performance of various DFO implementations in optimizing solver options, a setting that represents a true black-box optimization problem.

As in [32], we classify DFO approaches as local or global, depending on whether they involve steps aiming to escape from local minima. Depending on whether DFO algorithms build surrogate models or not, these algorithms are divided into direct and model-based methods. Finally, DFO algorithms are partitioned in stochastic and deterministic algorithms, based on whether random search strategies are used or not. For more details of DFO algorithms, we refer the reader to [32] and references therein.

3.1 Derivative-free optimization solvers

A large number of software implementations based on various DFO algorithms have been developed. Therefore, at least one corresponding implementation can be found for each algorithm mentioned above. This is especially true for several well-studied algorithms, such as trust-region methods [28] and DIRECT [23]. Moreover, modifications and extensions were introduced in nearly all implementations to improve performance in various aspects, which resulted in theoretical advances of many DFO methods. These DFO implementations have been developed in a variety of programming languages, i.e., Fortran, C/C++, and MATLAB. Table 2 provides information about 27 DFO solvers that we used in this paper to tune BARON's performance. For each solver, we present the version of the code used in the computational study and the algorithmic category (according to the classification in [32]). These implementations cover most state-of-the-art DFO algorithms. More details about these DFO solvers can be found in [32].

4 Computational results on tuning BARON's options

This section provides numerical results for tuning BARON's options. Initially, the computational environment and the problem instances that were used are presented. Moreover, the choice of the performance function is discussed in detail. Then, general solutions of identifying parameters for individual instances

Table 2 Derivative-free implementations used in computational experiments

Name	URL	Solver type
ASA 30.18	http://www.ingber.com/	SG
BOBYQA 2009	ccpforge.cse.rl.ac.uk/gf/project/powell/frs/	ML
CMA-ES 3.61beta	www.lri.fr/~hansen/cmaesintro.html	SG
DAKOTA/DIRECT 6.1	https://dakota.sandia.gov/	DG
DAKOTA/EA 6.1	https://dakota.sandia.gov/	SG
DAKOTA/PATTERN 6.1	https://dakota.sandia.gov/	DL
DAKOTA/SOLIS-WETS 6.1	https://dakota.sandia.gov/	SG
DFO 2.0	projects.coin-or.org/Dfo	ML
FMINSEARCH 1.1.6.2	www.mathworks.com	DL
GLOBAL 1	www.inf.u-szeged.hu/~csendes	SG
HOPSPACK 2.0.2	software.sandia.gov/trac/hopspack	DL
IMFIL 1.0.2	www4.ncsu.edu/~ctk/imfil.html	ML
MCS 2.0	www.mat.univie.ac.at/~neum/software/mcs/	DG
NEWUOA 2004	ccpforge.cse.rl.ac.uk/gf/project/powell/frs/	ML
NOMAD 3.7.1	www.gerad.ca/nomad/	DL
PRAXIS 1	http://netlib.org/opt/praxis	DL
PSWARM 2.1	www.norg.uminho.pt/aivaz/pswarm/	SG
SID-PSM 1.3	Available by email from alcustodio@fct.unl.pt	DL
SNOBFIT 2.1	www.mat.univie.ac.at/~neum/software/snobfit/	DG
TOMLAB/CGO 8.0	www.tomopt.com	DG
TOMLAB/GLB 8.0	www.tomopt.com	DG
TOMLAB/GLC 8.0	www.tomopt.com	DG
TOMLAB/GLCCLUSTER 8.0	www.tomopt.com	DG
TOMLAB/LGO 8.0	www.tomopt.com	SG
TOMLAB/MSNLP 8.0	www.tomopt.com	HG
TOMLAB/MULTIMIN 8.0	www.tomopt.com	HG
TOMLAB/RBF 8.0	www.tomopt.com	DG

DG: Deterministic global, DL: Direct local, HG: Hybrid global, ML: Model-based local, SG: Stochastic global

is given to indicate remarkable improvements obtained from parameter tuning. Two case studies are presented for detailed analysis, showing problem instances that lead to significant differences in optimal configurations and performance improvements. Finally, tuning options for the entire group is studied, showing that there are better parameter configurations than the default setting.

4.1 Computational environment and benchmark libraries

The computational study was performed on a workstation with two Intel Xeon processors E5-2660 v3 at 2.6GHz and 128GB of memory running CentOS 7. The version of BARON that was used is 15.5. The same parameter setting was used and the limit of function evaluations was set to $1,000$ for all DFO solvers examined. Most of the DFO solvers used in this computational study optimize in real spaces but some of the selected BARON parameters are integer valued. Integrality was handled by rounding real values to the nearest integer. The test set of this computational study includes 126 optimization problems from two public benchmark sets, specifically GLOBALLib and MINLPLib. All problem instances considered in this computational study take no more than 20 seconds for BARON to solve with the default option setting.

4.2 Performance function

The most widely used metric or performance function for tuning solvers is execution time, which is regarded as an important criterion to measure computational effort for solving optimization problems.

A number of previous works have focused on computational time and indicated that substantial runtime reductions can be obtained by parameter tuning. The runtime reduction is calculated as:

$$rr = \begin{cases} 1 - \frac{t_{DFO}}{t_{default}} & , \text{if } t_{DFO} >= t_{default} \\ -\left(1 - \frac{t_{default}}{t_{DFO}}\right) & , \text{if } t_{DFO} < t_{default} \end{cases}$$

where $rr \in [-1, 1]$ is the runtime reduction, t_{DFO} is BARON's execution time using the values of the options provided by a DFO solver, and $t_{default}$ is BARON's execution time with BARON's default options. A value of $rr > 0$ (resp. $rr < 0$) corresponds to an option combination that is better (resp. worse) than the default one.

Alternative performance functions have also been proposed to guide option searching. In [17], two metrics, i.e., runtime and optimality gap, were introduced and then compared on a large range of instances from various benchmark sets for three MIP solvers, specifically CPLEX, GUROBI, and LPSOLVE. For all MIP approaches, tuning parameters using both performance functions led to significant improvements. Furthermore, three metrics, i.e., computational time, best proven gap, and best feasible integer solution, were considered in [6]. Computational results for problem instances of the benchmark set MIPLIB showed that different performance functions had remarkably different impact on optimal parameters. Therefore, the metric must be chosen properly in order to meet the requirement for a given situation.

In this research, computational time is applied as the only metric in searching optimal parameters. The reason why measurements for solution quality, such as objective function values and solution optimality, are not considered is because if a problem can be solved within the time limit, BARON, as a global solver, can always find the same solution. In other words, for our problem instances, it is reasonable to simply focus on execution time since optimal parameters only bring about improvement in runtime rather than solution quality. Both our experiments and observations have indicated that a distinct optimal parameter setting may be attained for an individual instance or a family of similar problems. For this reason, previous tuning attempts on local solvers were considered for either individual problems or the entire group, depending on whether different instances shared similar features in objective functions, constraints, variables, etc. Tuning options for the entire group is also performed and we report the best found option combination when compared to the default.

4.3 Tuning options for individual problem instances

4.3.1 Best option configuration for individual problem instances

Since 27 DFO solvers are applied to find optimal options, various good option settings were identified in our runs for each individual problem instance. In order to demonstrate the benefit obtained from parameter tuning, only the best option configuration is considered in this section. An impressive average runtime reduction of about 57% across our group of 126 problems was achieved, which demonstrates that tuning options can substantially improve BARON's performance. The solution quality, such as the objective function value and the optimality gap, remains unchanged for all problems when new options are used.

Figure 1 presents the number of problems in different levels of runtime reductions. In Figure 1, we use the best option configuration from all DFO solvers. As shown in this figure, improvements in computational effort are attained for different instances. Considering the existence of disturbances from the computational environment, which may cause an obvious influence on runtime, a striking improvement is implied for a reduction larger than 10%. Hence, the execution time of 125 problems (around 99% of all problems) were significantly improved. Moreover, 84 of them (around 67% of all problems) had runtime reduced by more than 50% and an impressive runtime reduction of over 80% is obtained for 17 benchmarks (around 13% of all problems). Overall, the average runtime reduction was 57%, indicating that great benefits can be achieved by option tuning for individual problem instances.

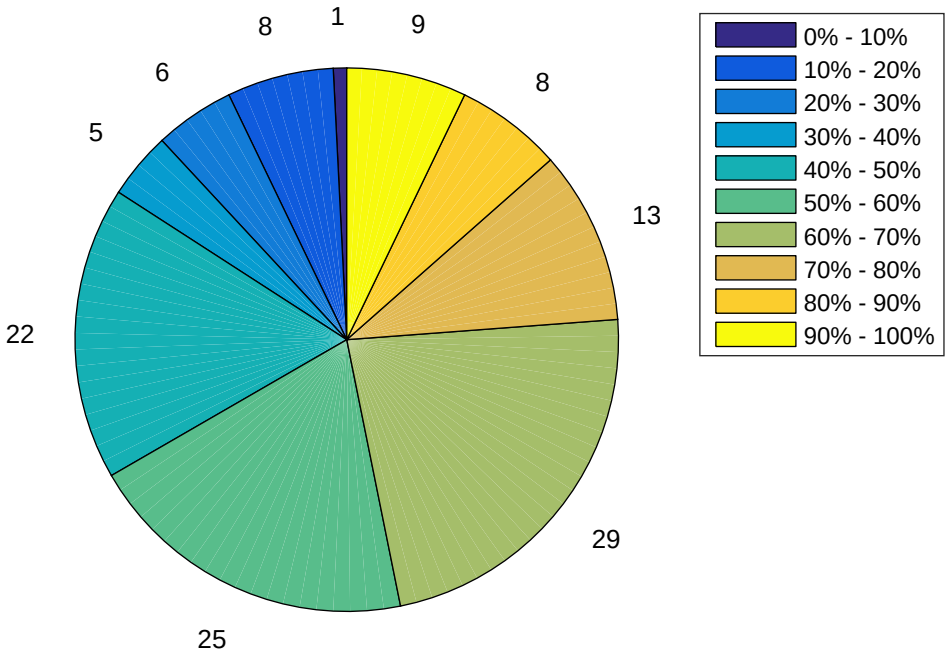

Fig. 1 Number of problems in different runtime reduction levels for the set of 126 problems (best option configuration from all DFO solvers)

On the other hand, only slight, or even no, improvements were attained for 1 problem (around 1% of all problem instances). Since there are no better options, this suggests that the default configuration is already 'optimal' for this benchmark.

Additionally, detailed analysis shows that optimal option configurations differ considerably for various problems. In other words, there exists no distinct optimal configuration that works equally well for all instances in our group of 126 problems, which verifies our previous assumption that the best solver options are strongly problem-dependent. Therefore, different runtime reductions as well as parameter configurations may result if tuning is conducted for a family of problems instead of individual problems.

Finally, it is worth noting that runtime reduction due to optimized configurations is oftentimes insufficient to recover the computational effort required for solving the corresponding tuning problems. For instance, the potential computational time reduction may only be several seconds for each instance, but the total execution time needed to find optimal options, even for the fastest DFO solvers, can be as high as several hours. Consequently, in practice, searching optimal option configuration for a small number of problems is not recommended if the goal is to reduce the total computational effort. However, option tuning, though time-consuming, is still worthwhile in several scenarios [6], including for single instances that are deemed representative of a large number of similar problems to be subsequently solved with the same algorithmic settings.

Table 3 Option comparison for problem instance *chenery*

Option	Default	Tuned option value			
		CMA-ES	DAKOTA/ DIRECT	TOMLAB/ LGO	PSWARM
BasFra	0.7	0.84	0.20	0.66	0.55
BrPtStra	0	1	1	0	1
BrVarStra	0	0	1	0	1
ConvexRatio	0.7	0.93	0.50	0.48	0.37
MaxNodePass	5	4	6	2	1
ModBrpt	1	1	0	0	1
NOuter1	4	2	2	2	2
NOutIter	4	1	2	8	1
PDo	-2	4	4	3	4
PFreq	3	10	5	8	10
ProFra	0.67	0.90	0.80	0.67	0.93
Execution time (sec)	4.5	0.6	0.6	0.7	0.6
Runtime reduction	–	87%	87%	85%	87%

4.3.2 Case studies

As mentioned in Section 4, the relationship between the configuration of the parameters and the optimization solver's performance is implicit and complicated, which leads to nonsmooth, nondifferential, and multi-modal performance surfaces. Among those features, the existence of multiple local solutions has the greatest impact on tuning options. Hence, local DFO solvers are highly likely to get stuck around local points and report relatively poorer performances, while global solvers have the ability to jump out from local optima and produce significantly better options. However, there is no guarantee that these global solvers will converge to the optimal point. Moreover, more computational effort is required to perform global searches.

By presenting various optimal option configurations for problem instances in two different types, the primary goal of this subsection is to gain insights in the impact of local optima on parameter tuning. The first benchmark is easy to tune. Due to a number of near-optimal configurations existing in the feasible region, computational time can be reduced significantly for this problem by most DFO solvers. For the second problem is much harder to tune because a large number of local optima configurations exist that are inferior to defaults. Trapped in these local points, several DFO solvers report tuned configurations with noticeably poorer performance. The study on existing multiple local solutions also provides deeper insight into the relationship between BARON's options and its performance.

Oftentimes different option configurations result in varying execution time reductions. For some problems, similar performance improvements are attained by a different combination of options. For instance, the best four option configurations identified for problem *chenery* from GLOBALLib are listed in Table 3, along with corresponding DFO solvers and the runtime reductions. Various options lead to nearly identical improvements, which means that there are a number of optimal points in the feasible region. Furthermore, there are no critical options that have the same value for all optimal configurations (except the option $NOuter1$ which has the same value on all four cases). Hence, a solver's performance is not sensitive to the options considered in our search and many local solutions are adequate. Therefore, it should be no surprise that runtime reductions in excess of 55% were reported by all DFO solvers.

In many instances multiple optimal solutions are observed. Unlike the case study presented above, there are dominant options that play a vital role in affecting a solver's performance. Table 4 presents the best four options configurations for the problem instance $nvs20$ of MINLPLib. Although the configurations of option values are not identical, the same choices are made for five options, specifically $BrPtStra$, $BrVarStra$, $NOuter1$, $NOutIter$, and PDo. Since special values are required for these parameters, they are dominant or critical options for this problem instance. On the other hand, other local points with

Table 4 Option comparison for problem instance *nvs*20

Option	Default	Tuned option value			
		DAKOTA/ DIRECT	TOMLAB/ GLB	TOMLAB/ GLC	TOMLAB/ RBF
BasFra	0.7	0.20	0.50	0.50	0.50
BrPtStra	0	1	1	1	1
BrVarStra	0	2	2	2	2
ConvexRatio	0.7	0.20	0.50	0.50	0.20
MaxNodePass	5	3	9	6	6
ModBrpt	1	1	0	0	1
NOuter1	4	6	6	6	6
NOutIter	4	6	6	6	6
PDo	-2	0	0	0	0
PFreq	3	3	6	9	9
ProFra	0.67	0.40	0.80	0.80	0.80
Execution time (sec)	2	0.2	0.2	0.2	0.2
Runtime reduction	–	90%	90%	90%	90%

different values in dominant options are far from optimal and sometimes only lead to a small reduction in execution time. As a result, the average reduction time for local DFO solvers is 83%, while the average reduction time for global DFO solvers is 89%.

4.4 Tuning options for the entire group

Other than tuning the execution time of each individual problem instance, we can also search for better option configurations for the entire group of 126 problems. Similarly, the total computational effort required for solving all benchmark problem instances is considered to be the only metric in assessing the performance of the global optimizer BARON. However, it is worth noting that the execution times required to solve different problem instances are remarkably different. Therefore, only focusing on execution time may overestimate the amount of savings for a small number of problem instances, but ignore the average percentage of improvement across the entire group. Hence, we divided the actual execution time by the corresponding execution time under BARON's default options to generate a normalized execution time. The objective function of this tuning problem is the summation of these times after normalization. Even though we tune and test quality of tuning on the same set of 126 problems, computations on the entire collection of GlobalLib and MINLPlib problems (a total of 580 problems) show that, in comparison to BARON's default options, one of the best DFO solver options (TOMLAB/LGO) reduced total CPU time by 5% and increased by 12 the total number of problems that BARON can solve to global optimality within 500 s.

Table 5 shows the runtime reduction for the group of 126 problems reported by each solver. Runtime reductions demonstrate that tuning options can also result in significant reductions in the total execution time. Almost all solvers (24 out of 27) report significant improvements ($> 10\%$). The best solver, SNOBFIT, obtains a runtime reduction of 34%, followed by CMA-ES and PSWARM that obtain a runtime reduction of 29%. On the other hand, DAKOTA/PATTERN, HOPSPACK, and SID-PSM failed to improve significantly the execution time for the entire set of 126 problems.

Table 6 shows the best four option configurations we found for the group of 126 problems. Various options lead to nearly identical improvements, which again proves that there are a number of optimal points in the feasible region. Moreover, there are no critical options. Hence, solver performance is not sensitive to the options considered in our search, and plenty of local solutions are adequate. Interestingly enough, the SNOBFIT options make BARON explore similar numbers of nodes of the branch-and-bound trees on average compared to the default options, while CMA-ES options lead to exploring three times

Table 5 Time reduction for the set of 126 problesm

DFO solver	Time reduction (%)	DFO solver	Time reduction (%)
ASA	22	NOMAD	20
BOBYQA	11	PRAXIS	11
CMA-ES	29	PSWARM	29
DAKOTA/DIRECT	20	SID-PSM	4
DAKOTA/EA	23	SNOBFIT	34
DAKOTA/PATTERN	7	TOMLAB/CGO	20
DAKOTA/SOLIS-WETS	11	TOMLAB/GLB	20
DFO	17	TOMLAB/GLC	20
FMINSEARCH	12	TOMLAB/GLCCLUSTER	20
GLOBAL	20	TOMLAB/LGO	25
HOPSPACK	8	TOMLAB/MSNLP	18
IMFIL	19	TOMLAB/MULTIMIN	21
MCS	23	TOMLAB/RBF	20
NEWUOA	11		

Table 6 Option comparison for the group of 126 problems

Option	Default	Tuned option value			
		CMA-ES	PSWARM	SNOBFIT	TOMLAB/ LGO
BasFra	0.7	0.59	0.32	0.10	0.69
BrPtStra	0	0	2	3	0
BrVarStra	0	1	0	0	0
ConvexRatio	0.7	0.95	0.69	0.37	0.24
MaxNodePass	5	3	9	10	7
ModBrpt	1	0	0	1	0
NOuter1	4	6	4	10	7
NOutIter	4	2	7	2	5
PDo	-2	3	3	2	3
PFreq	3	5	6	2	2
ProFra	0.67	0.46	0.48	0.50	0.36
Execution time (sec)	167.37	118.17	119.48	110.68	125.27
Runtime reduction	–	29%	29%	34%	25%

fewer nodes. Consequently, the size of the branch-and-bound tree is not indicative of the impact of different algorithmic options on BARON's CPU time consumption.

5 Comparison of DFO solvers

In this section, the options settings obtained by all DFO solvers for individual problem instances are presented and analyzed. A comparison of the solvers was made based on three criteria: (i) the number of significant improvements in comparison to BARON's default settings, (ii) the average runtime reduction, and (iii) the number of times a solver found better options than other solvers, i.e., the number of times a solver found the best solution for a tuning problem. Then, we present a comparison according to computational effort required to tune parameters, i.e., the number of the function evaluations and the total execution time, thus providing deeper insight on DFO solver ability to handle tuning problems. Finally, a comparison between different groups of solvers is presented, aiming to gain insights to general algorithmic strategies utilized by DFO algorithms.

5.1 Comparison of solution quality

In order to draw comprehensive conclusions, three metrics of solution quality, i.e., the number of significant improvements with respect to default options, the average runtime reduction with respect to default options, and the number of best solutions found by a solver, are considered to measure the performance of DFO solvers in searching optimal options for each individual problem instance. On average, global solvers perform well in solution quality.

5.1.1 Number of significant improvements

The number of improvements is one of the most straightforward criteria used to compare the tuning ability of different DFO solvers. In general, a larger number of improvements suggests a greater ability to find optimal option settings. Since the disturbances from the computational environment may have an effect on the execution time, runtime reductions between -10% and 10% are treated as insignificant. Therefore, a significant improvement corresponds to a runtime reduction that is larger than 10%. Table 7 presents the number of insignificant and significant runtime reductions for each solver. Moreover, it includes the number of best solutions found from each solver (for some instances, many solvers achieve the same runtime reduction), and the average runtime reduction. 21 DFO solvers successfully tuned configurations for more than 101 problem instances. On the other hand, BOBYQA, NEWUOA, and PRAXIS identified better options only for 83 instances, followed by DAKOTA/PATTERN that found better options for 86 problem instances. DAKOTA/SOLIS-WETS and FMINSEARCH also demonstrated poor performance, reporting optimized parameters for less than 96 instances. Moreover, for most DFO solvers, there was at least one failure (a runtime reduction lower than -10%) in option tuning. Only seven solvers, ASA, DAKOTA/EA, GLOBAL, PSWARM, SNOBFIT, TOMLAB/LGO, and TOMLAB/MULTIMIN, had no failures. As a result, the aforementioned seven solvers were the best solvers as they had significant improvements for at least 120 problems and encountered no failures.

5.1.2 Average runtime reduction

The average runtime reduction is an additional criterion that should be considered. We might intuitively expect that a good solver should report both high average reductions and large number of improvements. As seen in Table 7, PSWARM proves to be the best solver with an average runtime reduction of 55%. CMA-ES, DAKOTA/EA, SNOBFIT, TOMLAB/LGO, and TOMLAB/MULTIMIN follow PSWARM very closely with an average runtime reduction of at least 52%. All of these solvers have large numbers of significant improvements (more than 120), verifying our intuitive guess. On the other hand, DFO solvers that report small numbers of improvements, e.g., BOBYQA, DAKOTA/PATTERN, FMINSEARCH, NEWUOA, and PRAXIS, also present poor performance in terms of average runtime reduction. None of these solvers can achieve an average runtime reduction larger than 21%. BOBYQA is the worst solver with both the smallest number of improvements (83) and the lowest average runtime reduction (19%). From the analysis above, it is clear that better tuning ability can lead to greater performance, in terms of the number of affected problems and overall time requirements of the algorithm.

5.1.3 Number of best solutions

In this section, we compare DFO solver performance according to their ability to obtain the best option setting that leads to the largest runtime reduction. It is worth noting that the best configuration does not necessarily mean it is the optimal one. More specifically, it might even be a suboptimal or local solution. Therefore, the criterion is also called relative performance for it takes into account that we only consider the best solution available to us. As previously mentioned, a sufficiently large error range of computational time reduction is also needed to eliminate disturbances. Especially, if the difference between a runtime reduction and the best one is not greater than 5%, that reduction is also treated

Table 7 Number of improvements, number of best options, and average runtime reduction for the group of 126 problesm

DFO Solver	Runtime reduction			Best options found by the DFO solver	Average runtime reduction (%)
	>10%	<-10%	-10% - 10%		
ASA	123	0	3	29	49
BOBYQA	83	19	24	8	19
CMA-ES	122	1	3	50	52
DAKOTA/DIRECT	122	1	3	39	51
DAKOTA/EA	123	0	3	52	54
DAKOTA/PATTERN	86	18	22	11	20
DAKOTA/SOLIS-WETS	96	11	19	13	27
DFO	101	12	13	12	31
FMINSEARCH	91	20	15	13	21
GLOBAL	121	0	5	34	51
HOPSPACK	109	5	12	12	37
IMFIL	118	2	6	22	45
MCS	118	1	7	47	49
NEWUOA	83	21	22	7	19
NOMAD	115	3	8	29	42
PRAXIS	83	19	24	9	21
PSWARM	123	0	3	62	55
SID-PSM	120	2	4	27	45
SNOBFIT	123	0	3	41	53
TOMLAB/CGO	123	1	2	30	49
TOMLAB/GLB	122	1	3	28	49
TOMLAB/GLC	122	1	3	28	49
TOMLAB/GLCCLUSTER	119	2	5	28	49
TOMLAB/LGO	122	0	4	43	53
TOMLAB/MSNLP	120	1	5	27	51
TOMLAB/MULTIMIN	120	0	6	35	52
TOMLAB/RBF	123	1	2	27	48

as the best one and the corresponding setting is regarded as the best configuration. Moreover, many solvers report the same runtime reduction for some problem instances, especially those in which the total execution time to solve the problem is very small.

Table 7 shows the number of the best configurations found by each DFO solver. PSWARM performed best as it obtained the best configurations for 62 problems. CMA-ES, DAKOTA/EA, MCS, SNOBFIT, and TOMLAB/LGO also performed well, identifying best-tuned parameters for more than 41 problem instances. Moreover, it seems that no single DFO solver dominated all of the approaches across our group of 126 problems. Therefore, we are interested in finding a minimal subgroup of solvers that can report the best runtime reductions for all problem instances. Collectively, the global solvers CMA-ES, MCS, PSWARM, and SNOBFIT found the best solutions in 101 out of the 126 problem instances. In addition to these solvers, the minimal subset of solvers required to obtain the best known solutions for all 126 problem instances also includes DAKOTA DIRECT, DAKOTA EA, GLOBAL, IMFIL, PRAXIS, SID-PSM, SNOBFIT, TOMLAB/GLB, TOMLAB/GLC, TOMLAB/LGO, TOMLAB/MULTIMIN, and TOMLAB/MSNLP. However, adding the latter solvers produces relatively small improvements.

5.1.4 Summary

PSWARM was found to lead all approaches according to the three quality criteria considered, specifically the number of significant improvements, the runtime reduction, and the number of best solutions. It was followed closely by CMA-ES, DAKOTA/EA, SNOBFIT, and TOMLAB/LGO. SNOBFIT also achieved

the best runtime reduction for tuning the group of 126 problems. Outstanding performances were also reported by DAKOTA/DIRECT, GLOBAL, TOMLAB/MSNLP, and TOMLAB/MULTIMIN. All of these solvers: (i) reduced the average runtime by more than 51%, (ii) found more than 34 best options, and (iii) found significant improvements for more than 120 problem instances. Considering that the differences in performance between these solvers are small, we regard all these solvers as highly successful tuners. On the other hand, BOBYQA, DAKOTA/PATTERN, FMINSEARCH, NEWUOA, and PRAXIS demonstrated poor performance in tuning options as they only reduced execution time by less or equal than 21%. Additionally, they found best options for less or equal than 13 problem instances and found significant improvements for less than 92 problem instances.

Our results were also compared with those reported in [32]. Since the performance function in our research is both nonconvex and nonsmooth, we only focus on the solutions for nonconvex nonsmooth problem instances in [32], which can be found in Figures 10 and 20 of [32]. Our results were mostly consistent with the findings in [32]. Several solvers, such as PSWARM, CMA-ES, TOMLAB/LGO, and TOMLAB/MULTIMIN, performed well in both our computational study and in [32]. For some approaches, however, there are significant differences in the quality of the solver solution. For instance, solvers DAKOTA/EA and GLOBAL reported great runtime reductions in our computational study, but performed poorly in [32]. Conversely, the direct local solver FMINSEARCH, which was one of the poorest performers in our computational study, performed much better for the problems considered in [32].

5.2 Comparison on solver efficiency

It is clear that an optimal configuration can help reduce computational effort significantly. As already mentioned, identifying optimized parameters is a time-consuming task in many cases and requires more computational effort which cannot be recovered by runtime savings. Therefore, the computational effort of DFO solvers also plays a vital role in option tuning. For this reason, it is quite reasonable to look for adequate rather than optimal configurations in order to save total runtime required by option tuning.

In this section, two metrics are considered to perform comparisons on the computational effort of different DFO solvers: (i) the number of iterations, and (ii) the total execution time needed to tune a problem (time taken by the DFO solver plus time taken by BARON to perform function evaluations for the DFO solver). Then, by combining the analysis on solution quality and computational effort, a further analysis is proposed to show solver efficiency in coping with tuning problems.

The number of iterations is the most important information in determining the computational effort. Since more iterations require more calls to BARON, the number of iterations indicates the total number of function evaluations made by each DFO solver. Therefore, fewer evaluations are likely to cause less computational effort. The average number of iterations for each DFO solver is listed in Table 8. Note that function evaluations of various solvers differ greatly. ASA, DAKOTA/DIRECT, DAKOTA/EA, PSWARM, SNOBFIT, TOMLAB/LGO, and TOMLAB/MSNLP do not stop running until they reach the limit of 1000 iterations used in our study. In particular, ASA, DAKOTA/DIRECT, and DAKOTA/EA require more function evaluations because of their initial calculations on a large number of sampling points. On the other hand, the remaining 20 solvers are terminated by other criteria such as tolerance on global solution and feasibility, leaving the number of function calls below the limit. However, notable differences still exist among these solvers. Sixteen solvers have a large number of iterations (more than 800), while other solvers require fewer function evaluations. Specifically, six local DFO solvers, BOBYQA, DAKOTA/PATTERN, HOPSPACK, IMFIL, NEWUOA, and PRAXIS, need fewer than 250 iterations, which generates large savings in computational effort.

Table 8 shows that more function calls, in general, result in larger execution time, which is consistent with our intuitive assumption. ASA, DAKOTA/DIRECT, DAKOTA/EA, PSWARM, SNOBFIT, TOMLAB/LGO, and TOMLAB/MSNLP require more than 800 iterations on average and also have large average execution times. On the other hand, the solver with the fewest evaluations, BOBYQA, also has

Table 8 Computational effort of DFO solvers

DFO solver	Average number of iterations	Average execution time (sec)
ASA	1,051	1,447
BOBYQA	104	136
CMA-ES	836	959
DAKOTA/DIRECT	1,005	1,123
DAKOTA/EA	1,030	1,647
DAKOTA/PATTERN	241	383
DAKOTA/SOLIS-WETS	706	1,112
DFO	453	340
FMINSEARCH	777	1,123
GLOBAL	979	1,089
HOPSPACK	215	232
IMFIL	154	172
MCS	999	892
NEWUOA	150	122
NOMAD	556	569
PRAXIS	230	171
PSWARM	1,001	1,032
SID-PSM	367	363
SNOBFIT	1,001	1,502
TOMLAB/CGO	868	952
TOMLAB/GLB	863	924
TOMLAB/GLC	857	947
TOMLAB/GLCCLUSTER	907	1,053
TOMLAB/LGO	1,001	1,153
TOMLB/MSNLP	1,001	1,466
TOMLAB/MULTIMIN	994	1,039
TOMLAB/RBF	868	932

the smallest average execution time. Therefore, similar results are achieved by comparing computational effort in terms of average execution time.

5.2.1 Efficiency of DFO solvers

Figure 2 presents the efficiency for each DFO solver in tuning options for individual problem instances. As discussed in previous sections, the average number of iterations, which is chosen as the horizontal axis, presents the computational effort of each DFO solver, while the vertical axis indicates the quality of solutions in terms of average runtime reduction. Solvers that are located on the upper left corner of the figure indicate good efficiency. On the other hand, the approaches found on the lower right area have poor efficiency in tuning parameters. For further analysis, we separate the 27 solvers into three groups according to the number of average iterations.

In the first group, in the upper right corner of the figure, solvers ASA, CMA-ES, DAKOTA/DIRECT, DAKOTA/EA, GLOBAL, MCS, PSWARM, SNOBFIT, TOMLAB/CGO, TOMLAB/GLB, TOMLAB/GLC, TOMLAB/GLCCLUSTER, TOMLAB/LGO, TOMLAB/MSNLP, TOMLAB/MULTIMIN, and TOMLAB/RBF require more than 800 iterations and reduce the execution time by more than 48%. All of these solvers are global, including eight deterministic, six stochastic, and two hybrid solvers. Eight of these solvers achieve time reductions larger than 50%, while the average solution quality of this group is above 51%. Even the most inefficient solver in this group, TOMLAB/RBF, still obtains a runtime reduction of 48%. However, it is not surprising that these solvers report significant solution quality because they perform more function evaluations on different points. This makes it more likely for them to obtain better solutions. Therefore, larger computational effort produces a better solution quality in option tuning.

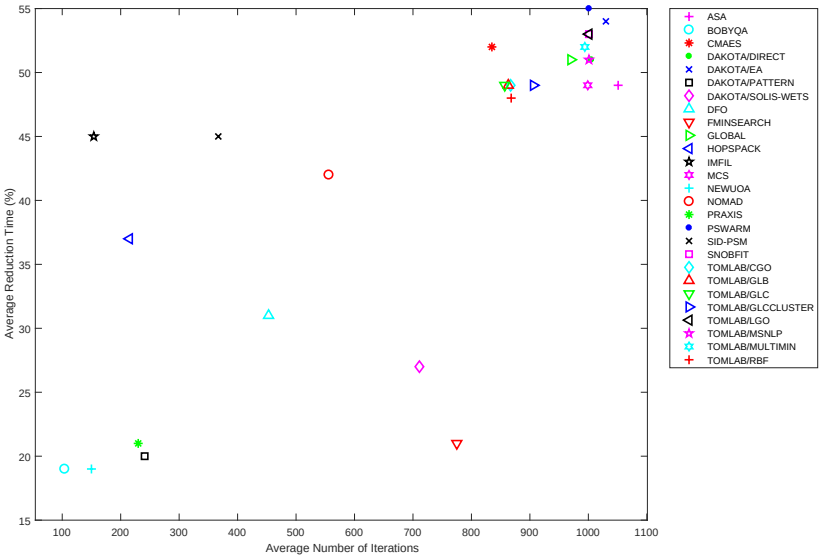

Fig. 2 Efficiency of DFO solvers

In the second group, solvers DFO, DAKOTA/SOLIS-WETS, FMINSEARCH, and NOMAD, require between 400 and 800 function evaluations on average. Similar to the solvers of the first group, it is reasonable to expect good solution quality for all solvers in this group. Unlike the former group, these solvers can be divided into two subgroups depending on their performance. NOMAD performs well in identifying optimal options by obtaining a runtime reduction of 42%. The other solvers in this group are not very efficient in obtaining runtime reductions of less than 32%. Moreover, the direct local solver FMINSEARCH is the worst implementation among all DFO solvers of this group. It requires a large number of iterations (777) and has the smallest runtime reduction (less than 21%).

BOBYQA, DAKOTA/PATTERN, HOPSPACK, IMFIL, NEWUOA, PRAXIS, and SID-PSM are included in the third and final group in our analysis. Significant differences between their performances are observed in Figure 2. All of them are local DFO solvers. Due to low requirements in computational effort, four local solvers report runtime reductions of less than 22%, which is significantly below average. Four direct local solvers derived from similar algorithms, BOBYQA, DAKOTA/PATTERN, NEWUOA, and PRAXIS, are located on the lower left corner with similar performances. Nevertheless, the other three solvers challenge our intuition by producing unexpectedly good performances. Particularly, the local solvers IMFIL and SID-PSM reduce the execution time by 45% and require a remarkably small number of function evaluations (less than 367). Moreover, HOPSPACK reduces the average runtime by 37% with 215 function evaluations. Hence, these solvers indicate remarkable efficiency in option tuning, making them suitable for searching for optimal options when the total execution time is strictly limited.

We also study the efficiency of each DFO solver when considering only the problems that BARON can solve within a fixed amount of time using the default options. Figures 3-6 present the efficiency for each DFO solver in tuning options for those problem instances that BARON can solve within 1, 2, 5, and 10 seconds, respectively. As it was expected, most solvers are more efficient on tuning easier problems. However, their efficiency does not drop significantly when tuning harder problems. Comparing the reduction time for problems solved by BARON within 1 second (Figure 3) with the reduction time for all problems (Figure 2), the efficiency of each DFO solver changes by at most 5%. On the other hand, TOMLAB/MSNLP and TOMLAB/MULTIMIN increase their average runtime reduction by 1% when tuning all problems.

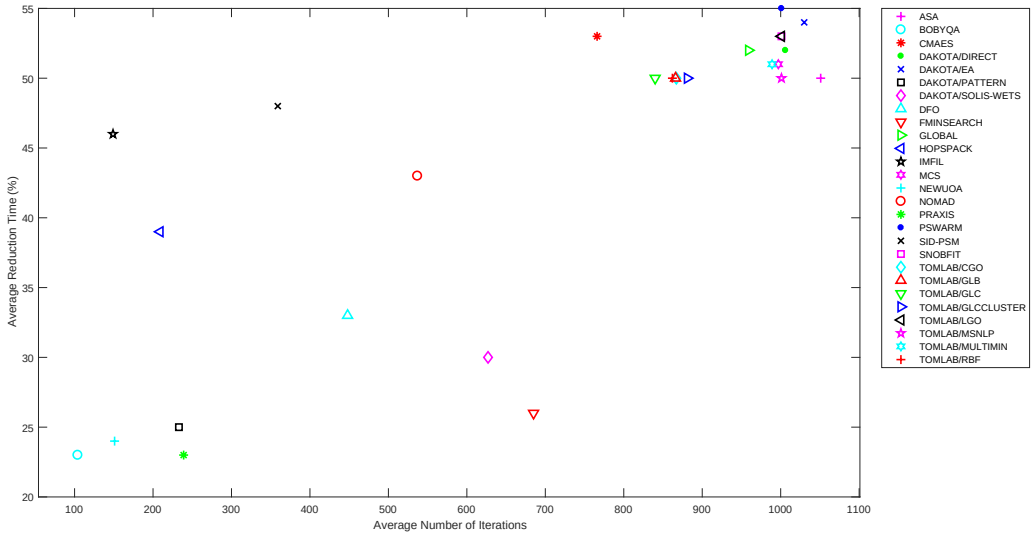

Fig. 3 Efficiency of DFO solvers for problems solved by BARON within 1 second using the default options

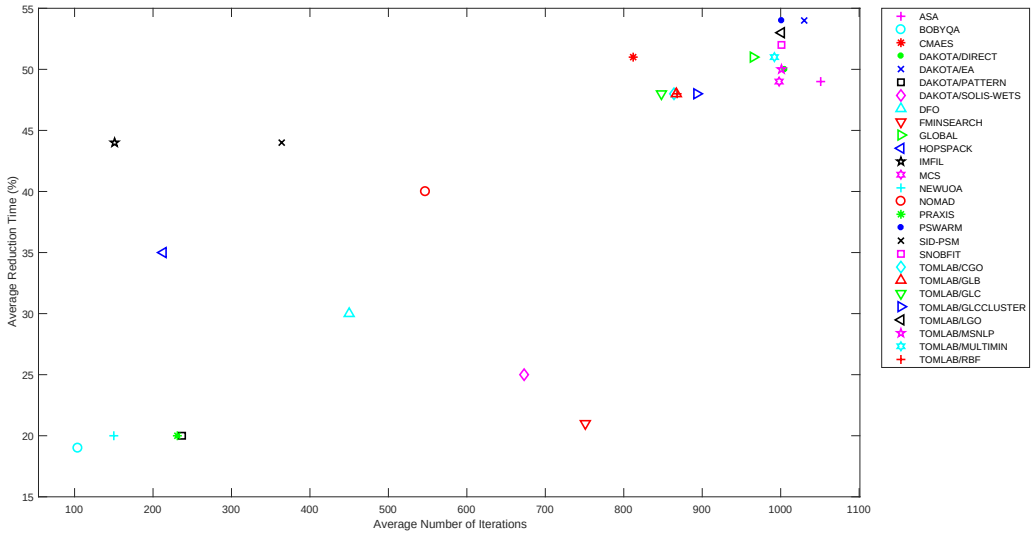

Fig. 4 Efficiency of DFO solvers for problems solved by BARON within 2 seconds using the default options

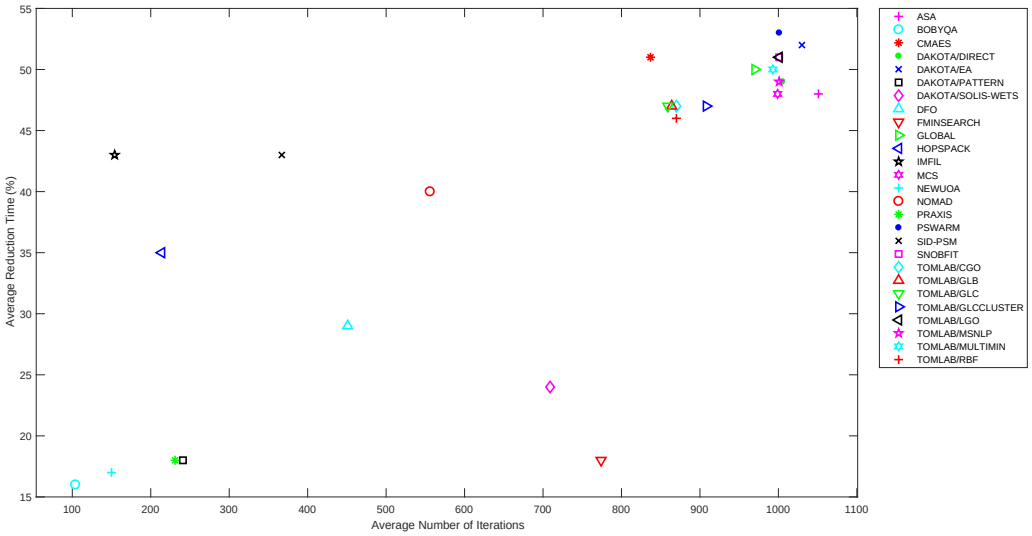

Fig. 5 Efficiency of DFO solvers for problems solved by BARON within 5 seconds using default options

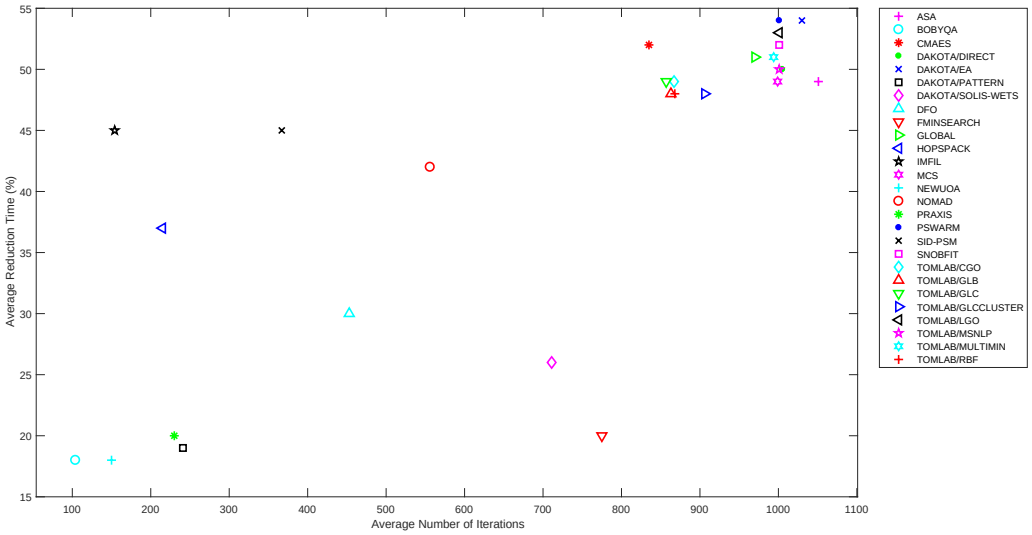

Fig. 6 Efficiency of DFO solvers for problems solved by BARON in less than 10 seconds using default options

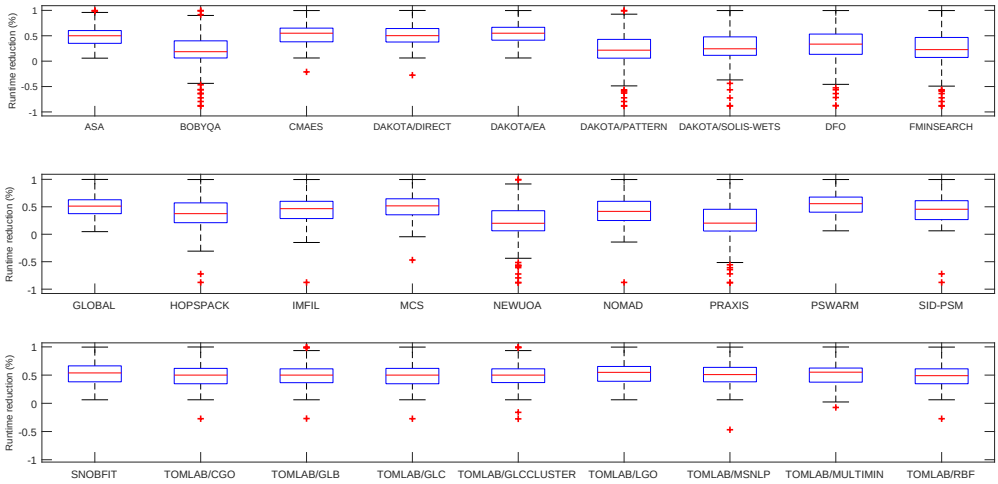

Fig. 7 Variance of the runtime reduction for each DFO solver

Figure 7 presents the box plot of the runtime reduction for each DFO solver. ASA, DAKOTA/EA, GLOBAL, PSWARM, SNOBFIT, and TOMLAB/LGO were able to tune all problems ($rr > 0$ for all problems). These solvers have also the best performance in terms of the average runtime reduction. Therefore, it is evident that the best solvers on this computational study can also improve BARON's execution time on all individual problems. On the other hand, all other DFO solvers have at least one problem for which they found a solution that decreased BARON's execution time. The average poor performance of BOBYQA, DAKOTA/PATTERN, FMINSEARCH, NEWUOA, and PRAXIS is mainly caused by their poor performance on specific instances. All these solvers decreased BARON's performance on more than 22 problem instances.

In general, a large number of iterations results in a better solution quality. However, solvers IMFIL and SID-PSM show outstanding efficiency by significantly reducing runtime while requiring a relatively small number of function evaluations. HOPSPACK is also an efficient solver. On the other hand, the MATLAB-based local solver FMINSEARCH is the most inefficient solver of this computational study.

5.3 Analysis

5.3.1 Comparison between local and global DFO solvers

As mentioned in Section 3, DFO solvers can be classified into two groups depending on whether they implement local or global algorithms. Our computational results show that global solvers greatly outperform local solvers in reducing computational effort. The average runtime reduction is about 49% for 17 global solvers and 30% for 10 local solvers. Even the best local solvers, IMFIL and SID-PSM with an average runtime reduction of at least 45%, perform worse than the average of global solvers. Therefore, it is not surprising that all efficient DFO tuning solvers are based on global algorithms, while the poor ones belong to the local group. As already mentioned, the objective function for option tuning is often nonsmooth and complex, which may result in a large number of local optima. Hence, local solvers are highly likely to be trapped in these local solutions and terminate their execution before they reach the global optima. On the other hand, global solvers have an ability to search for the best options throughout

Table 9 Performance of local DFO solvers

Direct solvers			Model-based solvers		
DFO Solver	Runtime reduction (%)	Average number of iterations	DFO Solver	Runtime reduction (%)	Average number of iterations
DAKOTA/PATTERN	20	241	BOBYQA	19	104
FMINSEARCH	21	777	DFO	31	453
HOPSPACK	37	215	IMFIL	45	154
NOMAD	42	556	NEWUOA	19	150
PRAXIS	21	230			
SID-PSM	45	367			
Average	31	398	Average	29	215

the feasible region. The only exception is the global solver DAKOTA/SOLIS-WETS, which results in an average runtime reduction of 27%, which is slightly worse than that of the average of local solvers.

Regarding the computational effort required by DFO solvers, global solvers require an average of 937 iterations, while local ones require an average of only 325 function evaluations. This confirms our intuitive assumption that more iterations tend to result in better option configurations. Interestingly, with almost three times ($937/325 = 2.9$) more computational effort, global DFO solvers increase BARON's performance nearly twofold ($49/30 = 1.6$).

5.3.2 Comparison between local DFO solvers

As discussed in Section 3, local DFO solvers are further classified into two groups, direct and model-based ones, with the latter utilizing surrogate models. Solvers based on direct algorithms obtained a runtime reduction of 31% on average, which slightly outperforms a runtime reduction of 29% obtained by model-based solvers. Yet, the highest runtime reduction was achieved by both a direct solver, SID-PSM, and a model-based solver, IMFIL. Additionally, to attain better performance in solution quality, direct local solvers need twice as many as function evaluations. Hence, no clear dominance relationship exists between these two groups of solvers.

Different solvers within the same group may lead to significantly different performance. For instance, in the model-based subgroup, solver IMFIL, based on the implicit filtering method, dramatically outperforms all other three trust region implementations. At the same time, BOBYQA and NEWUOA have similar performances in tuning parameters because both of them are derived from the same algorithm. However, even implementations based on similar algorithms can result in significantly different performances. For example, in the direct local solver subgroup, SID-PSM reduces execution time by 45%, while DAKOTA/PATTERN, though based on a similar algorithm, reduces the execution time by 20%. Moreover, different implementations may be suited for different problems. For instance, since the direct local solver DFO is initially designed to solve low-dimension smooth problems, it may not work well for our nonsmooth performance function.

Solvers IMFIL, NOMAD, and SID-PSM obtain good runtime reductions of over 42% by needing fewer than 600 function evaluations. These solvers are able to find optimized parameters, making them well suited for option tuning within strict limitations of the total computational effort.

5.3.3 Comparisons between global DFO solvers

A similar classification for global solvers introduced in Section 3, divides them into three classes depending on whether they apply random search strategies or not. As listed in Table 2, there are eight deterministic, seven stochastic, and two hybrid solvers. Hybrid solvers have significantly better performance (52% on average) than the deterministic (50% on average) and stochastic solvers (49 on average).

Table 10 Performance of global DFO solvers

Deterministic solvers			Stochastic/hybrid solvers		
DFO Solver	Runtime reduction (%)	Average number of iterations	DFO Solver	Runtime reduction (%)	Average number of iterations
DAKOTA/ DIRECT	51	1,005	ASA	49	1,051
MCS	49	999	CMA-ES	52	836
SNOBFIT	53	1,001	DAKOTA/EA	54	1,030
TOMLAB/ CGO	49	868	DAKOTA/ SOLIS-WETS	27	706
TOMLAB/ GLB	49	863	GLOBAL	51	979
TOMLAB/ GLC	49	857	PSWARM	55	1,001
TOMLAB/ GLCCLUSTER	49	907	TOMLAB/ LGO	53	1,001
TOMLAB/ RBF	48	842	TOMLAB/ MSNLP	51	1,001
			TOMLAB MULTIMIN	52	983
Average	50	918	Average	49	954

Different global solvers also result in significantly different performance. For most implementations, the average runtime reduction is constrained by a relatively small range from 48% to 55%. Hence, global solvers are very efficient in tuning parameters. However, the solver DAKOTA/SOLIS-WETS reported a runtime reduction of only 27% and obtained a much poorer solution quality among this subgroup. PSWARM is the best solver, reducing the execution time by 55%. CMA-ES, DAKOTA/EA, TOMLAB/LGO, TOMLAB/MULTIMIN and SNOBFIT also show great performance, obtaining a reduction time of at least 52%.

6 Conclusions

Option configuration plays an important role in improving performance of optimization solvers, both in terms of computational effort and solution quality. Option tuning can be treated as an optimization problem. However, the complex relationship between solver options and their performance is often nonsmooth and nondifferential, which makes optimization algorithms requiring explicit functional representations of the objective function or the constraints unsuitable to solve solver tuning problems. As a result, DFO methods are widely utilized to solve tuning problems. Previous works have implemented several different DFO strategies to tune various local or integer solvers and reported that tuning can lead to significant performance gains. The first goal of this paper was to investigate whether similar improvements in solver performance can also be attained by tuning options for global optimizer BARON. Our second objective was to perform a systematic comparison of a large number of DFO solvers in order to evaluate their abilities in terms of tuning a global solver.

Amongst the 27 DFO solvers that we applied to tune BARON's options, some solvers clearly stood out. For individual problems, PSWARM outperformed all other solvers using three performance metrics and was very closely followed by CMA-ES, DAKOTA/EA, TOMLAB/LGO, TOMLAB/MULTIMIN, and SNOBFIT. For the more difficult problem of tuning for an entire library of problems, SNOBFIT resulted in the highest runtime reduction, followed by CMA-ES and PSWARM. Also in the context of tuning for the entire library, the local solver IMFIL was identified as a highly effective solver because it resulted in significant runtime reduction after a very small number of function evaluations.

It is not surprising that BARON's performance could be improved for individual problems after parameter tuning with DFO solvers. Somewhat surprising was the fact that runtime reductions averaged 57% and exceeded 90% for certain problems. Even more surprising was the discovery of a set of options that resulted in 34% runtime reduction when applied to a library of 126 problems. Given that BARON has been in development for over two decades and is considered as the currently leading global solver, these results speak highly for the value of option tuning using DFO solvers and suggest that this approach be applied to other global solvers as well.

References

1. Audet, C., Dennis Jr., J.E.: Mesh adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization **17**, 188–217 (2006)
2. Audet, C., Orban, D.: Finding optimal algorithmic parameters using derivative-free optimization. Society for Industrial and Applied Mathematics **17**, 642–664 (2001)
3. Bao, X., Sahinidis, N.V., Tawarmalani, M.: Multiterm polyhedral relaxations for nonconvex, quadratically-constrained quadratic programs. Optimization Methods and Software **24**, 485–504 (2009)
4. Bartholomew-Biggs, M.C., Parkhurst, S.C., Wilson, S.P.: Using DIRECT to solve an aircraft routing problem. Computational Optimization and Applications **21**, 311–323 (2002)
5. Baz, M., Hunsaker, B.: Automated tuning of optimization software parameters. Tech. rep., Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA (2007)
6. Baz, M., Hunsaker, B., Prokopyev, O.: How much do we "pay" for using default parameters? Computational Optimization and Applications **48**, 91–108 (2011)
7. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib–A collection of test models for mixed-integer nonlinear programming. INFORMS Journal on Computing **15**, 114–119 (2003)
8. Chen, W., Shao, Z., Wang, K., Chen, X., Biegler, L.T.: Random sampling-based automatic parameter tuning for nonlinear programming solvers. Industrial & Engineering Chemistry Research **50**, 3907–3918 (2011)
9. Conn, A.R., Scheinberg, K., Toint, P.L.: On the convergence of derivative-free methods for unconstrained optimization. *In M. D. Buhmann and A. Iserles (eds.)*, Approximation Theory and Optimization, Tribute to M. J. D. Powell, *Cambridge University Press, Cambridge, UK* pp. 83–108 (1996)
10. Custódio, A.L., Dennis Jr., J.E., Vicente, L.N.: Using simplex gradients of nonsmooth functions in direct search methods. IMA Journal of Numerical Analysis **28**, 770–784 (2008)
11. Fan, S.S., Zahara, E.: A hybrid simplex search and particle swarm optimization for unconstrained optimization. European Journal of Operational Research **181**, 527–548 (2007)
12. Fowler, K.R., Reese, J.P., Kees, C.E., Dennis Jr., J.E., Kelley, C.T., Miller, C.T., Audet, C., Booker, A.J., Couture, G., Darwin, R.W., Farthing, M.W., Finkel, D.E., Gablonsky, J.M., Gray, G., Kolda, T.G.: A comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. Advances in Water Resources **31**, 743–757 (2008)
13. GLOBAL Library. http://www.gamsworld.org/global/globallib.htm
14. Gutmann, H.M.: A radial basis function method for global optimization. Journal of Global Optimization **19**, 201–227 (2001)
15. Han, J., Kokkolaras, M., Papalambros, P.Y.: Optimal design of hybrid fuel cell vehicles. Journal of Fuel Cell Science and Technology **5**, 041,014 (2008)
16. Hutter, F., Hoos, H.H., , Stützle, T.: Automatic algorithm configuration based on local search. *In A. Howe and R. C. Holte (eds.)*, Proceedings of the Twenty-second National Conference on Artificial Intelligence (AAAI'07), *AAAI Press / The MIT Press, Menlo Park, CA* pp. 1152–1157 (2007)
17. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. LNCS **6140**, 186–202 (2010)
18. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configurations. *In* Learning and Intelligent Optimization, *Springer* pp. 507–523 (2011)
19. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An antomatic algorithm configuration framework. Journal of Artificial Intelligence Research **36**, 267–306 (2009)
20. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. Journal of Global Optimization **14**, 331–355 (1999)
21. Hvattum, L.M., Glover, F.: Finding local optima of high-dimensional functions using direct search methods. European Journal of Operational Research **195**, 31–45 (2009)
22. Ingber, L.: Adaptive Simulated Annealing (`ASA`). http://www.ingber.com/#ASA
23. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Application **79**, 157–181 (1993)
24. Liu, M.L., Sahinidis, N.V., Shectman, J.P.: Planning of chemical process networks via global concave minimization. *In I. E. Grossmann (ed.)*, Global Optimization in Engineering Design, *Kluwer Academic Publishers, Boston, MA* pp. 195–230 (1996)

25. Mongeau, M., Karsenty, H., Rouzé, V., Hiriart-Urruty, J.B.: Comparison of public-domain software for black box global optimization. Optimization Methods & Software **13**, 203–226 (2000)
26. Moré, J., Wild, S.: Benchmarking derivative-free optimization algorithms. Optimization Online Digest pp. 1–20 (2008)
27. Nelder, J.A., Mead, R.: A simplex method for function minimization. Computer Journal **7**, 308–313 (1965)
28. Powell, M.J.D.: A direct search optimization method that models the objective and constraint functions by linear interpolation. In: S. Gomez, J.P. Hennart (eds.) Advances in Optimization and Numerical Analysis, pp. 51–67. Kluwer Academic, Dordrecht (1994)
29. Powell, M.J.D.: Recent research at Cambridge on radial basis functions. Tech. rep., Department of Applied Mathematics and Theoretical Physics, University of Cambridge (1998)
30. Puranik, Y., Sahinidis, N.V.: Bounds tightening based on optimality conditions for nonconvex box-constrained optimization. Journal of Global Optimization **67**, 59–77 (2017)
31. Puranik, Y., Sahinidis, N.V.: Domain reduction techniques for global NLP and MINLP optimization. Constraints **22**, 338–376 (2017)
32. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: A review of algorithms and comparison of software implementations. Journal of Global Optimization **56**, 1247–1293 (2013)
33. Ryoo, H.S., Sahinidis, N.V.: Global optimization of nonconvex NLPs and MINLPs with applications in process design. Computers & Chemical Engineering **19**, 551–566 (1995)
34. Ryoo, H.S., Sahinidis, N.V.: A branch-and-reduce approach to global optimization. Journal of Global Optimization **8**, 107–139 (1996)
35. Sahinidis, N.V.: BARON: A general purpose global optimization software package. Journal of Global Optimization **8**, 201–205 (1996)
36. Sahinidis, N.V.: Global optimization and constraint satisfaction: The branch-and-reduce approach. *In C. Bliek, C. Jermann, and A. Neumaier (eds.)*, Global Optimization and Constraint Satisfaction, *Lecture Notes in Computer Science, Vol. 2861, Springer, Berlin* pp. 1–16 (2003)
37. Sahinidis, N.V.: BARON 15.5.0: Global Optimization of Mixed-Integer Nonlinear Programs, *User's Manual* (2015)
38. Shectman, J.P., Sahinidis, N.V.: A finite algorithm for global minimization of separable concave programs. Journal of Global Optimization **12**, 1–36 (1998)
39. Spendley, W., Hext, G.R., Himsworth, F.R.: Sequential application for simplex designs in optimisation and evolutionary operation. Technometrics **4**, 441–461 (1962)
40. Stewart, C.R.: . Master's thesis, Vriginia Commonwealth University, Richmond, VA (2010)
41. Tawarmalani, M., Ahmed, S., Sahinidis, N.V.: Product disaggregation and relaxations of mixed-integer rational programs. Optimization and Engineering **3**, 281–303 (2002)
42. Tawarmalani, M., Sahinidis, N.V.: Convex extensions and convex envelopes of l.s.c. functions. Mathematical Programming **93**, 247–263 (2002)
43. Tawarmalani, M., Sahinidis, N.V.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**, 563–591 (2004)
44. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Mathematical Programming **103**, 225–249 (2005)
45. Torczon, V.J.: On the convergence of pattern search algorithms. SIAM Journal on Optimization **7**, 1–25 (1997)
46. Vaz, A.I.F.: `PSwarm` Home Page. http://www.norg.uminho.pt/aivaz/pswarm/
47. Vaz, A.I.F., Vicente, L.N.: A particle swarm pattern search method for bound constrained global optimization. Journal of Global Optimization **39**, 197–219 (2007)
48. Zorn, K., Sahinidis, N.V.: Global optimization of general nonconvex problems with intermediate bilinear substructures. Optimization Methods and Software **29**, 442–462 (2013)