Empirical comparisons of several derivative free optimization algorithms

A. Auger[1,2], **N. Hansen**[1,2], **J. M. Perez Zerpa**[1], **R. Ros**[1], **M. Schoenauer**[1,2]

[1] TAO Project-Team, INRIA Saclay – Ile-de-France
LRI, Bat 490 Univ. Paris-Sud 91405 Orsay Cedex France
[2] Microsoft Research-INRIA Joint Centre
28 rue Jean Rostand, 91893 Orsay Cedex, France

Abstract — In this paper, the performances of the quasi-Newton BFGS algorithm, the NEWUOA derivative free optimization algorithm, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), the Differential Evolution (DE) algorithm and a Particle Swarm Optimization (PSO) algorithm are compared experimentally on benchmark functions reflecting important challenges encountered in real-world optimization problems. Dependence of the performances in the conditioning of the problem and rotational invariance of the algorithms are in particular investigated.

1 Introduction

Continuous Optimization Problems (COPs) aim at finding the global optimum (or optima) of a real-valued function (aka *objective* function) defined over (a subset of) a real vector space. COPs commonly appear in everyday's life of many scientists, engineers and researchers from various disciplines, from physics to mechanical, electrical and chemical engineering to biology. Problems such as model calibration, process control, design of parameterized parts are routinely modeled as COPs. Furthermore, in many cases, very little is known about the objective function. In the worst case, it is only possible to retrieve objective function values for given inputs, and in particular the user has no information about derivatives, or even about some weaker characteristics of the objective function (e.g. monotonicity, roughness, ...). This is the case, for instance, when the objective function is the output of huge computer programs ensuing from several years of development, or when experimental processes need to be run in order to compute objective function values. Such problems amount to what is called *Black-Box Optimization* (BBO).

Because BBO is a frequent situation, many optimization methods (aka *search algorithms*) have been proposed to tackle BBO problems, that can be grossly classified in two classes: (i) deterministic methods include classical derivative-based algorithms, in which the derivative is numerically computed by finite differences, and more recent Derivative Free Optimization (DFO) algorithms [1], like pattern search [2] and trust region methods [3]; (ii) stochastic methods rely on random variables sampling to better explore the search space, and include recently introduced bio-inspired algorithms (see Section 3).

However, the practitioner facing a BBO problem has to choose among those methods, and there exists no theoretical solid ground where he can stand to perform this choice, first because he does not know much about his objective function, but also because all theoretical results either make simplifying hypotheses that are not valid for real-world problems, or give results that do not yield any practical outcome. Moreover, most of BBO methods require some parameter tuning,

and here again very little help is available for the practitioner, who is left with a blind and time-consuming test-and-trial approach.

In such context, this paper proposes an experimental perspective on BBO algorithms comparisons. Rigorous procedures to compare the results of different BBO algorithms have been proposed [4], taking into account the stochastic nature of many of them, and giving fair chances to each one of them. However, a critical issue in such experiments is that of the benchmark suite. And because no set of real-world problems can be guaranteed to cover all possible cases of difficult COPs, the approach that has been chosen here is to build artificial test functions with some precise characteristics that are known to be possible sources of difficulty for optimization (e.g. ill-conditioning, non-separability, non-convexity, ruggedness, ...). Such experimental results could then be cautiously generalized, leaving only a few good-performing algorithms in each specific context. Of course, in real-life BBO situations, it is assumed that nothing is known about the objective function. However, the user sometimes has some partial information (e.g. because his problem is known to be similar to other better-known problems) that might lead him to decide for a BBO method that is (experimentally) known to perform well, 'in vitro', in his precise situation. But on the other hand, assuming absolutely nothing is known in advance about the objective function, running the champion algorithms as identified in perfectly controlled environment might give him some information about his function (e.g. if numerical gradient-based algorithms perform 100 times better than all other methods, his problem is probably very similar to a quadratic problem). This paper is a first step toward such 'in vitro' results. Next, in Section 2, some characteristics of the objective function are surveyed that are known to make the corresponding BBO problem hard. Section 3 introduces the algorithms that will be compared here. Section 4 then introduces the test bench that illustrates the different difficulties highlighted in Section 2, as well as the experimental conditions of the comparisons. The results are presented and discussed in Section 5, and the paper ends with a summary in Section 6. This paper is organized as follows: in Section 2, some characteristics of the objective function are surveyed that are known to make the corresponding BBO problem hard. Section 3 introduces the algorithms that will be compared here. Section 4 then introduces the test bench that illustrates the different difficulties highlighted in Section 2, as well as the experimental conditions of the comparisons. The results are presented and discussed in Section 5.

2 What makes a search problem difficult?

In this section, we discuss problem characteristics that are especially challenging for search algorithms.

Ill-conditioning The conditioning of a problem can be defined as the range (over a level set) of the maximum improvement of objective function value in a ball of small radius centered on the given level set. In the case of convex quadratic functions ($f(x) = \frac{1}{2}x^T Hx$ where H is a symmetric definite matrix), the conditioning can be exactly defined as the condition number of the Hessian matrix H, i.e., the ratio between the largest and smallest eigenvalue. Since level sets associated to a convex quadratic function are ellipsoids, the condition number corresponds to the squared ratio between the largest and shortest axis lengths of the ellipsoid. Problems are typically considered as ill-conditioned if the conditioning is larger than 10^5. In practice we have seen problems with conditioning as large as 10^{10}.

Non-separability An objective function $f(x_1,\ldots,x_n)$ is separable if the optimal value for any variable x_i can be obtained by optimizing $f(\widetilde{x}_1,\ldots,\widetilde{x}_{i-1},x_i,\widetilde{x}_{i+1},\ldots,\widetilde{x}_n)$ for any fixed choice of the variables $\widetilde{x}_1,\ldots,\widetilde{x}_{i-1},\widetilde{x}_{i+1},\ldots,\widetilde{x}_n$. Consequently optimizing an n-dimensional separable objective

function reduces to optimizing n one-dimensional functions. Functions that are additively decomposable, i.e., that can be written as $f(x) = \sum_{i=1}^{n} f_i(x_i)$ are separable. One way to render a separable test function non-separable is to rotate first the vector x, which can be achieved by multiplying x by an orthogonal matrix B: if $x \mapsto f(x)$ is separable, the function $x \mapsto f(Bx)$ might be non-separable for all non-identity orthogonal matrices B.

Non-convexity Some BBO methods implicitly assume or exploit convexity of the objective function. Composing a convex function f to the left with a monotonous transformation $g : \mathbb{R} \to \mathbb{R}$ can result in a non-convex function, for instance the one-dimensional convex function $f(x) = x^2$ composed with $g(.) = sign(.)|.|^{1/4}$ becomes the non-convex function $\sqrt{|.|}$.

In this paper we will quantitatively assess the performance dependency on the conditioning of objective functions, investigate separable and non-separable problems as well as study the dependence of the performances in the convexity of the problem.

3 Algorithms tested

This section introduces the different algorithms that will be compared in this paper. They have been chosen because they are considered to be the champions in their category, both in the deterministic optimization world (BFGS and NEWUOA) and in the stochastic bio-inspired world (CMA-ES, DE and PSO). They will also be a priori discussed here with respect to the difficulties of continuous optimization problems highlighted in Section 2.

3.1 The algorithms

BFGS is a well-known quasi-Newton algorithm. It has a proven convergence to a stationary point provided the starting point is close enough from the solution, and the objective function is regular. Because it is blindly used by many scientists facing optimization problems, the Matlab® version of BFGS `fminunc` has been used here, with its default parameters. The algorithm is stopped whenever the objective improvement in one iteration is less than 10^{-25}. Of course, in BBO contexts, the gradients have to be computed numerically.

NEWUOA (NEW Unconstrained Optimization Algorithm) [3] is a DFO algorithm using the trust region paradigm. NEWUOA computes a quadratic interpolation of the objective function in the current trust region, and performs a truncated conjugate gradient minimization of the surrogate model in the trust region. It then updates either the current best point or the radius of the trust region, based on the a posteriori interpolation error. The implementation by Matthieu Guibert (`http://www.inrialpes.fr/bipop/people/guilbert/newuoa/newuoa.html`) has been used.

An important parameter of NEWUOA is the number of points that are used to compute the interpolation. As recommended [3], $2n+1$ points have been used. Other parameters are the initial and final radii of the trust region, respectively governing the initial 'granularity' and the precision of the search. After some preliminary experiments, values of 100 and 10^{-15} were chosen.

CMA-ES is an Evolution Strategy (ES) [5] algorithm: from a set of 'parents', 'offspring' are created by sampling Gaussian distributions, and the best of the offspring (according to the objective function values) become the next parents. The Covariance Matrix Adaptation [6] uses the path that has been followed by evolution so far to (i) adapt the step-size, a scaling parameter that tunes the granularity of the search, by comparing the actual path length to that of a random walk; (ii) modify the covariance matrix of the multivariate Gaussian distribution in order to increase the likelihood of beneficial moves. A single Gaussian distribution is maintained, centered at a linear

combination of the parents. CMA-ES proposes robust default parameters: the population size is set to $4 + \lfloor 3\log(n) \rfloor$ and the initial step-size to a third of the parameters range. The version used here (Scilab 0.92) implements weighted recombination and rank-μ update [7] (version 0.99 is available at `http://www.lri.fr/~hansen/cmaes_inmatlab.html`).

PSO (Particle Swarm Optimization) [8] is a bio-inspired algorithm based on the biological paradigm of a swarm of particles that 'fly' over the objective landscape, exchanging information about the best solutions they have 'seen'. More precisely, each particle updates its velocity, stochastically twisting it toward the direction of the best solutions seen by (i) itself and (ii) some parts of the whole swarm; it then updates its position according to its velocity and computes the new value of the objective function. A Scilab transcription of the Standard PSO 2006, the latter available at *PSO Central* `http://www.particleswarm.info/`, was used with its default settings.

Differential Evolution (DE) [9] borrows from Evolutionary Algorithms (EAs) the paradigm of an evolving population. However, a specific 'mutation' operator is used that adds to an individual the difference between two others from the population. Standard uniform crossover is also used. The implementation posted by the original authors at `http://www.icsi.berkeley.edu/~storn/code.html` was used here. However, the authors confess that the results highly depend on the parameter tuning. They propose 6 possible strategies, and extensive experiments (3×288 trials) on a moderately ill-conditioned problem lead us to consider the "*DE/local-to-best/1/bin*" strategy, where the difference vector is computed between a random point and the best point in the population. Also, the use of crossover seemed to have little beneficial impact on the results, so no crossover was used, thus making DE rotationally invariant. Moreover, the population size was set to the recommended value of $10n$, the weighting factor to $F = 0.8$.

3.2 Invariances

Some a priori comparisons can be made about those algorithms, related to the notion of *invariance*. Indeed, invariances add to the robustness of an algorithm: functions belonging to the same equivalence class with respect to some invariance property will look exactly the same for an algorithm that is invariant under the transformation defining this equivalence class.
Two sets of invariance properties are distinguished, whether they regard transformations of the objective function value or transformations of the search space. First, all comparison-based algorithms are invariant under monotonous transformations of the objective function, as comparisons are unaltered if the objective function f is replaced with some $g \circ f$ for some monotonous function g. All bio-inspired algorithms used in this paper are comparison-based, while the BFGS and NEWUAO are not.

Regarding transformations of the search space, all algorithms are trivially invariant under translation of the coordinate system. But let us consider some orthogonal rotations: BFGS is coordinate-dependent due to the computation of numerical gradients. NEWUOA is invariant under rotation when considering the complete quadratic model, i.e. built with $\frac{1}{2}(n+1)(n+2)$ points. This variant is however often more costly compared to the $2n+1$ one – but the latter is not invariant under rotation. The rotational invariance of CMA-ES is built-in, while that of DE depends whether or not crossover is used – as crossover relies on the coordinate system. This was one reason for omitting crossover here. Finally, PSO is (usually) not invariant under rotations, as all computations are done coordinate by coordinate [10, 11].

4 Test functions and experimental setup

Test functions The benchmark functions tested are given in Table 1. The functions are tested in their original axis-parallel version (i.e. B is the identity and $y = x$), and in rotated versions,

Table 1: Test functions with coordinate-wise initialization intervals and target function value, where $y := Bx$ implements an angle-preserving, linear transformation, *i.e.* B is orthogonal.

Function	α	Initialization	f_{target}
$f_{\text{elli}}(x) = \sum_{i=1}^{n} \alpha^{\frac{i-1}{n-1}} y_i^2$	$[1, 10^{10}]$	$[-20, 80]^n$	10^{-9}
$f_{\text{Rosen}}(x) = \sum_{i=1}^{n-1} \left(\alpha (y_i^2 - y_{i+1})^2 + (y_i - 1)^2\right)$	$[1, 10^{8}]$	$[-20, 80]^n$	10^{-9}
$f_{\text{elli}}^{1/4}(x) = \left(\sum_{i=1}^{n} \alpha^{\frac{i-1}{n-1}} y_i^2\right)^{1/4}$	$[1, 10^{10}]$	$[-20, 80]^n$	10^{-9}

where $y = Bx$. The orthogonal matrix B is chosen such that each column is uniformly distributed on the unit hypersphere surface [6], fixed for each run.

The Ellipsoid function f_{elli} is a convex-quadratic function where the parameter α is the condition number of the Hessian matrix that is varied between 1 and 10^{10} in our experiments. If $\alpha = 1$ the Ellipsoid is the isotropic separable sphere function. The function $f_{\text{elli}}^{1/4}$ has the same contour lines (level sets) as f_{elli}, however it is neither quadratic nor convex. For $\alpha \neq 1$, the functions f_{elli} and $f_{\text{elli}}^{1/4}$ are separable if and only if $B = I$.

The Rosenbrock function f_{Rosen} is non-separable, has its global minimum at $x = [1, 1, \ldots, 1]$ and, for large enough α and n, has one local minimum close to $x = [-1, 1, \ldots, 1]$, see also [12]. The contour lines of the Rosenbrock function show a bent ridge that guides to the global optimum (the Rosenbrock is sometimes called banana function) and the parameter α controls the width of the ridge. In the classical Rosenbrock function, α equals 100. For smaller α, the ridge becomes wider and the function becomes less difficult to solve. We vary α between one and 10^8.

Experimental Setup For each algorithm tested we conduct 21 independent trials of up to 10^7 function evaluations. For all algorithms, initial points have been sampled uniformly in the range $[-20, 80]^n$. If, for BFGS, no success was encountered, the number of trials was extended to 1001. We quantify the performance of the algorithms using the success performance $SP1$ used in [13], analyzed in [14], and also denoted as Q-measure in [15]. The $SP1$ equals the average number of function evaluations for successful runs divided by the ratio of successful runs, where a run is successful if the target function value 10^{-9} is reached before 10^7 function evaluations are exceeded. The $SP1$ is an estimator of the expected number of function evaluations to reach 10^{-9} if the algorithm is restarted until a success (supposing infinite time horizon) and assuming that the expected number of function evaluations for unsuccessful runs equals the expected number of evaluations for successful runs.

5 Results

Results are shown for dimension 20. Results for 10 and 40-D (see Appendix) reveal similar trends.

Ellipsoid functions: dependencies In Figure 1 a remarkable dependency of the performance ($SP1$) on the condition number can be observed in most cases. The two exceptions are PSO on the separable functions and DE. In the other cases the performance declines by at least a factor of ten for very ill-conditioned problems as for CMA-ES. The overall strongest performance decline is shown by PSO on the rotated functions. NEWUOA shows in general a comparatively strong decline, while BFGS is only infeasible for high condition numbers in the rotated case, reporting some numerical problems. The decline of CMA-ES is moderate.

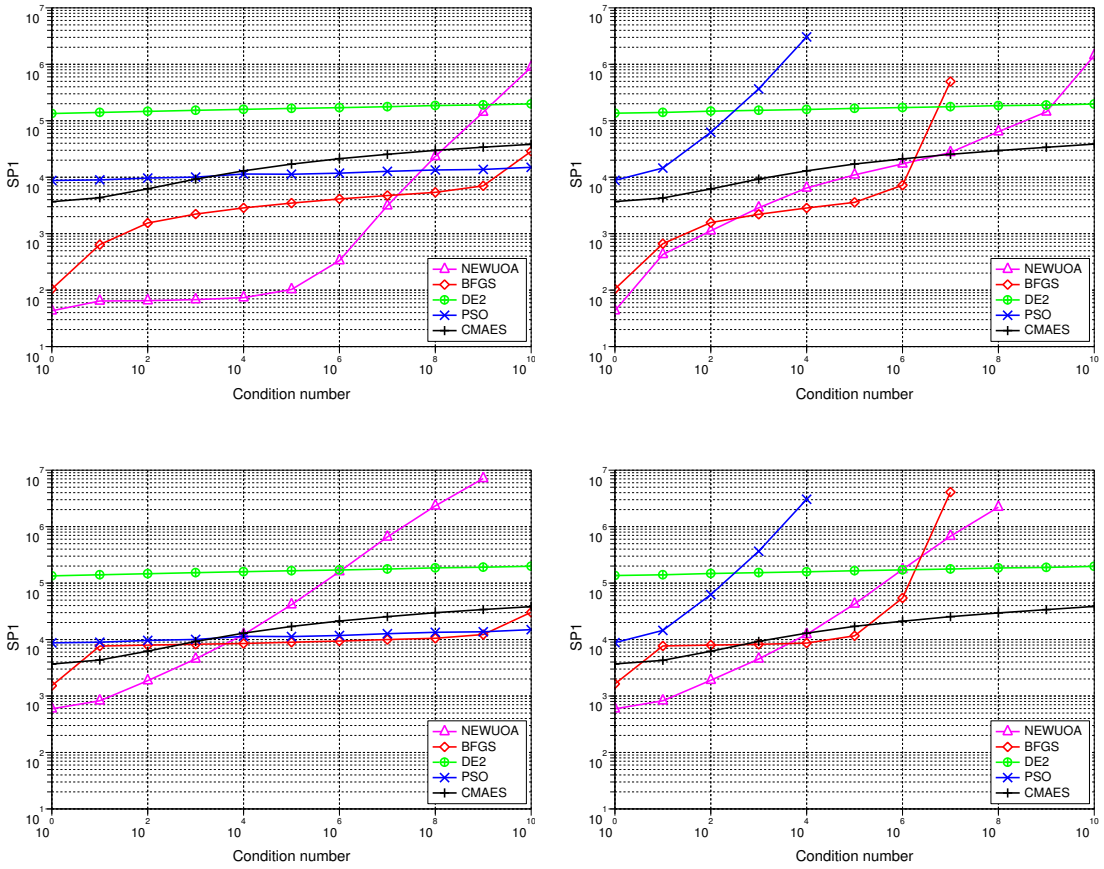

Figure 1: All ellipsoidal functions in 20D. Shown is $SP1$ (the expected running time or number of function evaluations to reach the target function value) versus condition number.

For CMA-ES and DE the results are (virtually) independent of the given ellipsoidal functions, where CMA-ES is consistently between five and forty times faster than DE. For PSO the results are identical on Ellipsoid and Ellipsoid$^{1/4}$, while the performance declines under rotation (left versus right figures) is very pronounced. A similar strong decline under rotation can be observed for NEWUOA on the Ellipsoid function for moderate condition numbers. BFGS, on the other hand, shows a strong rotational dependency on both functions only for large condition numbers $\geq 10^6$.

Switching from Ellipsoid (above) to Ellipsoid$^{1/4}$ (below) only effects BFGS and NEWUOA. BFGS becomes roughly five to ten times slower. A similar effect can be seen for NEWUOA on the rotated function. On the separable Ellipsoid function the effect is more pronounced, because NEWUOA performs exceptionally well on the separable Ellipsoid function.

Ellipsoid functions: comparison On the separable Ellipsoid function up to a condition number of 10^6 NEWUOA clearly outperforms all other algorithms. Also BFGS performs still better than PSO and CMA-ES while DE performs worst. On the separable Ellipsoid$^{1/4}$ function BFGS, CMA-ES and PSO perform similar. NEWUOA is faster for low condition numbers and slower for large ones. For condition number larger than 10^6, NEWUOA becomes even worse than DE.

On the rotated functions, for condition numbers larger than 10^3, PSO is remarkably outperformed by all other algorithms. On the rotated Ellipsoid function for moderate condition numbers BFGS and NEWUOA perform best and outperform CMA-ES by a factor of five, somewhat more for low condition numbers, and less for larger condition numbers. For large condition numbers

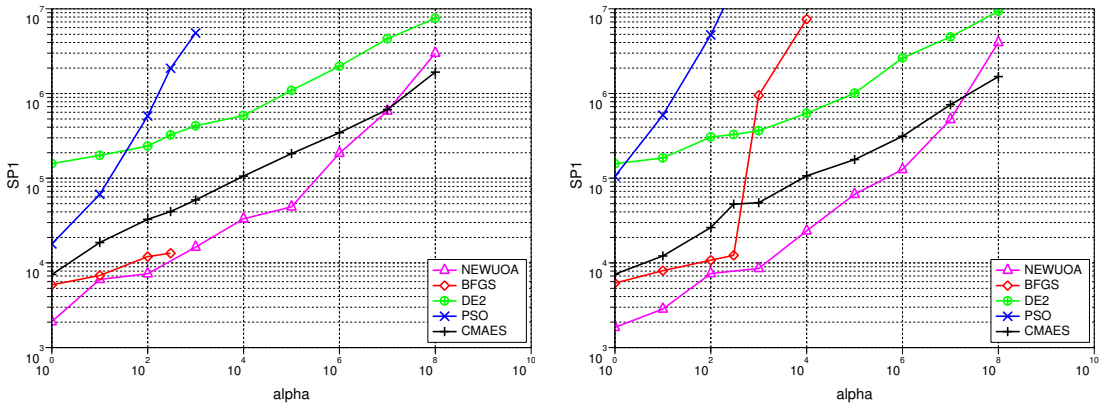

Figure 2: Rosenbrock function. Shown is $SP1$ (the expected running time or number of function evaluations to reach the target function value) versus conditioning parameter α.

CMA-ES becomes superior and DE is within a factor of ten of the best performance.

On the rotated Ellipsoid$^{1/4}$ BFGS and CMA-ES perform similar up to condition of 10^6. NEWUOA performs somewhat better for lower condition numbers up to 10^4. For larger condition numbers BFGS and NEWUOA decline and CMA-ES performs best.

Rosenbrock function On the Rosenbrock function NEWUOA is the best algorithm (Figure 2) . NEWUOA outperforms CMA-ES roughly by a factor of five, vanishing for very large values for the conditioning parameter α. For small α, BFGS is in-between, and for $\alpha > 10^4$ BFGS fails. DE is again roughly ten times slower than CMA-ES. Only PSO shows a strong dependency on the rotation of the function and the strongest performance decline with increasing α.

Scaling behaviors The scaling of the performance with search space dimension is similar for all functions (results not shown here for space reasons). CMA-ES, NEWUOA and PSO show the best scaling behavior. They slow down by a factor between five and ten in 40D compared to 10D. For BFGS the factor is slightly above ten, while for DE the factor is thirty or larger, presumably because the default population size increases linearly with the dimension.

6 Summary

In this paper we have conducted a comparison of BFGS, NEWUOA, and three stochastic bio-inspired optimization methods in a black-box optimization scenario. The empirical study was conducted on smooth functions with varying condition number. Aside from gradients being not provided, we consider these functions as the favorite playgrounds of BFGS and NEWUOA. We find that NEWUOA performs exceptional on separable quadratic functions, it performs in all cases very well with moderate condition numbers, but shows a comparatively steep performance decline with increasing ill-conditioning. BFGS performs well overall, but shows a strong decline on very ill-conditioned non-separable functions. For DE, the parameters are difficult to tune and yet it performs overall poorly with the single best parameter setting on our small function set. With the chosen parameters, DE shows the strongest robustness to ill-conditioning though. PSO performs similar to CMA-ES on the separable problems, with an even weaker dependency on the conditioning. On non-separable problems PSO performs very poorly even on moderately ill-conditioned functions. Finally, CMA-ES generally outperforms DE and PSO, while up to a moderate function conditioning BFGS and NEWUOA are significantly faster in most cases. Due to their invariance

properties, the performance results of CMA-ES and DE are the most stable ones and most likely to generalize to other functions.

Acknowledgements We would like to acknowledge Philippe Toint for his kind suggestions, and Nikolas Mauny for writing the Scilab transcription of the Standard PSO 2006 code.

References

[1] K. Scheinberg A. R. Conn and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79(3):397–415, 1997.

[2] Virginia Torczon. On the convergence of pattern search algorithms. *SIAM Journal on optimization*, 7(1):1–25, 1997.

[3] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. *Large Scale Nonlinear Optimization*, pages 255–297, 2006.

[4] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, KanGAL, IIT Kanpur, 2005.

[5] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, 1973.

[6] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[7] N. Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano et al., editor, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[8] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948, 1995.

[9] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, 1997.

[10] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. PSO facing non-separable and ill-conditioned problems. Research Report RR-6447, INRIA, 2008.

[11] D.N. Wilke, S. Kok, and A.A. Groenwold. Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on scale and frame invariance. *Int. J. Numer. Meth. Engng*, 70:985–1008, 2007.

[12] Yun-Wei Shang and Yu-Huang Qiu. A note on the extended rosenbrock function. *Evol. Comput.*, 14(1):119–126, 2006.

[13] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In Xin Yao et al., editors, *PPSN VIII, LNCS 3242*, pages 282–291. Springer, 2004.

[14] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

[15] V. Feoktistov. *Differential Evolution: In Search of Solutions*. Optimization and Its Applications. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006.

A All Results

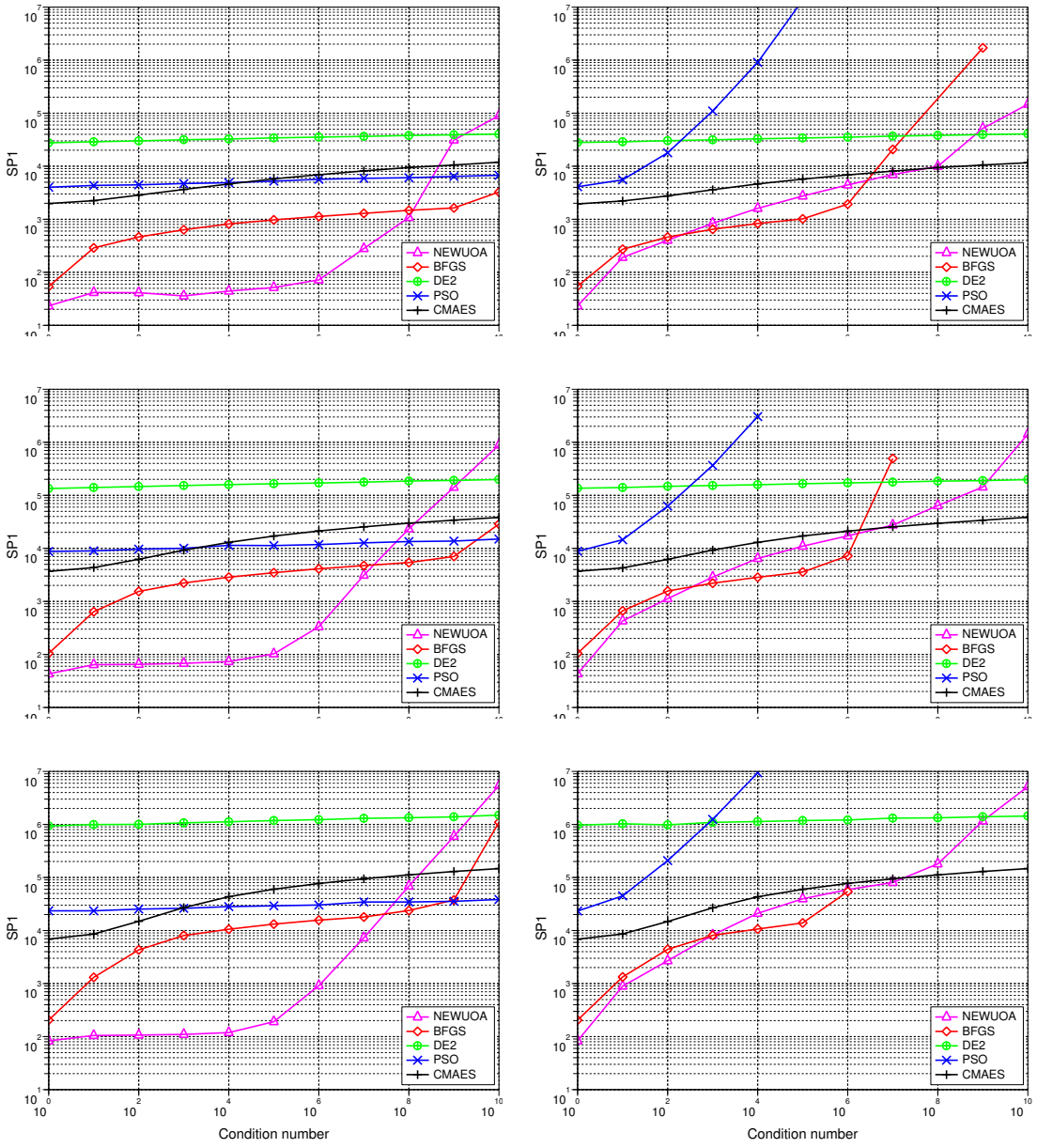

Figure 3: Ellipsoid function. Shown is *SP*1 (the expected running time or number of function evaluations to reach the target function value) versus condition number.

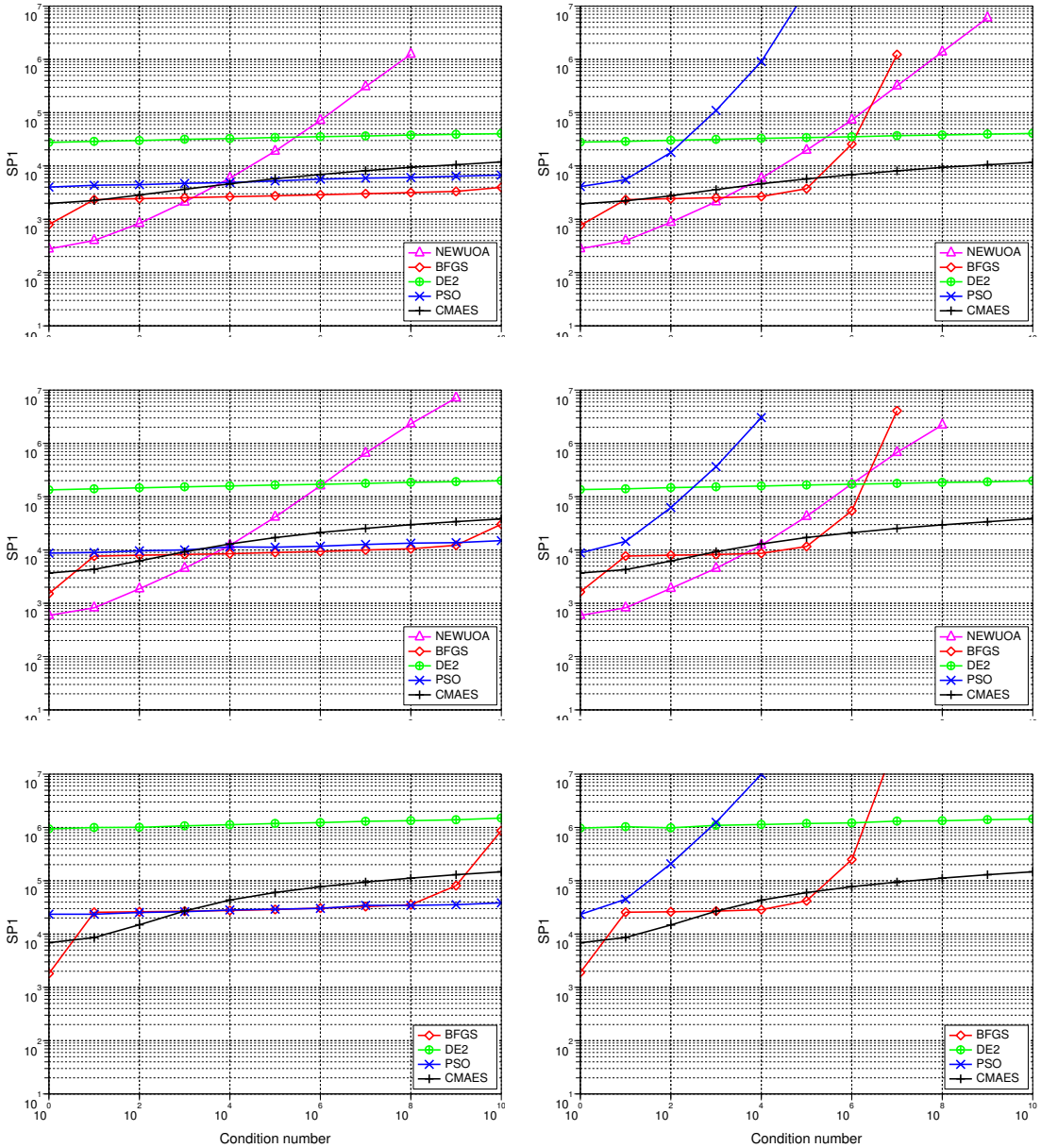

Figure 4: Ellipsoid$^{1/4}$ function. Shown is $SP1$ (the expected running time or number of function evaluations to reach the target function value) versus condition number.

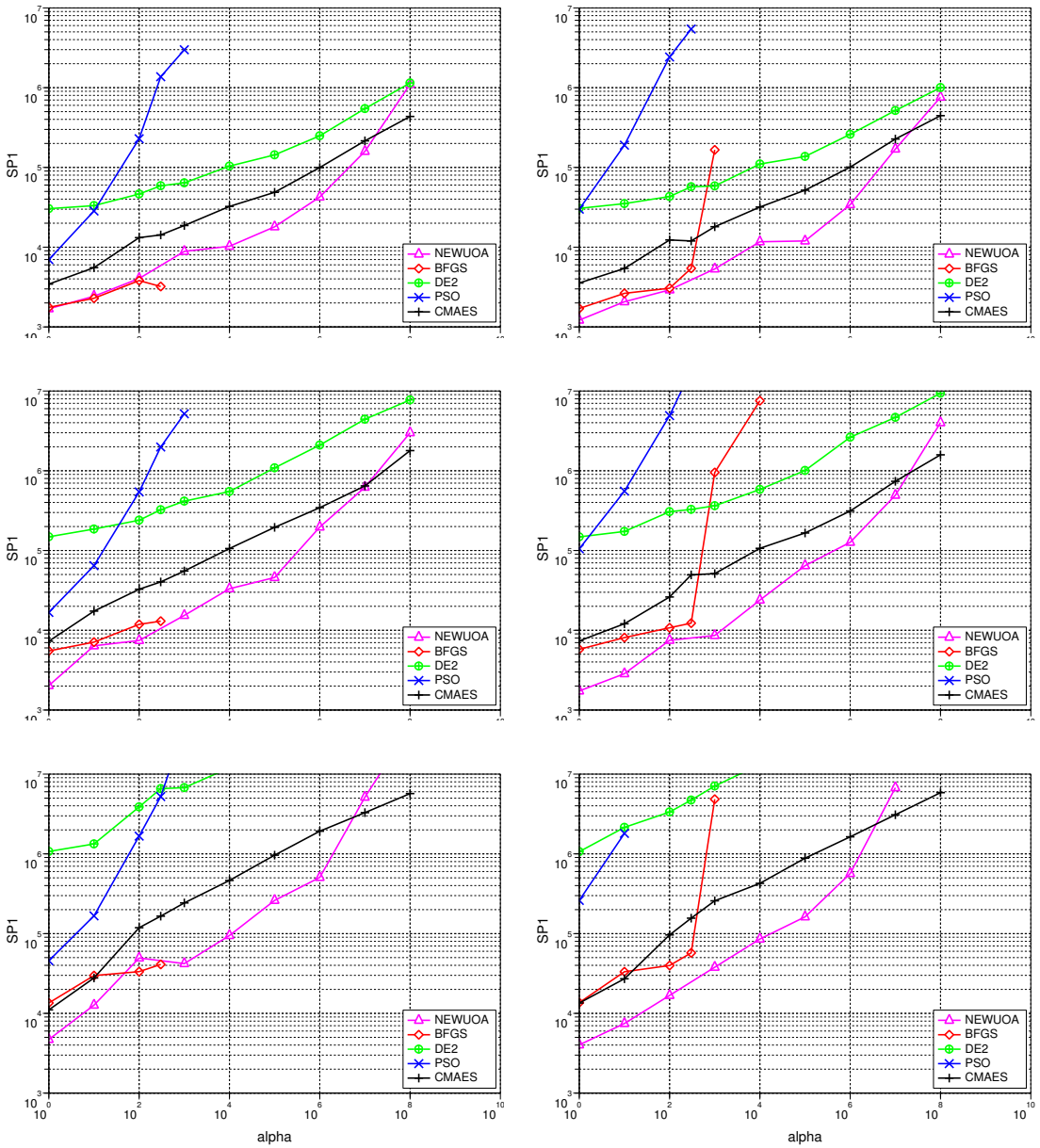

Figure 5: Rosenbrock function. Shown is $SP1$ (the expected running time or number of function evaluations to reach the target function value) versus conditioning parameter α.