A Simple Adaptive Algorithm for Numerical Optimization

Francisco Viveros-Jiménez[1], Jose A. León-Borges[2], and Nareli Cruz-Cortés[1]

[1] Centro de Investigación en Computación, Instituto Politécnico Nacional México D.F.,
[2] Universidad Politécnica de Quintana Roo, México
pacovj@hotmail.com, jleon@upqroo.edu.mx, nareli@cic.ipn.mx

Abstract. This paper describes a novel algorithm for numerical optimization, which we call Simple Adaptive Climbing (**SAC**). SAC is a simple efficient single-point approach that does not require a careful fine-tunning of its two parameters. Our algorithm has a close resemblance to local optimization heuristics such as random walk, gradient descent and, hill-climbing. However, SAC algorithm is capable of performing global optimization efficiently in any kind of space. Tested on 15 well-known unconstrained optimization problems, it confirmed that SAC is competitive against representative state-of-the-art approaches.

1 Introduction

Global optimization is the task of finding the point x^* with the smallest (minimization case) or biggest (maximization case) function value $f(x^*)$. In general, global optimization is a complex task that has remained unsolved until now. Optimizers are efficient tools used for a wide range of tasks such as: file compression [14], scheduling of an aircraft's engine maintenance [9], optimization of financial portfolios [20], evolution of neural networks for walking robots [16], among many others. Most of the recent efficient optimizers for solving unconstrained nonlinear optimization, such as restart CMA-ES [8], can be considered complex approaches because they use the Hessian and covariance matrix, which, in most cases, are very expensive to obtain. Those methods are very effective and greatly overcome simple heuristic approaches [11] [21].

However, we need to close the performance gap between simple heuristics and more complex approaches. This paper, presents a novel approach based on the idea of simplicity: Simple Adaptive Climbing (**SAC**). SAC is a single-point approach similar to hill-climbing or random walk algorithms.

The contents of this paper are organized as follows: First, we give a brief introduction to hill-climbing algorithm in Section 2. Section 3 describes the SAC-algorithm. Section 4 discusses the experimental design. Section 5 presents a comparison of SAC against four state-of-the-art approaches: Elitist Evolution, micro-Particle Swarm Optimization, Simple Adaptive Differential Evolution and Restart CMA-ES. Finally, Section 6 draws some conclusions.

2 Brief Review of the Hill Climbing-Like Algorithms

A hill-climbing like algorithm performs its search by moving an explorer (single point) through a mountain having a really thick fog (the target function). The explorer performs jumps for landing to a new point. If the new point is better, then the explorer stays in the new position. If the new point is not better, the explorer returns to its previous position and tries to perform a smaller jump. The jump range is known as step size and represents the current algorithm search ratio. This way, the explorer continues jumping until the jump range diminishes.

Hill-climbing like techniques are known for being fast on optimizing local optimum values, i.e. they are very good on descending/ascending a single hill function (unimodal functions). However, this strength is also its greater flaw: they could fail solving functions with multiple hills (multimodal funcions). Three main reasons have been detected behind this behavior [2]:

1. The foothill problem: the algorithm gets stuck in a local optimum, i.e. an optimum value that is not the global one.
2. The plateau problem: the algorithm gets stuck in mostly flat surfaces with few sharp peaks.
3. The ridge problem: the algorithm gets stuck because the ascent direction is not within the possible search directions.

On the other hand, population-based algorithms such as Differential Evolution and Genetic Algorithms have the capability of easily finding optimal value regions. However, they have a relatively slow local optimization speed. For this reason, hill-climbing algorithms are commonly used in combination with population algorithms to create efficient memetic algorithms [3,7].

3 Simple Adaptive Climbing

SAC is a single-point optimizer designed to be simple and competitive. The main idea is to modify the step size according to the current state of the search space. Further, a reinitialization process is added. Particularly, SAC general ideas are: (1) to increase search ratio while improving the solution, (2) to decrease search ratio when no improvements are made, and, (3) to restart the process when no improvements are made for a predetermined number of trials. The algorithm 1 shows the SAC technique for minimization. SAC requires the configuration of two parameters:

1. Base step sizes ($\mathbf{B} \in [\mathbf{0.0}, \mathbf{0.5}]$): B is the search space percentage to be explored in both directions of each dimension. For example, if B = 0.5 means that SAC will search half of the search space to the right and half to the left, meaning the whole search space to be explored. Figure 1 shows an example of the B search ratio in a 2D function.
2. Maximum consecutive errors ($\mathbf{R} > \mathbf{0}$): indicates the maximum consecutive errors necessary to restart the search process. Figure 3 shows an example of a restart in a 2D function.

SAC search by performing alterations in a random dimension subset as shown in Figure 2. It uses adaptive step sizes (as shown in lines 4-12 of Algorithm 1 where $b_j, j = 1, \ldots, D$, and D is the dimensionality of the problem) that is the key of exploration process. SAC adjusts its size accordingly to the current success/failure of the search:

- When a best point is found, b values became greater to encourage exploration of new search areas (see line 15 on Algorithm 1 and Figure 5 for an example).
- When no best point is found, b values became equal to the change between current and former points, encouraging exploration of nearby areas (see line 22 on Algorithm 1 and Figure 6 for an example).

In SAC, the search space is considered as a cincunference (Figure 4). So, if SAC tries to explore outside a variable bound, let us say the the upper limit, then it will explore a valid region near the variable lower limit. To avoid premature convergence on some functions, SAC keeps track of the consecutive unsuccessful explorations (*restart* variable on Algorithm 1). When *restart* reaches the user-defined limit R, the step-sizes and the current position are restarted as seen in line 24 on Algorithm 1.

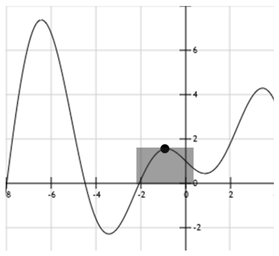

Fig. 1. B represents the initial and maximum search space. A $B = 0.1$ is depicted by the gray rectangle in this picture.

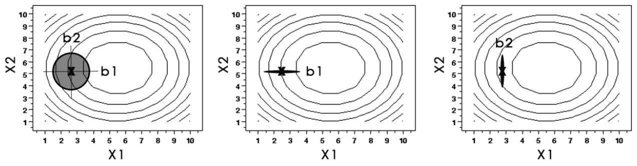

Fig. 2. SAC could search in any dimension subset of this 3D function with a maximum ratio of b_j

```
**Data**: $B \in [0.0, 0.5]$ (initial step size), $R > 0$ (maximum number of consecutive
    errors) and $MaxFEs$ (maximum number of function evaluations )
**Result**: $Xbest$ (best solution found)
Set $X$ as a random initial point and $Xbest = X$ as the best known solution;
Set $b_j = B, j = 1, \ldots, D$ as the initial step sizes and $restart = 0$;
**for** *g=1 To MaxFEs* **do**
    Set $P_a = \frac{rnd(1,D)}{D}$;
    **if** $flip_j(P_a), j = 1, \ldots, D$ **then**
        Set $O_j = X_j + rndreal(-b_j, b_j) \times (up_j - low_j)$;
        **if** $O_j > up_j$ **then**
            Set $O_j = low_j + (O_j - up_j)$;
        **if** $O_j < low_j$ **then**
            Set $O_j = up_j - (low_j - O_j)$;
    **else**
        $O_j = X_j$;
    **if** $f(O) < f(X)$ **then**
        Set $restart = 0$;
        Set $b_j = rndreal(b_j, B)$ for all $j$ dimensions where $O_j \neq X_j$;
        Set $X = O$;
        **if** $f(O) < f(Xbest)$ **then**
            Set $Xbest = O$;
    **else**
        Set $restart = restart + 1$;
        **if** $restart < R$ **then**
            Set $b_j = \left|\frac{X_j - O_j}{up_j - low_j}\right|$ for all $j$ dimensions where $O_j \neq X_j$;
        **else**
            Set $restart = 0$, $X = O$ and $b_j = B, j = 1, \ldots, D$;
```

Algorithm 1. Algorithm for SAC (minimization case)

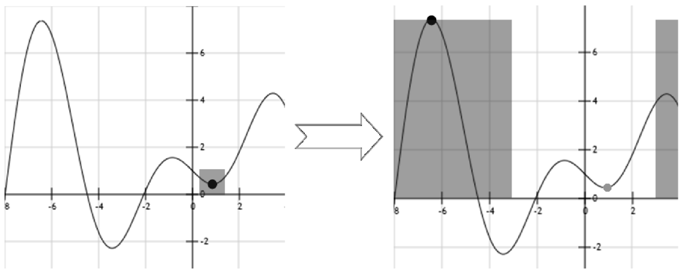

Fig. 3. Position and step sizes are restarted when SAC reaches R iterations without improving the solution. SAC keeps memory of the best solution found (represented as a gray dot in the second picture).

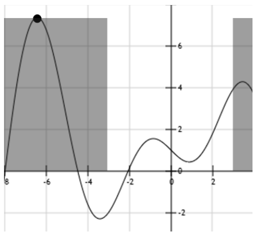

Fig. 4. Function are over a circunference in SAC, i.e., if SAC tries to explore outside the lower limit it will explore a valid region near the upper limit

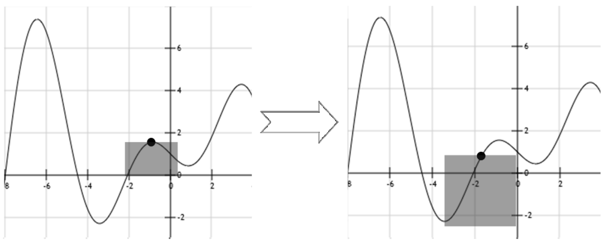

Fig. 5. When improving SAC increases the step sizes

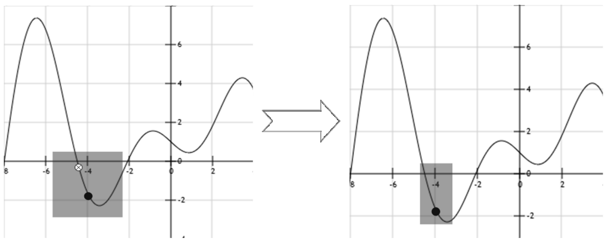

Fig. 6. When failing (white dot) SAC decreases the step sizes

Table 1. Test functions [12] [15]

Unimodal functions	**Multimodal functions**
Separable	
f_{sph} Sphere model	f_{sch} Generalized Schwefel's problem 2.26
$f_{2.22}$ Schwefel's problem 2.22	f_{ras} Generalized Rastrigin's function
$f_{2.21}$ Schwefel's problem 2.21	
f_{stp} Step function	
f_{qtc} Quartic function	
Non-separable	
$f_{1.2}$ Schwefel's problem 1.2	f_{ros} Generalized Rosenbrock's function
	f_{ack} Ackley's function
	f_{grw} Generalized Griewank's function
	f_{sal} Salomon's function
	f_{whi} Whitley's function
	$f_{pen1,2}$ Generalized penalized functions

4 Experimental Setup

Tests aim to confirm that the proposed algorithm is competitive in solving unconstrained functions. A comparison against some state-of-the-art was performed. The Error and Evaluation values for each trial was measured in a similar way to the one proposed in the test suite for CEC 2005 special session on real-parameter optimization [10]:

– $Error = F(x^o) - F(x^*)$, where x^o is the best reported solution for the corresponding algorithm and x^* is the global optimum value.
– *Evaluation* is the number of function evaluations (**FE**s) required to reach an error value of 10^{-8}.

The benchmark functions are specified in table 1. Thirty trials per test function were conducted and the number of successful trials that reached the target accuracy value were measured. The stop condition criterion for all approaches was $MaxFEs = 3E + 5$ function evaluations (**FE**s). The results were compared against two micro-EAs and two state-of the-art-approaches. Micro-population algorithms were selected because they are considered in between hill-climbers algorithms and other population algorithms. Also, they are competitive against their standard counterparts [1] [6] and are used to create memetic algorithms also [5] [17]. The selected approaches are:

– Elitist Evolution (**EEv**) [19]: The best micro-population algorithm as far as the authors knowledge.
– Micro Particle Swarm Optimization (μ-**PSO**) [13]: The micro population counterpart of PSO, that maintains the original algorithm performance.
– Simple Adaptive Differential Evolution (**SaDE**) [18] selected because it is a competitive Differential Evolution [4] variant representative of the state-of-the-art techniques.

– Restart CMA-ES [8] selected for measuring the gap against a technique that uses Hessian and covariance matrices. This was also the best technique on CEC 2005 special session on real-parameter optimization.

All the experiments were performed using a Pentium 4 PC with 512 MB of RAM, in C language over a Linux environment. The parameter sets for the techniques were:

1. **SAC**: $R = 150, B = 0.5$.
2. **μ-PSO**: $P = 6, C_1 = C_2 = 1.8, Neighborhoods = 2$, Replacement generation = 100, Replacement particles= 2, Mutation % = 10, based on [13].
3. **EEv**: $P = 5, B = 0.5$
4. **SADE**: set as in [18].
5. **Restart CMA-ES**: set as in [8].

5 Test Results and Analysis

5.1 Performance Evaluation on Functions with 30 Dimensions

A comparison of SAC against EEv, was performed μ-PSO, SADE and Restart CMA-ES. Tables 3 and Figure 7 show more detailed test results. Table 2 gives statistical significance to test results. Tests allow us to confirm the following:

1. SAC has a competitive performance in global optimization problems with a high dimensionality without requiring the fine-tuning of its two parameters.
2. SAC outperforms EEv and μ-PSO on most functions. It have more success rate and overall speed.
3. The difference between SAC's performance and EEv and μ-PSO' performances is statistical significant on most functions.
4. SAC is competitive against a state-of-the-art approach like SADE.
5. SAC's success rates competitively against CMA-ES'. However, CMA-ES is faster than SAC.

Table 2. Results of Mann-Whitney U paired test between SAC and other techniques. + means that the improvement is statistically significant. − means that the other technique outperforms SAC. = means that the difference is not statistically significant.

	μ-PSO	EEv	CMA-ES	SADE		μ-PSO	EEv	CMA-ES	SADE
f_{sph}	+	+	−	=	f_{ros}	=	=	−	−
$f_{2.22}$	+	+	−	−	f_{ack}	+	+	−	=
$f_{2.21}$	+	+	−	+	f_{grw}	=	=	−	−
f_{stp}	+	=	−	−	f_{pen1}	+	+	−	+
f_{qtc}	+	+	−	+	f_{pen2}	+	+	+	=
f_{sch}	+	+	+	+	f_{sal}	=	=	=	=
f_{ras}	+	+	+	+	f_{whit}	+	+	+	+
$f_{1.2}$	=	=	−	−					

Table 3. Mean Error values obtained on functions with $D = 30$. A $\star$ value means that 10^{-8} was reached in all runs (100% success rate). On values like X.XE+X(Y) Y represents the success rate (only when $Y \in [1\%, 99\%]$).

$D = 30$	SAC	μ-PSO	EEv	CMA-ES	SaDE
f_{sph}	$\star$	$\star$	$\star$	$\star$	$\star$
$f_{2.22}$	$\star$	$\star$	$\star$	$\star$	$\star$
$f_{2.21}$	$\star$	1.3E-2	9.1E-3	$\star$	4.5E+0
f_{stp}	$\star$	$\star$	$\star$	$\star$	$\star$
f_{qtc}	$\star$	$\star$	$\star$	$\star$	$\star$
$f_{1.2}$	3.2E-4	1.9E-2	6.1E-3	$\star$	$\star$
f_{sch}	4.8E-7(96%)	1.3E+3	1.5E+3	1.2E+4	3.9E+0(96%)
f_{ras}	$\star$	8.1E+0	$\star$	3.3E+0(10%)	7.9E-1(63%)
f_{ros}	1.3E+1	1.6E+1	4.1E+1	$\star$	3.9E-1(63%)
f_{ack}	$\star$	$\star$	$\star$	$\star$	3.1E-2(96%)
f_{grw}	2.3E-2(33%)	2.3E-2(30%)	2.6E-2(23%)	$\star$	2.7E-3(83%)
f_{pen1}	$\star$	$\star$	$\star$	$\star$	6.9E-3(93%)
f_{pen2}	$\star$	$\star$	$\star$	1.4E-3(86%)	$\star$
f_{sal}	6.6E-1	4.5E-1	6.3E-1	2.1E-1	2.0E-1
f_{whit}	6.9E+0(43%)	1.0E+2	1.0+1(40%)	4.8E+2	5.9E+1(20%)

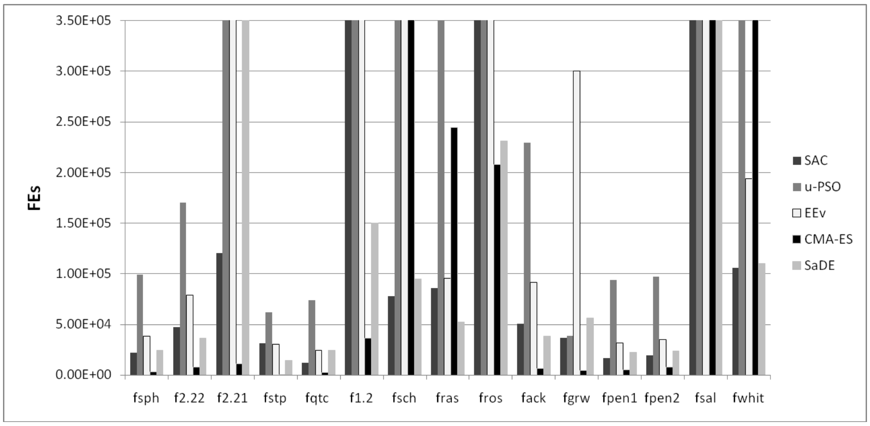

Fig. 7. Speed comparison (Evaluation values) on 30D functions

5.2 Sensibility to Parameter Adjustments

Tests with different B and R values were made to observe the effects of parameter configuration on SAC's performance. Results are shown in Figures 8 and 8, and, Tables 4 and 5. Tests allow observing that:

- SAC does not need complex configuration of its parameters.
- B is the reference search ratio of SAC. Greater B values encourage global exploration and smaller values encourage local exploitation.

Table 4. Changes in the success rates when using different B values. Only the functions with different success rates are shown. Best results are obtained with a $B = 0.5$.

$B =$ 0.1		0.2	0.3	0.4	0.5
f_{sch}	–	1%	76	86	96
f_{ras}	–	63	90	93	100
f_{grw}	20	16	13	36	10
f_{whit}	10	23	43	26	43

Table 5. Changes in the success rates when using different R values. Only the functions with different success rates are shown.

$R =$ 15		50	150	300	500	$R =$ 15		50	150	300	500
f_{sph}	—	100%	100	100	100	f_{ros}	—	3	—	3	3
$f_{2.22}$	—	—	100	100	100	f_{ack}	—	—	100	100	100
$f_{2.21}$	—	—	100	100	100	f_{grw}	—	30	10	30	30
f_{stp}	—	100	100	100	100	f_{pen1}	—	100	100	100	100
f_{qtc}	—	100	100	100	100	f_{pen2}	—	100	100	100	100
f_{sch}	—	60	96	96	90	f_{whit}	—	46	43	46	100
f_{ras}	—	100	100	96	93						

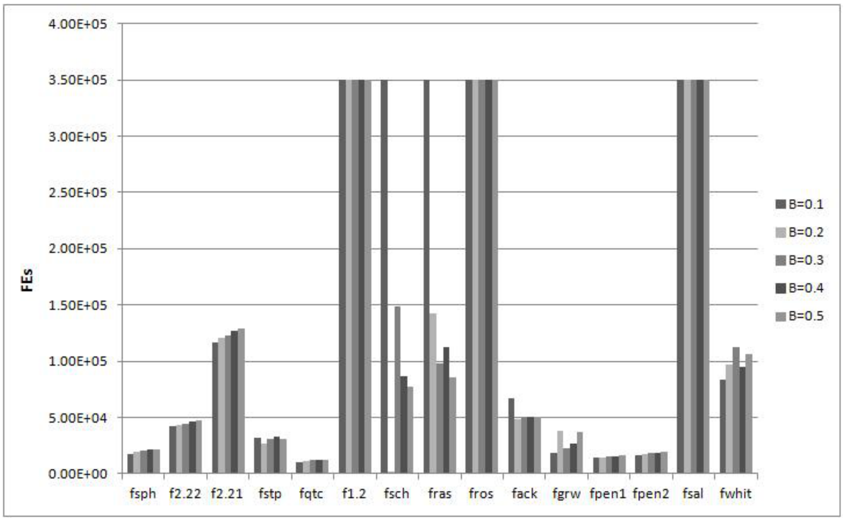

Fig. 8. Evaluation values registered with different B values on 30D functions

- R controls SAC's searching time over the current location. Smaller values allow SAC to move often, while bigger values provoke best exploitation of the current area.
- Smaller B values increase SAC performance on unimodal problems and decrease performance on the multimodal ones (SAC behaves more like a hill-climbing algorithm).

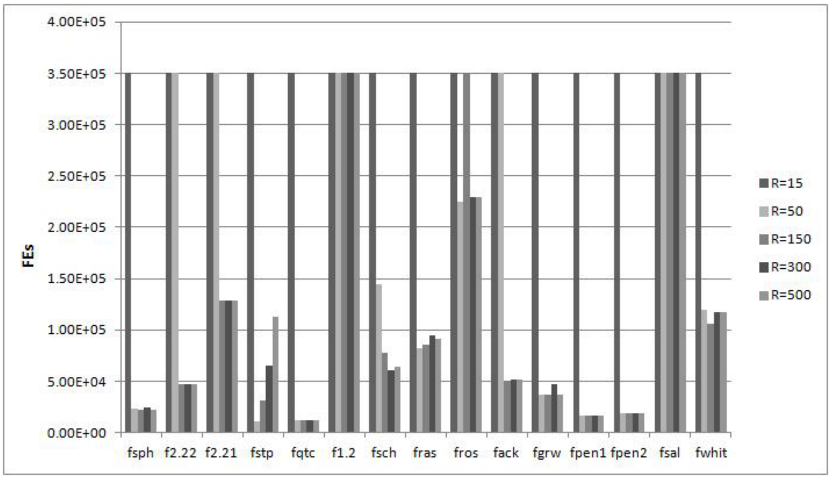

Fig. 9. Evaluation values registered with different R values on 30D functions

- Bigger B values have the best overall performance.
- Smaller R values have poor performance.
- Bigger R values have best performance on multimodal non-separable problems.

6 Conclusions and Future Work

This paper presented a novel optimizer called simple adaptive climbing ((SAC)) and tested it on 15 benchmark functions. SAC is a simple technique that uses a single exploration point and adaptive step sizes. Its main features are: (1) easy implementation, (2) state-of-the-art performance competitive against techniques such as: μ-PSO and SADE, (3) easy parameter configuration, (4) fast solution speed, and, (5) high success rate.

SAC uses 2 parameters: B and R. B is reference search ratio of SAC. Big B values encourage global exploration and small B values encourage local exploitation. R controls SAC's searching time over the current location. Small R values allow SAC to move often, while bigger values necessitate a more extensive exploration over the current search area.

More comparative studies and further analysis should be carried out to provide a more detailed understanding of SAC. It is planned to test SAC in constrained and in multi-objective optimization problems.

Acknowledgments. The first author acknowledges support from CONACyT through a scholarship. The authors acknowledge support from CONACyT through the project 132073 and SIP-IPN 20120002.

References

1. Krishnakumar, K.: Micro-genetic algorithms for stationary and non-stationary function optimization. In: SPIE: Intelligent Control and Adaptive Systems, vol. 1196, pp. 289–296 (1989)
2. Winston, P.H.: Artificial Intelligence, 3rd edn. Addison-Wesley Publishing Company, Reading (1992)
3. Renders, J.M., Bersini, H.: Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In: Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 312–317 (1994)
4. Storn, R., Price, K.: Differential Evolution - a simple and efficient heuristic for global optimization. Journal of Global Optimization 11(4), 341–359 (1997)
5. Kazarlis, S.E., Papadakis, S.E., Theocharis, J.B., Petridis, V.: Microgenetic Algorithms as Generalized Hill-Climbing Operators for GA Optimization. Evol. Comput. 5(3), 204–217 (2001)
6. Toscano-Pulido, G., Coello-Coello, C.A.: Multiobjective Optimization using a Micro-Genetic Algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 126–140. Springer, Heidelberg (2001)
7. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-Coded Memetic Algorithms with Crossover Hill-Climbing. Evolutionary Computation 12(3), 273–302 (2004)
8. Auger, A., Kern, S., Hansen, N.: A Restart CMA Evolution Strategy with Increasing Population Size. In: CEC 2005 Special Session on Real-Parameter Obtimization, Nanyang Technol. Univ., Singaporem IIT Kanpur (2005)
9. Kleeman, M.P., Lamont, G.B.: Solving the Aircraft Engine Maintenance Scheduling Problem Using a Multi-objective Evolutionary Algorithm. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 782–796. Springer, Heidelberg (2005)
10. Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.-P., Auger, A., Tiwari, S.: Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Obtimization. Nanyang Technol. Univ, Singaporem IIT Kanpur, 2005005 (2005)
11. Hansen, N.: Compilation of Results on the 2005 CEC Benchmark Function Set. Technical Report, CoLAB Institute of Computational Science, ETH, Zurich (2006)
12. Mezura-Montes, E., Coello Coello, C.A., Velazquez, R.J.: A comparative study of differential evolution variants for global optimization. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 485–492 (2006)
13. Fuentes Cabrera, J.C., Coello Coello, C.A.: Handling Constraints in Particle Swarm Optimization Using a Small Population Size. In: Gelbukh, A., Kuri Morales, Á.F. (eds.) MICAI 2007. LNCS (LNAI), vol. 4827, pp. 41–51. Springer, Heidelberg (2007)
14. Kattan, A., Poli, R.: Evolutionary lossless compression with GP-ZIP*. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO 2008, pp. 1211–1218. ACM (2008)
15. Noman, N., Iba, H.: Accelerating Differential Evolution Using an Adaptive Local Search. IEEE Transactions on Evol. Comput. 12(1), 107–125 (2008)
16. Valsalam, V.K., Miikkulainen, R.: Modular neuroevolution for multilegged locomotion. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO 2008, pp. 265–272. ACM (2008)

17. Parsopoulos, K.E.: Cooperative Micro-Particle Swarm Optimization. In: ACM 2009 World Summit on Genetic and Evolutionary Computation (2009 GEC Summit), pp. 467–474. ACM, Shanghai (2009)
18. Qin, K.A., Huang, V.L., Suganthan, P.N.: Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. IEEE Transactions on Evol. Comput. 13(2), 398–417 (2009)
19. Viveros Jiménez, F., Mezura-Montes, E., Gelbukh, A.: Elitistic Evolution: An Efficient Heuristic for Global Optimization. In: Kolehmainen, M., Toivanen, P., Beliczynski, B. (eds.) ICANNGA 2009. LNCS, vol. 5495, pp. 171–182. Springer, Heidelberg (2009)
20. Yan, W., Sewell, M.V., Clack, C.D.: Learning to optimize profits beats predicting returns -: comparing techniques for financial portfolio optimisation. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO 2008, pp. 1681–1688. ACM (2009)
21. Hansen, N., Auger, A., Finck, S., Ros, R.: Comparison tables: BBOB 2009 function testbed in 20-D. Technical Report, INRIA (2010)