2009 Ninth International Conference on Intelligent Systems Design and Applications

Unidimensional Search for solving continuous high-dimensional optimization problems

Vincent Gardeux*, Rachid Chelouah†, Patrick Siarry‡ and Fred Glover§

*Ecole Internationale des Sciences du Traitement de l'Information, Cergy-Pontoise, France
Email: vincent.gardeux@eisti.fr
†Ecole Internationale des Sciences du Traitement de l'Information, Cergy-Pontoise, France
Email: rachid.chelouah@eisti.fr
‡Université de Paris 12, LiSSi, Créteil, France
Email: siarry@univ-paris12.fr
§Leeds School of Business, University of Colorado, Boulder, CO 80309, USA
Email: fred.glover@colorado.edu

***Abstract*—This paper presents a performance study of two versions of a unidimensional search algorithm aimed at solving high-dimensional optimization problems. The algorithms were tested on 11 scalable benchmark problems. The aim is to observe how metaheuristics for continuous optimization problems respond with increasing dimension. To this end, we report the algorithms' performance on the 50, 100, 200 and 500-dimension versions of each function. Computational results are given along with convergence graphs to provide comparisons with other algorithms during the conference and afterwards.**

***Keywords*-metaheuristic; unidimensional; optimization**

I. Introduction

Many metaheuristics and especially evolutionary algorithms are designed to solve continuous optimization problems. A major challenge is to solve high-dimensional problems of this kind, which arise in critical applications such as data or web mining. However, many current algorithms are not designed to be dimensionally robust, and their performance deteriorates quickly as the dimensionality of the search space increases.

In order to compare these algorithms, performance is measured as the difference between the best solution found by each algorithm and the test function's global optimum. Speed of convergence is also an important criterion because it determines how fast an algorithm can find an acceptable solution, even if it's not the best one. Another relevant criterion, not so often highlighted, is the number of parameters needed to tune the algorithm for a particular problem. Such parameters require substantial computer and human time to be properly set.

The proposed benchmark is composed of 11 scalable functions which involve unimodalities or multimodalities, separable or non separable functions, etc. Our goal in this paper is to provide a new algorithm from the class known as greedy algorithms, whose purpose is to solve unimodal problems efficiently while maintaining the underlying algorithmic structure as simple as possible. We focus on unimodal problems because even complex multimodal problems require an ability to handle unimodal problems effectively, and multimodal algorithms typically can take advantage of an improved unimodal solution capability to enhance their performance. Gradient Descent [1] is a very simple instance of a greedy algorithm. To find a local optimum, it takes steps proportional to the negative (positive) gradient of the function at the current solution, for a minimization (maximization) objective. The search starts from an initial solution x_0, and computes the direction $d_0 = -\nabla f(x_0)$. In the simplest case, the method takes steps of unit size in the direction d_0 until the objective function can not be further improved. When no improving step exists the method stops. Otherwise, the procedure is restarted from the new solution. This method can only be performed if the objective function is known and differentiable.

In this paper, we present two variants of the gradient descent algorithm using a rudimentary search procedure that can be applied to any type of function. In our algorithms, the search direction is restricted to the set of those produced by varying a single coordinate at a time, but all such coordinate directions are examined.

In order to escape local optima, we augment this simple search with a guided restart procedure inspired from the diversification generation method used in Scatter Search [2]. The aim of this method is to generate a collection of diverse trial solutions and to select the one that lies farthest from a reference set of previously visited solutions. The presented algorithms are not only easy to implement but also parameter-free.

Section 2 describes the first version of this algorithm called Classical Unidimensional Search (CUS) and Section 3 gives an extended version called Enhanced Unidimensional Search (EUS). In Section 4, we present our simulation results with

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.191

tables and convergence graphs, and discuss the performance of our algorithms vs the different benchmark functions. A concluding summary is provided in Section 5.

II. CUS ALGORITHM

The CUS algorithm is designed to be easy to implement and robust in handling high-dimensional problems. The algorithm starts from a randomly generated initial solution of dimension D in the search space. A vector h is created and initialized as follows : each component h_i is set equal to the difference between the maximum and minimum search space bounds on the ith component of the solution vector. Then the algorithm iterates until a stopping criterion is reached.
At each iteration, the algorithm focuses on optimizing the $current$ solution on only one dimension i. (For simplicity, we represent the current solution as a vector which is given the name $current$). Thus, the method searches for the integer value v such that solution x, described as follows, has the best fitness value (i.e. objective function value):

$$x = [current_1, ..., current_i + v * h_i, ..., current_D] \quad (1)$$

The search starts by comparing $current$ (i.e. the solution for $v = 0$) with its 2 neighbor solutions (i.e. for $v = 1$ and -1). The best direction (positive or negative) from $current$ along the ith dimension is identified by the value of v that yields the best neighbor solution. After that, the iteration goes in only one direction : if $v = 1$ yields the best solution, the search continues in the positive direction (i.e. {2,3,...}), else it goes in the negative direction (i.e. {-2,-3,...}) until no better solution can be found. If this computed solution is out of bounds, it is corrected and set on the nearest bound. No additional values of v are examined beyond the first one found that fails to improve the objective function. Then, the $current$ solution is updated to be the best one obtained along the ith dimension and successive dimensions i are treated in the same manner. (In the case where no improving moves exist for a given i, the updating operation leaves the solution unchanged.).
Therefore, after one iteration, which consists of thus examining every dimension i, the algorithm finds a restricted local optimum relative to the precision given by the vector h. (We call this local optimum restricted, since if any change is produced during the iteration it is possible that a better solution could be produced by a new pass of the dimension i.) Often this restricted local minimum is a true local minimum relative to the current h employed, and so, after each iteration, h is decreased by a $ratio$ value fixed to 0.5 until h becomes smaller than a pre-selected vector of minimum h_i values denoted by $h(min)$, and then h is not decreased further. $h(min)$ values are all fixed to $1e-20$ in order to obtain a suitable precision. The parameter $ratio$ can be tuned but experiments show that the given fixed value is suitable.
The pseudo-code in Figure 1 details an iteration of the algorithm, where $next$, $previous$ and x are temporary solutions. The algorithm is not a population-based algorithm,

Procedure CUS
begin
  for $i = 1$ **to** D
    $next, previous = current$
    $next_i = current_i + h_i$
    $previous_i = current_i - h_i$
    evaluate $next$ and $previous$ fitness
    $x =$ best of 3 solutions
    while better solution found
      $x_i = x_i \pm h_i$ in best direction
    end
    $current = x$
  end
  $h = h * 0.5$ until $h < h(min)$
end

Figure 1. An iteration of CUS algorithm

so to avoid becoming trapped in a local minimum, we use a restart procedure that keeps the best solution found so far and re-initializes h to a new starting value after reaching the termination point given by $h < h(min)$. Therefore, increasing the vector $h(min)$ may increase the potential number of restarts of the algorithm, but tends to decrease the error accuracy. In order to better explore the search space, when the restart procedure is activated, a new solution is generated, that lies far from the reference set. This technique uses the diversification generation method from Scatter Search algorithm. It generates a collection of diverse trial solutions and selects the one farthest from a reference set of previously visited restricted local optima.

III. EUS ALGORITHM

This algorithm amends the CUS algorithm as follows. The variables D, h and its components h_i are initialized in the same manner as in the CUS algorithm. But at each iteration, when a good direction is found, there is no intensification. So, after one iteration, each dimension i of the $current$ solution can have 3 possible values, depending on which one produces the best $current$ fitness:

$$current_i = \begin{cases} current_i \\ current_i + h_i \\ current_i - h_i \end{cases} \quad (2)$$

with $i \in \{1, ..., D\}$.
This version is designed to solve more complex functions by taking smaller steps in a direction at a time. Indeed, changing the direction at each step reduces the probability of being trapped in a local minimum and permits to explore a larger part of the solution space. The pseudo-code of an iteration of this algorithm is given in Figure 2.

Procedure EUS
begin
  for $i = 1$ **to** D
    $next, previous = current$
    $next_i = current_i + h_i$
    $previous_i = current_i - h_i$
    evaluate $next$ and $previous$ fitness
    $current =$ best of 3 solutions
  end
  if no improvement has been found
    $h = h * 0.5$ until $h < h(min)$
  end
end

Figure 2. An iteration of EUS algorithm

IV. Experimental results

In order to compare our results with those of competing algorithms, our two algorithms have been tested on a fixed benchmark of 11 scalable functions. This process is designed to identify key mechanisms that make metaheuristics to be scalable on high-dimension problems.

A. Experimental setup

The proposed set of test problems includes 11 scalable functions with different characteristics (Table I). The first six are taken from the CEC'2008 test suite described in [3]. Other function definitions can be found in the workshop report [4]. A more detailed discussion of the Extended F_{10} function can be found in [5].

#	Function	Unimodal	Separable
1	Shifted Sphere	X	X
2	Shifted Schwefel Problem 2.21	X	-
3	Shifted Rosenbrock	-	-
4	Shifted Rastrigin	-	X
5	Shifted Griewank	-	-
6	Shifted Ackley	-	X
7	Schwefel Problem 2.22	X	-
8	Schwefel Problem 1.2	X	-
9	Extended F_{10}	-	-
10	Bohachevsky #1	-	X
11	Schaffer #2	-	-

Table I
Benchmark functions

For each function f, experiments are conducted in 50, 100, 200 and 500 dimensions. Each algorithm is run 25 times and the average of error e of the best solution x is computed:

$$e = f(x) - f(o) \text{ where } o = \text{optimum of the function} \quad (3)$$

The maximum number of fitness evaluations is $5000 * D$ and determines the stopping criterion for a run. The errors obtained by the EUS and CUS algorithms for each function are listed in Tables II-V. Error Value (Err. Val.) and Standard Deviation (Std. Dev.) are computed for 25 runs and are reported for both algorithms on each of the 11 functions. For function $\#4$, two different minima are found (local and global). It wouldn't be appropriate to compute average error for all 25 runs, therefore we calculated separately the average error on each minimum. An $error$**(n)** value indicates the average $error$ for n runs out of the 25 total.

# fn	CUS		EUS	
	Err. Val.	Std. Dev.	Err. Val.	Std. Dev.
1	4.18E-13	5.09E-14	4.14E-13	5.46E-14
2	1.24E2	7.04E0	1.90E-11	1.21E-11
3	3.01E2	2.35E3	1.07E1	2.74E2
4	5.43E-13**(12)** 1.49E0**(13)**	1.28E-13 5.01E-1	3.69E-13	3.64E-14
5	2.11E-13	2.80E-14	2.10E-13	4.25E-14
6	1.97E1	1.37E-1	1.97E1	1.83E-2
7	0.00E0	0.00E0	0.00E0	0.00E0
8	4.23E4	2.11E4	7.20E-10	2.39E-9
9	3.65E2	3.31E1	3.55E2	3.28E1
10	0.00E0	0.00E0	1.54E1	1.70E0
11	0.00E0	0.00E0	0.00E0	0.00E0

Table II
Error values achieved for D = 50

# fn	CUS		EUS	
	Err. Val.	Std. Dev.	Err. Val.	Std. Dev.
1	9.40E-13	7.72E-14	9.45E-13	7.42E-14
2	1.40E2	5.02E0	9.82E-11	2.67E-10
3	3.65E2	2.84E3	9.70E1	4.15E3
4	4.68E0	5.19E0	7.72E-13	4.25E-14
5	4.73E-13	4.30E-14	4.85E-13	3.44E-14
6	1.97E1	7.84E-2	1.97E1	4.93E-2
7	0.00E0	0.00E0	0.00E0	0.00E0
8	2.66E5	1.05E5	3.11E-3	3.73E-3
9	7.32E2	4.56E1	7.10E2	5.09E1
10	0.00E0	0.00E0	3.46E1	2.71E0
11	0.00E0	0.00E0	0.00E0	0.00E0

Table III
Error values achieved for D = 100

# fn	CUS		EUS	
	Err. Val.	Std. Dev.	Err. Val.	Std. Dev.
1	2.01E-12	1.00E-13	2.07E-12	8.37E-14
2	1.56E2	4.16E0	8.15E-10	1.69E-9
3	1.49E3	4.83E3	4.47E2	2.63E3
4	2.02E1	1.28E1	1.71E-12	8.15E-14
5	1.02E-12	7.10E-14	1.39E-12	6.82E-14
6	1.98E1	2.60E-2	1.98E1	7.48E-2
7	0.00E0	0.00E0	0.00E0	0.00E0
8	1.02E6	3.72E5	4.02E0	3.39E0
9	1.44E3	4.96E1	1.42E3	6.22E1
10	0.00E0	0.00E0	7.53E1	5.42E0
11	0.00E0	0.00E0	0.00E0	0.00E0

Table IV
Error values achieved for D = 200

# fn	CUS		EUS	
	Err. Val.	Std. Dev.	Err. Val.	Std. Dev.
1	5.40E-12	2.25E-13	5.37E-12	1.41E-13
2	1.69E2	2.33E0	1.14E-3	3.26E-4
3	1.34E3	2.75E3	4.78E2	2.13E2
4	1.11E2	2.41E1	4.55E-12	1.16E-13
5	2.89E-12	1.12E-13	3.53E-12	9.02E-14
6	1.98E1	1.63E-2	1.98E1	1.83E-2
7	0.00E0	0.00E0	0.00E0	0.00E0
8	8.11E6	3.29E6	6.53E3	3.35E3
9	3.65E3	9.95E1	3.52E3	9.68E1
10	0.00E0	0.00E0	2.00E2	7.94E0
11	0.00E0	0.00E0	0.00E0	0.00E0

Table V
ERROR VALUES ACHIEVED FOR D = 500

The convergence graphs of CUS and EUS algorithms on functions 1-5 and functions 6-11 are plotted in Figures 3-4 and 7-8 for 50 dimension problems, 5-6 and 9-10 for 500 dimension problems, respectively.

B. Discussion

For both algorithms, we observe that the convergence is very fast (fewer than 2.0E4 function evaluations for 50 dimensions and 2.0E5 for 500). Moreover, the algorithms keep this fast convergence behavior even if the dimension increases. The Rosenbrock and Ackley functions resist solution by these algorithms. The global minimum of Rosenbrock's function is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial, but to converge to the global minimum is difficult. Similarly, the landscape of the Ackley's function is very flat and the optimum is on a very localized peak. Our algorithms are not based on maintaining a population of solutions, so they may be inefficient in exploring particular regions of the search space if they are not guided, in particular in the case of high dimensions.

We observe that the CUS and EUS algorithms yield similar results on functions 1,5,6,7,9 and 11, but EUS yields better results on all others, except for Bohachevksy's (#10). Rastrigin's function (#4) is solved partially by CUS and performance decreases while the dimension increases, but it's always solved with an E-13 accuracy by EUS. Schwefel's problems 2.21 and 1.2 (#2 and #8) resist solution by CUS while EUS manages to solve them with E-10 accuracy in 50 dimensions. Smaller steps of EUS algorithm permit more complex problems to be solved even if convergence time is a bit longer than for CUS.

Some further experiments show that including a random parameter with h in equation (1) allows the EUS algorithm to solve Bohachevsky's and even Ackley's function. However, this does not help in the case of the Rosenbrock and Extended F_{10} functions.

In order to have a preliminary idea of EUS and CUS performance, we can compare them with the other tested algorithms from the CEC'08 competition. Functions of this

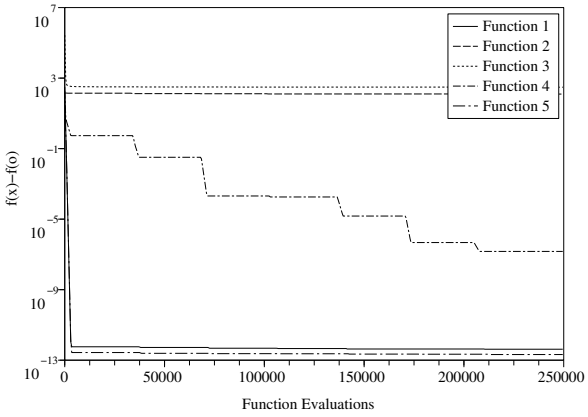

Figure 3. Convergence Graphs of CUS for D=50 (functions 1-5)

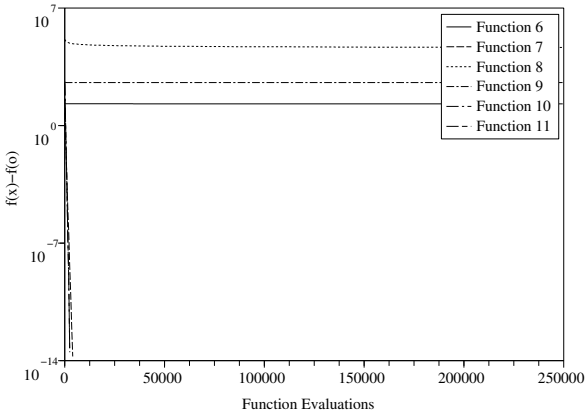

Figure 4. Convergence Graphs of CUS for D=50 (functions 6-11)

benchmark are the 6 first described in Figure 4. Results of this competition can be consulted in [6]. The algorithms are ranked according to their average performance on all functions. The 3 first ranked algorithms are MTS [7], LSEDA-gl [8] and jDEdynNP-F [9], respectively. The Table VI merges these 3 algorithms results with those of CUS and EUS for 1000 dimension problems. The experimental setup is the same as the one described in section 4.1.

#	CUS	EUS	MTS	LSEDA-gl	jDEdynNP-F
1	9.67E-12	9.66E-12	0.00E0	3.23E-13	1.14E-13
2	1.81E2	8.01E-1	4.72E-2	1.04E-5	1.95E1
3	2.52E3	8.24E2	3.41E-4	1.73E3	1.31E3
4	3.41E2	9.62E-12	0.00E0	5.45E2	2.17E-4
5	5.99E-12	6.39E-12	0.00E0	1.71E-13	3.98E-14
6	1.98E1	1.98E1	1.24E-11	4.26E-13	1.47E-11

Table VI
ERROR VALUES ACHIEVED FOR D = 1000

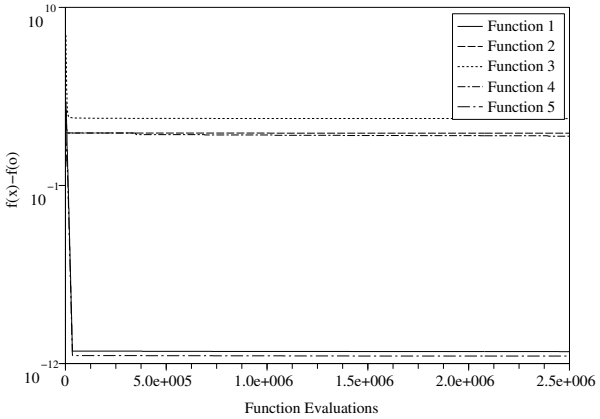

Figure 5. Convergence Graphs of CUS for D=500 (functions 1-5)

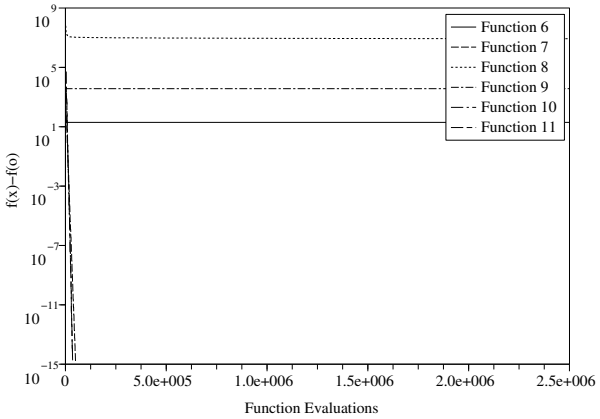

Figure 6. Convergence Graphs of CUS for D=500 (functions 6-11)

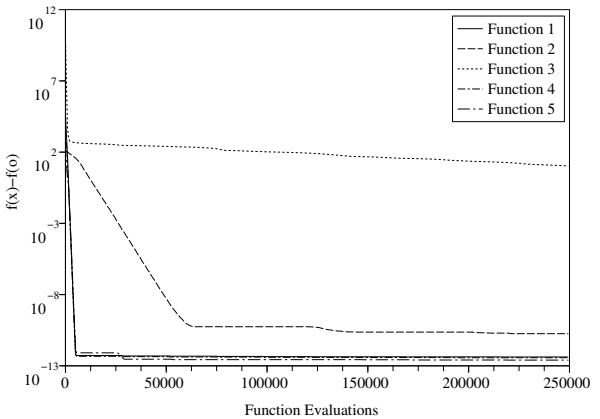

Figure 7. Convergence Graphs of EUS for D=50 (functions 1-5)

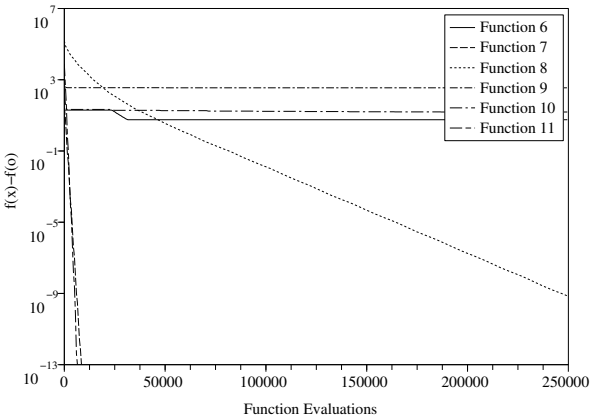

Figure 8. Convergence Graphs of EUS for D=50 (functions 6-11)

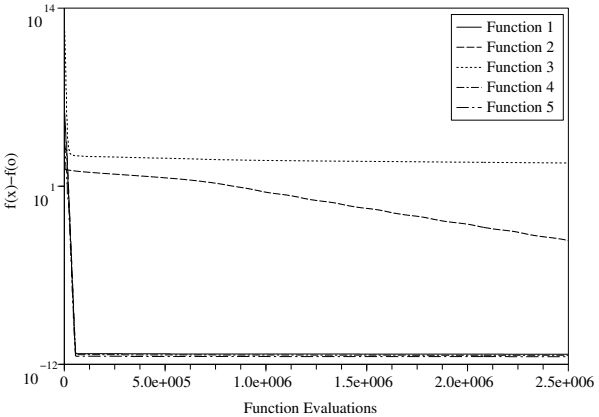

Figure 9. Convergence Graphs of EUS for D=500 (functions 1-5)

Figure 10. Convergence Graphs of EUS for D=500 (functions 6-11)

This table shows a similar behavior between the 5 algorithms on functions #1 and #5. However, EUS has better results than jDEdynNP-F for functions #2 and #4. Function #3 is not well solved by all algorithms, except for MTS. Ackley's function still resists solution for EUS and CUS. The MTS efficiency relies heavily on using 3 different local algorithms. Each one is tested on the neighborhood of a potential solution and the one whoever has the best results is chosen to improve the solution. Besides, our algorithms are stand-alone, in future research we may hybrid them to obtain better results. Overall, with EUS, we obtain competitive performance for 5 out of 6 of the previous problems, compared to efficient but more complex algorithms.

V. Conclusion

We present two main variants of a unidimensional search algorithm in this paper. Our methods, called CUS and EUS, are applied to a test suite that contains 11 high-dimension optimization problems. We utilize a dimension separation process that makes our algorithms robust in handling problems of high dimension. The experiments indicate that our methods achieve fast convergence on a wide range of functions in high dimension. Moreover, after comparison with the 3 best algorithms of the CEC'08 conference, our EUS algorithm shows similar performance results for the 5 first functions of the benchmark. These outcomes are noteworthy for an algorithm that is very simple to implement and that has no parameters to tune.

References

[1] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, January 1965.

[2] R. Marti, M. Laguna, and F. Glover, "Principles of scatter search," *European Journal of Operational Research*, vol. 169, pp. 359–372, 2006.

[3] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang, "Benchmark functions for the cec'2008 special session and competition on large scale global optimization," Nature Inspired Computation and Applications Laboratory, USTC , China, Tech. Rep., 2007, http://nical.ustc.edu.cn/cec08ss.php.

[4] F. Herrera and M. Lozano, "Benchmark functions 7-11," Workshop: Evolutionary Algorithms and other Metaheuristics for Continuous Optimization Problems - A Scalability Test, Tech. Rep., 2009, http://sci2s.ugr.es/programacion/workshop/functions7-11.pdf.

[5] D. Whitley, R. Beveridge, C. Graves, and K. Mathias, "Test driving three 1995 genetic algorithms: New test functions and geometric matching," *Journal of Heuristics*, vol. 1, pp. 77–104, 1995.

[6] K. Tang, "Summary of results on CEC'08 competition on large scale global optimization," University of Science and Technology of China, Tech. Rep., June 2008, http://nical.ustc.edu.cn/papers/CEC2008_SUMMARY.pdf.

[7] L.-Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," in *IEEE Congress on Evolutionary Computation*, 2008, pp. 3052–3059.

[8] Y. Wang and B. Li, "A restart univariate estimation of distribution algorithm: sampling under mixed Gaussian and Lévy probability distribution," in *IEEE Congress on Evolutionary Computation*, 2008, pp. 3917–3924.

[9] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer, "High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction," in *IEEE Congress on Evolutionary Computation*, 2008, pp. 2032–2039.