Random Lines: A Novel Population Set-based Evolutionary Global Optimization Algorithm

İsmet Şahin[1]

Department of Materials Science and Engineering
Room 2123, Chemical and Nuclear Engineering Building (Bldg 90)
University of Maryland, College Park, MD 20742-2115 USA
isahin@gmail.com

Abstract. In this paper, we present a new population set-based evolutionary optimization algorithm which aims to find global minima of cost functions. This algorithm creates random lines passing through pairs of points (vectors) in population, fits a quadratic function based on three points on each line, and then applies the crossover operation to extrema of these quadratic functions, and lastly performs the selection operation. We refer to the points determining random lines as parent points and the extremum of a quadratic model as the descendant or mutated point under some conditions. In the crossover operation, some entries of a descendant vector are randomly replaced with the corresponding entries of one parent vector and some other entries of the descendant vector are replaced with the corresponding entries of the other parent vector based on the crossover constant. The above crossover and mutation operations make this algorithm robust and fast converging. One important property of this algorithm is that its robustness in general increases with increasing population size which may become useful when more processing units are available. This algorithm achieves comparable results with the well-known Differential Evolution (DE) algorithm over a wide range of cost functions.

Keywords: Global Optimization, Continuous Variable Optimization, Direct Search Methods, Evolutionary Computation, Random Lines, Differential Evolution.

1. Introduction

Global extrema of a function play an important role in design and analysis of many science applications [1]. Without loss of generality, we discuss only minimization of functions; therefore, we refer to these functions as cost functions. Direct search algorithms are global optimization algorithms which rely on only cost function

[1] The author works in direct collaboration with the NIST Center for Neutron Research, National Institute of Standards and Technology, 100 Bureau Drive, MS 6100, Gaithersburg, MD 20899-6100 USA. This work is supported in part by the US National Science Foundation under grant DMR-0520547

evaluations for achieving minimization task [2]. As a result, these algorithms are very useful whenever derivative information (gradient or Hessian) about a cost function is not available due to its complexity, discontinuity, or nonnumeric structure. Multiple direct search algorithms [2] have been proposed and successfully used in literature including the Hooke and Jeeves algorithm [3] and the Downhill Simplex algorithm [4].

Population set-based algorithms are direct search algorithms which use multiple points simultaneously for creating variations and then selecting the variations yielding smaller function values than points of the current generation [5-11]. Points of the current generation are called target points and the points representing variations for this generation are called trial points. Variations are usually achieved by means of the mutation and crossover operations. One important example of population set-based algorithms is the DE algorithm [12,13] which is robust and efficient and therefore has been studied extensively and used successfully in many different applications [14-24]. The DE algorithm creates a mutated vector by choosing three different vectors, scaling the difference between two of these vectors, and then adding the scaled difference vector to the third vector [12]. In order to create a trial vector, the DE algorithm replaces some entries of the mutated vector with the entries of one parent vector in the crossover operation based on the crossover constant.

Quadratic functions have been used in many deterministic and stochastic optimization algorithms. The quasi-Newton algorithm in [25] finds a descent direction based on the BFGS (Broyden–Fletcher–Goldfarb–Shanno) Hessian update and performs an inexact search for the minimum along the line in this direction by using quadratic and cubic modals. The line search algorithm in [26] also makes use of quadratic interpolations. The stochastic algorithm Controlled Random Search [27] has a mutation operation which randomly chooses three points in population and fits a quadratic to these three points. A similar mutation operation for DE is also used in [28].

The proposed algorithm inherits main characteristics of population set-based direct search algorithms by using the mutation, crossover, and selection operations but it has major differences in using these operations compared to other population set-based algorithms. The most important difference is in the mechanism of creating variations. The proposed algorithm tries to learn cost function surface by fitting quadratic models and uses extrema of these quadratic models as mutated vectors under some conditions. This allows the algorithm to quickly locate the regions of search space with highly promising points and therefore to achieve fast convergence. In order to increase robustness, this algorithm uses a crossover operation for replacing some entries of a mutated vector with entries from two parent vectors rather than one vector as is usually performed in DE. Comparison of our algorithm with DE demonstrates its high efficiency and robustness over multiple cost functions.

2. Formulation

Consider a set of N-dimensional real vectors $X = \{x_1, x_2, ..., x_{N_P}\}$ with N_P elements. For each target point x_i in X, we randomly choose another point x_j in X

and construct the line passing through x_i and x_j which are called parent points. This line can be represented by $x_i + \mu(x_j - x_i)$ where μ is a real number. In Fig. 1, a generation with five points are demonstrated where each pair of square points represent a specific x_i and the corresponding x_j. Next, we evaluate the function value at a randomly sampled point x_k from the line passing through x_i and x_j:

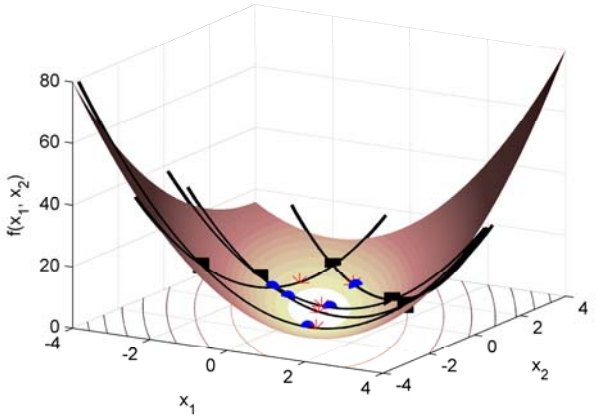
$$x_k = x_i + \mu_k(x_j - x_i) = x_i + \mu_k p \tag{1}$$

where $p = (x_j - x_i)$ and μ_k is a randomly chosen real number representing the step size constant, $\mu_k \neq 0$, $\mu_k \neq 1$. A step $\mu_k p$ from x_i toward x_j is taken if $\mu_k > 0$, otherwise a step in the opposite direction along the line is taken. We also note that the sampled point x_k is between x_i and x_j if $0 < \mu_k < 1$. Clearly, the point x_k will have larger distances from x_i for larger step sizes, therefore the local quadratic model may have larger mismatches with the underlying cost function surface. For this reason, we choose relatively small step sizes μ from the union of two intervals: [-0.95 -0.05] U [0.05 0.95] uniformly in this paper. Note that step sizes chosen from this distribution satisfy the conditions $\mu_k \neq 0$ and $\mu_k \neq 1$. The sampled points for the generation in Fig. 1 are represented by circle signs.

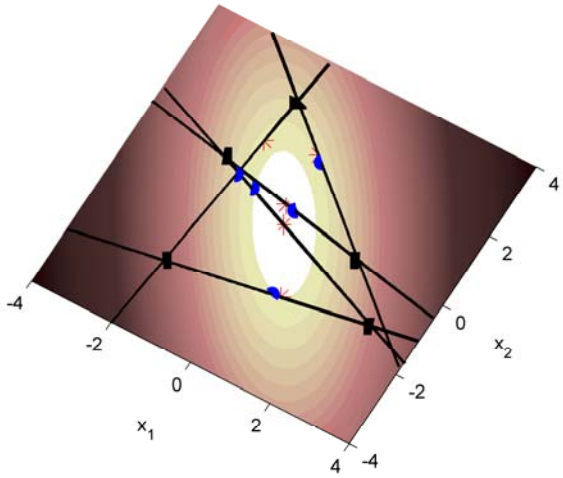
We consider one-dimensional real-valued function $\phi(\mu) = f(x_i + \mu p)$ which represents one-dimensional cross section of $f(x)$ along the line passing through x_i and x_j for the mutation operation. Since function values $\phi(0) = f(x_i)$, $\phi(1) = f(x_j)$, and $\phi(\mu_k) = f(x_i + \mu_k p)$ are known at three different points, a quadratic function $\hat{\phi}(\mu) = a\mu^2 + b\mu + c$ can be fit to $\phi(\mu)$. After some algebraic operations, we can show that the constraints $\hat{\phi}(0) = \phi(0)$, $\hat{\phi}(1) = \phi(1)$, and $\hat{\phi}(\mu_k) = \phi(\mu_k)$ uniquely determine the following coefficients of this quadratic model

$$\begin{aligned} a &= \phi(1) - \phi(0) - b = -\frac{1}{\mu_k - 1}\phi(1) + \frac{1}{\mu_k}\phi(0) + \frac{1}{\mu_k(\mu_k - 1)}\phi(\mu_k) \\ b &= \frac{\mu_k}{\mu_k - 1}\phi(1) - \frac{\mu_k + 1}{\mu_k}\phi(0) - \frac{1}{\mu_k(\mu_k - 1)}\phi(\mu_k) \\ c &= \phi(0) \end{aligned} \tag{2}$$

where $\mu_k \neq 0$ and $\mu_k \neq 1$. The critical point μ_* of this quadratic model is $\mu_* = -b/(2a)$, $a \neq 0$, and the corresponding point $x_* = x_i + \mu_* p$ represents a descendant or mutated point under some conditions. If $a > 0$, the quadratic model is convex with a unique minimum, and therefore x_* becomes the descendant point. Since all quadratic models in Fig. 1 are convex, their minima are the descendant points which are demonstrated with star signs. When $a < 0$, the model is concave with a maximum. In this case, we also use x_* as the descendant point if the function value at the sampled point is smaller than the function value of at least one parent point. This condition can be written as $a < 0$ and [$f(x_k) < f(x_i)$ or $f(x_k) < f(x_j)$]. If these conditions are not satisfied, x_* is not considered to be a descendant point and



(a)



(b)

Fig. 1. Surface of the cost function Schwefel12 [21] is shown from side and top views in (a) and (b) respectively. Five square points represent the current generation. For each square point x_i, another square point x_j is chosen randomly in order to draw a line passing through both points. From each line a point x_k is randomly sampled and the function value at this point is evaluated. The sampled points are demonstrated with circle signs. Since function values at three different points on each line are known, the quadratic function passing through these three points is determined as shown in (a) and its minimum is marked with a star sign in the above figures. Star points constitute descendant points which are clearly located at lower altitudes of the cost function compared to the parent (square) points.

the target vector x_i remains in the next generation. In addition to convex quadratic models, using concave quadratic models creates further diversity in search and therefore increases robustness of the proposed algorithm over multiple cost functions. When $a=0$ we have a linear model rather than a quadratic model; also in this case, x_i remains in the next generation.

```
1  Create an initial generation X_0 = {x_1, x_2, ...., x_{N_P}} and X_c = X_0
2  while (convergence is not achieved)
3      for each x_i ∈ X_c,  i = 1, 2, ...., N_P
4          randomly choose x_j ∈ X_c, x_i ≠ x_j and evaluate p = x_j − x_i
5          choose a step size μ_k uniformly from the interval [-0.95 -0.05] U [0.05 0.95]
6          evaluate f(x_k) = f(x_i + μ_k p) = φ(μ_k)
7          calculate a, b, and c by using equation (2)
8          if a > 0 or [a < 0 and (f(x_k) < f(x_i) or f(x_k) < f(x_j))]
9              evaluate μ_* = −b/(2a), x_* = x_i + μ_* p, and f(x_*)
10             calculate x̂_i by using the crossover operation in equation (3)
11             if f(x̂_i) < f(x_i)
12                 x̂_i ∈ X_+
13             else
14                 x_i ∈ X_+
15             end if
16         else
17             x_i ∈ X_+
18         end if
19     end for
20     X_c = X_+
21 end while
```

Fig. 2. The Random Lines (RL) Algorithm. The sign 'U' on line 5 denotes the union operation.

The crossover operation in our algorithm involves two parent points x_i and x_j, and the descendant point x_*. The trial vector $\hat{x}_i$ is determined based on the following rule:

$$\hat{\alpha}_k = \begin{cases} \alpha_k & \text{if } r_k \le 0.5(1-C_R) \\ \beta_k & \text{if } r_k \ge 0.5(1+C_R) \\ \gamma_k & \text{if } 0.5(1-C_R) < r_k < 0.5(1+C_R) \end{cases} \tag{3}$$

where $\hat{\alpha}_k, \alpha_k, \beta_k, \gamma_k$, and r_k are k^{th} entries of vectors $\hat{x}_i$, x_i, x_j, x_*, and r respectively and C_R is the crossover constant chosen between 0 and 1. The vector r contains N entries, each of which is drawn randomly from the uniform distribution $U[0,1]$. This

rule means that $100 \cdot C_R$ percent of the trial vector $\hat{x}_i$ is determined by the descendant vector x_*, half of the remaining entries of the trial vector is determined by the first parent x_i and the other half is determined by the second parent x_j. For instance, if $C_R = 0.7$, the contributions of x_*, x_i, and x_j vectors to the trial vector $\hat{x}_i$ are 70, 15, and 15 percents on average respectively.

We evaluate function value $f(\hat{x}_i)$ at the trial point for performing the selection operation. If the trial point $\hat{x}_i$ achieves a smaller function value than the target point x_i, the trial point replaces the target point in the next generation, otherwise the target point remains in the next generation. In order to summarize this algorithm, let X_C and X_+ denote the current and next generations respectively. In Fig. 2, we summarize the steps of the proposed Random Lines (RL) algorithm.

3. Performance Evaluation

In this section, we compare performance of the RL algorithm with the well-known global optimization algorithm DE [12]. Since the DE/rand/1/bin variant of the DE algorithm is robust and fast convergent for different cost functions, this variant is often used in performance analysis in literature [5,8,12,21]. We also use this variant through this section. We use 30 cost functions listed in Table 1 including unimodal, multimodal, separable, non-separable, and badly scaled functions. Each algorithm minimizes each of these cost functions 100 times and its total number of successes N_S and average number of function evaluations N_F are recorded for comparing robustness and efficiency of these algorithms. The average number of function evaluations is calculated based on only successful runs. In order to compare number of function evaluations of the algorithms, we use the acceleration rate (AR) defined as the ratio of average number of function evaluations for RL to average number of function evaluations for DE [21]. The acceleration rate is only defined for the cost functions for which both algorithms achieve at least 30 successes since the average number of function evaluations may deviate largely for smaller number of successful runs.

Since both algorithms use parameters C_R and N_P, we choose the same parameter values $C_R = 0.9$ and $N_P = 10N$ for both algorithms. The DE algorithm also uses $F = 0.5$ for scaling difference vectors. Multiple studies [5, 21, 29, 30] also use similar DE parameters as the DE algorithm usually achieves larger number of successes with smaller number of function evaluations with these parameter values. Both algorithms use the Mersenne Twister random number generator [13,35]. We say that an algorithm is successful or it achieves convergence when the algorithm finds an

Table 1. Cost functions and their limits and references. Here [*a*,*b*] means that each component of a vector in the initial generation is in interval [*a*,*b*]. Dimensions of functions are specified in paranthesis in first column.

Cost Functions	Limits	References
Ackley(*N*)	[-30, 30]	[21]
Alpine(*N*)	[-10, 10]	[21]
Beale(2)	[-10, 10]	[21]
Branin(2)	$x_1 \in$[-5, 10], $x_2 \in$[0, 15]	[21]
Brown Badly Scaled(2)	[-1e7, 1e7]	[32]
Camel6(2)	[-5, 5]	[21]
Colville(4)	[-10, 10]	[21]
Cubic Valley(2)	[-100, 100]	[34]
Dejong4(*N*)	[-1.28, 1.28]	[21]
GoldsteinPrice(2)	[-2, 2]	[33]
Griewangk(*N*)	[-600, 600]	[21]
Hartman3(3)	[0, 1]	[21]
Hartman6(6)	[0, 1]	[21]
Hyperellipsoid(*N*)	[-5.12, 5.12]	[21]
Kowalik(4)	[-5, 5]	[21]
Matyas(2)	[-10, 10]	[21]
Powell Badly Scaled(2)	[-10, 10]	[32]
Rastrigin(*N*)	[-5.12, 5.12]	[21]
Rosenbrock(*N*)	[-2.048, 2.048]	[21]
Schwefel12(*N*)	[-65, 65]	[21]
Schwefel221(*N*)	[-100, 100]	[21]
Schwefel222(*N*)	[-10, 10]	[21]
Shekel5(4)	[0, 10]	[21]
Shekel7(4)	[0, 10]	[21]
Shekel10(4)	[0, 10]	[21]
Shekel's Foxhole(5)	[0, 10]	[5]
Sphere(*N*)	[-5.12, 5.12]	[21]
Step(*N*)	[-100, 100]	[21]
Sum.Diff.Powers(*N*)	[-1, 1]	[21]
Zakharov(*N*)	[-5, 10]	[21]

acceptable point x_{best} satisfying $f(x_{best}) < f(x_{opt}) + \varepsilon$ where $\varepsilon = 10^{-5}$ and x_{opt} is the global minimizer. The algorithm is unsuccessful if it cannot find an acceptable point over 3000000 function evaluations. The algorithm is also unsuccessful if it cannot reduce the best function value over 500 generations.

Table 2 lists the number of successes and average number of function evaluations for DE and RL. For these results, we use two dimensional forms of the cost functions which can be extended to higher dimensions such as the Ackley and Rosenbrock's functions. Comparison of the results under the population size $N_P = 10N$ shows that RL achieves 2824 successes and DE achieves 2525 successes over 3000 runs. Both algorithms have the same number of successes over 16 functions. RL has larger number of successes than DE over 10 functions and DE has larger number of successes than RL over 4 cost functions. Both algorithms have at least 30 successes over 27 functions and DE is slightly more efficient than RL by requiring smaller number of function evaluations for 15 of these cost functions.

Table 2. The number of successes and average number of function evaluations for DE and RL. Hyphens signify that there is no successful run for the corresponding cost function and therefore N_F and **AR** are not defined. Extended functions such as Rosenbrock are 2-dimensional.

Cost Functions	DE $N_P = 10N$		RL $N_P = 10N$		AR	DE $N_P = 40N$		RL $N_P = 40N$		AR
	N_S	N_F	N_S	N_F		N_S	N_F	N_S	N_F	
Ackley	100	1165	100	2472	0.47	100	4479	100	6719	0.67
Alpine	100	1269	100	2752	0.46	100	5502	100	8672	0.63
Beale	99	691	100	733	0.94	100	2456	100	1819	1.35
Branin	100	799	100	415	1.93	100	3495	100	1211	2.89
Brown	--	--	100	2259	--	2	15240	100	9099	--
Camel	100	673	100	508	1.32	100	2690	100	1652	1.63
Colville	70	4840	100	8336	0.58	100	20259	100	28361	0.71
Cube	58	1443	100	4350	0.33	100	5487	100	10198	0.54
De Jong 4	100	149	100	114	1.31	100	387	100	313	1.24
Golds. Price	100	685	100	659	1.04	100	2458	100	1926	1.28
Griewangk	92	1765	57	2587	0.68	100	7115	100	6375	1.12
Hartman3	100	990	100	1142	0.87	100	3604	100	3454	1.04
Hartman6	35	5929	100	6146	0.96	36	29753	100	17075	1.74
Hyperellips.	100	488	100	183	2.67	100	1739	100	523	3.33
Kowalik	99	3318	100	1297	2.56	100	4265	100	2915	1.46
Matyas	100	450	100	138	3.26	100	1612	100	396	4.07
Powell	--	--	83	5980	--	--	--	100	23934	--
Rastrigin	95	1123	96	1556	0.72	100	4494	100	2808	1.60
Rosenbrock	83	670	100	1339	0.50	100	2517	100	4202	0.60
Schw. 1.2	100	695	100	238	2.92	100	2546	100	697	3.65
Schw. 2.21	100	1178	100	3136	0.38	100	4536	100	7141	0.64
Schw. 2.22	100	1000	100	1862	0.54	100	3816	100	4850	0.79
Shekel 5	90	3468	95	7811	0.44	100	13852	100	20348	0.68
Shekel 7	100	3242	96	7148	0.45	100	12787	100	19839	0.64
Shekel 10	100	3289	97	6647	0.49	100	12931	100	18995	0.68
Shek. Foxh.	4	5987	--	--	--	21	27219	4	134205	--
Sphere	100	475	100	179	2.65	100	1686	100	507	3.33
Step	100	289	100	137	2.11	100	950	100	375	2.53
Sum of pow.	100	264	100	140	1.89	100	820	100	374	2.19
Zakharov	100	534	100	297	1.80	100	1906	100	963	1.98

When the population size increases from $N_P = 10N$ to $N_P = 40N$, both algorithms have larger number of successes and function evaluations. RL converges successfully for 2904 runs and DE converges for 2659 runs over 3000 runs. They have the same number of successes for 26 cost functions. RL achieves larger number of successes than DE for the Brown, Hartman 6, and Powell cost functions, and DE has larger number of successes than RL for the Shekel's Foxholes function. From the last AR column, notice that RL requires smaller number of function evaluations than DE over 17 functions and DE requires smaller number of evaluations than RL over 10 functions.

Table 3. The number of successes and average number of function evaluations for DE and RL with 20-dimensional extended cost functions. Hyphens have the same meaning with hyphens in Table 2.

Cost Functions	DE $N_P = 15N$		RL $N_P = 15N$		AR	RL $N_P = 30N$		RL $N_P = 45N$	
	N_S	N_F	N_S	N_F		N_S	N_F	N_S	N_F
Ackley	100	294489	94	196893	1.50	100	292128	100	402248
Alpine	--	--	100	120937	--	100	208987	100	299383
Dejong4	100	82614	100	29052	2.84	100	28932	100	32616
Griewangk	100	436881	42	149311	2.93	68	289330	64	330154
Hyperellips.	100	169137	100	97157	1.74	100	161616	100	222191
Rastrigin	--	--	76	136360	--	100	222571	100	309498
Rosenbrock	100	500196	100	877172	0.57	100	1412765	100	1931361
Schwefel12	100	609909	100	294425	2.07	100	458028	100	613619
Schwefel221	100	578682	100	690687	0.84	100	747657	100	892602
Schwefel222	100	344670	100	137122	2.51	100	241600	100	343211
Sphere	100	148458	100	79955	1.86	100	134016	100	183455
Step	100	101364	7	73548	--	40	96062	71	121155
Sum of pow.	100	42546	100	3720	11.44	100	7092	100	10332
Zakharov	100	534828	100	216276	2.47	100	348900	100	477540

The number of successes and average number of function evaluations for 20 dimensional cost functions are given in Table 3. Since DE performance with $N_P = 30N$ and $N_P = 45N$ are usually worse than $N_P = 15N$ for these cost functions, only the DE results under $N_P = 15N$ are listed. Comparison of the results with $N_P = 15N$ shows that RL achieves slightly larger total number of successes than RL as they converge over 1219 and 1200 runs respectively out of 1400 runs. RL converges over 100 and 76 runs for the Alpine and Rastrigin cost functions respectively as DE does not converge for these cost functions. However, DE achieves 100 successes for the Step function while RL has 7 successes for this function. The sixth column specifies the acceleration rates for the results under $N_P = 15N$. From this column, notice that RL is more efficient over 9 functions and DE is more efficient over 2 cost functions. In particular, notice that DE requires approximately 11.4 times more function evaluations for the Sum of Powers function, 2.8 times more function evaluations for the Dejonk4 and Griewangk functions, and 2.5 times more function evaluations for the Schwefel222 and Zakharov functions than the RL algorithm.

Robustness of the RL algorithm in general increases with increasing population size. The number of successes for RL increases for the Ackley, Rastrigin, and Step functions with increasing population size in Table 3. Even though the number of function evaluations increases with increasing population size, the RL algorithm simultaneously achieves higher efficiency and robustness than DE. The total number of successes increases to 1308 and 1335 when population size is increased to $N_P = 30N$ and $N_P = 45N$ respectively for RL. For the population size $N_P = 30N$, RL is more efficient over 10 cost functions than DE and DE is more efficient than RL over 2 functions.

4. Conclusion

In this paper, we presented a new population set-based global optimization algorithm. The mutation operation of this algorithm makes an effort to learn cost function surface by constructing random lines passing through pairs of points in the current generation and then fitting a quadratic function by using three points on each line. The extrema of these quadratic models constitute descendant points which are subject to the crossover operation. The mutation operation with quadratic fit quickly finds regions of the search space with highly promising points and therefore allows this algorithm to achieve fast convergence. The crossover operation randomly selects components from two parents and replaces corresponding components of the descendant point. Using components from both parents increases diversity in search and therefore increases robustness. Minimization of 30 cost functions demonstrates that this algorithm achieves very promising results compared with the well-known DE algorithm. In particular, the robustness of this algorithm in general increases with increasing population size which becomes important when there are more processing units available.

Acknowledgements. I would like to thank Paul Kienzle, Florencia McAllister, Bulent Akgun, and Nuri Yilmazer for their useful comments.

References

1. Törn, A., Zilinskas, A.: Global Optimization. Springer-Verlag, New York (1989)
2. Kolda, T.G., Lewis, R.M., Torczon, V.: Optimization by direct search: New perspectives on some classical and modern methods. Siam Review 45(3), 385-482 (2003)
3. Hooke R., Jeeves, T.A.: Direct Search Solution of Numerical and Statistical Problems. Journal of the ACM 8(2), 212-229 (1961)
4. Nelder J.A., Mead, R.: A Simplex Method for Function Minimization. Computer Journal 7(4), 308-313 (1965)
5. Ali M.M., Törn, A.: Population Set-based Global Optimization Algorithms: Some Modifications and Numerical Studies. Computers and Operations Research 31(10), 1703–1725 (2004)
6. Back, T., Schewefel, H.P.: An Overview of Evolutionary Algorithms for parameter optimization. Evolutionary Computation 1(1), 1-23 (1993)
7. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, New York (2003)
8. Yao, X., Liu, Y., Lin, G.: Evolutionary Programming Made Faster. IEEE Transactions on Evolutionary Computation 3(2), 82-102 (1999)
9. Back, T., Hammel, U., Schwefel, H.P.: Evolutionary Computation: Comments on the History and Current State. IEEE Transactions on Evolutionary Computation 1(1), 3-17 (1997)
10. Fogel, D.B.: What Is Evolutionary Computation? IEEE Spectrum 37(2), 28-32 (2000)
11. Hinterding, R., Michalewicz, Z., Eiben, A.E.: Adaptation in Evolutionary Computation: A Survey. In: IEEE International Conference on Evolutionary Computation, pp. 65-69. IEEE Press, Los Alamitos (1997)

12. Storn, R., Price, K.: Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, 11(4), 41-359 (1997)
13. Price, K.V., Storn, R.M., Lampinen, J.A..: Differential Evolution: A Practical Approach to Global Optimization. Springer-Verlag, Berlin, Heidelberg (2005)
14. Veenhuis, C.B.: Tree Based Differential Evolution. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) LNCS, vol. 5481, pp. 208-219, Springer, Heidelberg (2009)
15. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Transactions on Evolutionary Computation 10(6) 646-657 (2006)
16. Montgomery, J., Chen, S.: An Analysis of the Operation of Differential Evolution at High And Low Crossover Rates. In: IEEE Congress on Evolutionary Computation, pp. 1-8. IEEE Press, Los Alamitos (2010)
17. Caponio, A., Neri, F.: Differential Evolution with Noise Analyzer. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G., Ekart, A., Esparcia-Alcazar, A., Farooq, M., Fink, A. Machado, P. (eds.) LNCS, vol. 5484, pp. 715-724, Springer, Heidelberg (2009)
18. Liu, G., Li, Y.X., He, G.L.: Design of Digital FIR Filters Using Differential Evolution Algorithm Based on Reserved Genes. In: IEEE Congress on Evolutionary Computation, pp. 1-7. IEEE Press, Los Alamitos (2010)
19. Das, S., Konar, A.: Automatic Image Pixel Clustering with an Improved Differential Evolution. Applied Soft Computing 9(1) 226-236 (2009)
20. Shi, Y., Teng, H., Li Z.: Cooperative Co-evolutionary Differential Evolution for Function Optimization. In: Wang, L., Chen, K., S. Ong, Y. (eds.) LNCS, vol. 3611, pp. 1080-1088, Springer, Heidelberg (2005)
21. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution. IEEE Transactions on Evolutionary Computation 12(1) 64-79 (2008)
22. Kundu, D., Suresh, K. Ghosh, S., Das, S., Abraham, A., Badr, Y.: Automatic Clustering Using a Synergy of Genetic Algorithm and Multi-objective Differential Evolution. In: Corchado, E., Wu, X., Oja, E., Herrero, A., Baruque, B. (eds.) LNCS, vol. 5572, pp. 177-186, Springer, Heidelberg (2009)
23. Abbass, H.A., Sarker, R., Newton, C.: PDE: a Pareto-frontier Differential Evolution Approach for Multi-objective Optimization Problems. In: Proceedings of the 2001 Congress on Evolutionary Computation, Seoul, South Korea, vol. 2, pp. 971-978. IEEE Press, Los Alamitos (2001)
24. Vesterstroem, J., Thomsen, R.: A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems. In: Proc. Congr. Evol. Comput. vol. 2, pp. 1980–1987 (2004)
25. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Inc., Englewood Cliffs, NJ (1983)
26. More, J.J., Thuente, D.J.: Line Search Algorithms with Guaranteed Sufficient Decrease. ACM Transactions on Mathematical Software 20, 286-307 (1992)
27. Mohan, C., Shanker, K.: A Controlled Random Search Technique for Global Optimization Using Quadratic Approximation. Asia-Pacific Journal of Operational Research 11, 93-101 (1994)
28. Thangaraj, R., Pant, M., Abraham, A.: New Mutation Schemes for Differential Algorithm and Their Application to the Optimization of Directional Over-current Relay Settings. Applied Mathematics and Computation 216, 532-544 (2010)
29. Liu, J., Lampinen, J.: A Fuzzy Adaptive Differential Evolution Algorithm. Soft Computing-A Fusion of Foundations, Methodologies and Applications 9(6) 448–462 (2005)

30. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Selfadapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Transactions on Evolutionary Computation 10(6) 646–657 (2006)
31. Storn, R.: On the Usage of Differential Evolution for Function Optimization. In: Proc. Biennial Conf. North Amer. Fuzzy Inf. Process. Soc., pp. 519–523 (1996)
32. More, J.J., Garbow, B.S., Hillstrom, K.E.: Testing Unconstrained Optimization Software. ACM Transactions on Mathematical Software 7(1) 17-41 (1981)
33. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems. Journal of Global Optimization 31, 635-672 (2005)
34. Pierre, D.A.: Optimization Theory with Applications. Dover Publications Inc., Mineola, NY (1969)
35. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation 8(1), 3-30 (1998)