An International Journal of Optimization
and Control: Theories & Applications
Vol.3, No.2, pp.111-119 (2013) © IJOCTA
ISSN:2146-0957 eISSN:2146-5703
DOI:10.11121/ijocta.01.2013.00167
http://www.ijocta.com

Minimization over randomly selected lines

İsmet Şahin

Mathematical and Computational Science Division
National Institute of Standards and Technology
Gaithersburg, MD 20899-8910 USA
Email: isahin@gmail.com

(Received April 25, 2013; in final form June 23, 2013)

Abstract. This paper presents a population-based evolutionary optimization method for minimizing a given cost function. The mutation operator of this method selects randomly oriented lines in the cost function domain, constructs quadratic functions interpolating the cost function at three different points over each line, and uses extrema of the quadratics as mutated points. The crossover operator modifies each mutated point based on components of two points in population, instead of one point as is usually performed in other evolutionary algorithms. The stopping criterion of this method depends on the number of almost degenerate quadratics. We demonstrate that the proposed method with these mutation and crossover operations achieves faster and more robust convergence than the well-known Differential Evolution and Particle Swarm algorithms.

Keywords: Random lines; nonlinear optimization; evolutionary optimization; population-based optimization; quadratic interpolation; crossover operator; mutation operator; stopping criterion; differential evolution; particle swarm.
AMS Classification: 90-04, 90-08, 90B50

1. Introduction

Population-based evolutionary optimization methods try to minimize a given cost function by using a set of points [1, 2, 3]. They are called evolutionary as they create variations in population through mutation and crossover operations and then select the variations that improve previous generation. In general they create variations based on only cost function evaluations thus they are easy-to-use for the cost functions whose gradient and Hessian are not known due to their complexity, discontinuity, or non-numeric structure. This paper presents a new mutation operation based on quadratic functions and a new crossover operation for creating variations.

Quadratic functions are frequently used in both deterministic and stochastic optimization methods. The Newton's method uses quadratic approximation of the cost function at current iterate and assigns its minimizer as the next best guess [4]. The quasi-Newton algorithm finds a descent direction based on the BFGS (Broyden–Fletcher–Goldfarb–Shanno) Hessian approximation and performs an inexact line search by using quadratic and cubic function models in [5]. The line search in [6] also uses quadratic interpolation for finding a minimizer over the line. The stochastic method Controlled Random Search [7] has a mutation operator which randomly chooses three points in population and fits a quadratic to these points. A similar mutation operator for Differential Evolution (DE) is also used in [8], which constructs the mutated vector by using either the DE mutation scheme or the quadratic interpolation based on a fixed mutation probability.

This work is supported in part by the US National Science Foundation under grant DMR-0520547.

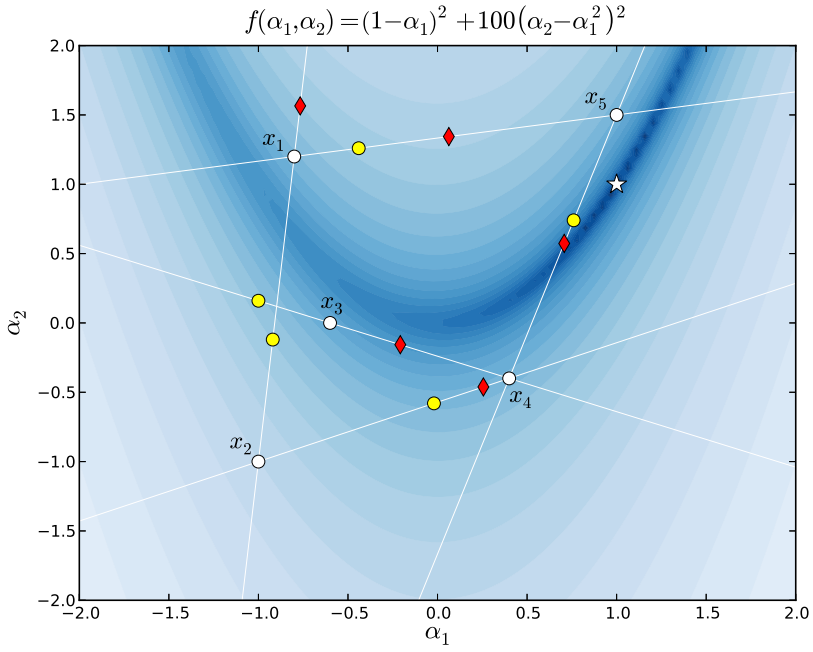

Figure 1. The contours of 2-dimensional Rosenbrock function with five random lines. The minimizer of this function $x_{opt} = [1, 1]^T$ is illustrated by the star sign. White circles represent parent points, yellow circles represent randomly sampled points, and red diamonds represent extrema of the quadratics passing through three different points over the lines.

The proposed mutation operator tries to learn cost function surface by fitting quadratics over one-dimensional slices taken from the search space. These slices are defined by lines passing through pairs of points in population. The points are randomly paired in order to achieve as uniform approximation as possible to the function. Using minima of convex quadratics as mutated points improves convergence rates as they often indicate regions of the search space with smaller function values. We note that this mutation approach is different than DE's mutation [9, 10, 11] which depends on only the points in population, whereas the proposed approach uses both the points as well as function values at these points. The crossover operation involves two points in population for improving robustness and efficiency. It replaces randomly selected entries of a mutated vector by corresponding entries of two other vectors in population. The proposed method's stopping criterion stops the search when at least a pre-specified number of quadratics are almost degenerate.

2. Formulation

Consider a set of n-dimensional real vectors $X = \{x_1, x_2, \ldots, x_{n_P}\}$ with n_P elements. The algorithm pairs each point x_i with another point x_j in X randomly. The paired points (x_i, x_j) are called the *parent points*. For instance a population of five points x_1, x_2, x_3, x_4, and x_5 are illustrated as white circles in Figure 1 for two dimensional Rosenbrock function, where random pairing resulted into these pairs: (x_1,x_5), (x_2,x_1), (x_3,x_4), (x_4,x_2), and (x_5,x_4). The parent point x_i is also called a *target point* since it competes with a *trial point* $\hat{x}_i$, which is a variational point constructed by the mutation and crossover operators described in this section. The pairing process results into n_P pairs, where i^{th} pair has x_i as its first entry thus guaranteeing each point in X to be a target point.

The parametric form of the line passing through x_i and x_j can be written as $x_i+\mu(x_j-x_i)$ where μ is a real number. For instance five lines passing through the paired points are illustrated in Figure 1. The algorithm randomly chooses a number μ_k and finds a third point $x_k = x_i + \mu_k p_i$ over the line where $p_i = x_j - x_i$. Note that a step $\mu_k p_i$ from x_i toward x_j is taken if $\mu_k > 0$, otherwise a step in the opposite direction is taken. When x_k is far from x_i and x_j, the resulting quadratic model may have large mismatches with the underlying cost function. Therefore we choose relatively small step sizes by uniformly drawing μ_k from the union of two intervals: $[-\mu_2, -\mu_1] \cup [\mu_1, \mu_2]$ where $0 < \mu_1 < \mu_2 < 1$. Note that μ_k chosen from this distribution satisfies $\mu_k \neq 0$ and $\mu_k \neq 1$ thus $x_k \neq x_i$ and

$x_k \neq x_j$, and a unique quadratic passing through $(x_i, f(x_i))$, $(x_j, f(x_j))$, and $(x_k, f(x_k))$ exists. In Figure 1 five sampled points are represented by yellow circles.

Consider the function $\phi(\mu) = f(x_i + \mu p_i)$ representing the one-dimensional cross section of $f(x)$ along the line passing through x_i and x_j. Since function values $\phi(0) = f(x_i)$, $\phi(1) = f(x_j)$, and $\phi(\mu_k) = f(x_k)$ are known, the coefficients of the interpolating quadratic function $\hat{\phi}(\mu) = a\mu^2 + b\mu + c$ can be found as:

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

The critical point of $\hat{\phi}(\mu)$ is $\mu_* = -b/(2a)$ and corresponding point is $x_* = x_i + \mu_* p_i$. If the quadratic is convex ($a > 0$), then x_* is assigned to be the mutated point. When the quadratic is concave, it has a maximizer instead of a minimizer. In this case, the step $\mu_* p_i$ from x_i leads to the maximizer, thus the step $-\mu_* p_i$ away from the maximizer is taken, i.e. $x_* = x_i + (b/(2a))\, p_i$. This concave case handling is in contrast to our previous study[12] which allows steps toward the maximizer under a condition. For numerical stability we treat the model being convex if $a > 10^{-6}$ and concave if $a < -10^{-6}$. When $|a| \leqslant 10^{-6}$, the quadratic is considered to be degenerate and x_i remains in the next generation. Figure 1 demonstrates a concave quadratic over the line passing through (x_1, x_5) and four convex quadratics over the other lines.

The crossover operator constructs the trial vector $\hat{x}_i$ by replacing some entries of x_* with the corresponding entries of either x_i and x_j by the following rule:

$$\hat{x}_i^k = \begin{cases} x_i^k & \text{if } r^k \leqslant \frac{1}{2}(1 - \rho_{CR}) \\ x_j^k & \text{if } r^k \geqslant \frac{1}{2}(1 + \rho_{CR}) \\ x_*^k & \text{if } \frac{1}{2}(1 - \rho_{CR}) < r^k < \frac{1}{2}(1 + \rho_{CR}) \end{cases} \tag{2}$$

where superscript k denotes the k^{th} entry of the vector, r^k is drawn from Uniform[0,1], the uniform distribution between 0 and 1, and ρ_{CR} denotes a pre-fixed crossover constant chosen between 0 and 1. This rule means that $100 \cdot \rho_{CR}$ percent of the trial vector $\hat{x}_i$ is determined by x_*, half of the remaining entries is determined by x_i, and the other half by x_j. For instance, if $\rho_{CR} = 0.8$, the contributions of x_*, x_i, and x_j vectors are 80, 10, and 10 percents on average respectively.

The selection operator selects $\hat{x}_i$ as the i^{th} member of the next generation if $f(\hat{x}_i) < f(x_i)$. Otherwise, the target vector x_i remains in the next generation.

The above mutation, crossover, and selection operations are performed until a stopping criterion is satisfied. The stopping criterion of this method depends on the number of almost degenerate quadratics. The method stops when the number of quadratics n_D satisfying $|a| < \epsilon_D$ and $|b| < \epsilon_D$ is equal to or larger than n_{D_max}, where ϵ_D is a small positive number and n_{D_max} is a number such that $1 \leqslant n_{D\_max} \leqslant n_P$. This means that if at least n_{D_max} quadratics are approximately constant functions, then the search stops. When this stopping criterion is not satisfied for some cost functions, the method also stops if improvement in the function value is smaller than a positive number ϵ_{NI} over at least n_{NI_max} generations [13].

Since minimization is based on extrema over randomly selected lines, we shortly refer to this approach as the Random Lines (RL) method and summarize it in Figure 2.

3. Performance Evaluation

Performance of a stochastic optimization method is often determined by its robustness and efficiency. When the same function is minimized multiple times, a robust method finds the global minimizer more frequently and an efficient method finds the minimizer after smaller number of function evaluations. We minimize each cost function 20 times and count total number of successes n_S as a measure of robustness. Since the stopping criterion using the absolute values of quadratic coefficients is only relevant to the RL method, for a systematic comparison of all methods we consider that a run is successful when a method finds a point x_{best} satisfying $f(x_{best}) - f(x_{opt}) < \epsilon_{opt}$ where x_{opt} is the known global minimizer and $\epsilon_{opt} = 10^{-5}$. The run is unsuccessful if improvement in function value is less than $\epsilon_{NI} = 10^{-12}$ for $n_{NI} = 50$ generations. In order to compare efficiency, the average of number of function evaluations n_{FE} for minimizing each cost function is also determined.

Since performance may vary for cost functions with different properties, we perform comparison over 50 cost functions listed in Table 1. The test suite includes separable and non-separable, normally-scaled and ill-scaled, unimodal and multimodal cost functions which are often used in optimization literature [14, 15, 17]. In order to compare scalability of the methods to higher dimensional problems, we minimize 2, 10, and 20-dimensional forms of the 16 extendible cost functions listed in Table 1.

The RL method is compared with the DE [9] and the Particle Swarm with constriction (PS)

```
1  Start with X_C = {x_1, x_2, ..., x_nP}                    //initialization...
2  f_best_prev = min(f(x_1), f(x_2), ..., f(x_nP))
3  X_+ = X_C
4  n_NI = 0
5  satisfied = 0
6  while(satisfied == 0)
7     foreach i in {1,2,...,n_P}
8        draw j from {1,2,...,i-1,i+1,...,n_P}              //pairing and finding x_k...
9        p_i = x_j - x_i
10       draw μ_k from [-μ_2,-μ_1]U[μ_1,μ_2]
11       x_k = x_i + μ_k p_i
12       b_i = (μ_k/(μ_k-1)) f(x_j) - ((μ_k+1)/μ_k) f(x_i) - (1/(μ_k(μ_k-1))) f(x_k)   //quadratic coefficients...
13       a_i = f(x_j) - f(x_i) - b_i
14       if a_i > 10^-6                                     //mutation operator...
15          μ_* = -b_i/(2a_i)        //μ_* for a convex quadratic
16       elseif a_i < -10^-6
17          μ_* = b_i/(2a_i)         //μ_* for a concave quadratic
18       else
19          continue                 //|a| ⩽ 10^-6, go to line 7
20       endif
21       x_* = x_i + μ_* p_i
22       foreach k in {1,2,...,n}                           //crossover operator...
23          draw r^k from Uniform[0,1]
24          if r^k ⩽ 0.5(1 - ρ_CR)
25             x_*^k = x_i^k
26          elseif r^k ⩾ 0.5(1 + ρ_CR)
27             x_*^k = x_j^k
28          endif
29       endfor
30       if f(x_*) < f(x_i)                                 //selection operator...
31          x_* ∈ X_+
32       endif
33    endfor
34    f_best = min(f(x_1), f(x_2), ..., f(x_nP)) where x_i ∈ X_+   //stopping criteria...
35    n_D is the total number of quadratics satisfying |a_i| < ε_D and |b_i| < ε_D
36    if n_D ⩾ n_D_max
37       satisfied = 1
38    else
39       if f_best < f_best_prev - ε_NI
40          n_NI = 0              //improvement on f_best
41       else
42          n_NI = n_NI + 1       //no improvement on f_best
43          if n_NI ⩾ n_NI_max
44             satisfied = 2
45          endif
46       endif
47    endif
48    X_C = X_+                                          //update fields...
49    f_best_prev = f_best
50 endwhile
```

Figure 2. The Random Lines (RL) method. The default parameter values are $\mu_1 = 0.3$, $\mu_2 = 0.7$, $\rho_{CR} = 0.9$, $\epsilon_D = 10^{-4}$, $n_{D\_max} = 0.2n_P$, $\epsilon_{NI} = 10^{-12}$, $n_{NI\_max} = 50$, $n_P = 10n$. The symbols X_C and X_+ denote the current and next generations. The keyword "continue" means to skip the rest of the for loop and continue with the next i. Comments are written after two slash characters. Increase n_P for increasing robustness. Increase n_{D_max} and/or decrease ϵ_D for increasing accuracy of solution.

Table 1. The cost functions used in performance comparison. The initial population is generated by uniformly choosing points from the hypercube defined by $\alpha_k \in [l_1, l_2]$ for $k = 1, 2, \ldots, n$ where $[l_1, l_2]$ are listed under search space columns. Dimensions of cost functions are listed in parenthesis, where the extendible functions are indicated by (n) symbol.

Cost Function	Search Space	Reference	Cost Function	Search Space	Reference
Ackley(n)	[-30, 30]	[14]	Hyperellipsoid(n)	[-5.12, 5.12]	[14]
Alpine(n)	[-10, 10]	[14]	Jennrich(2)	[-1,1]	[15]
Aluffi(2)	[-10, 10]	[16]	Kowalik(4)	[-5,5]	[14]
Bard(3)	[-10, 10]	[15]	Levy1(n)	[-10, 10]	[17]
Beale(2)	[-10, 10]	[14]	Matyas(2)	[-10, 10]	[14]
Becker(2)	[-10, 10]	[17]	Miele(4)	[-1, 1]	[17]
Bohachevsky1(2)	[-50, 50]	[17]	MultiGaussian(2)	[-2, 2]	[17]
Branin(2)	$\alpha_1 \in [-5,0], \alpha_2 \in [10,15]$	[14]	Periodic(2)	[-10, 10]	[17]
Brown(2)	[-1e7, 1e7]	[15]	Powell(2)	[-10, 10]	[15]
Camel3(2)	[-5, 5]	[14]	PowellsQ(4)	[-10, 10]	[17]
Camel6(2)	[-5, 5]	[14]	Rastrigin(n)	[-5.12, 5.12]	[14]
Colville(4)	[-10,10]	[14]	Rosenbrock(n)	[-2.048, 2.048]	[14]
Cubic(2)	[-100, 100]	[18]	Schaffer1(2)	[-100, 100]	[17]
Dejong4(n)	[-1.28, 1.28]	[14]	Schaffer2(2)	[-100, 100]	[17]
Dekker(2)	[-20, 20]	[17]	Schwefel12(n)	[-65, 65]	[14]
Easom(2)	[-10, 10]	[14]	Schwefel221(n)	[-100, 100]	[14]
Exponential(n)	[-1,1]	[17]	Schwefel222(n)	[-10, 10]	[14]
Freudenstein(2)	[-20, 20]	[15]	Shekel5(4)	[0, 10]	[14]
Gaussian(3)	[-10, 10]	[15]	Shekel7(4)	[0, 10]	[14]
Goldstein(2)	[-2, 2]	[17]	Shekel10(4)	[0, 10]	[14]
Griewangk(n)	[-600, 600]	[14]	Sphere(n)	[-5.12, 5.12]	[14]
Gulf(3)	$\alpha_1 \in [0,100], \alpha_2 \in [0,25], \alpha_3 \in [0,5]$	[15]	Step(n)	[-100, 100]	[14]
Hartman3(3)	[0, 1]	[14]	Sum.Diff.Powers(n)	[-1, 1]	[14]
Hartman6(6)	[0, 1]	[14]	Wood(4)	[-10, 10]	[17]
Helical(3)	[-10, 10]	[15]	Zakharov(n)	[-5, 10]	[14]

[19, 20] methods. All methods have the same population size of $n_P = 20n$. The DE and RL methods have the crossover constant of $\rho_{CR} = 0.9$. Since the DE/rand/1/bin variant of the DE algorithm is robust and fast convergent, this variant is used in many studies [9, 14, 21, 22], therefore it is also used in this section. The scaling factor for DE is $\rho_F = 0.5$. The constricted Particle Swarm method uses the default constriction constants $c_1 = 2.8$ and $c_2 = 1.3$. The RL uses the default parameters specified in Figure 2 except n_P. The initial generations are constructed by uniformly drawing points from the hypercubes specified in Table 1.

Table 2 presents performance results for the cost functions with fixed dimensions and for 2-dimensional forms of the extendible functions. We note that RL is the most robust method by achieving total number of 985 successes over 1000 runs, while PS and DE converge over 932 and 910 runs respectively. Since number of function evaluations for one cost function can be an outlier, total number of function evaluations may not indicate efficiency without bias. We count total number functions for which an algorithm requires the least number of function evaluations. In this sense, RL is the most efficient method by requiring least number of function evaluations for 29 functions, which is followed by DE and PS with 14 and 7 functions respectively.

To compare scalability to higher dimensional cost functions, the results for 10 and 20-dimensional forms of the extendible functions are listed in Table 3 and 4 respectively. Results for PS are not presented in these tables as its robustness decreases substantially for these cost functions. Table 3 shows that the RL is more robust than DE since they converge over 303 and 260 runs out of 320 runs respectively. Notice that DE cannot achieve any success for 3 functions whereas RL achieves at least 6 successes for any cost function. The RL method is more efficient for 12 cost functions while this number is 1 for DE. When the problem dimension increases to 20, robustness of DE decreases as its n_S reduces from 260 to 206. In contrast, robustness of RL slightly increases as its n_S increases from 303 to 308. RL achieves at least 10 successes for any 20-dimensional function while DE does not have any success for 4 functions. RL is also more efficient than DE for 11 functions and DE is more efficient for 1 function.

In the next experiment, the RL method described above remains the same but its crossover operation given by (2) is replaced with a crossover operation similar to DE's crossover:

Table 2. The total number of successes n_S and the average and standard deviation of n_{FE} for 50 cost functions

	PS			DE			RL		
Cost functions	n_S	$ave(n_{FE})$	$std(n_{FE})$	n_S	$ave(n_{FE})$	$std(n_{FE})$	n_S	$ave(n_{FE})$	$std(n_{FE})$
Ackley(2)	**20**	4,246	336.0	**20**	**2,248**	151.7	**20**	2,943	340.4
Alpine(2)	**20**	3,284	456.2	**20**	**2,726**	435.3	**20**	4,216	1,319.0
Aluffi(2)	**20**	1,064	198.7	**20**	824	111.9	**20**	**572**	178.4
Bard(3)	**20**	4,056	811.2	**20**	**3,195**	359.9	**20**	4,248	1,073.1
Beale(2)	**20**	1,774	348.8	**20**	1,220	164.9	**20**	**1,100**	301.4
Becker(2)	**20**	1,526	361.4	**20**	1,914	373.3	**20**	**704**	223.4
Bohachevsky1(2)	**20**	2,512	404.5	**20**	1,640	118.9	**20**	**740**	85.6
Branin(2)	**20**	1,468	256.7	**20**	884	109.3	**20**	**368**	51.3
Brown(2)	19	12,810	3,070.5	0	80,694	42,951.4	**20**	**4,432**	808.0
Camel3(2)	**20**	1,326	241.5	**20**	994	117.0	**20**	**468**	130.5
Camel6(2)	**20**	1,608	267.1	**20**	1,514	179.5	**20**	**852**	178.4
Colville(4)	**20**	**9,784**	5,749.6	**20**	9,804	1,345.3	**20**	17,415	5,134.1
Cubic(2)	**20**	**4,490**	1,507.7	18	4,784	6,113.2	18	5,977	1,365.2
Dejong4(2)	**20**	156	62.1	**20**	242	80.5	**20**	**152**	47.9
Dekker(2)	**20**	3,256	255.4	**20**	2,478	419.0	**20**	**1,284**	165.2
Easom(2)	**20**	1,428	346.3	**20**	**1,244**	126.4	**20**	1,721	457.6
Exponential(2)	**20**	616	180.9	**20**	582	96.7	**20**	**256**	69.2
Freudenstein(2)	**20**	3,068	1,013.0	19	1,990	706.5	**20**	**1,859**	546.8
Gaussian(3)	**20**	**3,309**	516.2	17	5,373	1,099.7	**20**	10,340	3,256.1
Goldstein(2)	**20**	1,952	274.3	**20**	1,310	102.1	**20**	**1,128**	234.1
Griewangk(2)	14	6,552	2,709.1	**20**	**3,408**	388.2	**20**	4,647	1,856.6
Gulf(3)	**20**	4,773	745.5	**20**	**3,447**	1,532.4	**20**	3,676	1,116.0
Hartman3(3)	**20**	2,358	310.9	**20**	1,923	152.6	**20**	**1,620**	275.2
Hartman6(6)	7	17,784	8,309.5	11	19,416	7,513.6	**20**	**8,880**	1,227.5
Helical(3)	**20**	4,299	269.9	**20**	3,855	362.0	**20**	**2,904**	454.2
Hyperellipsoid(2)	**20**	1,326	233.7	**20**	908	99.8	**20**	**280**	45.0
Jennrich(2)	**20**	2,354	341.5	**20**	**1,862**	157.6	**20**	2,204	252.2
Kowalik(4)	13	28,392	37,572.7	3	1,023,708	998,646.2	**16**	**27,363**	20,572.7
Levy1(2)	**20**	1,732	254.7	**20**	1,144	129.4	**20**	**584**	108.8
Matyas(2)	**20**	1,204	228.1	**20**	866	112.6	**20**	**224**	69.2
Miele(4)	**20**	**772**	208.1	**20**	1,264	237.3	**20**	3,286	720.9
MultiGaussian(2)	**18**	**2,232**	1,661.0	16	2,314	1,391.1	15	3,506	1,279.6
Periodic(2)	13	4,754	2,505.2	11	**3,776**	1,407.2	**20**	4,109	2,253.1
Powell(2)	**20**	**7,436**	1,086.1	0	24,020	5,821.4	19	11,749	2,315.6
PowellsQ(4)	**20**	6,168	1,032.7	**20**	4,888	311.4	**20**	**2,576**	777.3
Rastrigin(2)	19	2,908	1,081.6	**20**	2,282	225.0	**20**	**1,981**	453.0
Rosenbrock(2)	**20**	1,704	292.9	**20**	**1,322**	177.2	**20**	2,296	351.7
Schaffer1(2)	9	5,382	1,686.4	15	**5,362**	1,232.7	**17**	7,291	2,547.6
Schaffer2(2)	**20**	8,656	496.5	**20**	**5,534**	281.7	**20**	6,726	459.4
Schwefel12(2)	**20**	2,098	223.8	**20**	1,268	125.9	**20**	**400**	84.1
Schwefel221(2)	**20**	4,158	324.6	**20**	**2,284**	146.8	**20**	4,016	817.1
Schwefel222(2)	**20**	3,440	344.1	**20**	**1,956**	129.7	**20**	2,829	588.0
Shekel5(4)	12	9,940	4,718.2	**20**	6,876	602.3	**20**	**6,689**	2,261.1
Shekel7(4)	13	9,308	4,770.6	**20**	**6,344**	534.5	**20**	6,384	1,240.2
Shekel10(4)	15	8,996	4,700.8	**20**	6,536	520.5	**20**	**6,439**	1,327.8
Sphere(2)	**20**	1,238	189.2	**20**	884	104.6	**20**	**288**	44.2
Step(2)	**20**	564	207.2	**20**	540	74.0	**20**	**212**	79.0
Sum.Diff.Powers(2)	**20**	456	224.2	**20**	484	113.1	**20**	**204**	71.0
Wood(4)	**20**	**9,784**	5,749.6	**20**	9,804	1,345.3	**20**	17,399	4,776.7
Zakharov(2)	**20**	1,498	309.4	**20**	1,008	115.1	**20**	**512**	89.5
SUM	**932**	**225,999**	**99,445.6**	**910**	**1,274,943**	**1,079,153.5**	**985**	**202,049**	**64,479.6**

$\hat{x}_i^k = x_i^k$ if $r^k > \rho_{CR}$, otherwise $\hat{x}_i^k = x_*^k$. Notice from Table 5 that the RL crossover slightly increases the robustness as there are 13, 4, and 5 more number of successes for the cost functions in Tables 2, 3, and 4 respectively. The RL crossover also achieves smaller number of function evaluations than the DE crossover over 39, 11, and 11 functions in Tables 2, 3, and 4 respectively.

In order to determine suitable values for the stopping criterion parameters ϵ_D and n_{D_max} for the RL method, 12 cost functions are selected and minimized for 20 times. These cost functions are Ackley(2), Alpine(2), Bard(3), Branin(2), Camel6(2), Exponential(2), Goldstein(2), Rosenbrock(2), Schwefel12(2), Schwefel221(2), Schwefel222(2), and Sphere(2). Let f_{error} be the difference between the function value f_{best} at the converged point x_{best} and the function value f_{opt} at the global minimizer. Note that $f_{best} \leqslant f(x_i)$ for all $x_i \in X$. Table 6 shows the average f_{error} and the associated average number of function evaluations for different ϵ_D and n_{D_max} values, where $n_{D\_max} = \rho_{ratio} \cdot n_P$, ρ_{ratio} is a number satisfying $0 < \rho_{ratio} \leqslant 1$, and $n_P = 20n$. Notice that accuracy of the solution and the number of function evaluations increase for increasing ρ_{ratio} and decreasing ϵ_D values. For instance average error is less than 10^{-4} for $\epsilon_D \leqslant 10^{-3}$. Similarly error is smaller than 10^{-2} for all $\rho_{ratio} \geqslant 0.3$. Even

Table 3. The total number of successes n_S and the average and standard deviation of n_{FE} for 10-dimensional cost functions

	DE			RL		
Cost functions	n_S	$ave(n_{FE})$	$std(n_{FE})$	n_S	$ave(n_{FE})$	$std(n_{FE})$
Ackley(10)	**20**	70,560	1,275.9	**20**	**50,390**	2,262.8
Alpine(10)	0	58,960	15,781.5	**20**	53,534	23,125.7
Dejong4(10)	**20**	15,920	627.1	**20**	**4,560**	907.2
Exponential(10)	**20**	24,800	922.2	**20**	**12,637**	2,362.2
Griewangk(10)	0	39,920	7,887.5	**6**	73,447	18,076.3
Hyperellipsoid(10)	**20**	37,330	705.7	**20**	**23,157**	2,776.9
Levy1(10)	**20**	33,830	808.6	**20**	**20,333**	1,907.5
Rastrigin(10)	0	24,310	11,018.2	**18**	82,657	20,137.1
Rosenbrock(10)	**20**	**104,450**	5,464.9	**20**	156,173	10,318.1
Schwefel12(10)	**20**	75,600	1,711.9	**20**	**47,202**	6,695.7
Schwefel221(10)	**20**	98,520	2,275.2	**20**	**81,668**	5,401.8
Schwefel222(10)	**20**	72,580	1,511.9	**20**	**45,821**	1,925.8
Sphere(10)	**20**	33,970	1,001.6	**20**	**20,415**	2,549.3
Step(10)	**20**	22,380	979.6	19	**12,601**	4,856.3
Sum.Diff.Powers(10)	**20**	10,520	700.1	**20**	**2,580**	354.8
Zakharov(10)	**20**	62,170	1,969.5	**20**	**40,747**	4,590.1
SUM	**260**	**785,820**	**54,641.2**	**303**	**727,922**	**108,247.8**

Table 4. The total number of successes n_S and the average and standard deviation of n_{FE} for 20-dimensional cost functions

	DE			RL		
Cost functions	n_S	$ave(n_{FE})$	$std(n_{FE})$	n_S	$ave(n_{FE})$	$std(n_{FE})$
Ackley(20)	**20**	408,520	4,276.9	**20**	**194,309**	9,170.9
Alpine(20)	0	83,840	35,897.0	**20**	148,028	17,199.0
Dejong4(20)	**20**	114,460	3,998.5	**20**	**21,688**	6,813.2
Exponential(20)	**20**	154,660	3,339.8	**20**	**56,084**	5,116.1
Griewangk(20)	2	217,880	72,470.6	**10**	**188,474**	29,020.3
Hyperellipsoid(20)	**20**	234,380	3,562.3	**20**	**109,935**	7,197.8
Levy1(20)	**20**	207,060	5,025.0	**20**	**88,836**	5,976.9
Rastrigin(20)	0	62,660	29,701.5	**19**	174,422	16,419.2
Rosenbrock(20)	**18**	**657,900**	47,238.3	**20**	996,399	65,248.7
Schwefel12(20)	0	199,260	160,923.4	**20**	331,934	27,746.1
Schwefel221(20)	13	668,840	255,689.8	**20**	**583,869**	76,594.6
Schwefel222(20)	13	332,200	208,154.8	**20**	**162,684**	4,349.8
Sphere(20)	**20**	204,660	4,625.4	**20**	**93,155**	6,265.3
Step(20)	**20**	140,740	5,455.7	19	**61,721**	14,401.0
Sum.Diff.Powers(20)	**20**	59,760	3,335.1	**20**	**5,120**	729.5
Zakharov(20)	0	28,860	17,074.1	**20**	256,098	17,472.3
SUM	**206**	**3,775,680**	**860,768.2**	**308**	**3,472,756**	**309,720.8**

Table 5. Results for the RL method with the RL crossover given by (2) and with the DE's crossover operator. Last column specifies the number of cost functions for which a method requires less number of function evaluations than the other method.

Cost	n_S		$ave(n_{FE})$		n_{func}	
functions in	**DE**	**RL**	**DE**	**RL**	**DE**	**RL**
Table2	972	985	243,356	202,049	11	39
Table3	299	303	827,687	727,922	4	11
Table4	303	308	3,384,473	3,472,756	5	11

Table 6. The average number of function evaluations $ave(n_{FE})$ and average error in function values $ave(f_{error})$ after RL finds at least $n_{D\_max} = \rho_{ratio} \cdot n_P$ quadratics satisfying $|a| < \epsilon_D$ and $|b| < \epsilon_D$.

		Upper limits (ϵ_D) on $\|a\|$ and $\|b\|$					
	ρ_{ratio}	**1**	$\mathbf{10^{-1}}$	$\mathbf{10^{-2}}$	$\mathbf{10^{-3}}$	$\mathbf{10^{-4}}$	$\mathbf{10^{-5}}$
	0.1	502	871	1,216	1,680	2,295	2,855
	0.2	670	1,028	1,391	1,991	2,617	3,285
	0.3	782	1,170	1,582	2,329	3,034	3,751
$ave(n_{FE})$	**0.4**	912	1,338	1,820	2,777	3,561	4,232
	0.5	1,064	1,491	2,125	3,229	3,926	4,529
	0.6	1,512	2,014	2,912	4,037	4,633	5,013
	0.7	2,066	2,628	3,830	4,768	5,208	5,431
	0.1	3.07e-01	4.52e-03	7.75e-04	9.57e-05	2.50e-06	3.44e-07
	0.2	3.51e-02	2.32e-03	3.58e-04	1.33e-05	9.29e-07	1.73e-07
	0.3	8.13e-03	1.32e-03	1.67e-04	4.96e-06	3.47e-07	1.20e-07
$ave(f_{error})$	**0.4**	5.32e-03	8.08e-04	1.20e-04	1.98e-06	1.65e-07	9.51e-08
	0.5	3.97e-03	6.11e-04	4.77e-05	8.08e-07	1.21e-07	9.35e-08
	0.6	2.06e-03	2.44e-04	9.90e-06	3.10e-07	9.61e-08	9.27e-08
	0.7	1.35e-03	1.00e-04	1.81e-06	1.18e-07	9.35e-08	9.26e-08

though accuracy figures may vary largely for different cost functions, $\epsilon_D = 10^{-4}$ and $\rho_{ratio} = 0.2$ might be good initial choices for achieving smaller errors within smaller number of function evaluations. In order to achieve more accurate results, one needs to further increase ρ_{ratio} and/or decrease ϵ_D.

4. Conclusion

The paper proposes an evolutionary optimization method with new mutation and crossover operations. The mutation operation uses quadratic interpolating functions over randomly selected lines in the cost function domain. The crossover operation replaces entries of a mutated vector with the corresponding entries of either of two parent vectors. Its stopping criterion is satisfied if a pre-specified number of quadratics is almost degenerate. Numerical performance evaluation over many types of cost functions demonstrates that proposed method yields promising results compared to the well-known DE and PS algorithms. It is also shown that this method scales to minimize higher dimensional functions successfully.

Acknowledgments

The author thanks largely to Paul Kienzle at the National Institute of Standards and Technology, Brent Fultz and Mike McKerns at the California Institute of Technology, and Robert M. Briber at the University of Maryland for their encouragement and useful comments.

References

[1] Törn, A., *Global optimization*. Springer-Verlag (1989).

[2] Price, K., Storn, R.M., Lampinen, J.A., *Differential evolution: a practical approach to global optimization*. Springer-Verlag, Berlin (2005).

[3] Tu, T.V., Sano, K., Genetic algorithm for optimization in adaptive bus signal priority control. *An International Journal of Optimization and Control: Theories and Applications (IJOCTA)*, 3(1), 35–43 (2012).

[4] Luenberger, D.G., Ye, Y., *Linear and nonlinear programming*. Springer, New York (2008).

[5] Dennis, J.E., Schnabel, R.B., *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM (1987).

[6] More, J.J., Thuente, D.J., Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3), 286–307 (1994).

[7] Mohan, C., Shanker, K., A controlled random search technique for global optimization using quadratic approximation. *Asia-Pacific Journal of Operational Research*, 11(1), 93–101 (1994).

[8] Pant, M., Thangaraj, R., Singh, V.P., A new differential evolution algorithm for solving global optimization problems. International Conference on Advanced Computer Control (ICACC'09), 388-392 (2009)

[9] Storn, R., Price, K., Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341-359 (1997).

[10] Montgomery, J., Chen, S., An analysis of the operation of differential evolution at high and low crossover rates. IEEE Congress on Evolutionary Computation (CEC'2010), 1-8 (2010).

[11] Onwubolu, G., Davendra, D., Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2), 674–692 (2006).

[12] Şahin, İ., Random Lines: a novel population set-based evolutionary global optimization algorithm. Genetic Programming vol. 6621, 97-107, Springer-Verlag, Berlin (2011).

[13] Zielinski, K., Weitkemper, P., Laur, R., Kammeyer, K.-D., Examination of stopping criteria for differential evolution based on a power allocation problem. 10th International Conference on Optimization of Electrical and Electronic Equipment, Brasov, Romania (2006).

[14] Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A., Opposition-based Differential Evolution. *IEEE Transactions on Evolutionary Computation*, 12(1), 64-79 (2008).

[15] More, J.J., Garbow, B.S., Hillstrom, K.E., Testing unconstrained optimization software. *Acm Transactions on Mathematical Software*, 7(1), 17-41 (1981).

[16] Aluffipentini, F., Parisi, V., Zirilli, F., Global optimization and stochastic differential-equations. *Journal of Optimization Theory and Applications*, 47(1), 1-16 (1985).

[17] 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(4), 635-672 (2005).

[18] Pierre, D.A., *Optimization theory with applications*. Courier Dover Publications, Mineola(1987).

[19] Clerc, M., Kennedy, J., The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1), 58-73 (2002).

[20] Ali, M.M., Kaelo, P., Improved particle swarm algorithms for global optimization. *Applied Mathematics and Computation*, 196(2), 578-593 (2008).

[21] 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).

[22] Yao, X., Liu, Y., Lin, G., Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2), 82 -102 (1999).

İsmet Şahin *received the B.S. degree in electrical and electronics engineering from Cukurova University, Adana, Turkey, in 1996, the M.S. degree in electrical and computer engineering from the University of Florida, Gainesville, in 2000, and the Ph.D. degree in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, in 2006. Between July 2003 and July 2004, he was a Research Engineer in signal processing for laser- and radar-based security systems at Robert Bosch Corporation Research and Technology Center, North America, Pittsburgh, PA. He joined the Communication Systems Division, Compunetix, Inc., Pittsburgh, where he was a Software Engineer until 2007. From 2007 to 2009, he was a Research Associate with the Department of Biomedical Informatics, University of Pittsburgh. He is currently a Research Scientist at the Mathematical and Computational Science Division of the National Institute of Standards and Technology, Gaithersburg, MD, where he is engaged in developing deterministic and stochastic optimization algorithms for biomedical signal and image processing problems.*