Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 53 (2007) 1605–1614
An International Journal
computers & mathematics with applications
www.elsevier.com/locate/camwa

A novel population initialization method for accelerating evolutionary algorithms

Shahryar Rahnamayan, Hamid R. Tizhoosh*, Magdy M.A. Salama

Medical Instrument Analysis and Machine Intelligence Research Group, Faculty of Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

Received 28 March 2006; received in revised form 5 July 2006; accepted 12 July 2006

Abstract

Population initialization is a crucial task in evolutionary algorithms because it can affect the convergence speed and also the quality of the final solution. If no information about the solution is available, then random initialization is the most commonly used method to generate candidate solutions (initial population). This paper proposes a novel initialization approach which employs opposition-based learning to generate initial population. The conducted experiments over a comprehensive set of benchmark functions demonstrate that replacing the random initialization with the opposition-based population initialization can accelerate convergence speed.

Keywords: Evolutionary algorithms; Global optimization; Random initialization; Differential evolution (DE); Opposition-based learning

1. Introduction

Evolutionary algorithms (EAs) have been introduced to solve nonlinear complex optimization problems [1–3]. Some well-established and commonly used EAs are Genetic Algorithms (GA) [4] and Differential Evolution (DE) [6,5]. Each of these method has its own characteristics, strengths and weaknesses; but long computational time is a common drawback for all population-based schemes, specially when the solution space is hard to explore. Many efforts have been already done to accelerate convergence of these methods. Most of these works are focused on introducing or improving crossover and mutation operators, selection mechanisms, and adaptive controlling of parameter settings. Although, population initialization can affect the convergence speed and also the quality of the final solution, there is only a little reported research in this field. Maaranen et al. introduced quasi-random population initialization for genetic algorithms [7]. The presented results showed that their proposed initialization method can improve the quality of final solutions with no noteworthy improvement for convergence speed. On the other hand, generation of quasi-random sequences is more difficult and their advantage vanishes for higher dimensional problems (theoretically for dimensions larger than 12) [8].

* Corresponding author. Tel.: +1 519 888 4567x36751; fax: +1 519 746 4791.
E-mail addresses: shahryar@pami.uwaterloo.ca (S. Rahnamayan), tizhoosh@uwaterloo.ca (H.R. Tizhoosh), msalama@hivolt.uwaterloo.ca (M.M.A. Salama).

0898-1221/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2006.07.013

This paper presents a novel scheme for population initialization by applying opposition-based learning [9] to make EAs faster. The main idea behind the opposition-based learning is considering the estimate and opposite estimate (guess and opposite guess) at the same time in order to achieve a better approximation for current candidate solution. Unlike quasi-random number generation, the calculating of opposite candidates is not difficult or time consuming. Further, there is no dimensionality limitations. The idea is applicable to a wide range of optimization methods. Although the proposed scheme is embedded in a classical DE, it is general enough to be applied to all evolutionary algorithms. A test suite with 34 well-known benchmark functions has been utilized in the conducted experiments. Experimental results show efficiency of the proposed approach to accelerate the convergence speed.

Organization of the rest of the paper is as follows: in Section 2, the concept of opposition-based learning is briefly explained. The classical DE is briefly reviewed in Section 3. The proposed algorithm is presented in Section 4. Experimental results are given in Section 5. The results are analysed in Section 6. Finally, the work is concluded in Section 7. All benchmark functions are listed in Appendix.

2. Opposition-based learning

Generally speaking, evolutionary optimization algorithms start with some initial solutions (initial population) and try to improve performance toward some optimal solutions. The process of searching terminates when predefined criteria are satisfied. In the absence of a priori information about the solution, we always start with a *random guess*. Obviously, the computation time is directly related to distance of the guess from optimal solution. We can improve our chance to start with a closer (fitter) solution by checking the opposite solution simultaneously. By doing this, the closer one to solution (say guess or *opposite guess*) can be chosen as initial solution. In fact, according to probability theory, in 50% of cases the guess is farther from solution than the opposite guess; for these cases starting with opposite guess can accelerate convergence.

The concept of opposition-based learning was introduced by Tizhoosh [9] and its applications were introduced in [9–11]. Before concentrating on opposition-based optimization, we need to define opposite numbers [9]:

Definition. Let x be a real number in an interval $[a, b]$ ($x \in [a, b]$); the opposite number $\breve{x}$ is defined by

$$\breve{x} = a + b - x. \tag{1}$$

For $a = -b$ we receive $\breve{x} = -x$, and for $a = 0$ and $b = 1$ we receive $\breve{x} = 1 - x$. Similarly, this definition can be extended to higher dimensions as follows [9]:

Definition. Let $P(x_1, x_2, \ldots, x_n)$ be a point in n-dimensional space, where $x_1, x_2, \ldots, x_n \in R$ and $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, \ldots, n\}$. The opposite point of P is defined by $\breve{P}(\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n)$ where:

$$\breve{x}_i = a_i + b_i - x_i. \tag{2}$$

Theorem (*Uniqueness*). *Every point $P(x_1, x_2, \ldots, x_n)$ in the n-dimensional space of real numbers with $x_i \in [a_i, b_i]$ has a unique opposite point $\breve{P}(\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n)$ defined by $\breve{x}_i = a_i + b_i - x_i$, $i = 1, 2, 3, \ldots, n$.*

Proof. Consider the two space corners $A(a_1, a_2, \ldots, a_n)$ and $B(b_1, b_2, \ldots, b_n)$. According to the opposite point definition we have $\|P, A\| = \|\breve{P}, B\|$ or $\|\breve{P}, A\| = \|P, B\|$. Now, assume that a second point $Q(x'_1, x'_2, \ldots, x'_n)$ is also opposite of P. Then we should have $\|P, A\| = \|Q, B\|$ or $\|Q, A\| = \|P, B\|$. This, however, means $Q = \breve{P}$. Hence, $\breve{P}$ is unique. □

Now, by employing opposite point definition, the opposition-based optimization can be defined as follows:

Opposition-Based Optimization. Let $P(x_1, x_2, \ldots, x_n)$, a point in an n-dimensional space with $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, \ldots, n\}$, be a candidate solution. Assume $f(x)$ is a fitness function which is used to measure candidate optimality. According to opposite point definition, the point $\breve{P}(\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_n)$ is the opposite of $P(x_1, x_2, \ldots, x_n)$. Now, if $f(\breve{P}) \geq f(P)$, then point P can be replaced by $\breve{P}$; otherwise we continue with P. Hence, the point and its opposite point are evaluated simultaneously to continue with the fitter one.

Before introducing the opposition-based population initialization algorithm, the classical differential evolution (DE) algorithm is briefly reviewed in the following section.

3. The classical DE

Differential Evolution (DE) is a population-based direct search method [12]. According to comparative studies, DE outperforms many other evolutionary algorithms over both benchmark functions and also real-world optimization problems. Like other evolutionary algorithms, it starts with an initial population vector, which is generated randomly. Let assume that $X_{i,G}$, $(i = 1, 2, \ldots, n)$ are n N_v-dimensional parameter vectors of generation G (n is a constant representing the population size) [13]. In order to generate a new population of vectors, for each target vector in the population, three vectors are randomly selected and the weighted difference of two of them is added to the third one.

For classical DE, the procedure is as follows [13]:

(a) *Creating difference offspring*: For each vector i from generation G a mutant vector $V_{i,G+1}$ is defined by

$$V_{i,G+1} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G}), \tag{3}$$

where $i = \{1, 2, \ldots, n\}$ and r_1, r_2, and r_3 are mutually different random integer indices selected from $\{1, 2, \ldots, n\}$. Further, i, r_1, r_2, and r_3 are different so $n \geq 4$. $F \in [0, 2]$ is a real constant which determines amplification of the added differential vector of $(X_{r2,G} - X_{r3,G})$. Larger values for F result higher diversity in the generated population and the lower values faster convergence.

DE utilizes crossover operation to increase diversity of the population. It defines following trial vector:

$$U_{i,G+1} = (U_{1i,G+1}, U_{2i,G+1}, \ldots, U_{N_v i,G+1}), \tag{4}$$

where $j = 1, 2, \ldots, N_v$ and

$$U_{ji,G+1} = \begin{cases} V_{ji,G+1} & \text{if } rand_j(0, 1) \leq CR, \\ X_{ji,G} & \text{otherwise.} \end{cases} \tag{5}$$

$CR \in (0, 1)$ is the predefined crossover constant and $rand_j(0, 1) \in [0, 1]$ is jth evaluation of uniform random generator. Most popular value for CR is in the range of (0.4, 1) [16].

(b) *Fitness evaluation of trial vector*

(c) *Selection*: The approach must decide which vector, $U_{i,G+1}$ or $X_{i,G}$, should be a member of new generation, $G + 1$. Vector with the fitter value is chosen.

There are other variants of DE [5] but to maintain a general comparison, the classical version of DE has been selected to demonstrate the convergence improvement by the opposition-based population initialization.

4. Proposed algorithm

According to our review of optimization literature, in the absence of a priori information about solution, random number generation is the most commonly used method for almost all EAs to create initial population. But as mentioned in Section 2, the concept of opposition-based optimization can help us to obtain fitter starting candidate solutions even when there is no a priori knowledge.

We propose the following opposition-based population initialization algorithm which can be used instead of a pure random initialization:

(1) Generating uniformly distributed random population, $P(n)$; n is the population size;
(2) Calculating opposite population $OP(n)$; the kth corresponding opposite individual for $OP(n)$ is calculated by

$$OP_{k,j} = a_j + b_j - P_{k,j}, \quad k = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, N_v, \tag{6}$$

where N_v is the number of variables (problem dimension); a_j and b_j denote the interval boundaries of jth variable $(x_j \in [a_j, b_j])$;
(3) Selecting n fittest individuals from set the $\{P(n) \cup OP(n)\}$ as initial population.

The flowchart of random population initialization and above mentioned opposition-based population initialization are shown in Fig. 1.

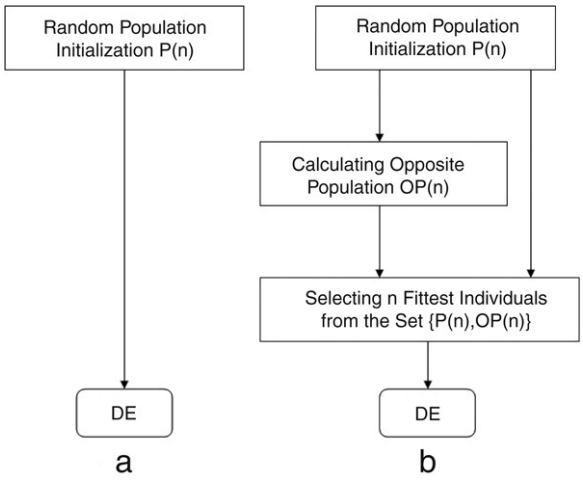


Fig. 1. DE with (a) random population initialization (DE_r) and (b) opposition-based population initialization (DE_o).

In all conducted experiments in the next section, the proposed opposition-based population initialization algorithm is embedded in the DE to increase convergence speed. In fact, the uniform random population initialization is replaced with opposition-based population initialization. By this way, we try to start with better (fitter) candidates instead of starting with pure random guesses.

5. Experimental results

5.1. Numerical benchmark functions

In order to compare convergence speed of DE with random population initialization (DE_r) and DE with opposition-based population initialization (DE_o), a test set with 34 numerical benchmark functions is employed. All selected functions are well-known in the global optimization literature [13,18,21]. The test set includes unimodal as well as highly multimodal minimization problems. The dimensionality of problems varies from 2 to 100 to cover a wide range of problem complexity. The definition, the range of search space, and also the global minimum of each function are given in Appendix.

5.2. DE settings

For all conducted experiments, three parameters of DE, namely, population size (n), scaling factor (F), and crossover probability constant (CR) are set to 100, 0.5, and 0.9, respectively. These values have been chosen according to reported setting in the literature (e.g. [21]). In order to have a fair comparison, these settings are kept the same for two competing algorithms over all benchmark functions during the simulations.

5.3. Comparison strategy

We compare the convergence speed of DE_r and DE_o by measuring the number of function calls (NFC) which is the most commonly used metric in the literature [18,15]. Smaller NFC means higher convergence speed (for more theoretical information about the convergence properties of evolutionary algorithms the reader is referred to [14]). The termination criterion is to reduce the best value found by algorithm to a value smaller than the value-to-reach (VTR) before meeting maximum number of function calls (MAX_{NFC}). The theoretical optimum value of all benchmark functions has been set to zero by shifting them, if needed. The MAX_{NFC} is set to 10^6 for all experiments. The VTR is set to 10^{-1} for all benchmark functions excepts for $\{f_9, f_{12}, f_{14}, f_{16}, f_{20}, f_{32}\}$: 10^{-7}, f_{30}: 10^{-14}, and f_{25}: 10^{-3}. In order to minimize the effect of stochastic nature of algorithms on measured metric, the reported number of function calls (NFC) for each algorithm is the average value over 100 runs for each test function (this number has commonly been set to a value between 30 and 100 for many studies [15–23]).

Table 1
Comparison of convergence speed (NFC) for DE with random population initialization (DE_r) and with opposition-based population initialization (DE_o) on 34 benchmark functions

Function	D	NFC(DE_r)	NFC(DE_o)
f_1	30	28 151	**26 983**
f_2	30	37 555	**36 708**
f_3	20	78 081	**76 035**
f_4	30	295 955	**295 689**
f_5	10	383 377	**360 994**
f_6	30	54 494	**53 311**
f_7	30	6101	**1689**
f_8	30	52 690	**51 619**
f_9	2	3831	**3744**
f_{10}	4	**7918**	7959
f_{11}	2	189 316	**125 758**
f_{12}	3	3650	**3511**
f_{13}	6	3176	**3171**
f_{14}	2	5290	**5155**
f_{15}	30	44 092	**41 843**
f_{16}	100	3132	**3050**
f_{17}	4	**35 370**	39 550
f_{18}	10	221 560	**196 980**
f_{19}	30	201 015	**196 567**
f_{20}	2	**4918**	4995
f_{21}	30	66 192	**63 763**
f_{22}	30	197 093	**148 739**
f_{23}	30	42 475	**41 533**
f_{24}	30	25 912	**24 248**
f_{25}	4	4181	**3580**
f_{26}	4	**6369**	6433
f_{27}	4	4611	**4475**
f_{28}	4	4452	**4380**
f_{29}	2	3906	**3841**
f_{30}	2	2442	**2431**
f_{31}	30	55 059	**52 833**
f_{32}	2	7398	**7303**
f_{33}	5	208 804	**150 639**
f_{34}	5	32 479	**31 286**
$\sum_{i=1}^{34} \text{NFC}_i =$		2321 045	**2080 795**

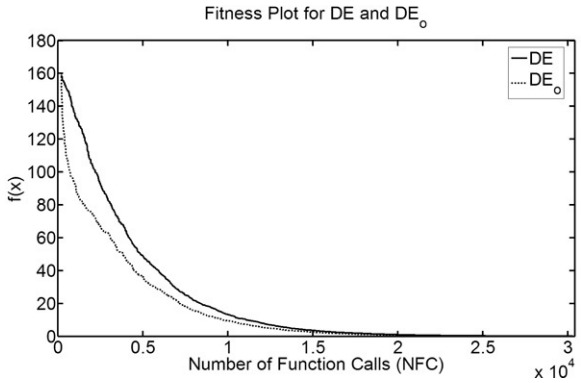
The better result for each case is highlighted in boldface. The reported numbers for each benchmark function are the average value of NFCs over 100 runs. D refers to the dimensionality of the problem.

5.4. Simulation results

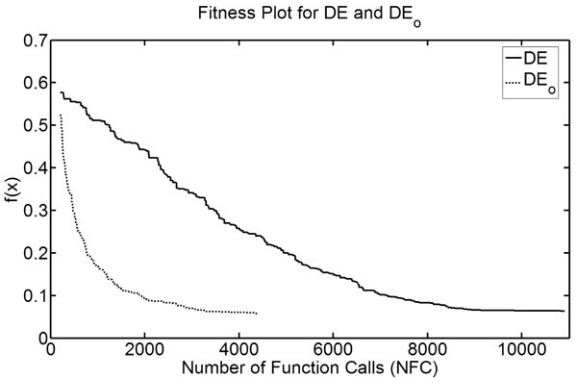
Numerical results are summarized in Table 1. It shows the convergence speed (number of function calls, NFC) of DE with random population initialization (DE_r) and DE with opposition-based population initialization (DE_o) on 34 benchmark functions. The better result for each case is highlighted in boldface. As seen, DE_o outperforms DE_r on 30 (out of 34) functions. It means applying opposition-based population initialization, instead of using pure random population initialization, speeds up the DE. Some examples for performance comparison are presented in Fig. 2. The graphs ($f(x)$ vs. NFC) show the progress toward optimum value (minimization, $f(x) = 0$). Experiments have been repeated 100 times to plot the average values. As seen in Table 1 and Fig. 2, starting with better (fitter) individuals as an initial population has not the same speedup effect on the convergence of functions with different characteristics and complexities.

6. Discussion

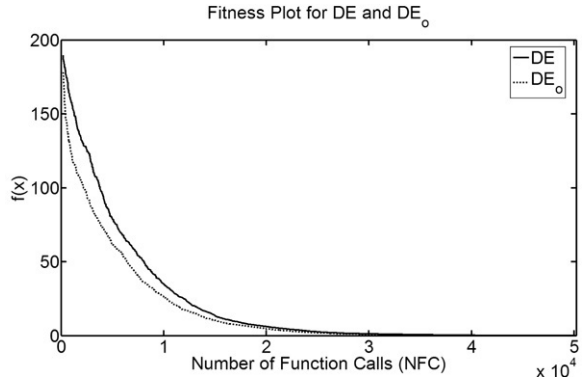
As shown in Table 1, the DE_o outperforms DE_r on 30 (out of 34) benchmark functions with respect to number of function calls. Just on four functions DE_r shows better result than DE_o. These functions are f_{10} (Colville function),



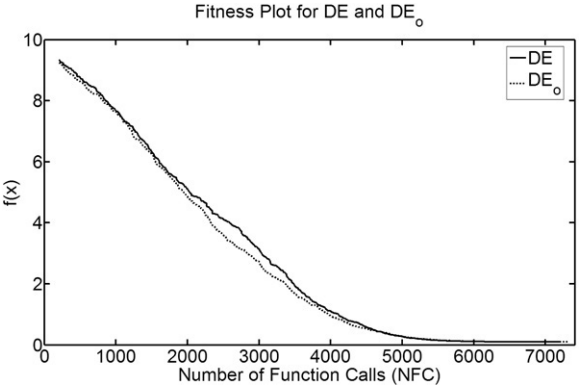
(a) f_1: Sphere model.



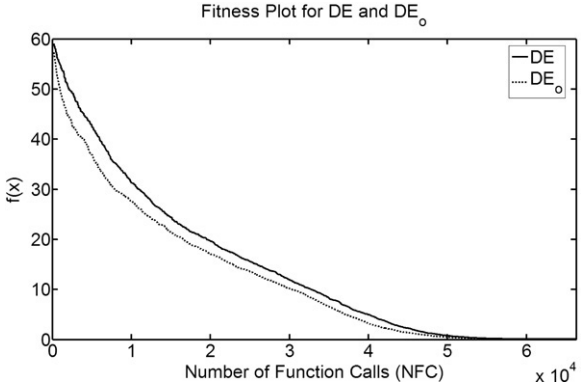
(b) f_7: Sum of different power.



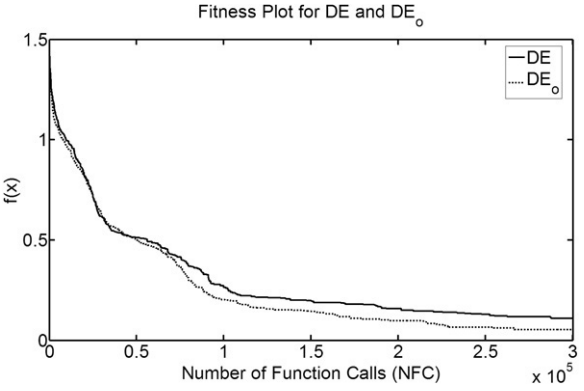
(c) f_{15}: Levy function.



(d) f_{27}: Shekel's family ($m = 7$).



(e) f_{31}: Alpine function.



(f) f_{33}: Pathological function.

Fig. 2. Examples for performance comparison between the DE with random population initialization and DE with opposition-based population initialization (DE_o) for minimization problems $f(x) = 0$. Experiments have been repeated 100 times to plot the average values.

f_{17} (Perm function), f_{20} (Branins's function), and f_{26} (Shekel's function) with 0.5%, 10.6%, 1.5%, and 1% smaller function calls, respectively. For functions f_{10}, f_{20}, and f_{26} the results are close because the difference ratio of function calls is less than 1.5%. Therefore, we can say the DE_r only over function f_{17} (Perm function), with 10.6% smaller function calls, surpasses DE_o outstandingly. For this function the global minimum is located at $\vec{x} = (1, 2, 3, 4)$ for search space of $-4 \leq x_i \leq 4$ where $i \in \{1, 2, 3, 4\}$. As seen, variables of optimal solution (1,2,3,4) are linearly spread over the search space ($x_4 = x_3 + 1$, $x_3 = x_2 + 1$, $x_2 = x_1 + 1$, so $x_i = x_{i-1} + 1$). So, for this special case

Table 2
Comparison of the overall acceleration rate ($\overline{AR}$) for DE_r and DE_o by partitioning the test suite in low ($D \leq 10$) and high ($D > 10$) dimensional functions

Group	N	N_r	N_o	$\sum_{i=1}^{N} \text{NFC}(\text{DE}_r)_i$	$\sum_{i=1}^{N} \text{NFC}(\text{DE}_o)_i$	$\overline{AR}$
$D \leq 10$	19	4	15	1133 048	966 185	14.7%
$D > 10$	15	0	15	1187 997	1114 610	6.2%

N indicates the number of functions in each group. N_r and N_o denote the number of functions which DE_r outperforms DE_o ($\text{NFC}(\text{DE}_r) < \text{NFC}(\text{DE}_o)$) and vice versa, respectively.

opposition-based initialization does not work properly, in fact, it has low chance of introducing a better candidate because by calculating the opposite guess some variables can be improved but, at the same time, others get worsen (because of linear spreading of variables of optimal solution over the search space).

As shown at the bottom of Table 1, for solving 34 problems the DE_r needs a total number of function calls of 2755883 but its competitor, DE_o, performs it with 2476078 function calls which means 10.35% overall reduction in NFCs.

Considering a set of N test functions, the overall acceleration rate $\overline{AR}$ can be calculated:

$$\overline{AR} = \left(1 - \frac{\sum_{i=1}^{N} \text{NFC}(\text{DE}_o)_i}{\sum_{i=1}^{N} \text{NFC}(\text{DE}_r)_i} \right) \times 100\%. \tag{7}$$

In order to investigate the overall acceleration rate $\overline{AR}$ for low and high dimensional problems, a current test suite has been partitioned in two groups, one with 19 problems and dimensionality of $D \leq 10$ and one with 15 problems with dimensionality of $D > 10$. Results for this comparison are given in Table 2. As it can be seen, the overall acceleration rate for the first group with $D \leq 10$ is higher than the second group with $D > 10$ (14.7% vs. 6.2%). On the other hand, for the first group, the DE_o outperforms DE_r in 79% of cases (15 out of 19 functions), but for the second group this number is 100% (15 out of 15). This means that even though the acceleration rate for higher diemensions is not as high as for lower diemnsions, DE_o is always faster than DE_r for more complex problems. By this way, the results confirm the *no-free-lunch theorem for optimization* [24,25]. This theorem states that "A general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration [26]."

7. Conclusions

The proposed approach employs opposition-based optimization for population initialization. In order to investigate the performance of the proposed algorithm, differential evolution (DE), an efficient and robust optimization method, has been utilized. A set of test functions including unimodal and multimodal benchmark functions is employed for experimental verification. The results demonstrate that the opposition-based population initialization makes convergence speed on average 10% faster; as mentioned before, the large portion of this acceleration comes from low dimensional functions. On the other hand, the DE_r outperforms DE_o just over four low dimensional functions. The proposed algorithm showed that, it is possible to start with better/fitter population even when there is no a priori information about the solution. The main idea is general and applicable to other population-based optimization algorithms such as genetic algorithms, which form our future work directions.

Acknowledgement

The authors would like to thank Erik Jonasson (visiting scholar at the University of Waterloo, Canada) for conducting experiments.

Appendix. List of numerical benchmark functions

- Sphere Model
  $f_1(x) = \sum_{i=1}^{n} x_i{}^2$, with $-5.12 \le x_i \le 5.12$, $\min(f_1) = f_1(0, \ldots, 0) = 0$.
- Axis parallel hyper-ellipsoid
  $f_2(x) = \sum_{i=1}^{n} \mathrm{i}x_i{}^2$, with $-5.12 \le x_i \le 5.12$, $\min(f_2) = f_2(0, \ldots, 0) = 0$.
- Schwefel's problem 1.2
  $f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$, with $-65 \le x_i \le 65$, $\min(f_3) = f_3(0, \ldots, 0) = 0$.
- Rosenbrock's valley
  $f_4(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$, with $-2 \le x_i \le 2$, $\min(f_4) = f_4(1, \ldots, 1) = 0$.
- Rastrigin's function
  $f_5(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$, with $-5.12 \le x_i \le 5.12$, $\min(f_5) = f_5(0, \ldots, 0) = 0$.
- Griewangk's function
  $f_6(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$, with $-600 \le x_i \le 600$, $\min(f_6) = f_6(0, \ldots, 0) = 0$.
- Sum of different power
  $f_7(x) = \sum_{i=1}^{n} |x_i|^{(i+1)}$, with $-1 \le x_i \le 1$, $\min(f_7) = f_7(0, \ldots, 0) = 0$.
- Ackley's path function
  $f_8(x) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}\right) + 20 + e$, with $-32 \le x_i \le 32$, $\min(f_8) = f_8(0, \ldots, 0) = 0$.
- Beale function
  $f_9(x) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$, with $-4.5 \le x_i \le 4.5$, $\min(f_9) = f_9(3, 0.5) = 0$.
- Colville function
  $f_{10}(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$, with $-10 \le x_i \le 10$, $\min(f_{10}) = f_{10}(1, 1, 1, 1) = 0$.
- Easom function
  $f_{11}(x) = -\cos(x_1)\cos(x_2)\exp((-(x_1 - \pi)^2 - (x_2 - \pi)^2))$, with $-40 \le x_i \le 40$, $\min(f_{11}) = f_{11}(\pi, \pi) = -1$.
- Hartmann function 1
  $f_{12}(x) = -\sum_{i=1}^{4} \alpha_i \exp(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2)$, with $0 \le x_i \le 1$, $\min(f_{12}) = f_{12}(0.114614, 0.555649, 0.852547) = -3.86278$. The value of α, A, and P are given in [17].
- Hartmann function 2
  $f_{13}(x) = -\sum_{i=1}^{4} \alpha_i \exp(-\sum_{j=1}^{6} B_{ij}(x_j - Q_{ij})^2)$, with $0 \le x_i \le 1$, $\min(f_{13}) = f_{13}(0.20169, 0.150011, 0.476874, 0.275332, 0.311652, 0.6573) = -3.32237$. The value of α, B, and Q are given in [17].
- Six Hump Camel back function
  $f_{14}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$, with $-5 \le x_i \le 5$ $\min(f_{14}) = f_{14}(0.0898, -0.7126)/(-0.0898, 0.7126) = 0$.
- Levy function
  $f_{15}(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n))$, with $-10 \le x_i \le 10$, $\min(f_{15}) = f_{15}(1, \ldots, 1) = 0$.
- Matyas function
  $f_{16}(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$, with $-10 \le x_i \le 10$, $\min(f_{16}) = f_{16}(0, 0) = 0$.
- Perm function
  $f_{17}(x) = \sum_{k=1}^{n} [\sum_{i=1}^{n} (i^k + 0.5)((\frac{1}{i}x_i)^k - 1)]^2$, with $-n \le x_i \le n$, $\min(f_{17}) = f_{17}(1, 2, 3, \ldots, n) = 0$.
- Michalewicz function
  $f_{18}(x) = -\sum_{i=1}^{n} \sin(x_i)(\sin(\mathrm{i}x_i^2/\pi))^{2m}$, with $0 \le x_i \le \pi$, $m = 10$, $\min(f_{18_{(n=2)}}) = -1.8013$, $\min(f_{18_{(n=5)}}) = -4.687658$, $\min(f_{18_{(n=10)}}) = -9.66015$.
- Zakharov function
  $f_{19}(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5\mathrm{i}x_i)^2 + (\sum_{i=1}^{n} 0.5\mathrm{i}x_i)^4$, with $-5 \le x_i \le 10$, $\min(f_{19}) = f_{19}(0, \ldots, 0) = 0$.
- Branins's function
  $f_{20}(x) = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e$, with $-5 \le x_1 \le 10$, $0 \le x_2 \le 15$, where $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $d = 6$, $e = 10$, $f = 1/(8\pi)$, $\min(f_{20}) = f_{20}(-\pi, 12.275)/(-\pi, 2.275)/(9.42478, 2.475) = 0.3979$.

- Schwefel's problem 2.22
  $f_{21}(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$, with $-10 \le x_1 \le 10$, $\min(f_{21}) = f_{21}(0, \ldots, 0) = 0$.
- Schwefel's problem 2.21
  $f_{22}(x) = \max_i\{|x_i|, 1 \le i \le n\}$, with $-100 \le x_1 \le 100$, $\min(f_{22}) = f_{22}(0, \ldots, 0) = 0$.
- Step function
  $f_{23}(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$, with $-100 \le x_1 \le 100$, $\min(f_{23}) = f_{23}(-0.5 \le x_i < 0.5) = 0$.
- Quartic function i.e. noise
  $f_{24}(x) = \sum_{i=1}^{n} \mathrm{i}x_i^4 + random[0, 1)$, with $-1.28 \le x_1 \le 1.28$, $\min(f_{24}) = f_{24}(0, \ldots, 0) = 0$.
- Kowalik's function
  $f_{25}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$, with $-5 \le x_i \le 5$, $\min(f_{25}) = f_{25}(0.19, 0.19, 0.12, 0.14) = 0.0003075$.
  The value of a and b are given in [17].
- Shekel's Family
  $f(x) = -\sum_{i=1}^{m} [(x_i - a_i)(x_i - a_i)^{\mathrm{T}} + c_i]^{-1}$, with $m = 5$, 7, and 10 for $f_{26}(x)$, $f_{27}(x)$, and $f_{28}(x)$, respectively, $0 \le x_j \le 10$, $\min(f_{26}) = f_{26}(4, 4, 4, 4) = -10.2$, $\min(f_{27}) = f_{27}(4, 4, 4, 4) = -10.4$, $\min(f_{28}) = f_{28}(4, 4, 4, 4) = -10.5$. The value of a and c are given in [17].
- Tripod function
  $f_{29}(x) = p(x_2)(1 + p(x_1)) + |(x_1 + 50p(x_2)(1 - 2p(x_1)))| + |(x_2 + 50(1 - 2p(x_2)))|$, with $-100 \le x_i \le 100$, $\min(f_{29}) = f_{29}(0, -50) = 0$, where $p(x) = 1$ for $x \ge 0$ otherwise $p(x) = 0$.
- De Jong's function 4 (no noise)
  $f_{30}(x) = \sum_{i=1}^{n} \mathrm{i}x_i{}^4$, with $-1.28 \le x_i \le 1.28$, $\min(f_{30}) = f_{30}(0, \ldots, 0) = 0$.
- Alpine function
  $f_{31}(x) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|$, with $-10 \le x_i \le 10$, $\min(f_{31}) = f_{31}(0, \ldots, 0) = 0$.
- Schaffer's function 6
  $f_{32}(x) = 0.5 + \frac{\sin^2\sqrt{(x_1^2 + x_2^2)} - 0.5}{1 + 0.01(x_1^2 + x_2^2)^2}$, with $-10 \le x_i \le 10$, $\min(f_{32}) = f_{32}(0, 0) = 0$.
- Pathological function
  $f_{33}(x) = \sum_{i=1}^{n-1} \left(0.5 + \frac{\sin^2\sqrt{(100x_i^2 + x_{i+1}^2)} - 0.5}{1 + 0.001(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2)^2}\right)$, with $-100 \le x_i \le 100$ $\min(f_{33}) = f_{33}(0, \ldots, 0) = 0$.
- Inverted cosine wave function (Masters)
  $f_{34}(x) = -\sum_{i=1}^{n-1} \left(\exp(\frac{-(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1})}{8}) \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}}\right)\right)$, with $-5 \le x_i \le 5$, $\min(f_{34}) = f_{34}(0, \ldots, 0) = -n + 1$.

References

[1] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, USA, ISBN: 0195099710, 1996.

[2] H.-P. Schwefel, Computational Intelligence: Theory and Practice, Springer-Verlag, New York, ISBN: 3540432698, 2003.

[3] T. Bäck, U. Hammel, H.-P. Schwefel, Evolutionary computation: Comments on the history and current state, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 3–17.

[4] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, New York, 1989.

[5] R. Storn, K. Price, Differential evolution — A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.

[6] K. Price, R.M. Storn, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series), first ed., Springer, ISBN: 3540209506, 2005.

[7] H. Maaranen, K. Miettinen, M.M. Mäkelä, A Quasi-Random Initial Population for Genetic Algorithms, in: Computers and Mathematics with Applications, vol. 47, 2004, pp. 1885–1895.

[8] P. Bratley, B.L. Fox, H. Niederreiter, Implementation and tests of low-discrepancy sequences, ACM Transaction on Modeling and Computer Simulation 2 (3) (1992) 195–213.

[9] H.R. Tizhoosh, Opposition-based learning: A new scheme for machine intelligence, in: Int. Conf. on Computational Intelligence for Modelling Control and Automation — CIMCA'2005, Vienna, Austria, 2005, pp. 695–701.

[10] H.R. Tizhoosh, Reinforcement learning based on actions and opposite actions, in: ICGST International Conference on Artificial Intelligence and Machine Learning (AIML-05), Cairo, Egypt, 2005.

[11] H.R. Tizhoosh, Opposition-based reinforcement learning, Journal of Advanced Computational Intelligence and Intelligent Informatics 10 (3) (2006) 578–585.

[12] K.V. Price, An introduction to differential evolution, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London, UK, ISBN: 007-709506-5, 1999, pp. 79–108.
[13] G.C. Onwubolu, B.V. Babu, New Optimization Techniques in Engineering, Springer, Berlin, New York, 2004.
[14] G. Rudolph, Convergence Properties of Evolutionary Algorithms, Verlag Dr. Kovač, Hamburg, 1997.
[15] O. Hrstka, A. Kučerová, Improvement of real coded genetic algorithm based on differential operators preventing premature convergence, Advance in Engineering Software 35 (2004) 237–246.
[16] S. Das, A. Konar, U. Chakraborty, Improved differential evolution algorithms for handling noisy optimization problems, IEEE Congress on Evolutionary Computation Proceedings 2 (2005) 1691–1698.
[17] M. Montaz Ali, C. Khompatraporn, Z.B. Zabinsky, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, Journal of Global Optimization 31 (2005) 635–672.
[18] J. Andre, P. Siarry, T. Dognon, An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization, Advance in Engineering Software 32 (2001) 49–60.
[19] V.K. Koumousis, C.P. Katsaras, A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance, IEEE Transactions on Evolutionary Computation 10 (1) (2006) 19–28.
[20] X. Yao, Y. Liu, G. Liu, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.
[21] J. Vesterstroem, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in: Proceedings of the Congress on Evolutionary Computation (CEC-2004), IEEE Publications, vol. 2, 2004, pp. 1980–1987.
[22] H.-Y. Fan, J. Lampinen, A trigonometric mutation operation to differential evolution, Journal of Global Optimization 27 (1) (2003) 105–129.
[23] J. Brest, S. Greiner, B. Bošković, M. Mernik, V. Žumer, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.
[24] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.
[25] D.H. Wolpert, W.G. Macready, No free lunch theorems for search, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.
[26] Y.C. Ho, D.L. Pepyne, Simple explanation of the no-free-lunch theorem and its implications, Journal of Optimization Theory and Applications 115 (3) (2002) 549–570.