Computer Science • 19(1) 2018 https://doi.org/10.7494/csci.2018.19.1.2522

Radhwan Al-Jawadi
Marcin Studniarski
Aisha Younus

NEW GENETIC ALGORITHM BASED ON DISSIMILARITIES AND SIMILARITIES

Abstract *Optimization is essential for finding suitable answers to real-life problems. In particular, genetic (or more generally, evolutionary) algorithms can provide satisfactory approximate solutions to many problems to which exact analytical results are not accessible. In this paper, we present both the theoretical and experimental results on a new genetic algorithm called Dissimilarity and Similarity of Chromosomes (DSC). This methodology constructs new chromosomes starting with the pairs of existing ones by exploring their dissimilarities and similarities. To demonstrate the performance of the algorithm, it is run on 17 two-dimensional, 1 four-dimensional, and 2 ten-dimensional optimization problems described in the literature, and compared with the well-known GA, CMA-ES, and DE algorithms.The results of the tests show the superiority of our strategy in the majority of cases.*

Keywords genetic algorithm, forma analysis, similarity and dissimilarity of chromosomes, chromosome injection

Citation Computer Science 19(1) 2018: 21–39

1. Introduction

Global optimization algorithms can be divided into two groups: deterministic algorithms and metaheuristic algorithms (see [11]). Metaheuristic methods are helpful for a wide class of optimization problems where deterministic algorithms are not suitable (for example, functions with a large number of local extrema). In particular, metaheuristic algorithms include Ant Colony Optimization (ACO), Genetic Algorithms (GAs), Bees Algorithms (BAs), and other bio-inspired techniques.

Evolutionary Algorithms (EAs) constitute a large class of optimization procedures (including classical GAs) that are inspired by the process of natural evolution. As Eiben and Smith [6] observe, different implementations of EAs (e.g., genetic algorithm, genetic programming, evolutionary strategy) can essentially be summarized by the following steps:

1. initialize a population randomly and evaluate each candidate,
2. select parents,
3. recombine pairs of parents,
4. mutate the resulting offspring,
5. evaluate each new candidate,
6. select individuals for next generation,
7. repeat from Step 2 until a stopping criterion is satisfied.

In this paper, a new evolutionary optimization algorithm is described that explores similarities and dissimilarities in pairs of chromosomes. This procedure divides each population into three unequal parts and then applies new genetic operators to the first two. Our algorithm is called Dissimilarity and Similarity of Chromosomes (DSC), and its purpose is to find optimal solutions in numerical optimization problems.

This concept of dividing a population into parts and then working with schemata and similarity for each part separately is already known in the literature. For example, in the paper by Han et al. [8], the population was divided into three parts based on the fitness of chromosomes (the best, middle, and worst fitness groups); then, the common schema in a population was discovered by using clustering. Later, for the first and third parts of a population, the number of chromosomes that have some similarity with the schema was calculated. The percentage of the positions on which the individual agrees with the schema defines the similarity between an individual and a schema.

A general approach to estimate the expected first hitting time (i.e., the time when the algorithm finds an optimal solution) was proposed by Yu and Zhou [18]. It is based on an analysis of EAs with different configurations. This method works with three mutation operators, a recombination operator, and a time-variant mutation operator. We are planning to examine the possibility of applying a similar theoretical analysis to our DSC algorithm in further research.

The organization of the paper is as follows. In Section 2, we introduce two genetic operators: the similarity operator and dissimilarity operator. These are defined in

terms of a forma analysis of Radcliffe [14]. Section 3 is devoted to a description of the DSC algorithm. Section 4 gives an analysis of the experimental results. Section 5 contains a discussion of the figures. Finally, the conclusions are presented in Section 6.

2. Forma analysis of genetic operators

In this section, we define and analyze the two genetic operators used in our DSC algorithm. We apply the abstract forma analysis presented in [14] so that our definitions may be applied in a more-general setting than only for binary schemata. First, we must review some definitions.

Let $\mathcal{S}$ be a finite search space of some genetic algorithm. Function $\psi : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$ is called an *equivalence relation over* $\mathcal{S}$ if and only if it satisfies the following three conditions:

1. $\forall x \in \mathcal{S} : \psi(x, x) = 1$,
2. $\forall x, y \in \mathcal{S} : \psi(x, y) = 1 \Longrightarrow \psi(y, x) = 1$,
3. $\forall x, y, z \in \mathcal{S} : \psi(x, y) = \psi(y, z) = 1 \Longrightarrow \psi(x, z) = 1$.

We define $\mathbb{E}(\mathcal{S})$ to be the set of all equivalence relations over $\mathcal{S}$. Given two equivalence relations $\psi, \phi \in \mathbb{E}(\mathcal{S})$, we define their *intersection* $\psi \cap \phi \in \mathbb{E}(\mathcal{S})$ by

$$(\psi \cap \phi)(x, y) := \psi(x, y) \wedge \phi(x, y),$$

where $\wedge$ denotes logical conjunction ("and").

For given set $\Psi \subset \mathbb{E}(\mathcal{S})$, we call subset $E \subset \Psi$ a *basis* for Ψ if and only if the following two conditions hold:

1. E *spans* Ψ; that is, each element of Ψ can be constructed by the intersection of some subset of E:

$$\Psi \subset \text{Span } E := \left\{ \epsilon \in \mathbb{E}(\mathcal{S}) : \exists A_\epsilon \subset E \text{ such that } \bigcap A_\epsilon = \epsilon \right\}.$$

2. E is *independent*; that is, no member of E can be constructed by an intersection of other members of E:

$$\forall \epsilon \in E, \nexists A_\epsilon \subset E \backslash \{\epsilon\} : \bigcap A_\epsilon = \epsilon.$$

Given equivalence relation $\psi \in \mathbb{E}(\mathcal{S})$, we define Ξ_ψ to be the set of *formae* (equivalence classes) induced by ψ. Further, given a set of equivalence relations $\Psi \subset \mathbb{E}(\mathcal{S})$, with $\Psi = \{\psi_1, \psi_2, ..., \psi_{|\Psi|}\}$, where $|\Psi|$ is the number of elements of Ψ, we define Ξ_Ψ to be the set of vectors of formae given as the Cartesian product

$$\Xi_\Psi := \Xi_{\psi_1} \times \Xi_{\psi_2} \times \cdots \times \Xi_{\psi_{|\Psi|}}.$$

A set of equivalence relations $\Psi \subset \mathbb{E}(\mathcal{S})$ is said to *cover* $\mathcal{S}$ if and only if, for each pair of different solutions in $\mathcal{S}$, there exists some relation in Ψ under which the pair are not equivalent:

$$\forall x \in \mathcal{S}, \forall y \in \mathcal{S} \backslash \{x\}, \exists \psi \in \Psi : \psi(x, y) = 0.$$

Let E be a basis for a set of equivalence relations $\Psi \subset \mathbb{E}(\mathcal{S})$ that covers $\mathcal{S}$. The members of E are called *basic equivalence relations*, or *genes*. For a given relation $\epsilon \in E$, the members of Ξ_ϵ are called *basic formae*, or *alleles*.

A set of equivalence relations $E \subset \mathbb{E}(\mathcal{S})$ is said to be *orthogonal* if and only if, given any $|E|$ equivalence classes induced by different members of E, their intersection is nonempty:

$$\forall \xi = (\xi_1, \xi_2, ..., \xi_{|E|}) \in \Xi_E : \bigcap_{i=1}^{|E|} \xi_i \neq \emptyset.$$

Let Ξ be a set of formae defined over search space $\mathcal{S}$, and let $L \subset \mathcal{S}$. The *similarity set* of L (defined with respect to Ξ and written $\Sigma(L)$) is the intersection of all those formae to which each solution in L belongs:

$$\Sigma(L) := \begin{cases} \bigcap \{\xi \in \Xi : L \subset \xi\}, & \text{if } \exists \xi \in \Xi : L \subset \xi, \\ \mathcal{S}, & \text{otherwise.} \end{cases}$$

For given set $E = \{\epsilon_1, \epsilon_2, ..., \epsilon_n\} \subset \mathbb{E}(\mathcal{S})$, we define the *genetic representation function* $\rho_E : \mathcal{S} \to \Xi_E$ by

$$\rho_E(x) := ([x]_{\epsilon_1}, [x]_{\epsilon_2}, ..., [x]_{\epsilon_n}),$$

where, for given $\epsilon \in \mathbb{E}(\mathcal{S})$ and $x \in \mathcal{S}$, we denote by $[x]_\epsilon$ the equivalence class of x under ϵ:

$$[x]_\epsilon := \{y \in \mathcal{S} : \epsilon(x, y) = 1\}.$$

Now, we are able to define the two genetic operators used in our DSC algorithm. The first one (the similarity operator) can be defined without any extra assumption on considered set Ψ of equivalence relations. It is, in fact, equal to the *random respectful recombination operator* $\mathbf{R}^3 : \mathcal{S} \times \mathcal{S} \times \mathbb{Z} \to \mathcal{S}$ [14, Def. 59] defined by

$$\mathbf{R}^3(x, y, k) := \sigma_{k'}(x, y),$$

where $\mathbb{Z}$ is the set of integers, $\sigma_i(x, y)$ is the ith element of the similarity set $\Sigma(\{x, y\})$ under some arbitrary enumeration, and $k' := k \pmod{|\Sigma(\{x, y\})|}$. The number k is interpreted as a random control parameter; thus, $\mathbf{R}^3(x, y, k)$ returns a randomly selected element of the similarity set of x and y. The *similarity operator* is defined as

$$\mathrm{sim}(x, y, k) := \mathbf{R}^3(x, y, k).$$

The second operator (the dissimilarity operator) is defined under the additional assumption that orthogonal basis $E = \{\epsilon_1, \epsilon_2, ..., \epsilon_n\}$ for Ψ is given that covers $\mathcal{S}$. Then, it follows from [14, Thm. 25] that ρ_E is a bijection. Moreover, we assume that each basic relation $\epsilon \in E$ divides search space $\mathcal{S}$ into two equivalence classes (i.e., for each

gene, there are only two alleles available). For each $x \in \mathcal{S}$, we can thus define the *complement* of class $[x]_\epsilon$, denoted by $\overline{[x]_\epsilon}$, as follows:

$$\overline{[x]_\epsilon} := \{y \in \mathcal{S} : \epsilon(x, y) = 0\}.$$

Of course, $\overline{[x]_\epsilon}$ is also some equivalence class under ϵ. Since ρ_E is bijective, we can also define the *opposite element* to x, denoted $\overline{x}$, as follows:

$$\overline{x} := \rho_E^{-1}\left(\overline{[x]_{\epsilon_1}}, \overline{[x]_{\epsilon_2}}, ..., \overline{[x]_{\epsilon_n}}\right).$$

Then, we define the *dissimilarity operator* (depending on two elements $x, y \in \mathcal{S}$ and random control parameter $k \in \mathbb{Z}$) by

$$\mathrm{dis}(x, y, k) := \mathrm{sim}(\overline{x}, y, k).$$

It follows from the theory presented in [14] that the similarity operator possesses some properties required by a "good" recombination (crossover) operator. In particular, it respects the formae with respect to which it is defined, in the sense that we always have $\mathrm{sim}(x, y, k) \in \Sigma(\{x, y\})$. On the other hand, the dissimilarity operator does not have such properties; it is a composition of the similarity operator and the operation of taking the opposite of the first argument.

In our DSC algorithm, the chromosomes (i.e., the values of ρ_E) are simply binary strings of a fixed length, and the basic equivalence relations in E are determined by fixed positions in a string (i.e., two strings are equivalent if they have the same value at a given position). Then, the equivalence relations from Span E are the usual schemata (each schema is determined by a finite number of fixed positions in a string).

In this particular case, the similarity operator is equivalent to the well-known *uniform crossover* (see [14, p. 370]), while the dissimilarity operator is equivalent to the uniform crossover applied to $\overline{x}$ and y.

3. DSC algorithm

We consider the following optimization problem:

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\text{minimize|maximize } f(x_1, \ldots, x_n) \text{ subject to}$$

$$x_i \in [a_i, b_i], i = 1, \ldots, n$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a given function.

In the algorithm described below, we use a standard encoding of chromosomes as found in the book of Michalewicz [12].

In particular, we use the following formula to decode real number $x_i \in [a_i, b_i]$:

$$x_i = a_i + \mathrm{decimal}(1001...001) \cdot \frac{b_i - a_i}{2^{m_i} - 1}$$

where m_i is the length of a binary string and "decimal" represents the decimal value of this string. The value of m_i for each variable depends on the length of interval $[a_i, b_i]$. To encode point $(x_1, \ldots x_n)$, we use a decimal string of length $m = \sum_{i=1}^{n} m_i$.

Let M be a positive integer divisible by 8. The DSC algorithm is described by the following steps:

1. Generate M chromosomes, each chromosome representing a point $(x_1, \ldots x_n)$.
2. Compute the values of fitness function f for each chromosome in the population.
3. Sort the chromosomes according to the descending (for maximization) or ascending (for minimization) values of the fitness function.
4. Copy C times the first chromosome and put it in C positions in the first half of the population randomly, replacing the original chromosomes, where $C = M/8$.
5. For chromosomes in the first quarter of the population (from 1 to $M/4$), apply the dissimilarity operator to the first and second chromosomes, replace the second chromosome by the offspring, then apply the dissimilarity operator to the (new) second and third chromosomes, and so on.
6. For chromosomes in the second quarter of the population (from $M/4+1$ to $M/2$), apply the similarity operator to the first and second chromosomes, replace the second chromosome by the offspring, then apply the similarity operator to the (new) second and third chromosomes, and so on.
7. Generate chromosomes randomly for the second half of the population. These will replace the second half of the chromosomes (in positions from $M/2+1$ to M).
8. Go to Step 2 and repeat until the stopping criterion is reached.

Note. The stopping criterion for our algorithm depends on the example being considered (see Section 4).

To maintain population diversity, Sultan et al. [17] proposed a simple injection strategy to the population. They used fix point injection, which means that they introduced new randomly generated chromosomes to the population for certain numbers of generations. We have applied a similar strategy in our DSC algorithm by generating the second half of each population randomly.

In the paper by M. Lewchuk [10], the author introduces a genetic invariance algorithm that is a modification of the classical GA. He uses a uniform crossover operator that is equivalent to our similarity operator, and he also uses a sorting of the population according to the fitness function values. However, the crossover is applied only to a pair of individuals for which the difference in their function values is minimum over all pairs. Note that the uniform crossover and sorting procedure are used in our DSC algorithm, but we also use a new dissimilarity operator and random regeneration of a part of the population in each iteration; these last two procedures do not appear in the genetic invariance algorithm.

In Berretta et al. [2], the authors define the Recombine() procedure (pp. 78–79) that contains three genetic operators called "rebel", "conciliator" and "obsequent".

They take some alleles from two parents P_1 and P_2 to copy in the offspring first as follows:

1) "rebel" copies alleles of P_2 that are different from P_1,

2) "conciliator" copies alleles in common to P_1 and P_2,

3) "obsequent" copies alleles of P_1 that are different from P_2.

Then, the procedure chooses the alleles for the remaining positions in the offspring. This can be done by using several different algorithms (random or deterministic). It should be noted that the "rebel" operator is very similar to our dissimilarity operator (in fact, they are equivalent if a random selection is chosen for the second part of the procedure). In the same way, the "conciliator" is equivalent to our similarity operator, and "obsequent" is equivalent to our dissimilarity operator applied to P_2 and P_1 (in reverse order).

4. Experimental results

In this section, we report on computational testing (by using the Matlab software) of the DSC algorithm on 19 test functions taken from the literature. After each test, the result of DSC has been compared with the known global optimum and with the result of a classical GA taken from the respective reference. The results are presented in Tables 1–5 below. We have applied the algorithm with 40 chromosomes (see the results in Table 2), 80 chromosomes (Table 3), and 160 chromosomes (Table 4).

The DSC algorithm has found optimum solutions for some optimization problems (like Easom, Both's, Schwefel's, and Shubert's) that the classical genetic algorithm cannot solve, with a minimum success rate of 92% with 80 chromosomes for Schwefel's function (Table 2) and a maximum success rate of 100% for the remaining problems. Observe that, with 160 chromosomes, we have achieved a 100% success rate even for Schwefel's example.

In Table 5, we compare the mean of the numbers of iterations for all successful runs of the proposed DSC (40, 80, and 160 chromosomes). Then, we compare the rates of success of the DSC and classical GA algorithms. The algorithm was stopped when either the maximum number of iterations was reached (fixed at 2500) or the difference between the obtained minimum/maximum fitness and global optimum was less than or equal to the threshold given in the second column.

The success rates for the GA presented for comparison in the last columns of Tables 2–5 were taken from the quoted literature; these results were obtained for populations of between 10 and 100 chromosomes, depending on the specific example.

We have recognized that, for most problems, using 80 chromosomes gives the best results in terms of both the success rate and number of function evaluations. The DSC algorithm keeps the best solution from each iteration in the first position until it is replaced by a better one. Note that the maximum average rate of iterations was especially high (561) for the Schwefel function, for which the classical genetic algorithm failed to find a solution (see also Table 3).

Table 1
Test functions

Function name	Interval	Function	Global optimum min/max
Easom	$x,y \in [-100,100]$	$f(x,y)=-\cos(x)\cos(y)\exp(-(x-\pi)^2+(y-\pi)^2)$	$f(\pi,\pi)=-1$, min
Matyas	$x,y \in [-10,10]$	$f(x,y)=0.26(x^2+y^2-0.48xy)$	$f(0,0)=0$, min
Beale's	$x,y \in [-4.5,4.5]$	$f(x,y)=(1.5-x-xy)^2+(2.25-x+xy^2)^2+(2.625-x-xy^2)^2$	$f(3,0.5)=0$, min
Booth's	$x,y \in [-10,10]$	$f(x,y)=(x+2y-7)^2+(2x+y-5)^2$	$f(1,3)=0$, min
Goldstein -Price	$x,y \in [-2,2]$	$f(x,y)=(1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2))\cdot(30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2))$	$f(0,-1)=3$, min
Schaffer N.2	$x,y \in [-100,100]$	$f(x,y)=0.5+\frac{\sin^2(x^2-y^2)-0.5}{(1+0.001(x^2+y^2))^2}$	$f(0,0)=0$, min
Schwefel's	$x_1,x_2 \in [-500,500]$	$f(x)=\sum_{i=1}^{n} x_i\cdot\sin\left(\sqrt{\lvert x_i\rvert}\right)$	$f(1,1)=0$, min
Branins's rcos	$x_1 \in[-5,10]$ $x_2 \in[0,15]$	$f(x_1,x_2)=a\cdot(x_2-b\cdot x_1^2+c\cdot x_1-d)^2+e\cdot(1-f)\cdot\cos(x_1)+e$ $a=1,\ b=\frac{5.1}{4\cdot\pi^2},\ c=\frac{5}{\pi},\ d=6,\ e=10,\ f=\frac{1}{8\pi}$	$f(\pi,2.275)$ or $f(9.42478,2.475)$ or $f(-\pi,12.275)=0.397887$, min
Six-hump camel back	$x_1 \in[-3,3]$ $x_2 \in[-2,2]$	$f(x_1,x_2)=\left(4-2.1x_1^{\frac{4}{3}}\right)\cdot x_1^2+x_1x_2+\left(-4+4x_2^2\right)\cdot x_2^2$	$f(-0.0898,0.7126)=-1.0316$, min
Shubert	$x_1,x_2 \in [-10,10]$	$f(x_1,x_2)=\left(\sum_{i=1}^{5} i\cos[(i+1)x_1+i]\right)\cdot\left(\sum_{i=1}^{5} i\cos[(i+1)x_2+i]\right)$	18 global min $f=-186.7309$, min
Martin and Gaddy	$x_1,x_2 \in[0,10]$	$f(x_1,x_2)=(x_1-x_2)^2\cdot\left((x_1+x_2-10)/3\right)^2$	$f(5,5)=0$, min
Zbigniew Michalewicz	$x_1 \in[-3,12.1]$ $x_2\in[-4.1,5.8]$	$f(x_1,x_2)=21.5+x_1\cdot\sin(4\pi x_1)+x_2\cdot\sin(20\pi x_2)$	$f(11.631407,5.724824)=38.81208$, max, [7]
Holder table	$x_1,x_2 \in [-10,10]$	$f(x_1,x_2)=-\left\lvert\sin(x_1)\cos(x_2)\exp\left(\left\lvert1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert$	$f(8.05502,9.66458)$ or $f(8.05502,-9.66458)$ or $f(-8.05502,9.66458)$ or $f(-8.05502,-9.66458)=-19.2085$, min
Drop-wave	$x_1,x_2 \in [-4.12,5.12]$	$f(x_1,x_2)=-\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5(x_1^2+x_2^2)+2}$	$f(0,0)=-1$,
Levy N.13	$x_1,x_2 \in [-10,10]$	$f(x_1,x_2)=\sin^2(3\pi x_1)+(x_1-1)^2[1+\sin(3\pi x_2)]+(x_2-1)^2[1+\sin(2\pi x_2)]$	$f(1,1)=0$,
Rastrigin's	$x_i \in [-5.12,5.12]$	$f(x_i)=10\cdot d\sum_{i=1}^{d}[x_i^2-10\cos(2\pi x_i^2)]$	$f(x_i)=0$ at $x_i=0$
Sphere $d=2,d=10$	$x_i\in [-5.12,5.12]$	$f(x_i)=\sum_{i=1}^{d} x_i^2$	$f(x_i)=0$ at $x_i=0$
Ackley $n=4$	$x_i \in[-32.768, 32.768]$	$f(x)=-a\cdot\exp\left(-b\cdot\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)+\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(c\cdot x_i)\right)+a+\exp(1)$, where $a=20,b=0.2,c=2\pi$	$f(x_i)=0$ at $x_i=0$
Sum of different powers $d=10$	$x_i\in[-1,1]$	$f(x_i)=\sum_{i=1}^{d}\lvert x_i\rvert^{i+1}$	$f(x_i)=0$ at $x_i=0$

Table 2

Best value of functions for 50 runs of DSC algorithm (40 chromosomes)

Function name	Threshold of best	Min number of iterations	Max number of iterations	Mean no. of iterations for all successful runs	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Easom	0.001	31	464	181	−0.99543	100%	0% [5]
Matyas	0.001	4	391	64	0.000505	100%	70% [15]
Beale's	0.001	5	1349	179	0.000517	98%	6% [9]
Booth's	0.005	13	1181	321	0.00274	98%	0% [15]
Goldstein–Price	0.001	46	896	387	3.00038	100%	72% [5]
Schaffer N.2	0.001	24	1533	476	4.11E-05	100%	0%
Schwefel's	0.01	94	2390	506	0.07317	50%	0% [5]
Branins's rcos	0.001	16	2332	171	0.39853	100%	100%
Six-hump camel back	0.001	9	215	73	−1.03125	100%	98% [13]
Shubert	1	5	149	67	−185.886	100%	0% [5]
Martin and Gaddy	0.001	7	438	53	3.95E-05	100%	1% [5]
Zbigniew Michalewicz	0.1	40	1500	346	38.81746	100%	73% [7]
Holder table	0.001	9	535	100	−19.2035	100%	78% [16] asynchronus EA
Drop-wave	0.001	30	1621	420	−0.99517	100%	30%
Levy N.13	0.001	47	1700	504	0.000583	100%	70%
Rastrigin's	0.001	16	330	127	0.00505	100%	100% with bit string 0% with Double vector(GA)
Sphere $d = 2$	0.001	15	395	155	0.003588	100%	100%
Ackley $n = 4$	0.001	403	2499	1661	0.280442	62%	100%

Table 3
Best value of functions for 50 runs of DSC algorithm (80 chromosomes)

Function name	Threshold of best	Min number of iterations	Max number of iterations	Mean no. of iterations for all successful runs	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Easom	0.001	16	286	88	−0.99579	100%	0% [5]
Matyas	0.001	6	97	31	0.000492	100%	70% [15]
Beale's	0.001	4	646	93	0.00059	100%	6% [9]
Booth's	0.005	5	980	151	0.003198	100%	0% [15]
Goldstein–Price	0.001	35	368	134	3.00036	100%	72% [5]
Schaffer N.2	0.001	5	1023	278	4.11E-05	100%	0%
Schwefel's	0.01	33	2287	561	0.015643	92%	0% [5]
Branins's rcos	0.001	3	890	86	0.39853	100%	100%
Six-hump camel back	0.001	6	115	39	−1.03129	100%	98% [13]
Shubert	1	4	139	32	−185.781	100%	0% [5]
Martin and Gaddy	0.001	6	151	36	3.02E+05	100%	1% [5]
Zbigniew Michalewicz	0.1	26	713	207	38.81257	100%	73% [7]
Holder table	0.001	4	163	47	−19.8125	100%	78% [16] asynchronus EA
Drop-wave	0.001	13	816	194	−0.99487	100%	30%
Levy N.13	0.001	5	1611	290	0.000547	100%	70%
Rastrigin's	0.001	14	181	71	0.007197	100%	100% with bit string 0% with Double vector (GA)
Sphere $d = 2$	0.001	19	186	75	0.004133	100%	100% with bit string 50% with Double vector (GA)
Ackley $n = 4$	0.001	372	2171	1042	0.07673	100%	100%

Table 4

Best value of functions for 50 runs of DSC algorithm (160 chromosomes)

Function name	Threshold of best	Min number of iterations	Max number of iterations	Mean no. of iterations for all successful runs	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Easom	0.001	11	141	61	−0.99927	100%	0% [5]
Matyas	0.001	2	29	13	0.000434	100%	70% [15]
Beale's	0.001	2	212	48	0.000523	100%	6% [9]
Booth's	0.001	6	1018	123	0.000595	98%	0% [15]
Goldstein–Price	0.001	12	106	44	3.000484	100%	72% [5]
Schaffer N.2	0.001	6	731	107	0.00045	100%	0%
Schwefel's	0.01	26	2301	517	0.07051	100%	0% [5]
Branins's rcos	0.001	2	324	40	0.398517	100%	100%
Six-hump camel back	0.001	1	41	14	−1.03106	100%	98% [13]
Shubert	0.01	10	457	111	−186.716	100%	0% [5]
Martin and Gaddy	0.001	3	38	14	0.000513	100%	1% [5]
Zbigniew Michalewicz	0.04	6	297	93	38.81715	100%	73% [7]
Holder table	0.001	6	725	84	−19.2078	100%	78% [16] asynchronus EA
Drop-wave	0.001	14	708	122	−0.99954	100%	30%
Levy N.13	0.001	4	538	117	0.000471	100%	70%
Rastrigin's	0.001	17	116	53	0.000442	100%	100% with bit string 0% with Double vector (GA)
Sphere $d = 2$	0.001	1	42	12	0.000445	100%	100% with bit string 50% with Double vector (GA)
Ackley $n = 4$	0.001	173	1379	552	0.071314	100%	100%
Sphere function $d = 10$	0.1	359	1396	746	0.090271	100%	90%
Sum of different powers d=10	0.1	2	39	14	0.020016	100%	80%

Table 5

Best value of functions for 50 runs of the DSC algorithm (40 vs 80 vs 160 chromosomes)

Function name	Mean no. of iterations for all successful runs 40 ch.	Mean no. of iterations for all successful runs 80 ch.	Mean no. of iterations for all successful runs 160 ch.	Rate of success DSC (40 ch.)	Rate of success DSC (80 ch.)	Rate of success DSC (160 ch.)	Rate of success GA
Easom	181	88	61	100%	100%	100%	0% [5]
Matyas	64	31	13	100%	100%	100%	70% [15]
Beale's	179	93	48	98%	100%	100%	6% [9]
Booth's	321	151	123	98%	100%	100%	0% [15]
Goldstein–Price	387	134	44	100%	100%	100%	72% [5]
Schaffer N.2	476	278	107	100%	100%	100%	0%
Schwefel's	506	561	557	50%	92%	100%	0% [5]
Branins's rcos	171	86	40	100%	100%	100%	100%
Six-hump camel back	73	39	14	100%	100%	100%	98% [13]
Shubert	500	198	111	100%	100%	100%	0% [5]
Martin and Gaddy	53	36	14	100%	100%	100%	1% [5]
Zbigniew Michalewicz	346	207	93	100%	100%	100%	73% [7]
Holder table	100	47	84	100%	100%	100%	78% [16] asynchronus EA
Drop-wave	420	194	122	100%	100%	100%	40%
Levy N.13	504	290	117	100%	100%	100%	70%
Rastrigin's	127	71	53	100%	100%	100%	100% with bit string 0% with Double vector (GA)
Sphere $d = 2$	155	75	12	100%	100%	100%	100%
Ackley $n = 4$	1661	1042	552	62%	100%	100%	100%

Table 6 presents a comparative study of the success rates and number of function evaluations for the CMA-ES (Covariance Matrix Adaptation Evolution Strategy), DE (Differential Evolution), and DSC algorithms; it shows that the DSC algorithm is the most-successful one (see, especially, the Drop-wave function). The Matlab codes for the CMA-ES and DE algorithms were taken from [3] and [4], respectively.

Table 6

Comparison of CMA-ES, DE, and DSC algorithms in terms of mean number of function evaluations and success rate (30 runs, max. 1000 iterations, 60 chromosomes)

Function name	CMA-ES success rate	Function evaluations of CMA-ES	DE success rate	Function evaluations of DE	DSC success rate	Function evaluations of DSC
Easom	70%	17.053	100%	3240	100%	7588
Schaffer N.2	90%	6726	100%	5016	100%	8356
Drop-wave	50%	26.470	94%	9.048	100%	13.788
Levy N.13	100%	606	100%	1958	100%	9216
Rastrigin's	80%	13.134	100%	2388	100%	8022

5. Discussion of figures

Figure 1 shows a two-dimensional view of Easom function. It can be seen that the DSC algorithm has reached the best solution at the blue point at $f(\pi, \pi) = -1$.

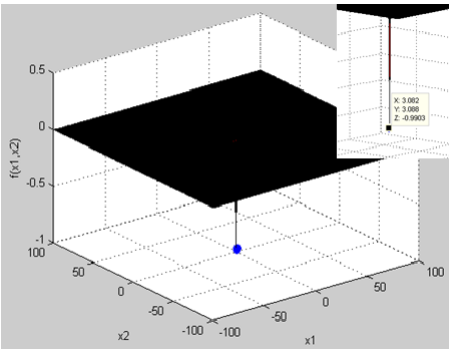

Figure 1. Solutions of Easom problem

Figure 2 shows a two-dimensional view of Schaffer's function. It can be seen that the DSC algorithm reached the best solution at the blue point on the focus view in the upper-right corner of the figure. For this function, it is difficult to reach an optimal solution, because it contains multi-local minimum solutions close to the best one.

Figures 3–10 show two-dimensional views of Shubert's problem with 18 optimal solution points, Branins's problem with 3 optimum optimal solution points, the Six-hump camel back problem with 2 optimum points, the Holder table problem with 4 optimum points, the Michalewicz problem, the Drop-wave problem, Schwefel's problem, and the Levy N.13 problem with one point optimum solution, respectively.

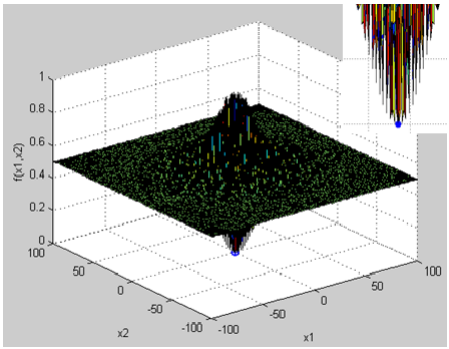

Figure 2. Solutions of Schaffer's problem

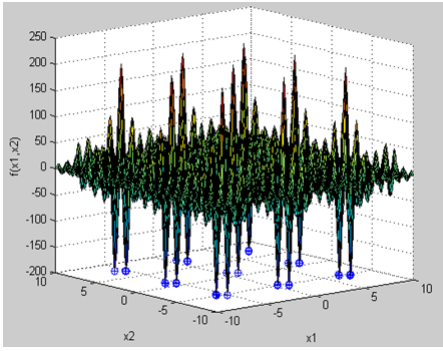

Figure 3. Solutions of Shubert's problem, 18 optimum solutions

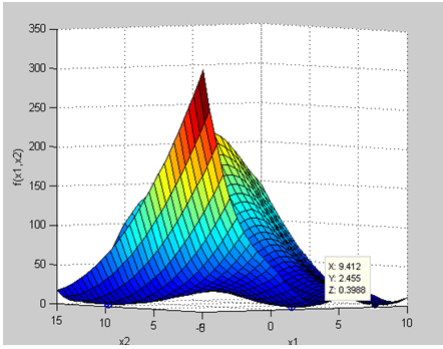

Figure 4. Solutions of Branins's problem

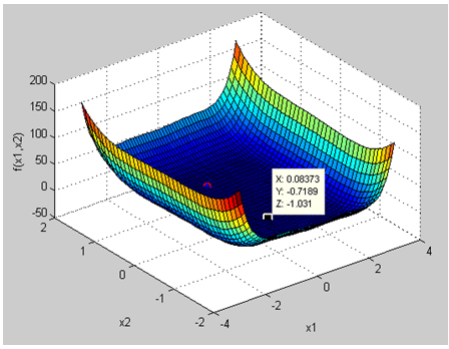

Figure 5. Solutions of Six-hump camel back problem

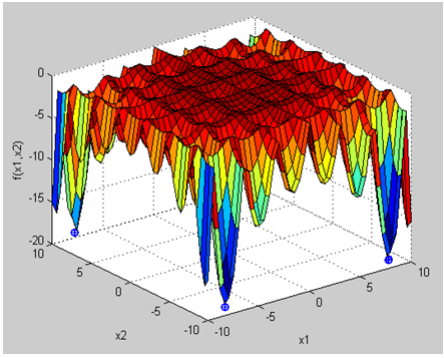

Figure 6. Solutions of Holder-table problem

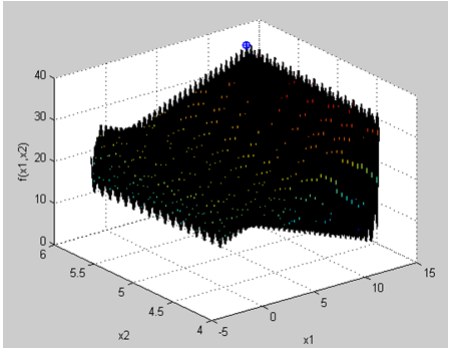

Figure 7. Solutions of Michalewicz

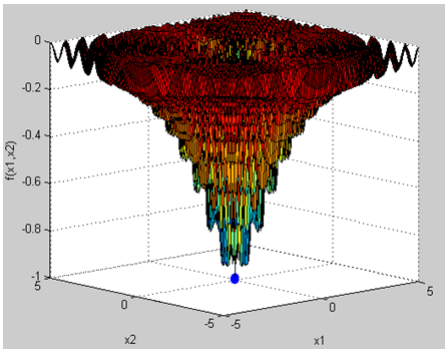

Figure 8. Solutions of Drop-wave problem

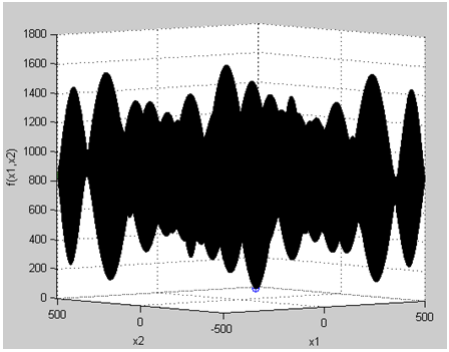

Figure 9. Solutions of Schwefel's problem

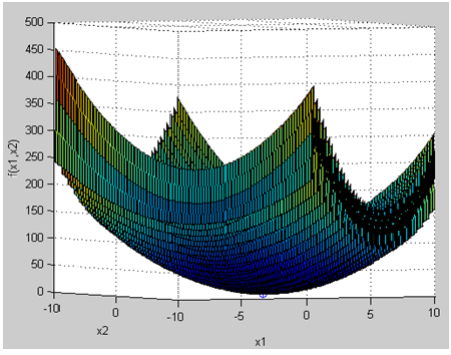

Figure 10. Solutions of Levy N.13 problem

Figures 11 and 12 show how the best fitness values of the population evolve with the number of iterations. Here, the red color represents a jump to a better solution.

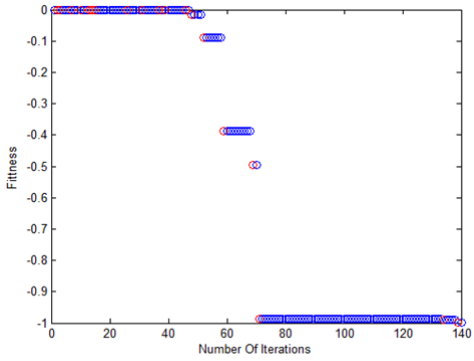

Figure 11. Finding best solution for Schaffer problem in 140 iterations

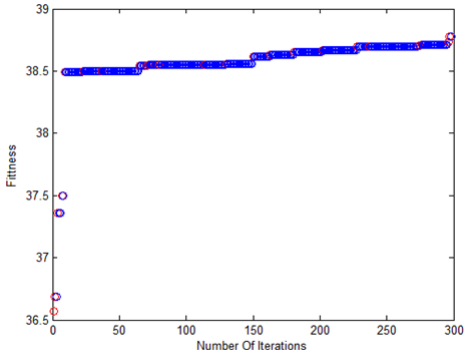

Figure 12. Finding best solution for Michalewicz problem in 300 iterations

6. Conclusion

In this paper, a new meta-heuristic optimization algorithm called Dissimilarity and Similarity of Chromosomes (DSC) is introduced. DSC can be simply implemented without too many parameters. It includes two genetic operators (the dissimilarity and similarity operators), population sorting, and random generation of a part of the population. The experiments have shown the quick convergence and good global searching ability of our algorithm. The DSC algorithm is easy to understand and uses a simple classical representation of points in $\mathbb{R}^n$.

The DSC algorithm has only one parameter to be set by the user: the number M of chromosomes. Therefore, it is easier to test than the classical GA where the user must try multiple runs to test different combinations of parameters. For all of our examples, 80 chromosomes are enough to solve the problem. As Table 5 shows, there is a significant difference in the rate of success between 40 and 80 chromosomes. Table 5 also shows that the rate of success of our algorithm is much better than for the classical GA that has a lot of parameters.

Acknowledgements

The first author would like to thank the Ministry of Higher Education and Scientific Research (MOHESR), Iraq.

References

[1] Auger A., Hansen N.: A restart CMA evolution strategy with increasing population size. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 1769–1776, 2005.

[2] Berretta R., Cotta C., Moscato P.: Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm. Results on the number partitioning problem. In: Resende M., Pinho de Sousa J. (eds.): *Metaheuristics: Computer-Decision Making*, pp. 65–90, 2003.

[3] CMA-ES. Matlab program. `https://www.mathworks.com/matlabcentral/fileexchange/52898-cma-es-in-matlab`.

[4] Differential Evolution Optimization Lecture. Matlab program. `http://www.dii.unipd.it/~alotto/didattica/corsi/Elettrotecnica%20computazionale/DE.pdf`.

[5] Eesa A., Brifcani A., Orman Z.: A New Tool for Global Optimization Problems-Cuttlefish Algorithm, *International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering*, vol. 8(9), pp. 1235–1239, 2014.

[6] Eiben A.E., Smith J.E.: *Introduction to Evolutionary Computing*, Springer, 2003.

[7] Gen M.: Network models and optimization: multiobjective GA approach, 2009. `http://logistics.iem.yzu.edu.tw/Gen/seminarII-YZU-IEM-MGen.pdf`.

[8] Han X., Liang Y., Li Z., Li G., Wu X., Wang B., Zhao G., Wu C.: An efficient genetic algorithm for optimization problems with time-consuming fitness evaluation, *International Journal of Computational Methods*, vol. 12(01), pp. 13501065–1–13501065–24, 2015.

[9] Iqbal M.A., Khan N.K., Mujtaba H., Baig A.R.: A novel function optimization approach using opposition based genetic algorithm with gene excitation, *International Journal of Innovative Computing, Information and Control*, vol. 7(7B), pp. 4263–4276, 2011.

[10] Lewchuk M.: *Genetic invariance: a new type of genetic algorithm.* Technical report TR 92-05, Dept. of Computing Science, University of Alberta, 1992.

[11] Manda K., Satapathy S., Poornasatyanarayana B.: Population based meta-heuristic techniques for solving optimization problems: A selective survey, *International Journal of Emerging Technology and Advanced Engineering*, vol. 2(11), pp. 206–211, 2012.

[12] Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolutions Programs*, Springer, Berlin, 1996.

[13] Odili J., Kahar M.: Numerical function optimization solutions using the african buffalo optimization algorithm (ABO), *British Journal of Mathematics & Computer Science*, vol. 10(1), pp. 1–12, 2015.

[14] Radcliffe N.: The algebra of genetic algorithms, *Annals of Mathematics and Artificial Intelligence*, vol. 10, pp. 339–384, 1994.

[15] Ritthipakdee A., Thammano A., Premasathian N., Uyyanonvara B.: An improved firefly algorithm for optimization problem, *The 5th International Symposium on Advanced Control of Industrial Processess (ADCONIP 2014)*, pp. 159–164, 2014.

[16] Scott E., Jong K.: Understanding simple asynchronous evolutionary algorithms, *FOGA'15 Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, pp. 85–98, 2015.

[17] Sultan A., Mahmod R., Sulaiman M., Bakar M.: Maintaining diversity for genetic algorithm: A case of timetabling problem, *Jurnal Teknologi*, vol. 44, pp. 123–130, 2006.

[18] Yu Y., Zhou Z.H.: A new approach to estimating the expected first hitting time of evolutionary algorithms, *Artificial Intelligence*, vol. 172(15), pp. 1809–1832, 2008.

Affiliations

Radhwan Al-Jawadi
University of Warsaw, Faculty of Mathematics, Informatics and Mechanics, Poland, (permanent address: Technical College of Mosul, FTE, Iraq), radwanyousif@yahoo.com

Marcin Studniarski
University of Lodz, Faculty of Mathematics and Computer Science, Lodz, Poland, marstud@math.uni.lodz.pl

Aisha Younus
University of Lodz, Faculty of Mathematics and Computer Science, Lodz, Poland, azeezzena74@yahoo.com

Received: 25.05.2017
Revised: 30.12.2017
Accepted: 30.12.2017