International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:8, 2016

An Optimization Algorithm Based on Dynamic Schema with Dissimilarities and Similarities of Chromosomes

Radhwan Yousif Sedik Al-Jawadi

***Abstract*—**Optimization is necessary for finding appropriate solutions to a range of real-life problems. In particular, genetic (or more generally, evolutionary) algorithms have proved very useful in solving many problems for which analytical solutions are not available. In this paper, we present an optimization algorithm called Dynamic Schema with Dissimilarity and Similarity of Chromosomes (DSDSC) which is a variant of the classical genetic algorithm. This approach constructs new chromosomes from a schema and pairs of existing ones by exploring their dissimilarities and similarities. To show the effectiveness of the algorithm, it is tested and compared with the classical GA, on 15 two-dimensional optimization problems taken from literature. We have found that, in most cases, our method is better than the classical genetic algorithm.

***Keywords*—**Genetic algorithm, similarity and dissimilarity, chromosome injection, dynamic schema.

I. Introduction

GLOBAL optimization algorithms are usually categorized as deterministic or meta-heuristic, see [1]. Meta-heuristic techniques are useful for many optimization problems where deterministic algorithms are difficult to apply (for example, ill-behaving functions with many jumps). Many of the meta-heuristic algorithms such as Genetic Algorithms (GA), Ant Colony Optimization (ACO), Bees Algorithms (BA) and other, which are bio-inspired, have previously been described in the literature.

The Evolutionary algorithms (EAs) are optimization techniques modeled on the process of natural evolution. As [2] observes, different implementations of EAs (e.g., GA, genetic programming, evolutionary strategy) can essentially be summarized by the following four steps:

1. Generate an initial population randomly;
2. Produce new solutions based on the current population;
3. Remove bad solutions from the population;
4. Repeat from Step 2 until a stopping criterion is satisfied.

In this paper, a meta-heuristic optimization algorithm is presented that is inspired by the schema theory and the mechanism of similarity and dissimilarity of chromosomes. This procedure depends on dividing each generation into four equal parts and then applying different genetic operators to each of them. The presented algorithm is called DSDSC and it is designed to find optimal solutions to numerical optimization problems.

The idea of dividing a population into some parts and then working with schemata and similarity for each part separately is already known in the literature. For example, [3] divides a population into three parts depending on the fitness of chromosomes (the best, the middle and the worst fitness groups) and then discover the common schema in a population by using clustering. Next, for the first and the third part of a population, they calculate the number of chromosomes that have some similarity with the schema. The similarity between an individual and a schema is defined as the percentage of positions on which the individual agrees with the schema.

Yu and Zhou [2] described a new general approach to estimating the expected first hitting time by analyzing EAs with different configurations. This method works with three mutation operators, a recombination operator and a time variant mutation operator. In a further research, we plan to examine the possibility of applying a similar theoretical analysis to our DSDSC algorithm.

Many studies have been suggested different approaches to be compare with the GA such as [4]-[9].

This paper is organized as follows. In Section II methodology of DSDSC algorithm are introduced. In Section III, we describe the DSDSC algorithm and show its flowchart. Section IV gives the schema analysis of the algorithm. Section V gives the analysis of experimental results. Section VI contains descriptions and discussion of figures. Finally, conclusions are presented in Section VII.

II. Methodology

The DSDSC algorithm starts with a population of M elements representing a number of solutions to the problem. This population is divided into four equal groups and we apply some different operators to these groups. This will be discussed in Section III.

Briefly, the DSDSC creates new chromosomes by exploring dynamic dissimilarity, similarity, dynamic schema and random generation of new chromosomes.

Table I shows all M chromosomes (Ch1..ChM) divided onto 4 groups (G1, G2, G3, G4).

III. The DSDSC algorithm

We consider the following optimization problem:

Radhwan Yousif Sedik Al-Jawadi is with the Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland (permanent address: Technical College of Mosul, FTE, Iraq; e-mail: radwanyousif@yahoo.com).

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$
$$\text{minimize|maximize } f(x_1, \ldots, x_n) \text{ subject to}$$
$$x_i \in [a_i, b_i], i = 1, \ldots, n$$

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

TABLE I
GROUPS OF CHROMOSOMES

Ch_1	
Ch_2	
Ch. …	G1: To the first group we apply the dynamic dissimilarity operator.
Ch. …	
$Ch_{M/4}$	
$Ch_{M/4+1}$	
$Ch_{M/4+2}$	
Ch. …	G2: To the second group we apply the similarity operator.
Ch. …	
$Ch_{M/2}$	
$Ch_{M/2+1}$	
$Ch_{M/2+2}$	
Ch. …	G3: To the third group we apply the dynamic schema operator.
Ch. …	
$Ch_{M/2+M/4}$	
$Ch_{M/2+M/4+1}$	
$Ch_{M/2+M/4+2}$	
Ch. …	G4: The fourth group is generated randomly.
Ch. …	
Ch_M	

In the algorithm described below, we use a standard encoding of chromosomes as in the book of Michalewicz [10]. In particular, we use the following formula to decode a real number $x_i \in [a_i, b_i]$:

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

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 the interval $[a_i, b_i]$. To encode a 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 DSDSC algorithm consist of the following steps:

1) Generate M chromosomes, each chromosome representing a point $(x_1, \ldots, x_n)$.
2) Compute the values of the 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. Then divide the population into four equals groups (G1,G2,G3,G4).
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) Apply the dynamic schema operator to the chromosomes Ch_1 and $Ch_{M/4}$ (that is, the chromosomes on the positions 1 and *M/4*, respectively). This operator works as follows (see Table II):

(a) First, we divide each chromosome onto n parts corresponding to variables $(x_1, \ldots, x_n)$, the i-th part having length m_i. Next, for each variable x_i , we select a random integer r_i from the set {3,…,8}, then let $R_i := round\ (m_i / r_i)$, where "round" means rounding a number to the nearest integer. We define the "gray" part of x_i as the first segment of length R_i of the string corresponding to x_i. We define the "white" part of x_i as the second segment of length $m_i - R_i$ of the same string.

(b) For the "white" parts of both chromosomes, if the two bits are not equal, put a star (*) in the schema, then copy this schema *M/4* times and put it in the third part of population (G3) between positions *M/2+1* and *M/2+M/4*, then put randomly 0 or 1 in the positions having *. We keep the positions marked in gray unchanged.

Note. The name "dynamic schema operator" is justified by the fact that the lengths of "gray" and "white" segments of chromosomes may vary from iteration to iteration.

TABLE II
THE DYNAMIC SCHEMA OPERATOR
Before change: an example for finding schema from the first chromosome and the chromosome on position *M/4*. Here shadow bits are not destroyed.

No. of Ch.	m_1						m_2			
	R_1		$m_1 - R_1$				R_2	$m_2 - R_2$		
Ch_1	1	1	0	0	1	0	1	0	1	0
$Ch_{M/4}$	0	1	1	0	0	1	0	0	0	1
Schema	1	1	*	0	*	*	1	0	*	*
After finding the schema: put it in M/2.M/2+M/4 positions, then put randomly 0 or 1 in (*) bits										
$Ch_{M/2+1}$	1	1	*	0	*	*	1	0	*	*
$Ch_{M/2+2}$	1	1	*	0	*	*	1	0	*	*
Ch. …	1	1	*	0	*	*	1	0	*	*
Ch. …	1	1	*	0	*	*	1	0	*	*
$Ch_{M/2+M/4}$	1	1	*	0	*	*	1	0	*	*
After change: put randomly 0 or 1 in (*) bits										
$Ch_{M/2+1}$	1	1	1	0	1	0	1	0	0	1
$Ch_{M/2+2}$	1	1	1	0	0	0	1	0	1	1
Ch. …	1	1	0	0	1	0	1	0	1	0
Ch. …	1	1	0	0	0	1	1	0	0	0
$Ch_{M/2+M/4}$	1	1	1	0	1	1	1	0	1	1

6) Compare pairs of chromosomes for the first half (G1,G2) of the population to explore dissimilarities and similarities. Check each two following chromosomes, i.e. the first and the second, the second and the third, and so on, by comparing the respective bits, as follows:

(a) For chromosomes in the first quarter (G1) of the population (from 1 to *M*/4), if the two bits are equal, put a star (*) in the second (following) chromosome; otherwise, leave this bit without a change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on. This dissimilarity operator also depends on dynamic part of chromosomes m_i/r_i.

(b) For chromosomes in the second quarter (G2) of the population (from *M*/4+1 to *M*/2), if the two bits are not equal, put a star (*) in the second (following) chromosome; otherwise leave this bit without a change in

the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on. this comparison is started from bit number 1 for each m_i in chromosomes (M/4+1 to M/2).

TABLE III
THE DYNAMIC DISSIMILARITY OPERATOR
Before change: an example for the first quarter of chromosomes.

	m_1						m_2			
	R_1		$m_1 - R_1$				R_2	$m_2 - R_2$		
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	1	0	0	1	0	0	1	1
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	1	*	0	1	1	*	*	1
After change: put randomly 0 or 1 in (*) bits										
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	1	1	0	1	0	1	0	1

TABLE IV
THE SIMILARITY OPERATOR
Before change: an example for the second quarter of chromosomes.

	m_1							m_2		
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	0	0	1	0	0	1	0	0	1	1
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	*	*	*	0	*	*	*	0	1	*
After change: put randomly 0 or 1 in (*) bits										
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	0	1	0	0	0	1	0	0

7) All chromosomes B created in this way replace the original ones in positions from 2 to M/2. The schema is also generated in the way described at Step 5 from M/2+1 to M/2+M/4. Then generate randomly chromosomes for the fourth group of the population. These will replace the fourth group of the chromosomes (on positions from M/2+M/4+1 to M).
8) Go to Step 2 and repeat until the stopping criterion is reached.

Fig. 1 represents the flowchart for DSDSC algorithm.

Notes:

(a) We call the genetic operator performing the operations shown in Table III on a pair of chromosomes A and B the *dynamic dissimilarity operator*, and the genetic operator performing the operations shown in Table IV the *similarity operator*.
(b) The *dynamic schema operator* is shown in Table II, that uses different sizes of fixed segments (gray color) and applies a *similarity operator* on the rest of chromosome.
(c) The stopping criterion for our algorithm depends on the example being considered, see Section V.
(d) To maintain population diversity, [11] proposed a simple injection strategy to the population. They use fixpoint injection, which means that they introduce new randomly generated chromosomes to the population for certain numbers of generations. We have applied a similar strategy in our DSDSC algorithm by generating the last quarter of each population randomly.

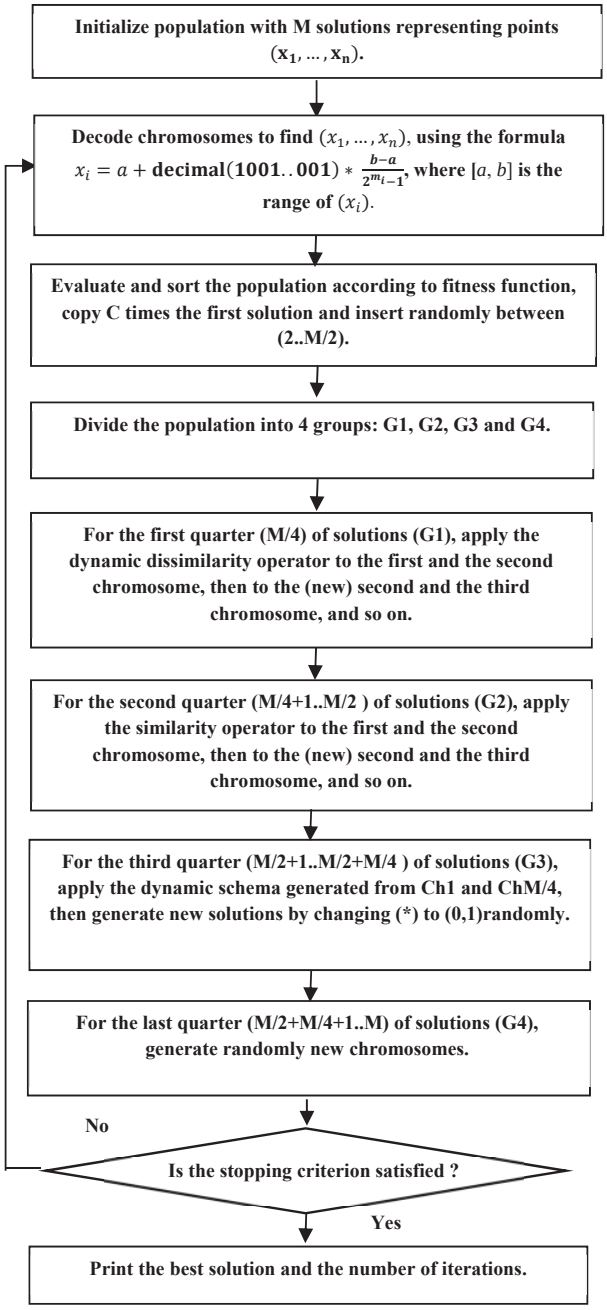

Fig. 1 Flowchart of the DSDSC algorithm.

IV. SCHEMA ANALYSIS

A schema represents a number of similar strings, thus, a schema can be thought of as a representation of a certain region in the search space. The schema that represents the region containing the best solution must increase in the population to get the solutions in the best region [10], [12]. For example, assume we have a part of the Zbigniew Michalewicz function $f(x_1, x_2) = 21.5 + x_1 \cdot sin(4\pi x_1) + x_2 \cdot sin(20\pi x_2)$, where x_1,$x_2 \in [0,1]$, as shown in Fig. 2, it is clear the maximum solution is between when $x_1 \in$ [0.6,0.8] in the

region [0,1]. This function has two local maximum solutions of which only one is global, as shown in Fig. 2. Consider this region [0,1] of x_1 represented by m bits $(1,\dots,m_i)$, then assume we have two types of schemata: H0= (0 * * . . .*) representing the left region where $x_1 \in [0,0.5]$, and H1= (1 * * . . . *) representing the right region, where $x_1 \in [0.5,1]$. Since we need to find a global optimum solution, we must focus on schema H1 since it represents the region of global solution. Also the same thing for x_2 [12]. However, it is possible that we do not find the region of global optimal solution this way. In such a case, the similarity operator and random generation of a part of chromosomes could help to find a better region.

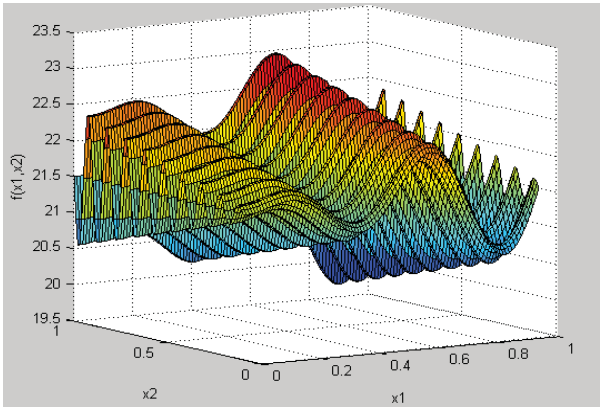

Fig. 2 A part of Zbigniew Michalewicz function

V. Experimental Results

In this section, we report on computational testing (by using the Matlab software) of the DSDSC algorithm on 15 test functions taken from literature. After each test, the result of DSDSC 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 Table VI.

In Table V, we mention all 15 functions with best solutions. We have applied the algorithm with 80 chromosomes with the stopping criterion that the difference between our best solution the and known optimal solution is less than the threshold specified in the second column (Table VI).

The DSDSC algorithm has found optimum solutions for some optimization problems (like Easom, Booth's, Schaffer, Schwefel's, Shubert) that the classical GA cannot solve as shown in Table VI. Column eight shows 0% success rate by using classical GA as it has been tested and also mentioned in [4], [5]. All success rates are 100% with 80 chromosomes for all problems.

The DSDSC algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Note that the maximum number of iterations to found the best solution was especially high (282) for the Schwefel's function, for which the classical GA failed to find a solution as shown in Table VI [4]. Also the maximum number of generations for Michalewicz problem was 280 compared with the classical GA algorithm where it was 396 generations to find the best solution [10]. On the other hand, column three in Table VI shows the minimum number of iterations for finding an optimal solution was between 2 and 9 for all 15 test functions. Column five shows the average number of iterations for all successful runs.

VI. Discussion of Figures

Fig. 3 shows a two-dimensional view of Easom function. It can be seen that the DSDSC algorithm has reached the best solution at the blue point at $f(\pi,\pi) = -1$.

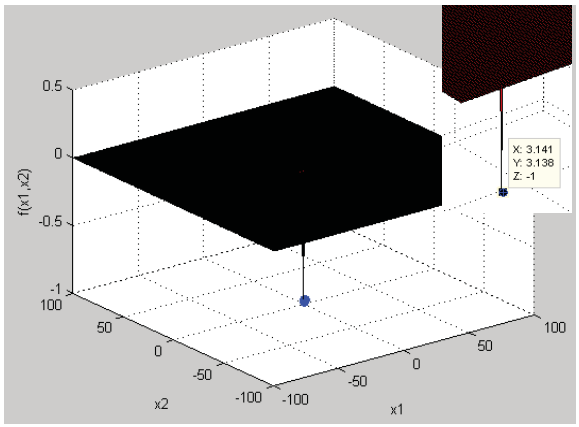

Fig. 3 Solutions of Easom problem

Fig. 4 shows a two-dimensional view of Schaffer's function. It can be seen that DSDSC algorithm has reached the best solution at the blue point on the focus view in the right up corner of the figure. For this function, it is difficult to reach an optimal solution because it contains multi-local minimum solutions near to the best one.

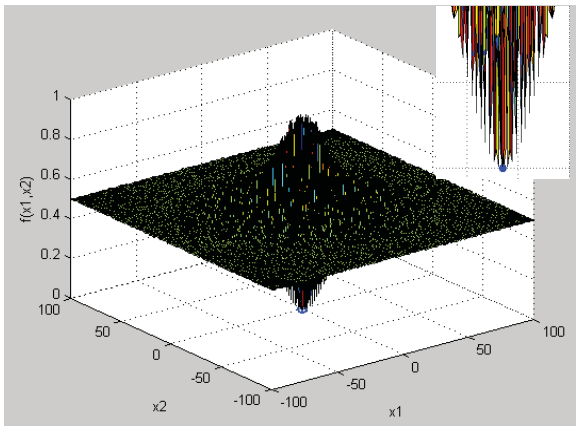

Fig. 4 Solutions of Schaffer's problem

Figs. 5-12 show two-dimensional views, Fig. 5 shows Shubert problem with 18 optimal solution at the blue points with 50 iterations, Branins's problem with 3 optimum optimal solution points, Six-hump camel back problem with two optimum points and Holder Table problem with 4 optimum points, Michalewicz problem, Drop-Wave problem,

Schwefel's problem, Levy N.13 problem with one point optimum solution, respectively.

TABLE V
TEST FUNCTIONS

Function Name	Interval	Function	Global Optimum Min/max
Easom	x,y ϵ [-100,100]	$f(x,y) = -cos(x)cos(y)exp(-(x-\pi)^2 + (y-\pi)^2))$	$f(\pi,\pi) = -1$, *min*
Matyas	x,y ϵ [-10,10]	$f(x,y) = 0.26(x^2 + y^2 - 0.48xy)$	$f(0,0) = 0$, *min*
Beale's	x,yϵ [-4.5,4.5]	$f(x,y) = (1.5 - x - xy)^2 + (2.25 - x + x\,y^2)^2 + (2.625 - x - xy^2)^2$	$f(3{,}0{,}5) = 0$, *min*
Booth's	x,y ϵ [-10,10]	$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$	$f(1{,}3) = 0$, *min*
Goldstein–Price	x,yϵ [-2,2]	$f(x,y) = (1 + (x + y + 1)^2\,(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)) * (30 + (2x - 3y)^2\,(18 - 32x + 12x^2 + 48y - 36xy + 27y^2))$	$f(0,-1) = 3$, *min*
Schaffer N.2	x,y ϵ [-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 ϵ [-500,500]	$f(x) = \sum_{i=1}^{n} -x_i\,.\,sin\left(\sqrt{\lvert x_i \rvert}\right)$	$f(1,1) = 0$, *min*
Branins's rcos	x_1 ϵ [-5,10] x_2 ϵ [0,15]	$f(x_1,x_2) = a \cdot (x_2 - b \cdot x^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot cos(x_1) + e$ $a = 1, b = \frac{5.1}{4 \cdot \pi^2}, c = \frac{5}{\pi}, d = 6, e = 10, f = \frac{1}{8 \cdot \pi}$	$f(\pi, 2.275)$ *or* $f(9.42478{,}2.475)$ *or* $f(-\pi, 12.275) = 0.397887$, *min*
Six-hump camel back	x_1 ϵ [-3,3] x_2 ϵ [-2,2]	$f(x_1,x_2) = \left(4 - 2.1x_1^{4/3}\right) \cdot x_1^2 + x_1x_2 + (-4 + 4x_2^2) \cdot x_2^2$	$f(-.0898{,}0.7126) = -1.0316$, *min*
Shubert	x_1,x_2 ϵ [-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 ϵ [0,10]	$f(x_1,x_2) = (x_1 - x_2)^2 \cdot ((x_1 + x_2 - 10)/3)^2$	$f(5{,}5) = 0$, *min*
Zbigniew Michalewicz	x_1 ϵ [-3,12.1] x_2 ϵ [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*, [11]
Holder table	x,y ϵ [-10,10]	$f(x_1,x_2) = -\lvert\, sin(x)\, cos(x)\, exp\left(\lvert 1 + \frac{\sqrt{x_1^2 + x_2^2}}{\pi} \rvert\right) \rvert$	$f(8.05502, 9.66458)$ *or* $f(8.05502, -9.66458)$ *or* $f(-8.05502, 9.66458)$*or* $f(-8.05502, -9.66458) =$ -19.2085, *min*
Drop-wave	x,y ϵ [-5.12,5.12]	$f(x_1,x_2) = -\frac{1 + \cos(12\sqrt{x_1^2 - x_2^2})}{0.5(x_1^2 + x_2^2) + 2}$	*f(0.0)=-1,min*
Levy N. 13	x,y ϵ [-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,min*

TABLE VI
BEST VALUE OF FUNCTIONS FOR 50 RUNS OF THE DSDSC ALGORITHM (80 CHROMOSOMES)

Function name	Threshold of best solution	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 DSDSC	Rate of success classical GA
Easom	0.001	6	238	**51***	-0.99553	100%	0% [4]
Matyas	0.001	2	**28**	**11**	0.0004317	100%	70% [5]
Beale's	0.001	5	166	**49**	0.000634	100%	6% [6]
Booth's	0.005	4	65	**20**	0.002966	100%	0% [5]
Goldstein–Price	0.001	4	85	**34**	3.00036	100%	72% [4]
Schaffer N.2	0.001	4	189	**71**	3.91E-05	100%	0%
Schwefel's	0.01	6	**282**	**41**	0.004635	100%	0% [4]
Branins's rcos	0.01	5	203	**28**	0.398947	100%	100%
Six-hump camel back	0.001	5	127	**18**	-1.03129	100%	98% [7]
Shubert	1	3	67	**19**	-185.835	100%	0% [4]
Martin and Gaddy	0.001	4	38	**15**	4.54E-05	100%	1% [4]
Zbigniew Michalewicz	0.04	9	280	**67**	38.81046	100%	73% [8]
Holder table	0.001	3	45	**12**	-19.2036	100%	78% asynchronous EA [9]
Drop-wave	0.001	4	172	**48**	-0.9954	100%	30%
Levy N. 13	0.001	5	202	**45**	0.000472	100%	70%

* For Easom function we use R between 4 and 8 instead of 3 and 8.

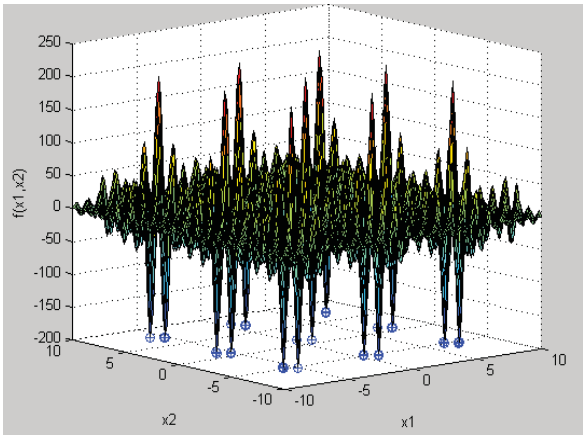

Fig. 5 Solutions of Shubert problem 18 optimum solution

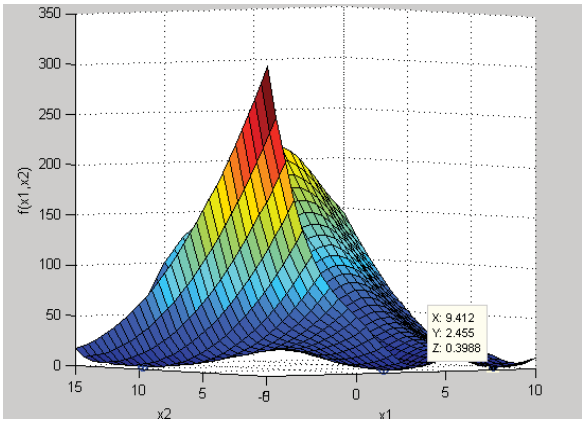

Fig. 6 Solutions of Branins's problem

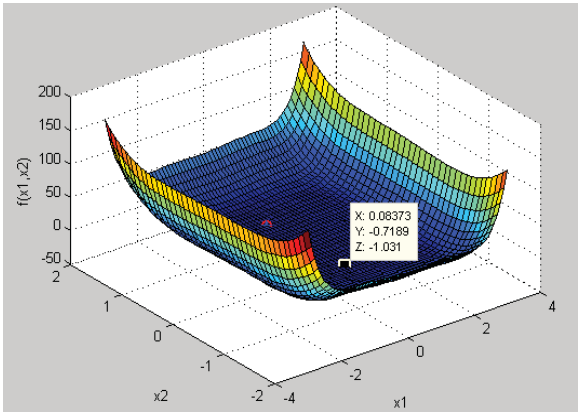

Fig. 7 Solutions of Six-hump camel back problem

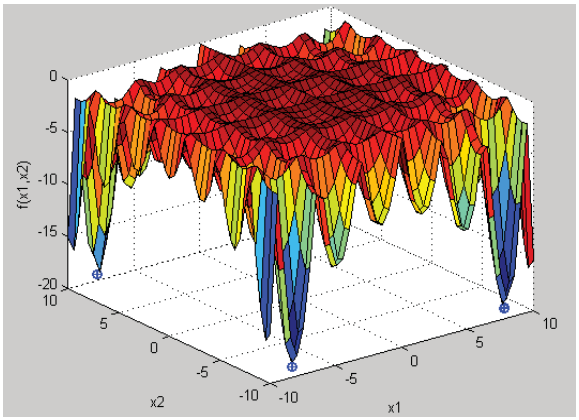

Fig. 8 Solutions of Holder-table problem

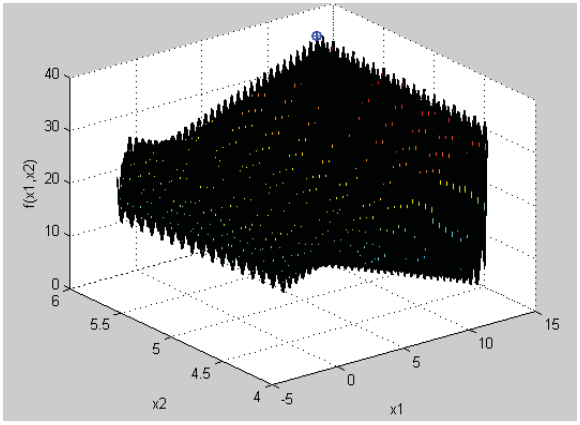

Fig. 9 Solutions of Michalewicz problem

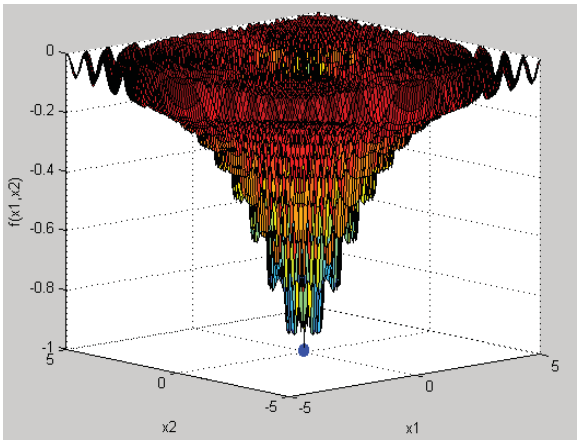

Fig. 10 Solutions of Drop-wave problem

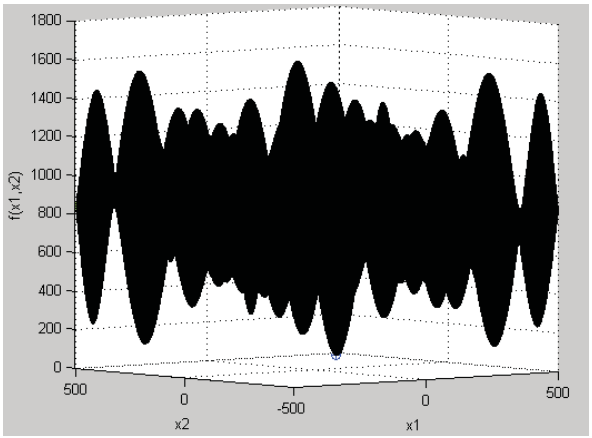

Fig. 11 Solutions of Schwefel's problem

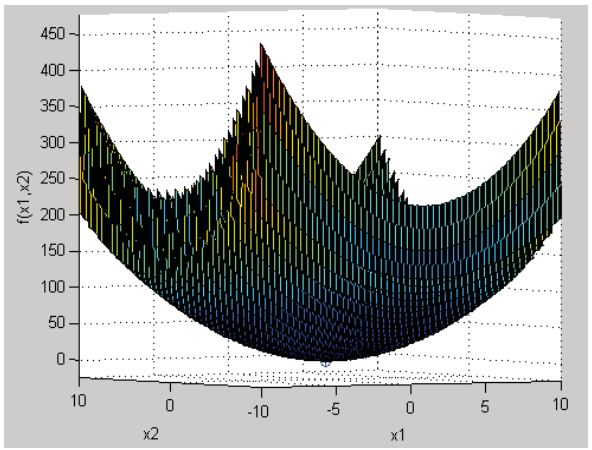

Fig. 12 solutions of Levy N.13 problem

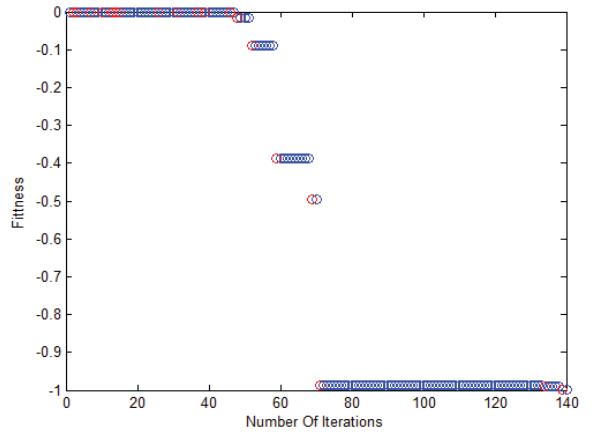

Fig. 13 Finding the best solution for Schaffer problem in 140 iterations

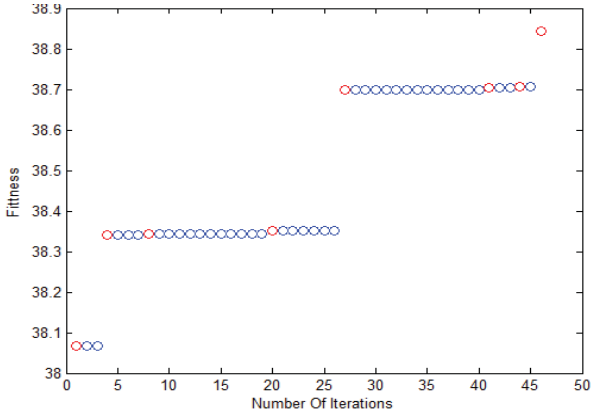

Fig. 14 Finding the best solution for Michalewicz problem in 47 iterations

Figs. 13, 14 show how the best fitness values of the population evolve with the number of iterations. Here the red color means jumping to a better solution, blue color means keeping the best solution from the previous iteration. Fig. 13 shows that for the Schaffer problem we have found the best solution in 140 iterations and Fig. 14 shows that for the Michalewicz problem we have found the best solution in 47 iterations.

VII. Conclusion

In this paper, a new meta-heuristic optimization algorithm called DSDSC is introduced. DSDSC can be simply implemented, without too many parameters. It includes three genetic operators (the dynamic schema, dissimilarity and similarity operators), population sorting and random generation of a part of the population.

The experiments have shown quick convergence and the good global searching ability of our algorithm. The DSDSC algorithm is easy to understand and uses a simple classical representation of points in $\mathbb{R}^n$.

The DSDSC 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 our examples, 80 chromosomes are enough to solve the problem. As Table VI shows, the rate of success of our algorithm is much better than for the classical GA that has a lot of parameters.

Acknowledgments

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

References

[1] K. Manda, S. C. Satapathy, and B. Poornasatyanarayana, "Population based meta-heuristic techniques for solving optimization problems : A selective survey, International Journal of Emerging Technology and Advanced Engineering IJETAE", vol. 2, no. 11, 2012.

[2] Y. Yu and Z. H. Zhou, "A new approach to estimating the expected first hitting time of evolutionary algorithms", *Artif. Intell.*, vol. 172, no. 15,

pp. 1809–1832, 2008.
[3] X. Han, Y. Liang, Z. Li, G. Li, X. Wu, B. Wang, G. Zhao, and C. Wu, "An Efficient Genetic Algorithm for Optimization Problems with Time-Consuming Fitness Evaluation", *Int. J. Comput. Methods*, vol. 12, no. 01, p. 1350106, 2015.
[4] A. S. Eesa, A. Mohsin, A. Brifcani, and Z. Orman, "A New Tool for Global Optimization Problems - Cuttlefish Algorithm", International Journal of Mathematical, Computational, Natural and Physical Engineering, vol. 8, no. 9, pp. 1203–1207, 2014.
[5] A. Ritthipakdee, A. Thammano, N. Premasathian, and B. Uyyanonvara, "An Improved Firefly Algorithm for Optimization Problems", ADCONP, Hiroshima, no. 2, pp.159-164, 2014
[6] J. Town, E. Sciences, and A. K. B. Road, "A Novel Function Optimization Approach Using Opposition Based Genetic Algorithm with Gene Excitation", International Journal of Innovative Computing, Information and Control, vol. 7, no. 7, pp. 4263–4276, 2011.
[7] J. B. Odili, M. Nizam, and M. Kahar, "Numerical Function Optimization Solutions Using the African Buffalo Optimization Algorithm (ABO)", British Journal of Mathematics & Computer Science, vol. 10, no. 1, pp. 1–12, 2015.
[8] G. Mitsuo, Invited Talk: Network Models and Optimization : moGA Network Models and Optimization , Graduate School of Information, Production and Systems, WASEDA University, March, 2009.
[9] E. O. Scott and K. A. De Jong, "Understanding Simple Asynchronous Evolutionary Algorithms", In: FOGA '15 Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, pp. 85-98.
[10] Michalewicz Z., "Genetic Algorithms + Data Structures = Evolution Programs (3ed).PDF." Springer, Berlin,1996.
[11] R. Mahmod, "Maintaining diversity for genetic algorithm: a case of timetabling problem", Jurnal Teknologi (Universiti Teknologi Malaysia) vol. 44, no. D, pp. 123–130, 2007.
[12] Deb K., "Multi-Objective Optimization Using Evolutionary Algorithms", F. Edition, Chichester, U.K.: Wiley, 2001.