I.J. Engineering and Manufacturing, **2017, 6, 1-11**
Published Online November 2017 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijem.2017.06.01

Available online at http://www.mecs-press.net/ijem

A Novel Genetic Operator for Genetic Folding Algorithm: A Refolding Operator and a New Genotype

Mohammd A. Mezher [a], Maysam F. Abbod [b]

[a] *Computer Science Department, Fahad Bin Sultan University, Tabuk, KSA*
[b] *Electronic and Computer Engineering Department, Brunel University London, UK*

Received: 03 February 2017; Accepted: 28 March 2017; Published: 08 November 2017

Abstract

Genetic Folding algorithm uses linear chromosomes composed of organized genes in floating-numbers manner, in which each genes chain fold back on themselves to form the final GF chromosome. In this paper, a novel genotype representation and a novel genetic operator were proposed. The paper was applied using MATLAB code to illustrate the beneficiary, flexibility and powerful of the Genetic Folding algorithm solving Santa Fe Trail problem. The problem of programming an artificial ant to follow the Santa Fe Trail is used as an example of program search space.

To evaluate the efficiency and feasibility of the proposed methods, a comparison was held between the various types and sizes through the Santa Fe Trail problem. Several test functions along with various levels of difficulty were also conducted. Results of this proposal clearly show significant results of the proposed genotype and the genetic operator also.

Index Terms: Genetic Folding Algorithm, genotype representation, refolding operator, Evolutionary Algorithm, Genetic Programming, Genetic Algorithm, GF, GPLab.

1. Introduction

The Evolutionary Algorithms are optimization and search techniques based on the principles of genetics and natural selection. Genetic Folding Algorithm (GF) is a member of the evolutionary algorithms as it uses population of individuals, elitism of chromosome and the reproduction operators. Genetic Algorithm (GA), Genetic Programming (GP) and recently Gene Expression Programming [10] (GEP) are all members of the

* Corresponding author. Tel.: 00966581681999
E-mail address: mmezhr@fbsu.edu.sa

evolutionary algorithms. However, the essential difference among these algorithms reside on the nature of the chromosomes that being generated. GF algorithm presents a new way of representing problems by generating a linear floating number folded back nonlinearly on themselves.

Like in biology, GF algorithm mimics the architecture of RNA/DNA “secondary structure” which the helices of chains are represented in the GF algorithm as terminal and nonterminal. GF therefore, mimics the RNA/DNA folding mechanisms as it allows each gene in the chromosome mapped with a complementary gene in the same chromosome. [1]

During the folding process, GF algorithm bound genes using different sort of mathematical operators, logical operators or user-defined operators. Each GF strings fold back on themselves to create chromosomes as feasible programs. In these folded chromosomes, each gene carry the corresponding information. GF life cycle operates on a user-defined number of generations. Each population contains on a user-defined number of populations of potential non-linear mapping solutions. Each GF chromosome may undergo by a set of reproduction operators that lead in generating new population for the next generation. This population contains on several potential chromosomes evaluated each time by a fitness functions which applying the principal of survival of the fittest.

In GF, the linear chromosomes work as the genotype and the parse trees as the phenotype, creating a genotype/phenotype system. This genotype/ phenotype system is efficient in thus encoding a very length parse tree in each chromosome. This means that the computer programs created by GF are composed of a very length parse tree. GF algorithm shown an effective strategy in various types of computer problems such as binary and multi-classification and regression datasets. For example, GF for binary classification [7], multi-classification [5] and regression [6] have demonstrated how GF used to derive superior results in comparing to other members of the evolutionary algorithm’s family.

Fig. 1. shows the general GF life cycle starting from the initializing step up to the refolding operator process passing by the life cycle stages. In the encoding/decoding process the GF chromosome is encoding every gene in the GF approach where every gene has numbers separated by dots. Further details in the GF chromosomes are given in the next section.

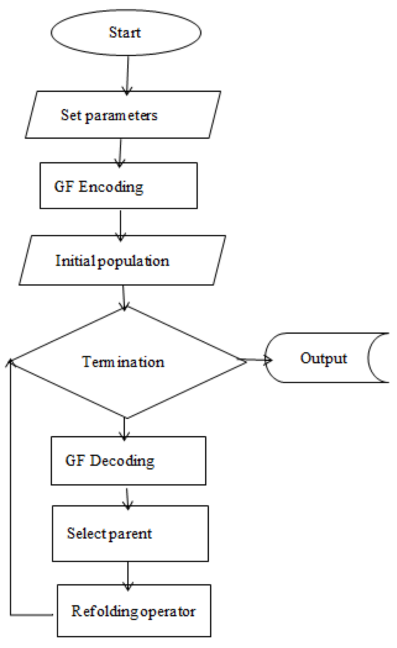


Fig.1. Genetic Folding Life Cycle

2. Genetic Folding Chromosome

RNAs are full of short sequences that are "accidentally" complementary, and RNA chains fold back on themselves to form helices. In general, the chromosomes (individuals) represent set of genes, which represent the code of the dependent mapping genes. For this, every chromosome symbolizes a folding solution of the given problem. From other hand, the genes hold floating-numbers that represent a meaning related to the next gene. A set of different chromosomes forms a population that runs through several generations.

In this paper, GF can create computer programs or models by a simple linear string. Although computer programs may be complex tree structures GF could represent the program by learning and adapting their sizes. In [4-8], all papers addressed the GF chromosome, as it is the standard GF chromosome genotype format. However, in this paper we introduced GF algorithm for a new sort of genotype which being used for three functions and three terminals including three string styles. Table I shown the functions and terminals were applied in the implementation. Where Table II shows an example of three genotype styles. Each style has a different meaning which either composed of three numbers, two numbers or one number depending on the arity of that defined GF decoding string.

Basically, the GF chromosome was divided into two segments; a head segment, which contains on the functions only and a tail segment, which contains on the terminals only. However, the size of the head segment must be determined beforehand but for the size of tails segment no need as the GF algorithm predict the number of genes required based upon the arity required for the drawn functions (shown in Table I). Though, to draw new chromosomes using GF algorithm, every time the GF algorithm generates several functions randomly at first the GF algorithm will predict again the number of terminals required. Simply, GF uses the equations of (1) and (2).

Assume GF algorithm generate randomly the following two genes for the head segment:

Antprogn3 *Antif*

Then, GF algorithm will reserve five rooms for the corresponding two functions drawn earlier. Therefore, the new GF chromosome will be something like:

Antprogn3 *Antif* <u>AntMove</u> <u>AntRight</u> <u>AntLeft</u> <u>Antright</u>

Therefore, every time GF needs to find the number of tail required for a chromosome will use the following equation:

$$\text{GF\_Tail} = \text{number\_of \_Arity} - 1 \tag{1}$$

Likewise, for finding the full length of a GF chromosome we use the equation in (2) and substituting the value of term in (1):

$$\text{GF\_Length} = \text{number\_of\_functions} + \text{GF\_Tail} \tag{2}$$

Table 1. Type of Functions and Terminals

Type of operator	**Name of steps**	**No of Arity**
Function	Antif	2
	antprogn2	2
	antprogn3	3
Terminal	Antright	1
	Antleft	1
	Antmove	1

The genome of GF consists of a linear, symbolic string or chromosome of fixed length composed of equal size. These genes, despite their fixed length, code for floating-numbers strings. An example of a chromosome with different arity is the following string: (Notably, the tail was formatted as underlined style where the head was formatted in an italic style.)

Table 2. GF Encoding Example

Index	**1**	**2**	**3**	**4**	**5**	**6**
GF (String) Decoding	*Antprogn3*	*Antif*	Antmove	Antmove	Antmove	Antmove
Arity	3	2	1	1	1	1
GF Encoding	2.4.5	3.6	3	4	5	6

Where the first row represents the index number of each gene in the chromosome. The second row represents the arity of a GF chromosome composed of functions or terminals were randomly generated. The third row is for the arity each type of operator required.

As GF chromosome composed of both head and tail segments. In the above example, only functions were generated with a size of three and two arities respectively. The tail size thus was generated using the equation in (1). For both encoding and decoding processes, GF algorithm generates in every generation several populations of chromosomes contain on a random helix coding numbers. For the helix coding numbers generated in each cell of the chromosome we call this type of relationship a Helix-language or in short H-Language. Decoding GF genes to expression tree is very simple and a straightforward procedure. For example, the GF chromosome shown in Table 2 will be presented using GF tree expression as follows:

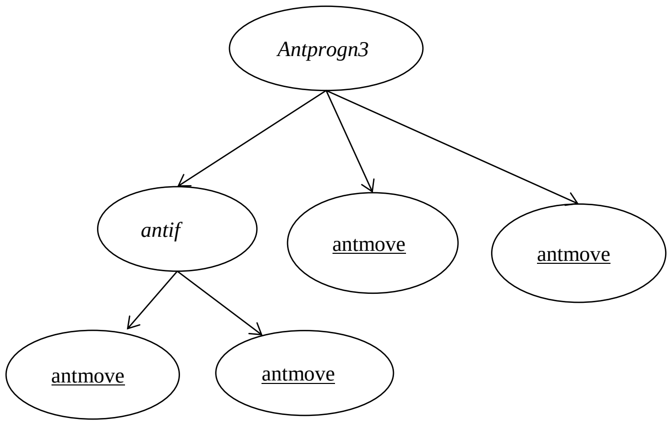


Fig.2. Expression Tree of Genetic Folding Chromosome

3. Genetic Folding Operator

To comprehend the EA as a whole, it is necessary to understand the role of a gene representation and genetic operators. The two most commonly employed genetic search operators are crossover and mutation. Crossover produces offspring by recombining the information from two parents. While mutation prevents convergence of the population by flipping a small number of randomly selected bits to continuously introduce variation. A genetic operator is a process used in GAs to maintain genetic diversity [4].

The GF algorithm is a flexible technique in both genotype and phenotype based on the principals of genetics representation and genetic operators. The genetic operators proposed here allow a fittest population to specify genetic folding programs (chromosome) that maximizes the fitness function. By means of evolutionary operators, here a refolding operator of an offspring population is introduced for the first time to maintain genetic diversity.

The refolding operator is a genetic operator that creates from the parent pool a chromosome by folding its genes on themselves to produce a new chromosome in the offspring pool. However, the refolding operator combines genes from a selected parent and produced new offspring including the folded genes. The idea behind refolding operator is that each offspring chromosome will keep folding on itself until an optimum folding found. And by this GF chromosomes will have a modification chance every time the chromosome fold again and again over the time.

The powerful idea behind the refolding operator proofed to replace both operators and found reasonable results. The refolding operator can redraw different possibilities of chromosomes out of one GF chromosome. By this the GF algorithm will map all genes with its complementary genes for a numerous number of combinations and for finding out the best folding format.

In the way of folding one gene into another gene a plenty of folding results may obtain based upon the length of the chromosome. However, each gene of parent will be folding back with its corresponding place of the complementary gene within the inside chromosome as sown below. Now, assume the refolding operator is occurred for the following GF individual. Let an individual holds a string of: antprogn3(antif(antmove,antmove), antmove,antmove) with a fitness value of 3.

1	**2**	**3**	**4**	**5**	**6**
Antprogn3	*Antif*	<u>Antmove</u>	<u>Antmove</u>	<u>Antmove</u>	<u>Antmove</u>
2.6.5	4.3	3	4	5	6

Consider now, the result of the refolding operator works on the GF individual as one of possible folding results as:

1	**2**	**3**	**4**	**5**	**6**
Antprogn3	*Antif*	<u>Antmove</u>	<u>Antmove</u>	<u>Antmove</u>	<u>Antmove</u>
2.6.3	5.4	3	4	5	6

Which gave the string of antprogn3(antif(antmove, antmove), antmove,antmove) with a fitness value of 3.

4. Related Works

In the ant trail problem, the task is to direct an ant moving on a virtual plane so that it collects maximum number of foods distributed randomly on the plane. The widely-used benchmark in evolutionary algorithms is the Santa Fe problem [2][3][11]. However, the techniques used to solve such benchmark could be divided into three major techniques; Fitness evaluations, Grammatical Evolutions, and ants' energies. Some researches provide modified algorithms to generate programs or subprograms to study empirically the effects of various fitness metrics on training and testing performance of the Santa Fe problem [12][13]. The evolutionary methodologies were adopted also in various research using the known ant's energy levels in literature with different initial range of energy [14]. Many other techniques within the field of GP is Grammatical Evolution. GE is a grammar-based form of GP which performed a mapping from a linear genotype to phenotypic GP trees.

5. Ant Colony Optimization

Before we present our results, we will explain in a brief the problem we are facing to resolve using the refolding operator. In general, the artificial ant must follow the "Santa Fe trail", which consists of 32*32 squares. The Santa Fe Trail problem is a good exercise in which there are 89 food units distributed non-uniformly along it. Every time the artificial ant enters a square containing food eat it per a programmed set of instructions. The amount of food eaten is used as the fitness measure of the control program. [3] However, each H-language of a GF chromosome was finally converted to a tree program by labeling its nodes with a function or terminal of the correct arity chosen uniformly at random.

However, the Santa Fe Trail problem is a well-known model problem that has been studied over the past two decades and is still being used as a GP benchmark. The layout of food pellets in the Santa Fe Trail problem has become a standard for comparing different EAs. This problem is known for its status of being "hard" by quality of evolutionary computing methods not solving it. The hardness has been attributed to a fitness landscape that is difficult to found. [2]

6. Experimental Results

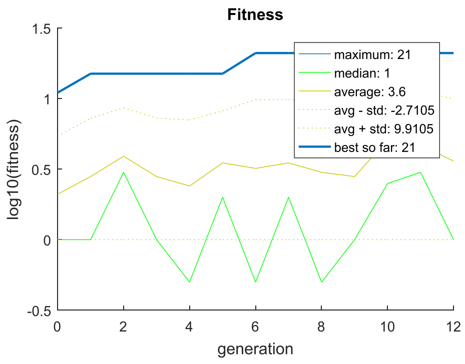
The GF algorithm was tested on various numbers of parameters to verify its ability of finding good results. The GF algorithm was identically relying on the GPlab toolbox [9] in drawing GF tree and in some other drawing features. Although, GF algorithm was conducted in a wide range of population and generation sizes, the refolding operator could find an optimum result at small sizes. Table 3 shows the maximum fitness values found in percentages of different sizes accompanied for a convenience reading.

Table 3. GF Algorithms Results for Different Sizes

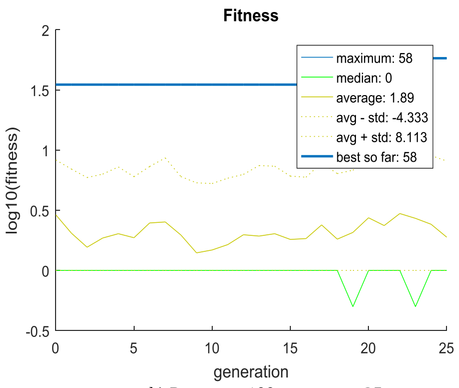
	Pop. Size	Gen.	Best found	Fitness	Depth	Nodes
Small	10	12	6	21/89	7	31
Small 2	100	25	22	58/89	6	16
Mid	200	50	49	62/89	9	31
Large	400	100	42	54/89	6	15
Large 2	400	200	72	62/89	7	15

In Fig. 3, GF was tested again on five different sizes of both generations and populations. Fig.3 (e) and (c) were an example of medium and high number of populations. Even though GF algorithm could find the best ant pathway with a highest number of pallets not after 49 and 72 generations in both figures respectively (see Table III). However, Fig. 3 (d) was unsuccessful finding the highest number of pallets to be eaten in comparing to the artificial ant shown in the Fig. 3 (c).

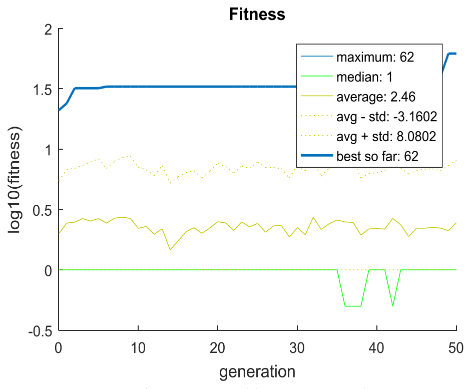
In the Appendix, we enclosed other examines and results in full details. The appendix shows results of different sizes conducted in the paper such as; the structural complexity of the GF folded chromosomes, best GF chromosome drawn as a tree structure and the ant pathways that each artificial ant passed through comparably.



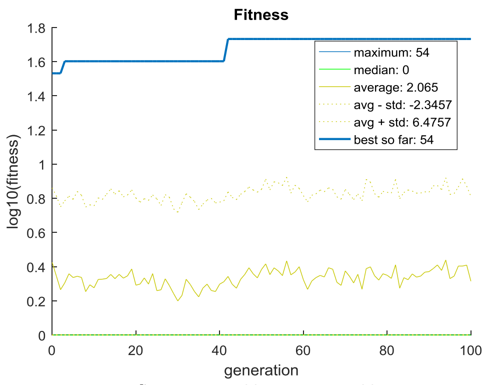
a) Pop no. = 10, gen. no. = 12,



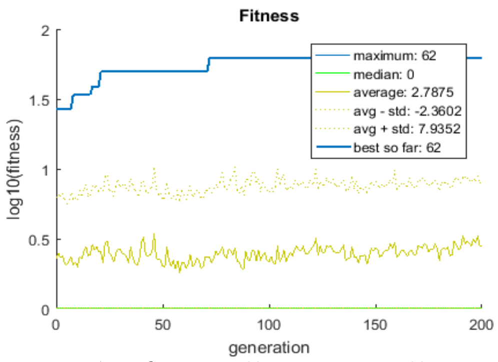
b) Pop no. = 100, gen. no. = 25



c) Pop no. = 200, gen. no. = 50



d) Pop no. = 400, gen. no. = 100



e) Population no. =400, generation no. = 200

Fig.3. Best Fitness Values for Different Sizes

7. Conclusion and Future Works

The increasing importance of Genetic Folding algorithm in solving NP-problem motivated us of developing a new genetic operator in contrast to the conventional genetic operators. The GF algorithm shows significant results of finding a high number of pallets for the artificial ants in aids of using the refolding operator proposed here. The genotype proposed here was also sufficient for the problem in hand to predict the best chromosome structure even for a small and medium population size.

Refolding operator introduced within a novel genotype as a simple and easy to implement operator, yet fast and powerful in reproducing new population. Instead of using a self-adaptive operator or using more than one of the traditional genetic operators the refolding operator may replace them all.

Appendix A

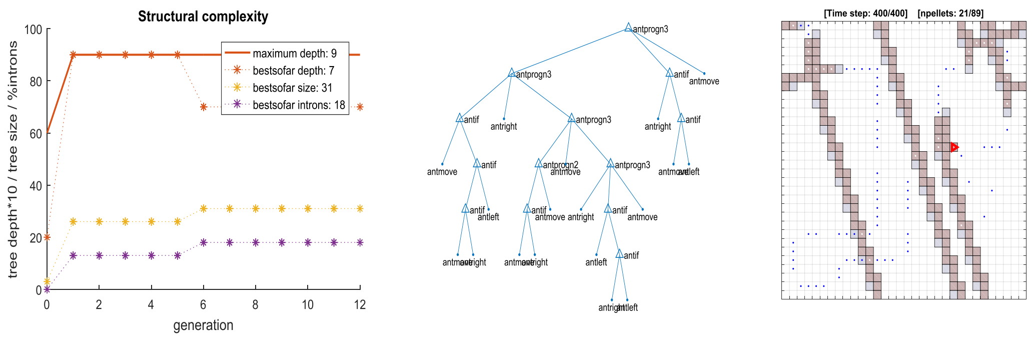


Fig.4. Best Result of Population no. = 10 and Generation no. = 12

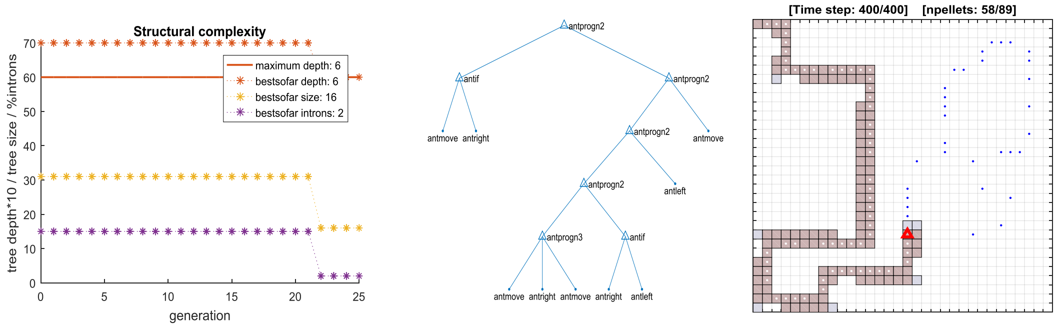


Fig.5. Best Result of Population no. = 100 and Generation no. = 25

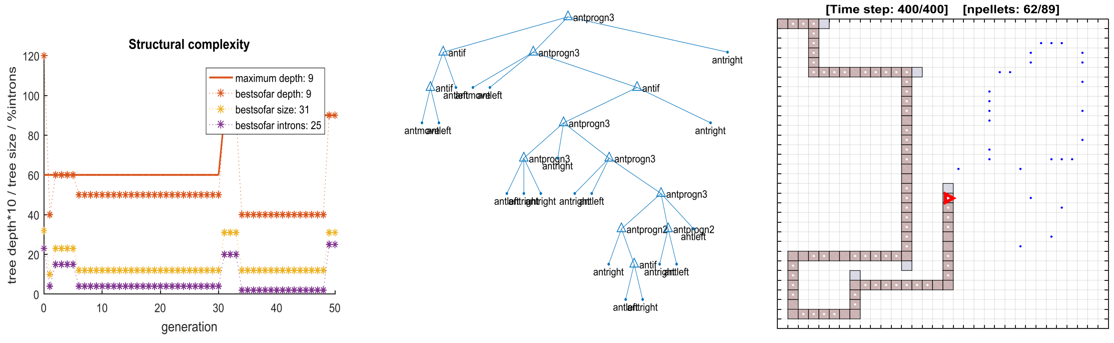


Fig.6. Best Result of Population no. = 200 and Generation no. =50

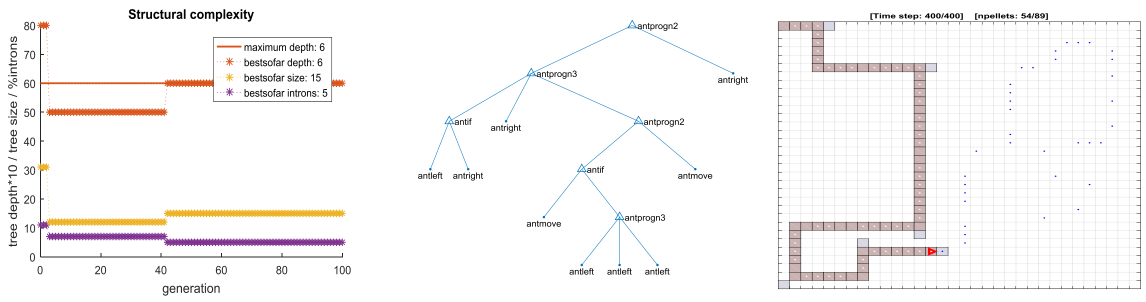


Fig.7. Best Result of Population no. =400 and Generation no. = 100

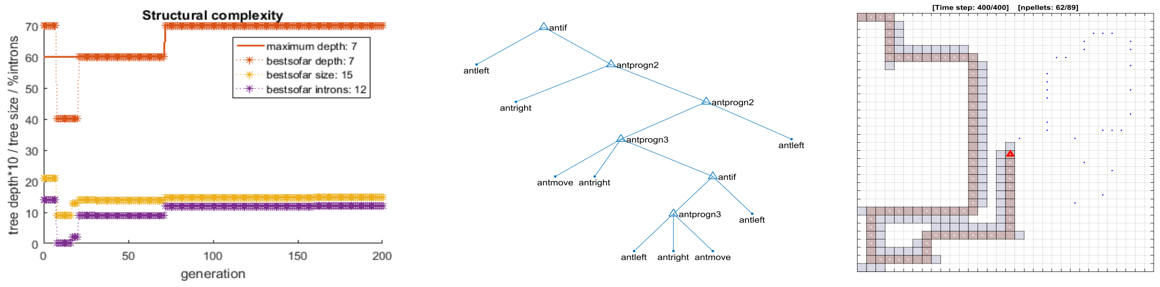


Fig.8. Best Result of Population no. =200 and Generation no. = 400

Table 4. Best Result of Population no. = 10 and Generation no. = 12

GF String	antprogn3(antprogn3(antif(antmove,antif(antif(antmove,antright),antleft)),antright,antprogn3(antprogn2(antif(antmove,antright),antmove),antmove,antprogn3(antright,antif(antleft,antif(antright,antleft)),antmove))),antif(antright,antif(antmove,antleft)),antmove)
GF chromosome	[13.8.24][17.18][19.23][30.14][29.7][9.21.12][4.20][28.11][3.16][31.2][25.26][22.10.15][5.27.6][14][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31]

Table 5. Best Result of Population no. = 100 and Generation no. = 25

GF String	antprogn2(antif(antmove,antright),antprogn2(antprogn2(antprogn2(antprogn3(antmove,antright,antmove),antif(antright,antleft)),antleft),antmove))
GF chromosome	[4.10][11.27][26.15][23.32][31.7.24][20.30][28.13][8.22][9.25][2.21][12.3][18.29.16][14.33.17][19.5.6][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31][32][33]

Table 6. Best Result of Population no. = 200 and Generation no. = 50

GF String	antprogn3(antif(antif(antmove,antleft),antleft),antprogn3(antmove,antleft,antif(antprogn3(antprogn3(antleft,antright,antright),antright,antprogn3(antright,antleft,antprogn3(antprogn2(antright,antif(antleft,antright)),antprogn2(antright,antleft),antleft))),antright)),antright)
GF chromosome	[11.8.30][20.14][17.24][27.23.9][10.13][19.3][29.22][26.21.5][6.2.15][12.28.4][7.25][18.31.16][13][14][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31]

Table 7. Best Result of Population no. = 400 and Generation no. = 100

GF String	antprogn2(antprogn3(antif(antleft,antright),antright,antprogn2(antif(antmove,antprogn3(antleft,antleft,antleft)),antmove)),antright)
GF chromosome	[13.22][17.31.30][21.32][29.7][10.6][18.16][23.28.26][14.8][4.27][5.24][19.20][25.12][11.15.9][2.3][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31][32]

Table 8. Best Result of Population no. = 400 and Generation no. = 200

GF String	antif(antleft,antprogn2(antright,antprogn2(antprogn3(antmove,antright,antif(antprogn3(antleft,antright,antmove),antleft)),antleft)))
GF chromosome	[31.9][4.17][19.25][32.24.12][21.22.18][27.28][7.5][29.11][23.2][6.15][13.3][14.16][8.10][26.30.20][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31][32]

References

[1] Peter B. Moore, The RNA World, 2nd Ed.: The Nature of Modern RNA Suggests a Prebiotic RNA World. Pages 381-401. Volume 37. 1999

[2] Dominic Wilson, Devinder Kaur, How Santa Fe Ants Evolve. Neural and Evolutionary Computing. 2014

[3] W. B. Langdon and R. Poli. Why Ants are Hard. Genetic Programming, Pages 193-201. 1998

[4] Mohammad Mezher, Maysam Abbod. Genetic folding: Analyzing the mercer's kernels effect in support vector machine using genetic folding. World Academy of Science, Engineering and Technology. Pages 1342 – 1347. Volume 5. 2011.

[5] Mohammad Mezher, Maysam Abbod. Genetic folding for solving multiclass SVM problems. Applied Intelligence Journal. Pages 464-472. Volume 41. 2014.

[6] Mohammad Mezher, Maysam Abbod. A new genetic folding algorithm for regression problems. Computer Modeling and Simulation (UKSim), UKSim 14th International Conference On. Pages 46-51. 2012.

[7] Mohammad Mezher, Maysam Abbod. Genetic Folding: A New Class of Evolutionary Algorithms. Research and Development in Intelligent Systems. Springer. Pages 279-284. 2011.

[8] Mohammad Mezher. Genetic Folding Algorithm: An Introduction to a New Evolutionary Algorithm. LAP Lambert Academic Publishing. 2012.

[9] Sara Silva, Jonas Almeida. GPLAB-a genetic programming toolbox for MATLAB. Proceedings of the Nordic MATLAB conference. Pages 273-278.

[10] Candida Ferreira. Gene Expression Programming: A new Adaptive Algorithm for Solving Problems. Complex Systems, Vol. 13, issue 2. 2001.

[11] Kushchu, I. Genetic programming and evolutionary generalization. IEEE transactions of Evolutionary Computation Vol. 6 issue 5. 2002.

[12] Lehman, J. Stanley, K. Exploiting open-endedness to solve problems through the search for novelty. Proceedings of the international conference on artificial Life. MIT press, Cambridge. 2008.

[13] J. Doucette, M. 1. Heywood, "Novelty-based fitness: An evaluation under the santa fe trail", Genetic Programming 13th European Conference EuroGP 2010 ser. Lecture Notes in Computer Science, pp. 50-61, 2010.

[14] D. Oghorodi, P. Asagba. Determining an Optimal Energy Level of the Artificial Ant in the Classical Santa Fe Artificial Ant Problem on the platform of Genetic Programming. African Journal of Computing and ICT. Vol 8 issue 2. 2015.

Authors' Profiles

Mohammad A. Mezher is assistant professor and chair of the department of computer science at Fahad Bin Sultan University, where he has been since 2013. He received a BSc. From Alzaytoonah University in 2004, and MSc. From the Sains Malaysia University in 2006. He received his Ph.D. in Computer Science from the University of Brunel in 2011. Much of his work has been on improving the evolutionary algorithms, mainly Genetic Folding Algorithm. Where he has invented and introduced GF for the first time in 2010 at SAGAI, Cambridge, UK.

Dr Maysam F. Abbod (MIEE, CEng) He received BSc degree in Electrical Engineering from University of Technology in 1987. PhD in Control Engineering from University of Sheffield in 1992. From 1993 to 2006 he was with the Department of Automatic Control and Systems Engineering at the University of Sheffield as a research associate and senior research fellow. He is recently a senior lecturer in Intelligent Systems and a Course Director for EEE programmes at Brunel University.

How to cite this paper: Mohammd A. Mezher, Maysam F. Abbod,"A Novel Genetic Operator for Genetic Folding Algorithm: A Refolding Operator and a New Genotype", International Journal of Engineering and Manufacturing(IJEM), Vol.7, No.6, pp.1-11, 2017.DOI: 10.5815/ijem.2017.06.01