Evolutionary Algorithm Based on Automatically Designing of Genetic Operators

Dazhi Jiang, Chenfeng Peng
Department of Computer Science
Shantou University
Shantou, China
{dzjiang,11cfpeng}@stu.edu.cn

Zhun Fan
Department of Electronic and Information Engineering
Shantou University
Shantou, China
zfan@stu.edu.cn

***Abstract*—at present there is a wide range of evolutionary algorithms available to researchers and practitioners. Despite the great diversity of these algorithms, virtually all of the algorithms share one feature: they have been manually designed. Can evolutionary algorithms be designed automatically by computer? In this paper, a novel evolutionary algorithm based on automatically designing of genetic operators is presented to address this problem. The resulting algorithm not only explores solutions in the problem space, but also automatically generates genetic operators in the operator space for each generation. In order to verify the performance of the proposed algorithm, comprehensive experiments on 23 well-known benchmark optimization problems are conducted, and the results show that the proposed algorithm can outperform standard Differential Evolution (DE) algorithm.**

Keywords—Evolutionary Algorithm; Automatically Designing; Space of Genetic Operators;

I. INTRODUCTION

At present there is a wide range of evolutionary algorithms available to researchers and practitioners. Despite the great diversity of these algorithms, virtually all of the algorithms share one feature: they have been manually designed. As a result, current evolutionary algorithms inevitably incorporate human biases and preconceptions in their designs.

In recent years, several automatic algorithm design techniques were proposed to overcome this limitation. Hyper-heuristics includes search methods that automatically select and combine simpler heuristics, creating a generic heuristic that is used to solve more general instances of a given type of optimization problem. Hence, hyper-heuristics searches in the space of heuristics, instead of in the problem solution space [1], raising the level of generality of the solutions produced by the hyper-heuristic evolutionary algorithm. Ant Colony algorithms are population-based methods widely used in combinatorial optimization problems. Jorge Taveres and Francisco B. Pereira [2] proposed a grammatical evolution [3] approach to automatically design ant colony optimization algorithms. The grammar adopted by this framework has the ability to guide the learning of novel architectures, by rearranging components regularly found on human designed variants. Furthermore, Jorge Taveres and Francisco B. Pereira [4] proposed a strongly typed genetic programming [5] approach to automatically evolve the communication mechanism that allows ants to cooperatively solve a given problem. For these two applications, results obtained with several TSP instances show that the evolved pheromone update strategies are effective, exhibit a strong generalization capability and are competitive with human designed variants. For rule induction algorithms, Gisele L. Pappa and Alex A. Freitas [6] proposed the use of Grammar-based Genetic Programming (GGP) to automatically evolve rule induction algorithms. The experiments involving 11 data sets show that novel rule induction algorithms can be automatically generated using GGP. Mihai Oltean and Grina Grosan [7] used Multi Expression Programming (MEP) [8] technique to evolve evolutionary algorithm, in which Each MEP chromosome encodes multiple EAs.

Although the aforementioned automatic algorithms have different emphases on research objectives and contents, one thing in common is that they use automatic method to design algorithms, which shows that automatic programming method can build algorithms to solve problems automatically.

As the core components of the evolutionary algorithms, the genetic operators, such as mutation, combination, etc., are more variable and complicated compared with other components, such as initialization, selection, etc. in the algorithm framework. Therefore, more innovation may be achieved if we focus on design of genetic operators. This paper proposes a novel approach to design genetic operators in evolutionary algorithm, namely the evolutionary algorithm based on automatically designing of genetic operators (EA^2DGO), which uses MEP with a new encoding scheme [9] to automatically generate genetic operators in the evolutionary algorithm to solve problems. Organization of this paper is as follows. In Section II, the generic expression of genetic operators is introduced. The framework of EA^2DGO are described in Section III. Experimental verifications are given in Section IV. The last section gives conclusions.

II. THE GENERIC EXPRESSION OF GENETIC OPERATORS

It is important to investigate what expressions of genetic operators are amenable to automatic design, for which we can get inspirations from analyzing a Standard Genetic Algorithm (SGA), Particle Swarm Optimization (PSO) [10] and Differential Evolution (DE) [11].

SGA with the real coding usually adopts arithmetic crossover as one of the genetic operators. Take total arithmetic crossover for example, assume N is a constant number which presents the size of the population, and D is the dimension of the parameter vector. The population P is then expressed as $p = \{X_i(t), i = 1,2,\cdots,N\}$, t is the generation. Select two individuals $X_j(t)$, $X_k(t)$ from the population according to certain rule, where $j,k \in \{1,2,\cdots,N\}$, $j \neq k$. The child vector $X^{(1)}(t)$ and $X^{(2)}(t)$ could be generated by arithmetic operators and expressed as following:

978-1-4799-2548-3/13 $31.00 © 2013 IEEE
DOI 10.1109/CIS.2013.21

$$X^{(1)}(t) = \alpha X_j(t) + (1-\alpha)X_k(t) = X_k(t) + \alpha(X_j(t) - X_k(t)) \quad (1)$$

$$X^{(2)}(t) = \alpha X_k(t) + (1-\alpha)X_j(t) = X_j(t) + \alpha(X_k(t) - X_j(t)) \quad (2)$$

where $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_D\}$ and $\alpha_l \in [0,1]$, $l = 1, 2, \cdots, D$.

PSO, like other evolutionary algorithms, is also a population-based search algorithm and starts with an initial population of randomly generated solutions called particles. Each particle in PSO has a velocity and a position. The velocity V_i^d and position X_i^d of the *d*th dimension of the *i*th particle are updated according to the following equations:

$$V_i^d(t+1) = V_i^d(t) + c_1 * rand1_i^d * (pbest_i^d(t) - X_i^d(t)) + c_2 * rand2_i^d * (gbest^d - X_i^d(t)) \quad (3)$$

$$X_i^d(t+1) = X_i^d(t) + V_i^d(t+1) \quad (4)$$

Where $i = 1, 2, \cdots,$ is the particle's index, $X_i = (X_i^1, X_i^1, \ldots, X_i^D)$ is the position of the *i*th particle; $V_i = (V_i^1, V_i^1, \ldots, V_i^D)$ represents velocity of *i*th particle. $pbest_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^D)$ is the vest previous position yielding the best fitness value for the *i*th particle. $gbest = (gbest^1, gbest^2, \ldots, gbest^D)$ is the best position discovered by the whole population. $rand1_i^d$ and $rand2_i^d$ are two random numbers independently generated within the range of [0, 1], c_1 and c_2 are two learning factors reflecting the weighting of stochastic accelerations terms that pull each particle toward *pbest* and *gbest* positions, respectively. $t = 1, 2, \cdots$, indicates the iterations.

DE is a population-based, direct, robust and efficient search method. Like other evolutionary algorithms, DE starts with an initial population vector randomly generated in the solution space. The main difference between DE and other evolutionary algorithms, such as SGA and PSO, is its new generation vectors generating method. In order to generate a new population vectors, three vectors in population are randomly selected, and weighted difference of two of them is added to the third one. The procedure could be illustrated as following:

Mutation: For each vector *i* from generation *t*, a mutant vector $X_i(t+1)$ is defined by

$$X_i(t+1) = X_{r_1}(t) + F(X_{r_2}(t) - X_{r_3}(t)) \quad (5)$$

Where $i \in \{1, 2, \cdots, N\}$ and $r_1, r_2, r_3 \in [0, N]$, i, r_1, r_2 and r_3 are different.

Through the analysis above, the following observations are made:

i. Genetic operator is a formula which is composed by objects (such as $X_j(t)$ and $X_k(t)$), arithmetic operators (+,-,*) and parameters (such as α, *F*).

ii. The formula representing the genetic operator can have many variants. For example, DE and PSO have similar but different formulas, which is again different from the formula of SGA. For the automatically designing of genetic operators, the most different characteristic compared with traditional genetic operators is that its structure could be reconstructed by computer, which means that the genetic operators could be generated adaptively according to the requirements of problem.

iii. While existing evolutionary algorithms (including SGA, DE, PSO, etc.) have different formulas, they actually share significant commonality. An automatic way of generating novel formula (using the existing objects, arithmetic operators, and parameters) may lead to very novel design of evolutionary algorithm which can adapt itself to address problems with very dynamic and changing nature.

According to the i, ii and iii, we could design a scheme to represent the genetic operator for automatic design. Only consider of Eq.(1). Suppose $X_k(t) = a, X_j(t) = b, \alpha = c$, the Eq.(1) could expressed by an expression tree as in Fig.1.

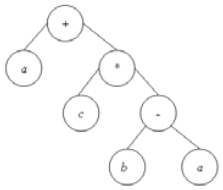

Figure 1. The expression tree of genetic operator Eq.(1) in SGA

In automatic programming, such as GEP (Gene Expression Programming)and MEP, an expression tree is a phenotype of chromosome, and the phenotype could be translated into an equivalent linear genotype. Suppose the length of one chromosome is 7, $T = \{a, b, c\}$ and $O = \{+, -, *\}$, the expression tree in Fig.1 could be expressed as a genotype as follows.

1	2	3	4	5	6	7
+	***a***	*	***c***	-	***b***	***a***
2，3		4，5		6，7		

Figure 2. the equivalent genotype of Eq.(1) in SGA

Each gene in the equivalent genotype encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments. Function arguments always have indices of higher values than the position of the function itself in the chromosome [9]. Phenotype translation is obtained by parsing the chromosome right-left. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol. For simplicity, the expression tree expressed by the first symbol is represented as the chromosome's final presentation.

The chromosome could be reconstructed with no difficulty. Every gene could be changed into another terminal or function symbol, so does the function arguments. When the gene or function arguments are changed, the genotype and phenotype of chromosome are transformed in the meanwhile. If we can design a self-organized framework to adjust the structure of chromosome adaptively in the problem solving, the genetic operators could be designed automatically.

III. EVOLUTIONARY ALGORITHM BASED ON AUTOMATICALLY DESIGNING OF GENETIC OPERATORS

Single-objective optimization problems are adopted to verify the validity of the EA^{2}DGO algorithm. This means that the genetic operators represented by above scheme are used to manipulate the individuals in the population in the problem space, and the goal is to find the problems' global optimal solution. However, in the automatically designing of genetic operators, the genetic operators it not predefined by designer before problem solving. Actually, the genetic operators is searched and designed in the process of problem solving. Thus, the framework of the EA^{2}DGO consists two core components, one is function optimization unit, which is searching in the problem solution space, and, the other one is automatically designing genetic operators unit, which is searching in the genetic operators' space.

1. **Begin**
2. **Input**: *NP*, *NOP*, *F*, *CR*, *OMR*, Max_Fes, *h*, *t*, *times*, *T*, and *O*;
3. *G*=0
4. Create the function optimization population $\vec{x}_G^i,\ \forall i,\ i=1,\cdots,NP$
5. Create the genetic operators population $\vec{o}_G^k,\forall k,\ k=1,\cdots,NOP$
6. $f(\vec{o}_G^i)=0,\forall k,\ k=1,\cdots,NOP$
7. Evaluate $f(\vec{x}_G^i),\forall i,\ i=1,\cdots,NP$
8. **For***G*=1 to *Max_FES* **Do**
9. Find the best $\vec{x}_G^{best}$ in *NP*
10. *Call Function Optimization unit*
11. **If**($rand[0,1)<OMR$) **Then**
12. *Call Automatically Designing Genetic Operators unit*
13. **End If**
14. *G*++
15. **End For**
16. Output $\vec{x}_G^{best}$
17. **End**

Figure 3. The general framework of EA^{2}DGO

The general framework of EA^{2}DGO is given in Fig.3. Where *NP* denotes the size of function optimization population; *NOP* denotes the size of operator generating population; *F* denotes scaling factor; *CR* denotes the probability of crossover; *OMR* denotes the probability of Mutation for operators in operator generating population; *Max_FEs* denotes the max number of function calls; *h* is the of head length of chromosome in new encoding scheme MEP; *t* is the tail length; times denotes the number of repeat times which randomly select individuals for mutation manipulation from population in function optimization unit; *T* is the terminal symbol set; *O* is the function symbol set.

The function optimization unit focuses on the solving of global optimal solution in the problem solution space. The framework is given in Fig.4. According to the framework, we can see that the function optimization unit is very familiar with DE, including the population initialization, crossover manipulation, individual fitness assessment, individual selection, etc. For automatically designing genetic operators, an individual $\vec{o}_G^k$, $k\in[1,NOP]$, is selected from the population according to the Roulette Wheel Selection, and the selected individual will be used in mutation process in the function optimization unit. The GeneCalculate function focuses on the fitness calculate gene by gene. The input includes several individuals selected from function optimization population and one individual selected from genetic operators' population. The specific fitness calculation method could see [11]. After *D* times GeneCalculate, a new candidate $\vec{u}_{j,G+1}^i$ is generated finally.

1. **Begin**
2. Suppose the function optimization population is $\vec{x}_G^i,\ \forall i,\ i=1,\cdots,NP$
3. Suppose the genetic operators population is $\vec{o}_G^k,\forall k,\ k=1,\cdots,NOP$
4. **For** *i*=1 to NP **Do**
5. Select randomly $r_1\neq r_2\neq r_3\neq i,\ r_1,r_2,r_3\in[1,NP]$:
6. Select $\vec{o}_G^k, k\in[1,NOP]$ by Roulette Wheel Selection
7. j_{rand} =randint(1,D)
8. **For** *j*=1 to D **Do**
9. **If**($rand_i[0,1)<CR$ or $j=j_{rand}$) **Then**
10. $u_{i.G+1}^i=\vec{x}_{i.G}^i+$ GeneCalculate($\vec{x}_{i.G}^{r_1},\vec{x}_{i.G}^{r_2},\vec{x}_{i.G}^{r_3},\vec{x}_{i.G}^{best},\vec{o}_G^k$)
11. **Else**
12. $\vec{u}_{i.G+1}^i=\vec{x}_{i.G}^i$
13. **End If**
14. **End For**
15. **If** (*better*($\vec{u}_{i.G+1}^i\ \vec{x}_{i.G}^i$)) **Then**
16. $\vec{x}_{G+1}^i=\vec{u}_{G+1}^i$
17. $f(\vec{o}_G^k)++$
18. **Else**
19. $\vec{x}_{G+1}^i=\vec{x}_G^i$
20. **End If**
21. **End For**
22. **End**

Figure 4. The framework of Function Optimization

In automatically designing genetic operators unit, the most important work is the manipulation and evolution of the genetic operators. Suppose the individual before genetic manipulation is $\vec{o}_G^k$, and the new generated individual after genetic manipulation is $\vec{o}_G'^k$). A proposed way to assess the candidate $\vec{o}_G'^k$ in this paper is: within a certain times, we repeatedly select individuals from population in function optimization unit, then generate new candidates $\vec{u}_G^r$ and $\vec{v}_G^r$

by $\vec{o}_G^k$ and $\vec{o}_G'^k$ respectively. Counting the times that the fitness of child better than its parent $\vec{x}_G^r$, donate by $ST(\vec{o}_G^k)$ and $ST(\vec{o}_G'^k)$ respectively. If $ST(\vec{o}_G^k) < ST(\vec{o}_G'^k)$, we think that the candidate $\vec{o}_G'^k$ is better than $\vec{o}_G^k$, and $\vec{o}_G^k$ is replaced by $\vec{o}_G'^k$. The most disadvantage of this method is that, although this method can effectively assess the individual before and after genetic manipulation, it cost more computing resources.

The general framework of automatically designing genetic operators unit is given in Fig.5.

1. **Begin**
2. Input: $\vec{o}_G^k = \{\vec{o}_{1,G}^k, \cdots, \vec{o}_{m,G}^k\}, k \in [1, NOP]$;
3. Select randomly t, $t \in [1, m]$:
4. Select a new operator to $\vec{o}_{t,G}^k$ which will generate a new chromosome $\vec{o}_G'^k$
5. **For** *i*=1 to *times* **Do**
6. Select randomly $r_1 \neq r_2 \neq r_3 \neq r$, $r, r_1, r_2, r_3 \in [1, NP]$:
7. j_{rand} =randint(1,D)
8. **For** *j*=1 to D **Do**
9. **If**($rand_j[0,1) < CR$ or $j = j_{rand}$) **Then**
10. $\vec{u}_{j,G}^r = \vec{x}_{j,G}^r + \text{GeneCalculate}(\vec{x}_{j,G}^{r_1}, \vec{x}_{j,G}^{r_2}, \vec{x}_{j,G}^{r_3}, \vec{x}_{j,G}^{best}, \vec{o}_G^k)$
11. $\vec{v}_{j,G}^r = \vec{x}_{j,G}^r + \text{GeneCalculate}(\vec{x}_{j,G}^{r_1}, \vec{x}_{j,G}^{r_2}, \vec{x}_{j,G}^{r_3}, \vec{x}_{j,G}^{best}, \vec{o}_G'^k)$
12. **Else**
13. $\vec{u}_{j,G}^r = \vec{v}_{j,G}^r = \vec{x}_{j,G}^r$
14. **End If**
15. **End For**
16. **If** (*better*($\vec{u}_G^r, \vec{x}_G^r$)) **Then**
17. $ST(\vec{o}_G^k)++$
18. **End If**
19. **If** (*better*($\vec{v}_G^r, \vec{x}_G^r$)) **Then**
20. $ST(\vec{o}_G'^k)++$
21. **End If**
22. **End For**
22. **If** ($ST(\vec{o}_G^k) > ST(\vec{o}_G'^k)$) **Then**
24. $\vec{o}_{G+1}^k = \vec{o}_G^k$
25. **Else**
26. $\vec{o}_{G+1}^k = \vec{o}_G'^k$
27. **End If**
28. **End**

Figure 5. The framework of automatically designing genetic operators unit

IV. EXPERIMENTAL VERIFICATION

For experiment verification, 23 benchmark functions, which are well-known and frequently used, are selected from the [13]. In order to verify the effectiveness and efficiency of EA^{2}DGO algorithm, we carry out experiment based on these 23 benchmark functions.

In our experiment, we set *NP*=100, *F*=0.7, *CR*=0.8, *OMR*=0.15, *Max_FEs*=250000, for new encoding scheme MEP in automatically designing genetic operators unit, *NOP*=5, *h*=5, *t*=6, *L*=11, *times* =50, $T = \{a, b, c, d, F\}$ (*a* is $\vec{x}_{j,G}^{r_1}$, *b* is $\vec{x}_{j,G}^{r_2}$, *c* is $\vec{x}_{j,G}^{r_3}$, *d* is $\vec{x}_{j,G}^{best}$) and $O = \{+, -, *\}$. In order to verify the performance of the proposed algorithm, Standard DE algorithm is conducted. In DE algorithm, *NP*=100, *F*=0.7, *CR*=0.8, *Max_FEs*=250000. The obtained results are presented in Table 2. Simulation is carried out in Eclipse and run on an AMD laptop with 2G RAM under Windows XP platform. For each test problem, 25 independent runs were conducted with different random initialization.

The results on functions F1 to F23 are summarized in Table I. For functions F1-F5, F7, F9-F11, the EA^{2}DGO achieved better than DE. For functions F6, F12-F14, F16-F23, all two algorithms obtain the exactly same result. DE achieved the best in function F8 and F15. In F15, while the result of the EA^{2}DGO closes to DE, and both algorithms have the ability to get best solution. The most difference from the results is the function F8, in which DE can easily obtain the global minima but, the new algorithm presented in this paper easily falls into the local minima.

The convergence comparisons between the EA^{2}DGO and DE are shown in Fig.7. For simplicity, each Figure shows the result of a random trial. Because of space limitation, just some samples (F2, F4, F7 and F9) are selected and presented. According to Fig.7, we can find that EA^{2}DGO can converge to a better optimal solution quickly than DE in every trail.

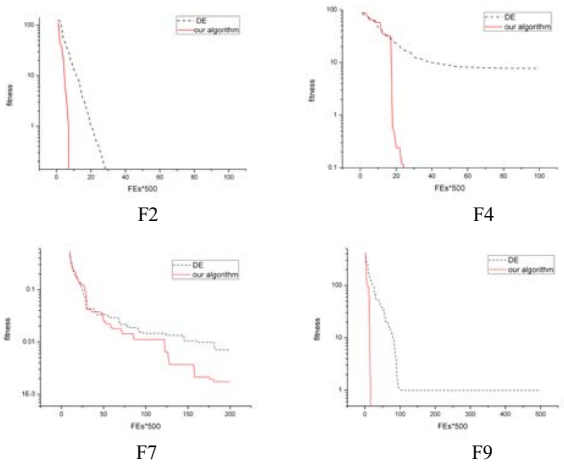

Figure 6. Convergence curves of the automation of EA for single objective optimization algorithm and DE for partial test functions. *X* axis represents number of function calls and *Y* axis represents the best fitness.

V. CONCLUSION

A novel evolutionary algorithm based on automatically designing of genetic operators is presented to tentatively solve algorithm designed automatically problem. EA^{2}DGO is not only searching in the problem solution space, but also searching in the space of genetic operators, which means the genetic operators it not predefined by designer before problem solving, but searched and designed automatically in the process of problem solving. For future work, the validity analysis for EA^{2}DGO should be carried out urgently. Moreover, parameters in presented algorithm are not analyzed. Further research could focus on the parameters researching

and the adaptations of parameters' settings during the evolving procedure.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their very detailed and helpful review. This work was supported by the National Natural Science Foundation of China (No.: 61175073) , in part by Natural Science Foundation of Guangdong Province (No.: S2013010013974), in part by the Leading Talent Project of Guangdong Province, in part by the Shantou Science and Technology Planning Project (No.:150) and Li Ka-Shing Foundation Project.

TABLE I. THE RESULTS ACHIEVED FOR F1 TO F23 USING EA^{2}DGO FOR SINGLE OBJECTIVE OPTIMIZATION ALGORITHM AND DE ALGORITHM

	EA^{2}DGO					DE				
	Best	Median	Worst	Mean	Std	Best	Median	Worst	Mean	Std
F1	0.0	0.0	0.0	**0.0**	0.0	2.38E-90	7.58 E-89	3.90E-88	1.07 E-88	9.22E-89
F2	0.0	0.0	0.0	**0.0**	0.0	3.07E-52	1.0E-51	6.34E-51	1.36 E-51	1.17E-51
F3	0.0	0.0	0.0	**0.0**	0.0	9.84E-87	1.8 E-85	6.8 E-85	2.23 E-85	1.7 E-85
F4	0.0	0.0	0.0	**0.0**	0.0	1.349	9.494	13.075	8.293	2.836
F5	0.0	22.117	29.0	**13.001**	10.839	1.810	18.356	71.343	23.886	14.486
F6	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	0.0	**0.0**	0.0
F7	3.74E-5	8.61 E-5	1.78 E-3	**8.82E-4**	4.26E-4	2.1E-3	3.3E-3	4.76E-3	3.3 E-3	5.81E-4
F8	-12569.487	-11858.856	-5897.496	-10864.79	1755.355	-12569.487	-12451.048	-12095.733	**-12419.463**	118.28
F9	0.0	0.0	0.0	**0.0**	0.0	0.0	1.99	6.96	2.12	1.32
F10	4.441 E-16	4.441 E-16	4.441 E-16	**4.441 E-16**	0.0	4.0E-15	7.55E-15	7.55E-15	5.89E-15	1.37E-15
F11	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	7.4E-3	4.93E-4	1.43E-3
F12	1.571 E-32	1.571 E-32	1.571 E-32	**1.571 E-32**	0.0	1.571 E-32	1.571 E-32	1.571 E-32	**1.571 E-32**	0.0
F13	1.35 E-32	1.35 E-32	1.35 E-32	**1.35 E-32**	0.0	1.35 E-32	1.35 E-32	1.35 E-32	**1.35 E-32**	0.0
F14	0.998	0.998	0.998	**0.998**	3.2E-8	0.998	0.998	0.998	**0.998**	8.6E-17
F15	3.075 E-4	3.075 E-4	1.791 E-3	4.496 E-4	1.024 E-4	3.075 E-4	3.075 E-4	1.22E-3	**3.69E-4**	1.77E-4
F16	-1.03163	-1.03163	-1.03163	**-1.03163**	0.0	-1.03163	-1.03163	-1.03163	**-1.03163**	0.0
F17	0.398	0.398	0.398	**0.398**	1.715 E-4	0.398	0.398	0.398	**0.398**	0.0
F18	3	3	3	**3**	2.045E-14	3	3	3	**3**	6.88E-16
F19	-3.86	-3.86	-3.86	**-3.86**	2.66E-9	-3.86	-3.86	-3.86	**-3.86**	9.25E-13
F20	-3.32	-3.32	-3.32	**-3.32**	4.79E-7	-3.32	-3.32	-3.32	**-3.32**	6.27E-12
F21	-10.153	-10.153	-10.153	**-10.153**	1.01E-9	-10.153	-10.153	-10.153	**-10.153**	1.38E-15
F22	-10.403	-10.403	-10.403	**-10.403**	2.59E-9	-10.403	-10.403	-10.403	**-10.403**	1.38E-15
F23	-10.536	-10.536	-10.536	**-10.536**	1.03E-9	-10.536	-10.536	-10.536	**-10.536**	1.85E-15

REFERENCES

[1] Dorigo, M. and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne,M. Dorigo, and F. Glover, eds., New Ideas in Optimization, pp. 11–32. McGraw-Hill, 1999.

[2] Jorge Taveres and Francisco B. Pereira, Automatic Design of Ant Algorithms with Grammatical Evolution, 15th European Conference on Genetic Programming,2012, pp.206–217.

[3] Ryan C., Collins J.J., O'Neill M. Grammatical Evolution: Evolving Programs for an Arbitrary Language. Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming 1998, pp.83-95.

[4] Jorge Taveres and Francisco B. Pereira, designing pheromone update strategies strongly typed genetic programming, 14th European conference on Genetic programming, 2011, pp.85-96.

[5] Koza, J. R., Genetic Programming: On the Programming of Computers by Meansof Natural Selection, MIT Press, Cambridge, MA, 1992.

[6] Pappa, G. L., FREITAS, A. A. Towards a genetic programming algorithm for automatically evolving rule induction algorithms,ECML/PKDD 2004 - Workshop on Advances in Inductive Learning, 2004, pp.93-108.

[7] Mihai Oltean and Grina Grosan, Evolving Evolutionary Algorithms using Multi Expression Programming,Advances in Artificial Life, Lecture Notes in Computer Science, Vol.2801, 2003, pp.651-658

[8] Oltean M., Dumitrescu D.,Multi Expression Programming, technical report,UBB-01-2002, Babes-Bolyai University, Cluj-Napoca, Romania.

[9] Dazhi Jiang, Zhifei Wang, Haojun Sun and Yulin Du, A Unified Fitness Calculation Method for Automatic Modeling Algorithms, Proceedings of the 8thWorld Congress on Intelligent Control and Automation, July 6-9 2010, Jinan, China,pp1569-1573.

[10] Kennedy J, Eberhart RC. Particle swarm optimization. In: Proc. of the IEEE Conf. on Neural NetWork, IV.Perth: IEEE Press, pp.1942-1948, 1995.

[11] Storn R, Price K, "Differential Evolution-A simple and efficient heuristic for global optimization over continuous spaces", *J. Global Optim.*, vol.11,1997, pp.341-359.

[12] Dazhi Jiang, Zhijian Wu, and Lishan Kang, "New Method Used in Gene Expression Programming: GRCM," Journal of System Simulation, vol.18, Jun. 2006, pp.1466-1468

[13] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster", ; IEEE Trans. Evolutionary Computation, 1999, pp.82-102.