Journal of Information Hiding and Multimedia Signal Processing ©2015 ISSN 2073-4212
Ubiquitous International Volume **6**, Number **5**, September **2015**

Combinatorial Test Suites Generation Method Based on Fuzzy Genetic Algorithm

Chang-An Wei, Yun-Long Sheng and Shou-Da Jiang
Department of Automatic Testing and Control
Harbin Institute of Technology
no. 2 Yi-Kuang Street, Nangang District, Harbin, 150080, China
weichangan@hit.edu.cn; 13B901008@hit.edu.cn; jsd@hit.edu.cn

Jian-Feng Wang
China Shipbuilding Industry Technology and Economy Research Institute
no. 70 Southern College Road, Haidian District, Beijing, 10081, China
hit08b901005@126.com

Received March, 2015; revised May, 2015

Abstract. *In this paper, we present an algorithm for generating combinatorial test suites using Fuzzy Genetic Algorithm (FGA). According to the problem of falling into local optimum of the traditional Genetic Algorithm (GA) method, we introduce the fuzzy control method to select the cross and variation probability adaptively using the entropy and discrete degree in the population, which improves the efficiency and reduces the time of generating the test data. Compared to other well-known algorithms, the final experiment results show the competitiveness of our algorithm both in the test suite size and the running time.*

Keywords: Combinatorial Testing, Software Test, Fuzzy Genetic Algorithm.

1. **Introduction.** Software errors are usually caused by interaction of parameters. A study found that software errors caused by a single parameter only account for 20% to 40% of the total, caused by interaction of two parameters account for 70%, and caused by three parameters account for about 90% [1, 2]. With the increasing number of parameters, the scale of the test suites and the degree of the complexity of algorithms grows exponentially. That is why pairwise testing has always been the research hotspot in the field of combinatorial testing.

As a special form of software, embedded software has characteristics of real-time, embedded, high-reliability requirements and complex parameters, which makes it particularly necessary and more complex to take depth testing on it. In this paper, we apply the combinatorial testing methods to embedded software testing. The automatic test of embedded software can be implemented through automatically injecting test data and getting test result through hardware interfaces. The generation of the test suites is an important part of embedded software test. This paper focuses on the method of generating combinatorial test suites of embedded software. At present, the test suites generation method includes algebraic construction, greedy algorithm, heuristic intelligent algorithm, etc [3]. Orthogonal design [4] and TCconfig [5] are the chief methods among the algebraic construction. Greedy algorithm consists essentially of AETG, TCG, IPO, etc. Ant Colony Algorithm [6], Particle Swarm Optimization [7, 8, 12], Evolutionary Algorithm [9] and Genetic Algorithm [10, 11] belong to intelligent algorithm. Compared to greedy algorithm, heuristic algorithm can return better result, but most heuristic algorithms require multiple matrix computing, which consume a lot of time.

Genetic Algorithm is widely used in generating combinatorial test suites, but it has the problem of falling into local optimum, which will cause large amount of test data and long generation time. To deal with this problem, we introduce fuzzy control method to improve Genetic Algorithm. By selecting the genetic operator under the adaptively control of the entropy and discrete degree in the population, increasing the cross and variation probability when the diversity of the population becomes worse, the population will evolve in the direction of global optimum, making a smaller amount of test data and a shorter generation time.

2. **Background and Definitions.**

Considering a Software Under Test (SUT) that has k parameters, assume that these parameters each has $v_1, v_2, \cdots, v_k$ values, that is to say, for $i (1 \leq i \leq k)$, there are v_i values which are represented by $0, 1, \cdots, v_i - 1$. Use notation $[0, v_i - 1]$ to represent the set$\{0, 1, \cdots, v_i - 1\}$. Meantime, assume that each parameter is independent in value. Referring to the current common combinatorial test model, make definitions as followed:

Definition 2.1. *Interaction, Point. Assume* $I = \{(i_1, a_{i_1}), (i_2, a_{i_2}), \cdots, (i_t, a_{i_t})\}$*, and* i_j *differs from each other,* $a_{i_j} \in [0, v_{i_j} - 1] (j = 1, 2, \cdots, t)$*, then it is named a t-way interaction if* $|I| = t$*, the 1-way interaction* $\{(i_j, a_{i_j})\}$ *can be named point, devoted by* (i_j, a_{i_j}) *as simple. In addition, let* $E_I = \{i_1, i_2, \cdots, i_t\}$ *devote the set of all factors corresponding to an interaction* I*, and* $H_t = \{I | I = \{(i_1, a_{i_1}), (i_2, a_{i_2}), \cdots, (i_t, a_{i_t})\}\}$ *devote all of the t-way interaction in the SUT.*

Definition 2.2. *Test Data. A test data is a k-tuple* $T = (t_1, t_2, \cdots, t_k), t_i \in [0, v_i - 1] (i = 1, 2, \cdots, k)$*. A test data* $T = (t_1, t_2, \cdots, t_k)$ *covers interaction*
$I = \{(i_1, a_{i_1}), (i_2, a_{i_2}), \cdots, (i_t, a_{i_t})\}$ *if* $T(i_j) = a_{i_j} (j = 1, 2, \cdots, t)$*, devoted by* $T \supseteq I$*. Let* $H_{T,t}$ *devote the set of all t-way interactions covered by* T*. Thus, a test data covers exactly* $\begin{pmatrix} k \\ t \end{pmatrix}$ *t-way interactions.*

Definition 2.3. *Mixed Cover Array. Assume that* A *is a* $n \times k$ *matrix. The factor of the* i^{th} *column has values from alphabet* $[0, v_i - 1]$ *and every possible t-way interaction is covered by some row, or in other words,* $\forall I \in H_t$ *, there exists at least one row* r *(corresponding to a test case) such that* $A[r, i_j] = a_{i_j}$*. Then,* A *is called a mixed covering array, denoted by* $MCA(n; t, (v_1, v_2, \cdots, v_k))$*. Especially, if* $v_1 = v_2 =, \cdots, = v_k = v$*, we call the objects simply covering arrays (CAs) and simplify the notation to* $CA(n; t, k, v)$*.*

In addition, if some columns take the same possible values, the number of factors k is omitted, so an $MCA(n; t, (v_1, v_2, \cdots, v_k))$ can be also devoted by
$MCA(N; t, S_1^{p_1}, S_2^{p_2}, \cdots, S_r^{p_r}), k = \sum_{i=1}^{r} p_i$, and the same as CAs.

Definition 2.4. *Discrete Degree of Two Test Data. Considering test data*
$T_1 = (t_{11}, t_{12}, \cdots, t_{1k})$ *and* $T_2 = (t_{21}, t_{22}, \cdots, t_{2k})$*,* $t_{1j}, t_{2j} \in [0, v_j - 1] (j = 1, 2, \cdots, k)$ *define* $D_{T_1,T_2} = \sum_{j=1}^{k} \frac{|t_{1j} - t_{2j}|}{v_j - 1} / k$ *as the discrete degree of* T_1 *and* T_2*.*

3. **Test Suites Generating Algorithm Based on GA.**

We adopt the one-test-at-one-time strategy to generate each test data by selecting, crossing and variation using GA, Repeat the process till the terminal condition is met.

Definition 3.1. *Chromosome. Let each test data* $T = (t_1, t_2, \cdots, t_k), t_i \in [0, v_i - 1] (i = 1, 2, \cdots, k)$ *as a chromosome,* t_i *is a gene of the chromosome and* v_i *is the gene bank.*

Definition 3.2. *Adaptive Value Function. Let A be the set of test suites generated already, U_c be the set of all the interactions that not covered by any test data in A, that is, $U_c = \{I | I \in H_t, \forall A_i \in A, I \notin H_{A_i,t}\}$. So, for a test data x, its adaptive value function is defined as$f(x) = |\{H_{x,t} \bigcap U_c\}|$.*

When an evolution period ends, according to the one-test-at-one-time strategy, select the test data x that the adaptive value is largest into A, and remove all the t-way interaction covered by A from U_c. That is,
$\forall I = \{(i_1, a_{i_1}), (i_2, a_{i_2}), \cdots, (i_t, a_{i_t})\} \in x$, if $\exists I \in U_c$, then remove I from U_c.

Recalculate the adaptive value of the test suites except for x and reserve test data y that the adaptive value is largest. Reserving y to next generation helps the population mature rapidly.

The Test Suites Generating Algorithm Based on GA is given as Algorithm 1.

Algorithm 1 Test Suites Generating Algorithm Based on GA

```
1: Initialize the generation number $M$ as zero. Initialize the test suites $A = \Phi$. Initialize
   the scale of population $N$. Initialize the set of uncovered t-way interaction $U_c$;
2: Initialize the population randomly and evaluate each gene of the chromosome;
3: (Selection) Calculate the adaptive value of each chromosome, and sort them in-
   creasingly. Let $f(x_i), i \in N$ be the adaptive value of the $i^{th}$ chromosome. Generate a
   random number $p \in [0,1]$.If $\sum_{j=1}^{n} f(x_i) / \sum_{k=1}^{N} f(x_k) \leq p$, select the $n_{th}$ chromosome
   to be the result. Repeat the process $N$ times to renew the population;
4: (Cross) Match two chromosomes in population $N$ randomly, and generate a random
   number $q \in [1,k]$ for each pair of chromosomes. Cross the gene at position $q$ by
   probability $p_c$.
5: (Variation) Generate a random number $t \in [1,k]$ for each chromosomes, mutate the
   gene at position $t$ by probability $p_m$.
6: Calculate the adaptive value of each chromosome. If the largest adaptive value of the
   population is $C_k^2$ or $m$ equals to $M$, go to Step 7, otherwise, let $m = m + 1$, go to
   Step 3.
7: Put the chromosome which has the largest adaptive value into A. Renew $U_c$, if $U_c$ is
   empty, end the algorithm and A is the final test suites, otherwise, go to Step 2.
```

4. **Test Suites Generating Algorithm Based on FGA.**

In GA, the cross and variation of chromosomes are in given probabilities. But it is better to reduce the cross probability p_c and the variation probability p_m when the diversity of population is good, and increase p_c and p_m when it is worse. We introduce entropy and discrete degree of the population to value the diversity of a population.

Definition 4.1. *Entropy of the Population. Assume that the k^{th} generation has R subsets $Q_{k1}, Q_{k2}, \cdots, Q_{kR}$,the i^{th} subset contains $|Q_{ki}|$ individuals, and $\exists q \in \{1, 2, \cdots, R\}$, $\bigcup_{q=1}^{R} Q_{kq} = B_k$, B_k is the set of the k^{th} population, then the entropy of the population is defined as:*

$$S_N = (-\sum_{j=1}^{R} P_j \log P_j) / \log N \tag{1}$$

$P_j = \frac{|Q_{kj}|}{N}$, N is the scale of the population. When $R = 1$, $S_N = 0$, when $R = N$, $S_N = 1$. The entropy reflects the distribution of different individuals of the population, that is, the larger entropy, the better diversity of the population.

Definition 4.2. *Discrete Degree of the Population. According to the discrete degree D_{T_1,T_2} of two test data T_1 and T_2, define the discrete degree of the population as follows:*

$$D_N = \frac{|Q_{k1}||Q_{k2}|D_{12} + \cdots + |Q_{k(R-1)}||Q_{kR}|D_{(R-1)R}}{|Q_{k1}||Q_{k2}| + \cdots + |Q_{k(R-1)}||Q_{kR}|} \tag{2}$$

Where, $|Q_{k1}||Q_{k2}|, i, j \in [1, R]$ and $i \neq j$ is the weight of the discrete degree between subset Q_i and Q_j. When $R = 1$, $D_N = 0$, When $R = N$, $D_N = 1$. The discrete degree also reflects the diversity of the population. That is, the larger discrete degree, the better diversity of the population.

Both the entropy and the discrete degree describe the diversity of a population, but they have no deterministic functional relationship with the cross probability and variation probability. We change the cross probability p_c and the variation probability p_m according to the entropy and the discrete degree adaptively using the Fuzzy Genetic Algorithm(FGA)[13].

We design a controller of a fuzzy interference system (FIS), the entropy and the discrete degree be the inputs, the cross probability p_c and variation probability p_m be the outputs of controller. Describe the membership of output and input using triangular function. We divide entropy and discrete degree into three fuzzy states: large, medium and small. And the functions are defined as Fig. 1 and Fig. 2.

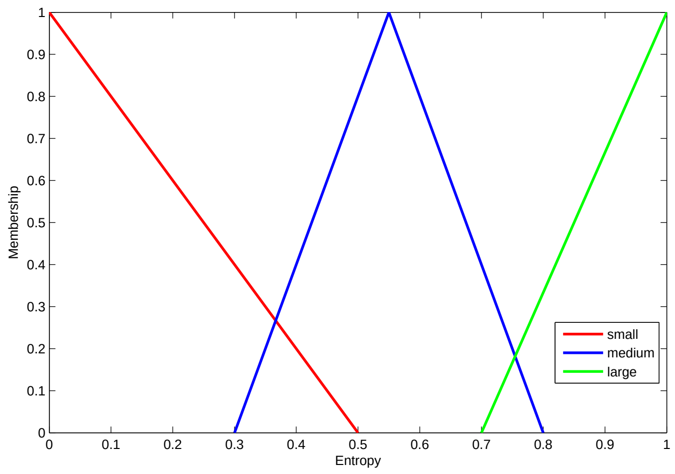

Figure 1. Entropy membership function.

The fuzzy rule of the population diversity with the entropy and discrete degree for the cross probability p_c and variation probability p_m is defined as table 1.

Table 1. Fuzzy rules of genetic operators

Population diversity \ Discrete degree / Entropy	Small	Medium	Large
Small	Bad	Bad	Good
Medium	Bad	Good	Excellent
Large	Good	Excellent	Excellent

It is suggested that the population evolves using small cross and variation probability when the population diversity is excellent, and using large cross and variation probability when the population diversity is bad.

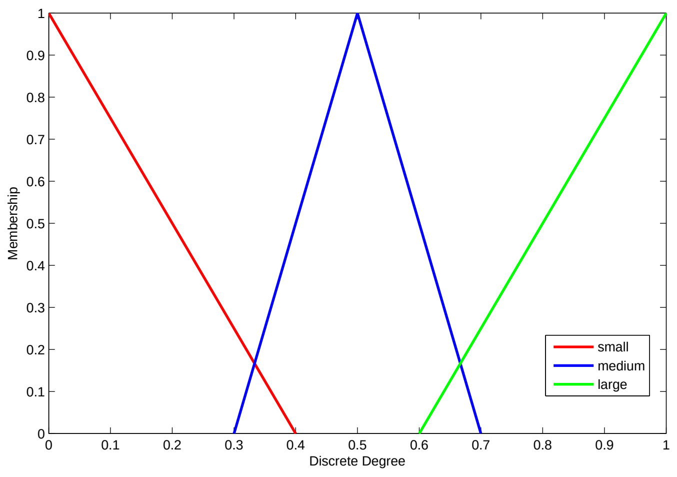

FIGURE 2. Discrete degree membership function.

The process of calculating cross probability and variation probability using FGA is as Algorithm 2.

Algorithm 2 Test Suites Generating Algorithm Based on FGA

```
1: (Fuzzing) Calculate the gradation of entropy and discrete degree according to the functions given above.
2: (Fuzzy inference) Use the entropy and discrete degree to reason the membership of the activated fuzzy rules according to the minimization method.
3: (De-fuzzing) According to the gradation of every fuzzy rules, calculate the cross probability and variation probability using weighted average method, $z_0 = \sum_{i=1}^{n} z_i u_c(z_i) / \sum_{i=1}^{n} u_c(z_i)$. Here, the weight is 1.0 when the Population diversity is bad, 0.5 when the Population diversity is good, and 0 when the Population diversity is excellent.
```

Example 4.1. *Let us consider the following example. When entropy $S = 0.4$, and discrete degree $D = 0.7$, we get the Entropy membership from Fig. 1, and get the Discrete degree membership from Fig. 2.*

TABLE 2. the Entropy membership when $S = 0.4$

Fuzzy State	*Entropy membership*
Small	*0.2*
Medium	*0.35*
Large	*0*

Using the non-zero value of table 2 and table 3, when Entropy fuzzy state is small and Discrete degree fuzzy state is large, according to table 1, the result is Good, so the weight is 0.5; when Entropy fuzzy state is medium and Discrete degree fuzzy state is large, the result is excellent, so the weight is 0.

When Entropy fuzzy state is small, the Entropy membership is 0.2, when Discrete degree fuzzy state is large, the Discrete degree membership is 0.22, using the minimization

Table 3. the Discrete degree membership when $D = 0.7$

Fuzzy State	*Discrete degree membership*
Small	*0*
Medium	*0*
Large	*0.22*

method, we get the membership is $\min(0.2, 0.22) = 0.2$*. Similarly, When Entropy fuzzy state is medium and Discrete degree fuzzy state is large, the membership is* $\min(0.35, 0.22) = 0.22$*.*

Here we set $p_c = p_m$*, then we get the cross probability* p_c *and variation probability* p_m *as:*

$$p_c = p_m = \sum_{i=1}^{n} z_i u_c(z_i) / \sum_{i=1}^{n} u_c(z_i) = (0.5 \times 0.2 + 0 \times 0.22)/(0.2 + 0.22) \approx 0.23; \qquad (3)$$

5. **Experiment Results.** In this section, the proposed method is used to generate test suites in some typical SUTs and compared with other heuristic algorithms.

Table 4. the test suites size in some typical SUTs

SUT	test data number						
	AETG	SA	GA	ACA	CE	PSO	PGA
$CA(N; 2, 4, 3)$	9	9	9	9	9	9	9
$CA(N; 2, 13, 3)$	17	16	17	17	17	18	18
$MCA(N; 2, 3^{12}4^5)$	31	NA	NA	NA	28	28	28
$CA(N; 2, 6, 4)$	28	NA	NA	NA	22	22	22
$MCA(N; 2, 5^1 4^4 3^{11} 2^5)$	21	28	21	26	25	28	25
$MCA(N; 2, 5^1 3^8 2^2)$	20	15	15	16	NA	20	18
$CA(N; 2, 10, 10)$	NA	NA	157	159	NA	168	154
$MCA(N; 3, 5^2 4^2 3^2)$	114	100	108	106	NA	108	105

Table 5. the running time in some typical SUTs

SUT	running time(s)						
	AETG[1]	SA[1]	GA[2]	ACA[2]	CE[3]	PSO[4]	PGA[4]
$CA(N; 2, 4, 3)$	NA	NA	NA	NA	NA	0.85	0.03
$CA(N; 2, 13, 3)$	NA	NA	NA	NA	NA	42	15
$MCA(N; 2, 3^{12}4^5)$	NA	NA	NA	NA	NA	81	23
$CA(N; 2, 6, 4)$	NA	NA	NA	NA	NA	6.2	5
$MCA(N; 2, 5^1 4^4 3^{11} 2^5)$	368	379	124	154	593	146	59
$MCA(N; 2, 5^1 3^8 2^2)$	58	214	22	31	91	23	10
$MCA(N; 2, 6^1 5^1 4^6 3^8 2^3)$	378	579	149	188	556	141	36
$CA(N; 2, 10, 10)$	NA	NA	886	1180	985	178	84
$MCA(N; 3, 5^2 4^2 3^2)$	NA	NA	NA	NA	NA	38	35

Table 4 and table 5 show the experiments results of our algorithm (FGA) with AETG, SA [14], GA [14], ACA [14], CE [15] and PSO in some typical SUTs. In order to remove the effect of different environments on the running time, we code and run FGA and PSO under the same conditions of hardware and software (C++, Windows 7, INTEL Core(TM)$_2$ 2.5GHz) . And the data of SA, GA, ACA, CE and AETG are according to the paper declared.

Notes:1. C++,Linux, INTEL Pentium IV 1.8GHz; 2. C, Windows XP, INTEL Pentium IV 2.26GHz; 3. Matlab, Windows XP, INTEL Core(TM)$_2$ 2.66GHz; 4. C++, Windows XP, INTEL Core(TM)$_2$ 2.5GHz.

According to table 4 and table 5, for the typical SUTs, the size of test suites generated by FGA is a little smaller than other algorithms, and the running time of FGA is less than others. The more complex of the SUT, the better results getting by our method FGA.

Furthermore, in order to eliminate the influences of experiment environment and programming methods, we compare FGA with PSO [15] in some SUTs under the same experiment environment. The statistical results are provided in Table 6 and table 7.

Table 6. Comparison of the test suites size in SUTs between PSO and FGA

SUT	test data number	
	PSO	PGA
$CA(N;2,11,10)$	189	163
$CA(N;2,11,11)$	232	197
$CA(N;2,12,12)$	290	237
$CA(N;2,12,13)$	347	272
$MCA(N;2,13^5 14^6)$	363	298
$MCA(N;2,14^5 15^7)$	440	344
$CA(N;3,6,10)$	1499	1455
$CA(N;3,7,5)$	231	212
$MCA(N;3,10^1 6^2 4^3 3^1)$	393	371

Table 7. Comparison of the running time in SUTs between PSO and FGA

SUT	running time(s)	
	PSO	PGA
$CA(N;2,11,10)$	263	190
$CA(N;2,11,11)$	347	243
$CA(N;2,12,12)$	615	459
$CA(N;2,12,13)$	751	582
$MCA(N;2,13^5 14^6)$	595	446
$MCA(N;2,14^5 15^7)$	1120	811
$CA(N;3,6,10)$	1020	788
$CA(N;3,7,5)$	186	150
$MCA(N;3,10^1 6^2 4^3 3^1)$	256	233

Let the scale $NP = 500$ and the iteration times $m = 20$. The testing environment is C++, Windows XP, INTEL Core(TM)$_2$ 2.5GHz.

Compared to PSO, FGA has the advantages both in the size of test suites and the running time. This is because the characteristic that PSO algorithm is very easy to fall into local optimum affects the result more apparently with the growth of factors. So it need more iterations to get the better result, therefore, the number of iterations is 1000 in paper [16]. From Table 6 and Table 7, our algorithm FGA improves the quality efficiency of test suites effectively, that is, reduces the running time by $10\% - 30\%$, and the size of test suites is reduced in different degrees as well.

6. **Conclusions.** Based on genetic algorithm to generate combinatorial test data, this paper introduced fuzzy control method and designed membership functions and fuzzy rules to adaptively change the cross and variation probability according to the diversity of the generation. It efficiently reduces prematurity issues and makes the generation evolved in the right direction. Experiments have verified that this algorithm is better than others in generation time and the scale of test data is smaller, so it can be effectively applied to generating combinatorial test data.

Acknowledgment. This work was supported by Heilongjiang Postdoctoral Grant in China (no. LBH-Z11171). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] D. R. Kuhn and M. J. Reilly, An investigation of the applicability of design of experiments to software testing, *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pp. 91–95, 2002.

[2] D. R. Kuhn, D. R. Wallace and J. AM. Gallo, Software Fault Interactions and Implications for Software Testing, *IEEE Trans. on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[3] X. Chen, Q. Gu, X. P. Wang and D. X. Chen, Research Advances in Interaction Testing, *Computer Science*, vol. 37, no. 3, pp. 1C-5, 2010.

[4] R. Brownlie, J. Prowse and M. S. Phadke. Robust Testing of AT&T PMX/StarMAIL Using Oats, *AT&T Technical Journal*, Wiley, vol. 71, no. 3, pp. 41C-47, 1992.

[5] A. W. Williams, *Determination of test configurations for pair-wise interaction coverage*, Testing of Communicating Systems, Springer, pp. 59C-74, 2000.

[6] C. Y. Mao, L. C. Xiao, X. X. Yu and J. F. Chen. Adapting ant colony optimization to generate test data for software structural testing. *Swarm and Evolutionary Computation*, vol. 20, pp. 23–36, 2015.

[7] J. F. Wang, C. Sun and S. D. Jiang, Improved algorithm for combinatorial test data generation based on particle swarm optimization, *Journal of Harbin Engineering University*, vol. 34, no. 4, pp. 478–482, 2013.

[8] C. L. Sun, J. C. Zeng and J. S. Pan, An Improved Vector Particle Swarm Optimization for Constrained Optimization Problems, *Information Sciences*, vol. 181, no. 6, pp. 1153–1163, 2011.

[9] Y. L. Zhang and N. Jian. Study of automatically generate test data based on evolutionary algorithm method. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 1, pp. 818–822, 2014.

[10] Y. Cao, C. H. Hu and L. M. Li, An approach to generate software test data for a specific path automatically with genetic algorithm, *8th International Conference on Reliability, Maintainability and Safety*, pp. 888–892, 2009.

[11] S. T. Pan and T. P. Hong, Robust Speech Recognition by DHMM with A Codebook Trained by Genetic Algorithm, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 3, no. 4, pp. 306–319, 2012.

[12] H. T. Yin, J. Q.p Qiao, P. Fu and X. Y. Xia, Face Feature Selection with Binary Particle Swarm Optimization and Support Vector Machine, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 5, no. 4, pp. 731–739, 2014.

[13] X. G. Zhang,S. Hu,D. Chen and X. L. Li, Fast Covariance Matching With Fuzzy Genetic Algorithm. *IEEE Trans.s on Industrial Informatics*, vol. 8, no. 1, pp. 148–157, 2012.

[14] T. Shiba, T. Tsuchiya and T. Kikuno, Using artificial life techniques to generate test cases for combinatorial testing. *Computer Software and Applications Conference, Proceedings of the 28th Annual International*, IEEE, pp. 72–77, 2004.

[15] R. J. Zha, D. P. Zhang, C. H. Nie and B. W. Xu, Test Data Generation Algorithms of Combinatorial Testing and Comparison Based on Cross-Entropy and Particle Swarm Optimization Method, *CHINESE JOURNAL OF COMPUTERS*, vol. 33, no. 10, pp. 1896–1908, 2010.

[16] X. Chen, Q. Gu, Z. Y. Wang and D. X Chen, Framework of Particle Swarm Optimization Based Pairwise Testing, *Journal of Software*, vol. 22, no. 12, pp. 2879–2893, 2011.