International Journal of Hybrid Information Technology
Vol.8, No.9 (2015), pp.121-130
http://dx.doi.org/10.14257/ijhit.2015.8.9.13

Combinatorial Test Generation Using Improved Harmony Search Algorithm

Xiaoan Bao[1], Shuhan Liu[1], Na Zhang[1*] and Meng Dong[1]

The institute of software of Zhejiang Sci-Tech University
zhangna@zstu.edu.cn

Abstract

Combinatorial testing can effectively detect the faults triggered by interactions among the various factors in software system. Harmony Search Algorithm (HS), which is a new optimization algorithm and has been widely applied in the fields of portfolio design, repeatedly adjusts the solution variables in harmony memory to reach the optimum. In order to improve the convergence speed of HS, we propose an improved HS algorithm (IHS) and uses one-test-at-a-time strategy to generate a set of optimum initial solutions in IHS. To avoid the algorithm falling into local optima, we dynamically adjust the values of HMCR and PAR in the new algorithm. Compared to some existing algorithms and tools, the improved harmony search algorithm performs more stably and efficiently in generating optimum combinatorial test cases.

Keywords: *Combinatorial Testing; Harmony Search Algorithm; Software Testing*

1. Introduction

The normal operation of a complex software system is restrict to lots of factors, these factors includes configuration of the system, the external input and internal events. Interactions between factors will lead into some potential problems, even cause the system to be crashed seriously. In software testing, only considering the software failures caused by a single factor will obviously be sufficient for software testing, we need to consider all the interactions between factors as many as possible. However, testing all the combinations of these factors is impractical. Therefore, various methods have been proposed to test combinations coverage and these methods are referred to as the Interaction Testing or Combinatorial Testing [1-4]. Premising that the error detection capability of these methods is ensured, these methods use fewer test cases to test the software fault caused by some interactive factors of the system. A large number of references indicate that the problem of generating a combinatorial test case suite is an NPC problem [5-6]. The researchers tried to use intelligent search algorithms, such as genetic algorithms, ant colony optimization algorithm, particle swarm optimization algorithm [7-9], to search out the approximately optimal solutions about test cases which have exact requirement for the combination coverage.

Geem ZW *et al.*, [10-11] proposed Harmony Search algorithm (HS) by simulating the process of debugging musical harmony, they repeated adjusting the solution variables, realized the convergence of iterations as the time of iteration increases, and completed optimization. The method uses few parameter and is easy to implement, it has been applied to solve numerical optimization problems, pipeline scheduling and structural engineering optimization [12-14] and so on. In software testing, Abdul Rahman A.Alsewari [15] based on combinatorial tests to generate combinatorial test cases efficiently, and proved that the harmony search algorithm perform excellently in solving highly interactive combinatorial problems. HS algorithm is effective in the neighborhood of solution suite, however, the random initial solutions generated by standard HS algorithms affects the performance of the algorithm to a great extent, and the value of key

ISSN: 1738-9968 IJHIT

parameter is almost fixed and computed by experience, these factors results in that it is easy to fall into local optimum convergence instability and other problems in the latter.

Thus, in this paper, to gain a harmony memory (HM) which is filled with better initial solutions, we first give an accurate description of the interaction of the system and use the Greedy Algorithm to generate test case by the way of one-test-at-a-time strategies, and then dynamically adjust the Harmony Memory Considering Rate(HMCR) and Pitch Adjusting Rate(PAR) to improve the speed of searching optimal solutions. At last, we emulate and analyze the improved algorithm; numbers of experimental results prove that the algorithm proposed in this paper can gain more efficient and stable test case.

2. Related Works

2.1. The Model of Combinatorial Testing

Supposing there are n factors that influence the Software Under Test(SUT), and these factors form a finite set $F = \{f_1, f_2, \cdots, f_n\}$. Each factor f_i contains a_i discrete values and these values are independent after some pre-processing operations such as equivalence partitioning . Assume that the set of optional values for these factors is $V_i = \{1, 2, \cdots, a_i\}(1 \le i \le n)$, where V_i represents the set of possible values of f_i, a_i represents the number of V_i ($a_i = |V_i|$).

Definition 1. The N-tuple $test = \{v_1, v_2, \cdots, v_n\}(v_1 \in V_1, v_2 \in V_2, \cdots, v_n \in V_n)$ is a test case of the software, accordingly, the suite T which is consist of several $test$ is marked as a test case suite of SUT.

Definition 2. Given a matrix $T = (V_{i,j})_{m \times n}$ and each row of the matrix represents a test case, T is definited as a test case suite of SUT whose size is m. The combination test cases are also marked as Coveting Array(CA).

Definition 3. Given the $j-th$ column of matrix $A = (a_{i,j})_{m \times n}$ indicates the parameter f_j of SUT , where all elements are taken from the set $Vj(j = 1, 2, \cdots, n)$, that is, $a_{i,j} \in V_j(i = 1, 2, \cdots, m)$. Given a positive integer $N(1 \le N \le n)$, if every N columns cover all the sub arrays of A at least once, calling the A as a $N-way$ covering array which is marked as $CA(m; N, F)$. Obviously, a combination covering of its sub-matrices which are taken from the column A are all composed of N strengths, therefore, $N-way$ covering array is also called as the fixed combination covering array.

Definition 4. For a $N-way(N \ge 2)$ covering array A , if t disjoint sub set $F_i \in F(i = 1, 2, \cdots, t)$ constitute a subset of C , which contain n_i factors corresponding to the columns of $N_i - way(N \le N_i \le n_i)$ covering array, where $N_i > N$, then A is a variable strength covering array. It could marked as $VSCA(m; N, F, C)$.

Definition 5. If the T is the optimal test set, then T needs to meet the standard of combination coverage and the m must be the minimum size.

2.2. The Basic HS Algorithm

The Harmony Search algorithm is first proposed by Geem. Similar to other heuristic intelligent optimization algorithms, the HS algorithm is inspired from nature and imitate behavior of musicians while composing a musical performance, and the process is exactly same as that of searching out the best objective function. Besides, aesthetic evaluation of music is determined by music pitch set of all music instruments, likewise, the objective function is determined by values of all variables.

The core idea of harmony search algorithm is as follows: 1. By some way, the HS algorithm first generates HMS initial solution vector (harmony), and add them to the Harmony Memory(HM). 2. The HS randomly generates new harmonics from HM in a permitted range. Geem designed Harmony Memory Considering Rate(HMCR) and Pitch Adjusting Rate(PAR) of HM, and applied them to adjust the process of debugging optimal solution. Then, the HS generated a random harmony $rand$ between 0 and 1. If $rand$ is less than HMCR, it will generate a new harmony from HM randomly, and if the new harmony is better than the worst one in HM, the worst one will be replaced and then HS use PAR to regulate a new solution in local scope. Otherwise, the HS will gain a new harmony outside of HM and the new harmony is within the permissible range. 3. Finally, the objective function produced by the new solution will be compared with the function produced by the worst solution of HM, if the new solution is better, it will be adopt and HM will be updated, or it will be abandon. At last, the HS repeats step 1 and step 2, and end until the maximum number of iterations is satisfied with T_{max}. The pseudo code of HS algorithm is listed as follow:

Algorithm1:

define objective function $f(x)$*;*
define HMCR, PAR and T_{max}*;*
generate HM with random harmonies;
while $t < max$ *number of iterations do*
  while $i <=$ *number of variables do*
    if ($rand < HMCR$*) then*
      choose a value from HM for the variable ;
      if ($rand < PAR$*) then*
        Adjust the value by adding small random amount;
      end if
    else
      Choose a random value;
    end if
  end while
  Accept the new harmony(solution) if better;
 end while
 return the current best solution;

3. Improved Harmony Search Algorithm

In this article, in order to enhance convergence speed, we improve standard HS algorithm, and the new method we proposed, which is based on harmony search algorithm and is used to generate combinatorial test suite, is composed of two parts. Firstly, depending on the factors that impact examined system, the range of value and constraints between factors, we use one-test-at-a-time strategy to generate a set of optimal initial solutions and apply them to initial harmony memory. Secondly, the improved algorithm adjusts HMCR and PAR adaptively. Lastly, we use the basic frame of improved harmony search algorithm to generate the final optimal test suite.

3.1. Initialize HMS

As harmony search algorithm is mainly based on the field of search, so the performance of initial solution greatly influences the search results. In particular, for some optimization problems with complex constraints, random initial solution is likely to be faulty, and even causes that multi-step search can not acquire a viable initial solution. Then, we use greedy algorithm or other method to search a feasible solution for specific constraints.

In this paper, one-test-at-a-time strategy, which is effective, easy to expand and most widely used in generating combinatorial test suite, is utilized to generate an initial HM. We use this strategy to generate combinatorial test suite: First, we base on factors suite F and interactions R of SUT to generate a suite called $CombSet$ which contains all combinations of distinct factors need to be covered by the test suite. In the initial stage, we initialize an empty test case set T . Thereafter, we choose a test case $TEST$ from the test case suite every time and add it to test case suite T .

Algorithm 2:
input: *factor collection* F *and interactions* R
output: *Combination test suite* T
1. Initialize $T[0\cdots 0][1\cdots n]$ *;*
2. base on factor suite F *and interactions* R *to generate* $CombSet$ *;*
3. $UncombSet := CombSet$ *;*
4. ***while(*** $UncombSet = \varphi$ *)*
5. Generate $TEST$ *and add it to* T *;*
6. Update $UncombSet$ *and delete combination covered by* $TEST$ *;*
7. ***end while***

3.2. The Process of Searching

In harmony search algorithm, due to the generating of new solutions is determined by HMCR, the HMCR value should be set relatively high. PAR controls local search in harmony search algorithm and help avoid problems of local optima. To search out feasible solutions, in the early iterations we must choose appropriate HMCR and PAR to expand search scope as possible. In later iterations, in order to avoid results falling into local optimal we can reduce HMCR and increase PAR to broaden search scope. Generally, the PAR value changes small to large, and in the early stage of optimization, a smaller value is more conducive to find local optimal solution. While in the latter stage, a larger value can increase the diversity of solutions. Therefore, we can optimize adaptive setting as following:

$$HMCR = HMCR_{max} - \frac{HMCR_{max} - HMCR_{min}}{T_{max}} \times k \qquad 3\text{-}1$$
$$PAR = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{T_{max}} \times k$$

In formula 3-1, T_{max} is the number of iteration , k is the number of current iteration, $HMCR_{max}$ and $HMCR_{min}$ represent the maximum and minimum of HMCR value in memory library respectively, PAR_{max} and PAR_{min} stand for maximum and minimum of adjustable probability respectively.

3.3. Process of Generating Optimal Test Cases

We apply the basic framework of improved harmony search algorithm to generate the optimal test cases, the steps are as following:

Step 1. Specify Optimization Problem and Initialize Parameter

The basic parameters of IHS algorithm include harmony memory size(HMS), Harmony Memory Consideration Rate (HMCR), Pitch Adjusting Rate (PAR), the total number of iteration(T_{max})and the initial iteration value $k(0 \le k \le T_{max})$. The optimization problem can be specified as follows:

$$f(x) = \{ y \in UncombSet : x \supseteq y \mid x = x_1, \cdots, x_i in P_1, \cdots, P_i \} (i = 1, \cdots, n) \qquad 3\text{-}2$$

Where $f(x)$ is an objective function, x is the set of decision variables x_i. $UncombSet$ is the set of the of non-covered interaction tuples(y), the objective value is the number of non-covered interaction tuples covered by x , P_i is the set of possible range of values for each decision variable (P_i = discrete decision variables $x_i(1) < x_i(2) < \cdots < x_i(k)$), N is the number of decision variables and k is the number of possible value for the discrete variables.

Step 2. Initialize HM

In this step, we use one-test-at-a-time strategy to generate HMS harmony variables $x^1, x^2, \cdots, x^{HMS}$, and add them to harmony memory(HM), the definition of HM is:

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \cdots & \vdots \\ x_1^{HMS} & x_2^{HMS} & \cdots & x_N^{HMS} \end{bmatrix} = \begin{bmatrix} f(x^1) \\ f(x^2) \\ \vdots \\ f(x^{HMS}) \end{bmatrix} \quad 3\text{-}3$$

On the left hand side of Eq.3-3 are the values of the decision variable of the solutions vectors, while the right hand side contains the fitness evaluation of the solution vectors in the HM. Firstly, we add solution vector, which covers the maximum interaction, to the final test case suite and remove the corresponding interaction tuples from the interaction tuple list. Then, to generate the HM, we fill the HM with solution vectors generated by Greedy Algorithm which is based on one-test-at-a-time strategy. Thirdly, the same step is repeated until no solution vectors in HM cover the maximum interaction.

Step 3. Generate a New Harmony Solution Vector

Three methods can be used to generate a new harmony solution vector from new harmony $x_i' = (x_1', x_2', \cdots, x_N')$ and each pitch $x_i'(i = 1, 2, \cdots, N)$, these methods are: 1. studying HM; 2. pitch-adjusted; 3.random pitch.

First, we choose a random rand, when the value of rand is larger than HMCR, we can choose any vector from P_i .Or we have to choose a vector x_i^{best} from HM. The value of x_i' is computed as formula 3-4.

$$x_i' = \begin{cases} x_i' \in (x_i^1, \ldots, x_i^{HMS}), & rand < HMCR \\ x_i' \in X_i, & other \end{cases}, i = 1, 2, \ldots, N \quad 3\text{-}4$$

$rand$ represent a random number which is between 0 and 1.

If x_i' is from HM, we need to adjust pitch by PAR. If $rand_i (1 \le i \le N)$ is less than PAR, x_i' will be moved either to the left or to the right until reaching the border, then move to the opposite. Otherwise do nothing. Formula is as follows:

$$x_i' = \begin{cases} x_i' \in (x_i^1, \ldots, x_i^{HMS}), & rand < HMCR \\ x_i' \in X_i, & other \end{cases}, i = 1, 2, \ldots, N \quad 3\text{-}5$$

Once a new solution vector is generated, the algorithm will produce a new iteration, and update HMCR and PAR based on the number of current iteration. The formula 3-1 is used to update HMCR and PAR.

Step 4. Update the Harmony Memory

This step, we need to assess the new harmony solution vector generated in step 3. If the new solution vector excludes forbidden tuples, the tuples coverage is computed. While if the new vector is better than the worst vector in HM, the current vector will be updated

and rearranged accordingly. If the new solution vector includes forbidden tuples, its contribution is completely ignored. In a word, the result must be better than the worst one in HM, and the computing process is formula 3-6:

$$if\ f(x') < f(x^{worst}) = \max f(x^j)$$
$$then\ x^{worst} = x' \qquad 3\text{-}6$$

Step 5. Check the Stopping Criterion

Repeat step 3 and step 4, and end until the maximum number of iterations is satisfied. Upon the iterations terminate, the best solution vector in HM is added into the final test suite and the covered interactions are removed from the interaction list.

Step 6: Check Exit Criteria for Interaction Coverage

When the interaction tuples are all covered, the iteration stops. Otherwise, iteration is restarted from Step 2.

4. Experimental Results

In this section, we conduct 3 simulations to evalute the effectiveness of the Improved Harmony Search Algorithm (IHS) and the excellence of HIS compared with standard HS. In this part, the datas of HS are from literature [16]. For HIS, we adopt the parameter settings: HMS=100, HMCR=0.7, PAR=0.2, T_{max}=1000. In order to compare IHS strategy with other intelligent strategies, we list 14 system configurations to verify the results:
CA1-4 3-valued parameters, t=2; CA2-13 3-valued parameters, t=2;
CA3-10 10-valued parameters, t=2; CA4-10 15-valued parameters, t=2;
CA5-10 5-valued parameters, t=2; CA6-6 3-valued parameters, t=3;
CA9-6 6-valued parameters, t=3; CA10-7 5-valued parameters, t=3;
CA11-1 5-valued parameters, 8 3-valued parameters, 2 2-valued parameters, t=2;
CA12-1 7-valued parameters, 1 6-valued parameters, 1 5valued parameters, 6 4-valued parameters, 8 3-valued parameters, 3 2-valued parameters, t=2;
CA13-2 5-valued parameters, 2 4-valued parameters, 2 3-valued parameters, t=3;
CA14-1 10-valued parameters, 2 6-valued parameters, 3 4-valued parameters,1 3-valued parameters, t=3；

To ensure fairness, we have downloaded and employed all strategies within our environment, which consists of a desktop PC with Window XP, 2.8 GHz Core 2 DUE CPU, 2GB of RAM and JDK 1.5 installed.

A. Comparison of Test Cases Size between Ihs And Hs

Test cases size and running time are two crucial factors to determine the performance of harmony search algorithm, thus to evaluate the effectiveness of IHS, we gain a sufficient amount of test cases size of IHS and compare them with HS at first. The test cases size of HS and IHS are generated with 7-parameters, 2-values, and the results are show in Table.1 and Figure 1:

Table 1. Comparison of Test Cases Size between IHS and HS

t	IHS	HS	R(%)
2	14	14	0
3	48	50	4.0
4	132	157	15.92
5	390	437	10.76
6	811	916	11.46

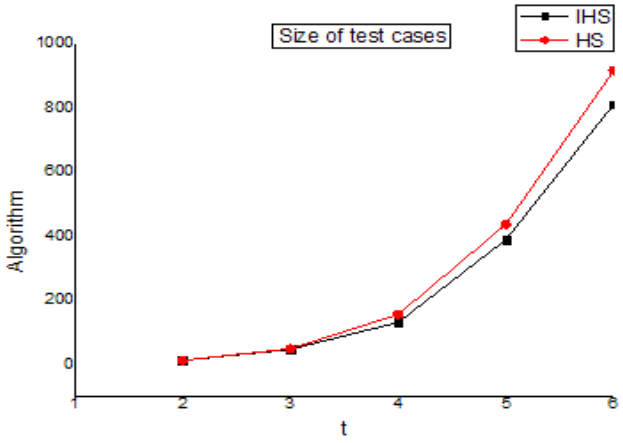


Figure 1. Size of Test Cases

Table 1 proves that when the variable strength is 2-way, the size of test cases is both 14. While along with the variable strength increasing, the size of test cases generated by IHS is less than that generated by HS. For example, When the variable strength is 6, the size of test cases generated by IHS is 811, while the size of test cases generated by HS is 916, thus the size of test cases generated by IHS is 11.46% less than HS. Besides, we can conclude that IHS is more efficient in decreasing test case size obviously. Thus, In combinatorial testing of complex variable strength, the IHS is more effective.

B. Comparison of Running Time between Ihs And Hs

Table 2. Comparison of Running Time between IHS and HS

t	IHS	HS
2	1396	1398
3	4800	4950
4	13068	15229
5	38221	42384
6	81913	89767

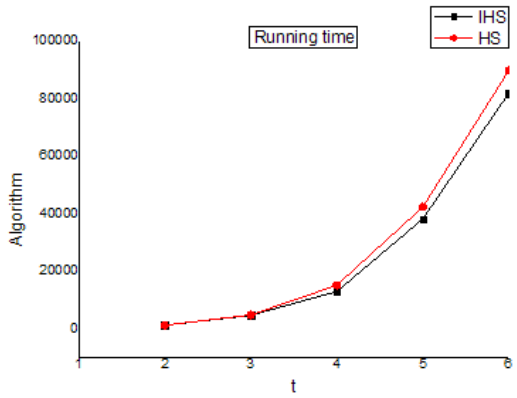


Figure 2. Running Time

Table 2 and Figure 2 list the running times of IHS and HS, the experimental results show that when the variable strength is small, the running time of searching out test case suite by IHS is nearly same as HS, for example, when the variable strength is 2, the running time of IHS is 1396ms, and the running time of HS is 1398ms. While if the variable strength is bigger, the running time of IHS significantly decreases compared with HS, for example, when the variable strength is 6, the running time of IHS is 81913ms, while the running time of HS is 89767ms. Thus, along with variable strength increasing, the running time of IHS will be decreasing.

The comparison with test case size and running time of IHS and HS proves that IHS is more effective in test case size and running time. Using the Greedy Search Algorithm in the initial stage makes IHS algorithm obtain better initial test cases than HS, so IHS could generate better test cases in the latter stage. In the intermediate stage, we use adaptive pitch adjusting to automatically generate test cases within less convergence time, this resulting in less running time spend on generating test case suite. All experimental results certificate that the Improved Harmony Search algorithm performs more effectively in automatically generating test cases, what is more, as the variable strength increases, the effectiveness will be more obvious.

C. Comparison with other intelligent algorithms

To evaluate the performance of IHS algorithm when compared with other related intelligent algorithms, we change the number of parameter and the number of values of these parameters used in software system, and then we compare the performance in test case size. Table.3 lists the results of IHS, GA, ACA and SA.

Table 3. Comparison of Test Cases Size between IHS and Other Algorithms

St	IHS	SA	GA	ACA	St	IHS	SA	GA	ACA
CA1	9	9	9	9	CA8	113	152	125	125
CA2	16	16	17	17	CA9	287	300	331	330
CA3	156	NA	157	159	CA10	186	201	218	218
CA4	339	NA	NA	NA	CA11	14	15	15	16
CA5	42	NA	NA	NA	CA12	40	42	42	42
CA6	36	33	33	33	CA13	96	100	108	106
CA7	58	64	64	64	CA14	353	360	360	361

Table 3 shows the number of test cases generated by different algorithms. It is evident that IHS strategy outperforms others (*e.g.*, SA, GA, ACA) and it seems to give an optimal size in most part of the experiments. When using the system CA1, the size of test case is the same between IHS, SA, GA and ACA. When using the system CA2, the test cases size of IHS is equal to GA. While, the IHS has the largest size of the test cases when we test system CA6 and the SA, GA and ACA have the least size.

5. Conclusion

In this paper, we propose and illustrate our efficient strategy, namely HIS for t-way combinatorial test case generation using improved initial solutions generated by one-test-

at-a-time strategy and adjusting the HS parameter adaptively. Experimental results demonstrate that IHS performs more efficiently in generating t-way combinatorial test case.

Of course, our research work still has limitations. In the future, we should consider the variable strength and the constraints of test cases. Besides, we should consider the impact of diversity of initial solutions of every parameter.

Acknowledgements

This research was supported in part by the National Natural Science Foundation of China (No.61379036); the Natural Science Foundation of Zhejiang Province, China (No.Y13F020175); 521 ta-lent project of Zhejing Sci-Tech University; the New-shoot Talents Program of Zhejiang province (Grant No.2014R406073); Public technology research plan of Zhejiang Province(Grant No. 2014C33102); Major scientific and technological special key industrial projects of Zhejiang Province, China (Grant No. 2014C01047).

References

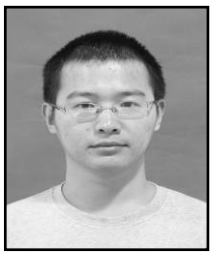
[1] Mandl R. Orthogonal latin squares: An application of experimental design to compiler testing. Communications of the ACM. 28,10:1054-1058(**1985**).
[2] Li Longshu, Cui Yingxia and Yang Yun. Combinatorial test cases with constraints in software systems. IEEE 16th Computer Supported Cooperative Work in Design. (**2012**): 195-199; Wuhan,China
[3] KUHN D R, and REILLY M J. An investigation of the applicability of design of experiments to software testing. IEEE Software Engineering Workshop.(**2002**):91-95;Los Alamitos, USA
[4] Yan Jun, and Zhang Jian. Combinatorial testing principles and methods. Journal of Software. 120, 6:1393-1405 (**2009**)
[5] Williams A W, and Probert R L. Formulation of the Interaction Test Coverage Problem as an Integer Program. Proceedings of 14th International Conference on the Testing of Communicating Systems. (**2002**) March:283-298
[6] Smith B, Feather M S, and Muscettola N. Challenges and Methods in Testing the Remote Agent Planner. Proc. 5th Int'1 Conf. (**2000**):254-263
[7] Lodha, G.M, and Gaikwad, R.S. Search based software testing with genetic using fitness function. Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH), 2014 Innovative Applications of. (**2014**):159-163
[8] X. Chen, Q. Gu, A. Li, and D. Chen, Variable strength Interaction Testing with an Ant Colony System Approach. Proceedings of the 16th Asia-Pacific Software Engineering Conference, (**2009**):160–167
[9] T. Shiba, T. Tsuchiya, and T. Kikuno, Using Artificial Life Techniques to Generate Test Cases for Combinatorial Ttesting. Proceedings of the 28th Annual International Computer Software and Applications Conference, (**2004**):72–77.
[10] Geem ZW, and Kim JH. A New Heuristic Optimization Algorithm: Harmony Search. Simulation.76, 2: 60−68 (**2001**)
[11] Kim JH, Geem ZW, and Kim ES. Parameter estimation of the nonlinear muskingum model using harmony search. Journal of the American Water Resources Association. 37, 5: 1131−1138 (**2001**)
[12] Yadav, P. Kumar, R. Panda, S.K. Chang, and C.S. Optimal Thrust Allocation for Semisubmersible Oil Rig Platforms Using Improved Harmony Search Algorithm. The IEEE Journal of Oceanic Engineering 3, 39(**2014**)
[13] Valian, Ehsan; Tavakoli, Saeed; Mohanna, and Shahram. An intelligent global harmony search approach to continuous optimization problems. APPLIED MATHEMATICS AND COMPUTATION, 670-684(**2014**)
[14] Manjarres, D. ;Landa-Torres, I.; and Gil-Lopez, S. A survey on applications of the harmony search algorithm. ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE. 8, 26: 1818-1831(**2013**)
[15] Abdul Rahman A. Alsewari,and Kamal Z.Zamli. Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support .Information and Software Technology. 54:553-568(**2012**)
[16] Alsewari, A.A. Zamli, ansd K.Z. Interaction test data generation using Harmony Search Algorithm. Industrial Electronics and Applications. 2011 IEEE Symposium on . (**2011**):559-564.

Authors

Bao Xiao'an, born in 1973, M.S. He is a professor and mainly researches adaptive software, software testing and intelligent information processing.

Liu Shuhan, born in 1991, Master candidate. Her main research interests include software testing.

Zhang Na, born in 1977, M.S. She mainly researches software engineering and software testing technology.



Dong Meng, born in1989. Master candidate. His main research interests include reliability model and analysis of software testing.