Combinatorial Test Case Suite Generation Based on Differential Evolution Algorithm

Xu LIANG
Software Technology Institute, Dalian Jiao Tong University, Dalian, China
Email: liangxu00@263.net
Shujie GUO
Mechanical Engineering Institute, Dalian Jiao Tong University, Dalian, China
Email: shujieguo@126.net
Ming HUANG
Software Technology Institute, Dalian Jiao Tong University, Dalian, China
Xuan JIAO
Management Institute, Dalian University of Technology, Dalian, China

***Abstract*—The combinatorial testing, an effective way to improve the efficiency of software testing, is an important means to ensure the quality of software. In the combinatorial testing, the key is the combinatorial test case suite generation. According to the characteristics of the combinatorial test case suite generation problem, this paper proposes a differential evolution algorithm based on the one-test-at-a-time strategy for the solution to this problem. Through the experiments, this paper compares optimized performances of different mutations, explores the influence of algorithm parameters on the optimized performance and verifies the effectiveness and advancement of the solution.**

***Index Terms*—Combinatorial Testing; Test Case Generation; Differential Evolution Algorithm.**

I. INTRODUCTION

The combinatorial testing is a kind of test case generation technology based on the convention, which can reduce the scale of test cases on the basis of ensuring the defect detection capabilities. With the trend of the software developing towards highly configuration, the combinatorial testing has become a widely-applied means to conduct a test in an effective way. The combinatorial test case suite generation is an NP-complete problem, which is the key of combinatorial testing. More and more researchers have paid more attention to the combinatorial test case suite generation and proposed a lot of solutions [1]. Wang Ziyuan proposed a new method of generating the variable strength combinatorial test suite by focusing on the actual interactions between factors [2]. Huanglong proposed the method of generating the n-way combinatorial covering test case with the improved AETG algorithm and IPO algorithm [3]. Shi Liang proposed a solution space tree model, which generates the test data through the path search [4]. Charles proposed the method of generating the combinatorial test suite based on the interactive tree model [5]. Chen Xiang proposed the pair-wise combinatorial test algorithm framework based on particle swarm optimization [6]. Cha Rijun proposed a method of generating the pair-wise combinatorial test data with the cross-entropy method and particle swarm optimization algorithm [7]. Liang Yalan analyzed how the algorithm parameters affect the results when using the genetic algorithm to generate coverage table [8]. McCaffrey proposed a method of using the genetic algorithm to generate the pair-wise combinatorial test case [9]. As the intelligent optimization algorithm is an effective way of solving the NP-complete problem, more and more researchers have begun to use the intelligent optimization method such as the genetic algorithm, particle swarm optimization algorithm, to solve the generation problem of combinatorial test data. As the differential evolution algorithm, as an emerging intelligent optimization technique, can effectively solve the complex optimization problems [10], this paper proposes the method of solving the generation problem of combinatorial test case with the differential evolution algorithm.

II. COMBINATORIAL TEST

The software system, which can be viewed as a processor, processed the input data based on certain rules, and outputs the processing results. These input data is referred to as "input parameters". The statistical result shows that 90% of the errors are triggered by the interaction of multiple parameters [11]. This paper proposes the concept of combinatorial testing in order to improve the error detection rate through thoroughly testing how the interactions between various input parameters impact the system.

Definition 1. Entry: The entry refers to the factors which influence the software system processing flow or the output result, including the software configuration parameters, external events, input data, etc.

Definition 2. Optional value: The optional value is a possible value of an entry. For the input data type entry,

Project supported by the National High Technology Research and Development Program of China (No. 2012AA041402-4);
Manuscript received July 15, 2013; revised October 21, 2013; accepted October 25, 2013.

doi:10.4304/jsw.9.6.1479-1484

the black-box testing techniques, such as the Boundary value analysis or equivalence partitioning, are often used to process the input value in order to obtain the optional values of a finite number of discretions.

Definition 3. Combined strength: In the case of a given test case suite "TR", if the TR package covers all the combinations of any t entries, the combined strength of TR is t.

Provided that a test software system "T" has n separate entries I={I1, I2, ..., In}, each of which contains a finite number of possible values, i.e. Vi={vi1, vi 2, ..., vi n}, the combinatorial testing needs to generate the combinatorial test case suite "TR" according to the requirements of covering intensity in order to test the system. For the sake of the cost of testing, we shall use the fewest number of TR if possible.

III. Generation Solution

A. Differential Evolution Algorithm

Differential Evolution (DE) is a kind of simple and effective new evolutionary algorithm, which has been successfully applied in engineering optimization, mechanical design, electrical industry, environmental industry, water conservancy and other fields [12-15]. Standard differential evolution algorithm concerns four steps: initial population, mutation operation, crossover operation and selection operation.

Initial population: consists of N individuals randomly generated.

Mutation operation: In each generation search, DE algorithm generates a target individual "$t_i(g)$" (g marks the evolutional generation) for each individual in the current population "$x_i(g)$" through mutation operation. DE algorithm has different versions of the mutation mechanism, which can be expressed as the form of DE / a / b. "a" represents the type of mutation operation base, including "rand" and "best" types. "rand" means that a randomly selected individual is taken as the mutation operation base; "best" means that the currently optimal individual is taken as the mutation base. "b" represents the number of differential items at mutation. The most common mutation mechanism is DE/rand/1: $t_i(g) = x_{r1}(g) + F \times [x_{r2}(g) - x_{r3}(g)]$, where F is the scaling factor at 0-1.

Crossover operation: After the mutation operation, use part of variables of the target individual t_i to replace the corresponding current individual x_i according to a certain probability, and then get the tested individual v_i. The crossover methods include the binomial crossover and index crossover.

Selection operation: Select a better one from the tested individual v_i and the current individual x_i to enter into the next generation search. Namely

$$x_i(g+1) = \begin{cases} t_i(g), f[t_i(g)] < f[x_i(g)] \\ x_i(g), \text{Otherwise} \end{cases}$$

Scholars have proposed many solutions of improving the classical differential evolution algorithm, which mainly focuses on two aspects: the adaptive adjustment of parameters [16-20] and the hybrid algorithm [14] [21]. It is worth mentioning that the experimental results show that these improved algorithms are not better than the classical algorithm in solving the problem of combinatorial test case generation. We experimented with the parameter adjustment related to the evolutional generation and the parameter adjustment related to the optimized stall generation, formed a hybrid algorithm with the harmony search algorithm, and used the improvement strategy, specifically, adopted the radical change strategy for 20% of the individuals with the lower fitness every certain generation. In the experiment, after each algorithm ran independently 20 times, we first obtained the mean value of use cases and their variance through the comparison of individual algorithm and then judged the merits or demerits of each algorithm.

B. Data Storage Structure

Definition 4. Configuration: All the combinations of t entries of the software constitute a set, which is called the configuration with the intensity "t". Provided that a system has a total of four entries: I_1, I_2, I_3 and I_4, its configurations with the intensity "3" are: {(I_1, I_2, I_3),(I_1, I_2, I_4),(I_1, I_3, I_4),(I_2, I_3, I_4) }.

Definition 5. Configuration table: For each element in the configuration, the combinations covering all the optional values constitute a set, which is called the configuration table. Provided that the entries I_1、I_2 have two optional values 0 and 1, for the configuration elements (I_1, I_2), the set covering all the optional values is: {(0,0), (0,1), (1 , 0), (1,1)}.

The storage of optional values: Use the chain table to store the optional value of each entry. Using this storage method, we can establish the corresponding relationship between the optional value and its location in the chain table. In the optimization process, on the basis of setting the sequential order of entries, we can use the position value of the optional value in the chain table to replace the specific value of the optional value, solving the problem of real-coding in DE solution.

Suppose the system S has three entries, and their order is artificially regulated like this: I1, I2, I3. Of them, I1 has two optional values V1={true, false}; I2 has three optional values V2={0, 10, 20}; I3 has two optional values V3={"PRC", "USA"}. In that case, the test cases {false, 20, "PRC"} are encoded as {1, 2, 0}, as is shown in Figure 1.

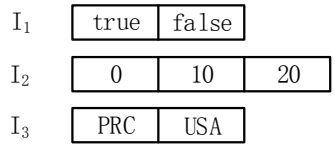

Figure 1. The Storage Structure of Optional Values

The storage of the configuration: Store all the entries of the system in the chain table Li according to the predetermined sequence; and then place the entries in the configuration to be stored in the locations in Li in the form of the chain table, which constitutes the storage way of the configuration. The configuration storage structure of the system S is shown in Figure 2.

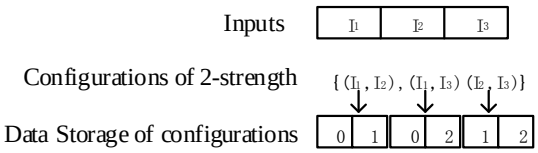

Figure 2. The Storage Structure of the Configuration

The storage of the configuration table: Use the dictionary structure to store the configuration table so as to consult the configuration table through the configuration. The Key values in the dictionary are the elements in the configuration, such as (0, 1), (1, 2), etc.; Value is the chain table composed of the combinations this element in the configuration table is corresponding to. In the system S, in the case of Key=(0, 1), Value={(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)}

C. Combination Coverage Evaluation Solution

We judge whether the test case is good or not through the number of effective configuration elements covered by it. Assume that, the number of test cases included in the current test case set TR (i) is m, the part in the configuration table covered by m test cases is C and the uncovered part is UC, and the test case TC_j can cover n in the UC. In this case, the fitness of TC_j is n.

D. Test Case Suite Generation Process

The combinatorial test case suite is generated through the adoption of the one-dimensional expansion strategy characterized by the expansion item by item, that is, the so-called one-test-at-a-time strategy. Use the differential evolution algorithm to generate a single test case, and add the test case suite until all the elements in the configuration table are covered. The generation process is divided into three steps: initialize; randomly generate n test cases; use the DE to generate other test cases item by item. At the stage of initialization, the main tasks are: to complete the configuration of the parameters and reading of the data files, to initialize the configuration chain table according to the data file and generate the configuration table dictionary. When the cases in the test case suite TR are smaller in number, there is almost no difference in the fitness of individuals whether they are generated in the way of optimized search or at random as a large number of uncovered elements exist in the configuration table "TCon". To improve efficiency, prior to the generation for optimization, generate n test cases at random and store them in the TR, and remove the elements covered by n test cases from the TCon. The experiment of the 4^{10} combined strength "3" shows that it is more appropriate when n is 0.5% of the number of configurations. When using the DE to generate other test cases item by item, first randomly generate N individuals to form an initial population, and then optimize the search according to the basic steps of standard differential evolution algorithm, and finally add the optimal solution to TR and remove the elements covered by the test case represented by the optimal individual from the TCon; this cycle repeats until the TCon is empty.

IV. Experiment and Analysis

A. The Expression Approach of Experimental Data

When generating the combinatorial test case data, three parameters shall deserve our special attention: the entry, the optional value of the entry and combined strength of the entry. Provided that the system to be tested has m entries with n selectable values, the generation problem of the combinatorial test data with the combined strength "t" can be expresses as $n^m - t$. For example, provided that some test system include 3 entries with 5 optional values, 2 entries with 3 optional values and 4 entries with 10 optional values, the generation problem of the combinatorial test case suite with the combined strength 3 can be expressed as $5^3 3^2 10^4 - 3$.

B. Performance Comparison of Mutation Mechanisms DE/rand/1 and DE/best/1

Typically, the standard DE has two kinds of mutation: DE/best/1 and DE/rand/1. Through the experiment, we compared the effects of two mutation mechanisms in solving the generation problem of combinatorial test data. In the experiment, we solved the same problem in two different ways of mutation. For each way, we kept it independently running 10 times. After that, we determined the merits or demerits of each mutation through comparing the calculated average number of test cases. The comparison result is shown in Table 1. From Table 1, it can be easily seen that, for the generation of combinatorial test case suite, the random mutation base mutation approach is better than the optimal mutation base mutation approach.

TABLE I. The Comparison Result of Different Mutation Approaches

Problem	Mutation approach	Average value
410-2	best	29.5
	rand	29.4
410-3	best	145
	rand	143.3
410-4	best	664.75
	rand	660.5
210-5	best	66
	rand	64.6
210-6	best	118
	rand	115.6

C. The Influence of the Algorithm Parameters on Performance

Differential evolution algorithm parameters include the scale factor, crossover probability, population size, optimized generation, etc. As the algorithm has stronger parameter sensitivity, in order to find a better parameter configuration solution, we designed the comparative experiments to explore how these parameters affect the performance of the algorithm. In the experiment, we first selected a parameter A while keeping other parameters constant, and then assigned "A" a value "a1", and finally

obtained the average number of test cases " $\overline{A_{a1}}$ " through calculating 10 times; we assigned "A" a value "a2" again, and then obtained the average number of test cases " $\overline{A_{a2}}$ " through calculating 10 times; this process continues until the end of the experiment on the parameter A. The experimental result is shown in Figure 3 - 6.

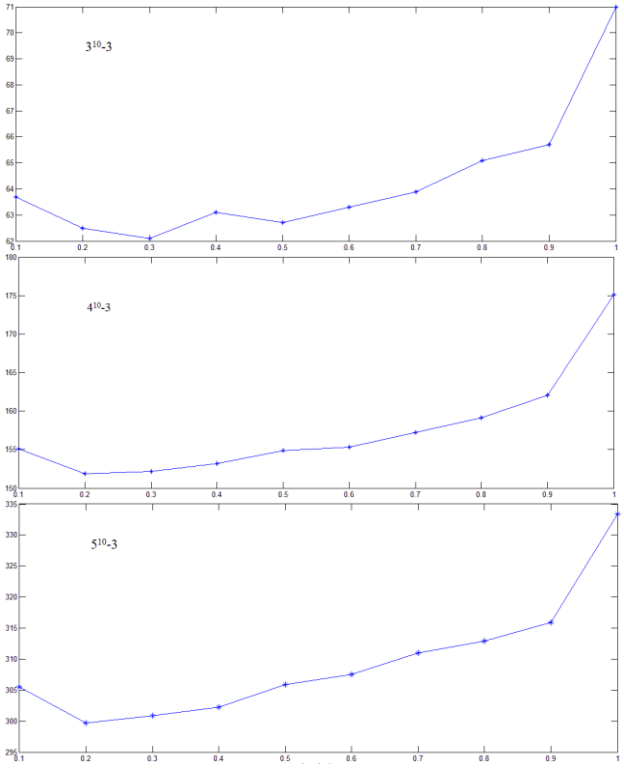

Figure 3. The Influence of Crossover Probability on the Algorithm Performance

It can be seen from Figure 3 that the number of test cases generated by the algorithm presents the trend of rising before inhibition and the optimal points appear nearby 0.2-0.3.

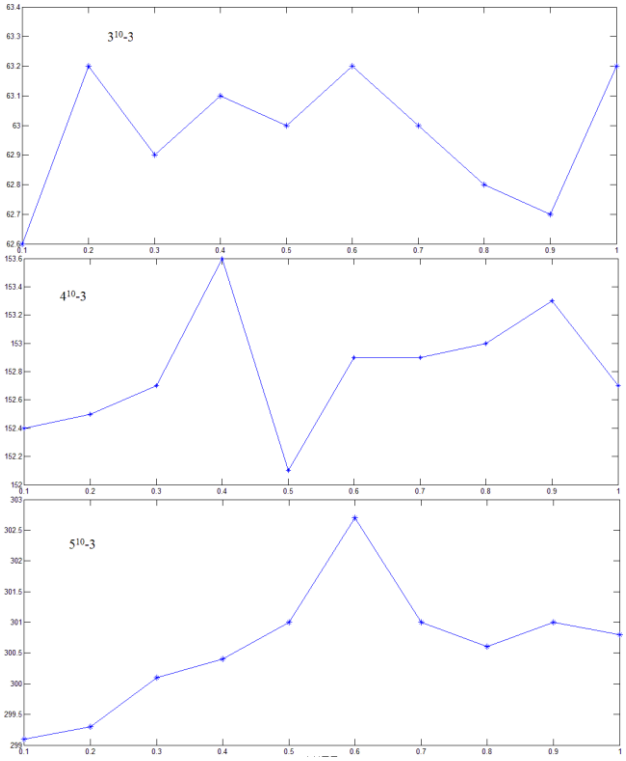

Figure 4. The Influence of the Scale factor on Algorithm Performance

It can be seen from Figure 4 that the scale factor's influence on the algorithm presents uncertainty and the relative optimal solutions often appear at 0.1.

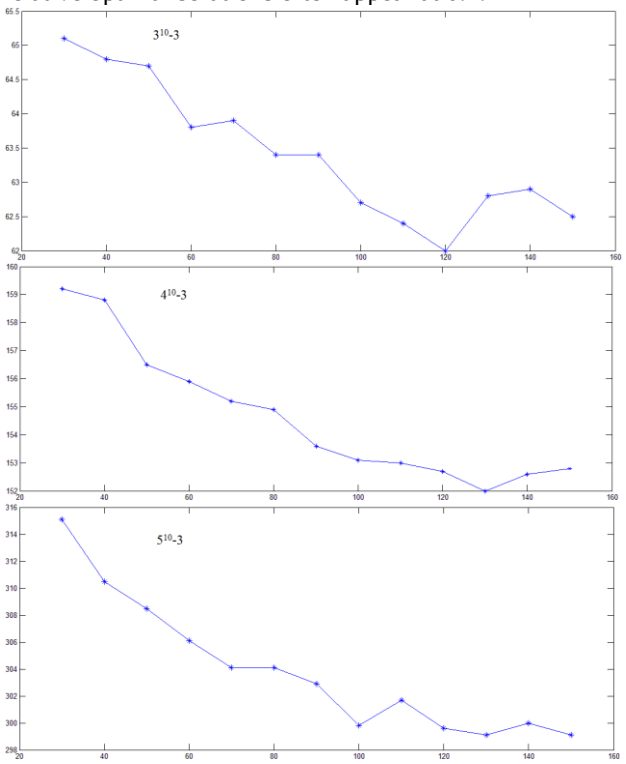

Figure 5. The Influence of Population Size on Algorithm Performance

It can be seen from Figure 5 that the number of test cases generated by the algorithm decreases with the increase of the population, but when the population is

greater than 100 in size, the number change rate of test cases found by the algorithm tends to flatten.

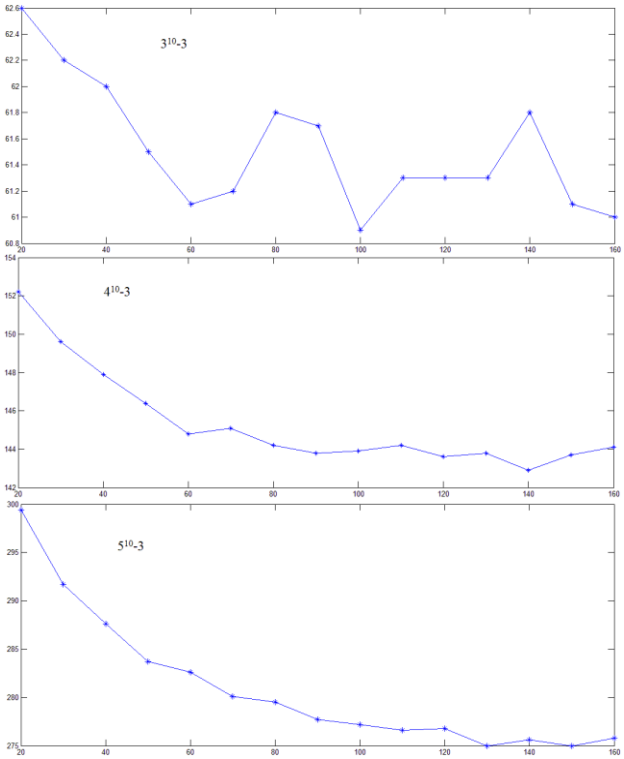

Figure 6. The Influence of Evolutional Generation on Algorithm Performance

It can be seen from Figure 6 that the number of test cases generated by the algorithm decreases with the increase of the evolutional generation, but when the evolutional generation is greater than 60 in size, the number change rate of test cases found by the algorithm tends to flatten.

It should be noted that, in the above experiment, we just considered how each parameter influences on the algorithm, so we were ignorant of the interactions between the parameters. In addition, we just studies three kinds of problems: 3^{10}-3, 4^{10}-3 and 5^{10}-3, so the experimental samples are still not rich enough.

D Comparison with other Intelligent Optimization Algorithm

In order to test the performance of the solution, we experimented with the genetic algorithm, particle swarm optimization and other intelligent optimization algorithms respectively. The results of the comparison experiment are shown in Table 2.

TABLE II. COMPARISON TABLE OF TEST CASES IN NUMBER

Problem	Combinatorial number	Number of test cases		
		GA	PSO	DE
4^{10}-2	720	32	30	28
4^{10}-3	7680	161	156	140
4^{10}-4	53760	733	704	658
4^{10}-5	258048	3124	3112	2814
4^{10}-6	860160	12731	12373	12027

As can be seen from Table 2, for the generation problem of the combinatorial test data, the differential evolution algorithm is superior to the genetic algorithm and particle swarm optimization.

E. Comparison with greedy algorithms

In order to further test the performance of the solution, we experimented with the commonly used combinatorial test case generation solutions provided by National Institute of Standards and Technology (NIST) [22]. The results of the comparison experiment are shown in Table 3.

TABLE III. COMPARISON TABLE OF TEST CASES IN NUMBER

Problem	Combinatorial number	Number of test cases			
		IPOG	IPOG-F	PICT	DE
4^{10}-2	720	30	29	31	28
4^{10}-3	7680	150	157	160	140
4^{10}-4	53760	695	731	742	658
4^{10}-5	258048	3079	3162	3161	2814
4^{10}-6	860160	12778	12727	12473	12027

As can be seen from Table 3, regardless of the combinatorial number, the differential evolution algorithms can generate fewer test cases. When the combinatorial number is smaller, there is little difference in the number of test cases generated by the differential evolution algorithm and other three kinds of algorithms respectively. But with the increase of the combinatorial number, the difference in the number is growing bigger and bigger.

ACKNOWLEDGMENT

Thanks National High-Tech Research and Development Program of China (863 Program) for its funding of sub-project “Testing of the operating system kernel module and software library” (No. 2012AA041402-4). We are grateful to other members of the project team for their valuable help and suggestions.

REFERENCES

[1] YAN Jun， ZHANG Jian, “Combinatorial Testing: Principles and Methods ”, *Journal of Software*, Vol.20, No.6, pp.1393 − 1405, June 2009

[2] WANG Ziyuan， QIAN Ju， CHEN Lin, “Generating Variable Strength Combinatiorial Test Suite With One-test-at-a-time Strategy”, *Chinese Journal of Computers* Vol.35 No.12, pp.2541-2532,2012

[3] HUANG Long, YANG Yuhang, LI Hu, “Rearch on Algorithm of Parameter Pairwise and n-way Combinatorial Coverage, ”, *Chinese Journal of Computers* Vol.35 No. 2, pp.257-269, 2012

[4] SHI Liang, NIE Changhai,XU Baowen, “Pairwise Test Data Generation Based on Solution Space Tress”, *Chinese Journal of Computers* Vol.29 No. 6, pp.849-853, 2006

[5] Charles Song, Adam Porter, Jeffrey S. Foster, “iTree: Efficiently Discovering High-Coverage Configurations Using Interaction Trees”, *ICSE 2012. NJ, USA: IEEE Press*, pp. 903-913, 2012

[6] CHEN Xiang, GU Qing WANG ZiYuan, "Framework of Particle Swarm Optimization Based Pairwise Testing", *Journal of Software*, Vol.22 No.12 , pp.2879-2893,2011

[7] ZHA RiJun, ZHANG DePing, NIE ChangHai, "Test Data Generation Algorithms of Combinatorial Testing and Comparison Based on Cross-Entropy and Particle Swarm Optimization Method ", *Chinese Journal of Computers* 2010 Vol.33 No. 10, pp.1896-1908

[8] LIANG YaLan, NIE ChangHai, "The Optimization of Configurable Genetic Algorithm for Covering Arrays Generation ", *Chinese Journal of Computers*, Vol.35 No. 7, pp.1522-1552,2012

[9] McCaffrey, J.D, "Generation of Pairwise Test Sets Using a Genetic Algorithm", *COMPSAC '09. Washington, DC, USA: IEEE Computer Society*, pp. 626 - 631, 2009.

[10] He YC, Wang XZ, Liu KQ, "Convergent Analysis and Algorithmic Improvement of Differential Evolution *Journal of Software*, Vol.21, No.5, pp.875− 885, 2010.

[11] Kuhn DR, Reilly MJ, "An investigation of the applicability of design of experiments to software testing", *In: Caulfield M, ed. Proc. of the Annual NASA/IEEE Software Engineering Workshop (SEW). Los Alamitos: IEEE Press*, pp. 91 − 95, 2002.

[12] QIN Hui, ZHOU Jianzhong, WANG Guangqian, "Multi – objective optimization of reservoir flood dispatch based on multi-objective differential evolution algorithm," *Journal of Hydraulic Engineering*Vol.40 No.5 , pp.513-519 , 2009.

[13] LIU Bo, WANG Ling ,JIN Yihui, "Advances in differential evolution ", *CONTROL AND DECISION* Vol.22 No.7 , pp.721-729, 2007.

[14] WANG Ling, QIAN Bin, *Hybrid differential evolution algorithm and scheduling*, Beijing Tsinghua University Press, pp. 33-48, 2012.

[15] MENG Hong-Yun, ZHANG Xiao-Hua, LIU San-Yang, "A Differential Evolution Based on Double Populations for Constrained Multi-Objective Optimization Problem", *Chinese Journal of Computers*, Vol.31 No.2 , pp.228-235, 2008.

[16] XIAO Shujun, ZHU Xuefeng, "A modified fast and highly efficient differential evolution algorithm ",*JOURNAL OF HEFEI UNIVERSITY OF TECHNOLOGY* Vol.32 No.11 , pp.1700-1703, 2009

[17] ZHANGXue-xia, CHENWei rong, DAI Chao-hua, "Dynamic Multi-group Self adaptive Differential Evolution Algorithm with Local Search for Function Optimization ", *ACTA ELECTRONICA SINICA*,Vol.38 No.8 , pp.1825-1830, 2008

[18] GE Jianwu, QI Rongbin, QIAN Feng, "A Modified Adaptive Differential Evolution Algorithm ", *Journal of East China University of Science and Technology (Natural Science Edition)* ,Vol.35 No.4 , pp.600-605,2009

[19] Quan-Ke Pan， P.N. Suganthan，Ling Wang etc. , "A differential evolution algorithm with self-adapting strategy and control parameters", *Computers & Operations Research*, No.38, pp.394–408, 2011

[20] R. Mallipeddi, P.N. Suganthan, Q.K. Pan etc. , "Differential evolution algorithm with ensemble of parameters and mutation strategies", *Applied Soft Computing* No.11, pp.1679–1696, 2011

[21] T. Warren Liao, "Two hybrid differential evolution algorithms for engineering design optimization", *Applied Soft Computing*,No.10, pp.1188–1199, 2010

[22] http://csrc.nist.gov/groups/SNS/acts/#briefings