Test Case Prioritization using Multi Objective Particle Swarm Optimizer

Manika Tyagi
Department of CSE
U.I.E.T., Kurukshetra University
Kurukshetra, Haryana, India
tmanika@gmail.com

Sona Malhotra
Department of CSE
U.I.E.T., Kurukshetra University
Kurukshetra, Haryana, India
sonamalhotrakuk@gmail.com

***Abstract*— The goal of regression testing is to validate the modified software. Due to the resource and time constraints, it becomes necessary to develop techniques to minimize existing test suites by eliminating redundant test cases and prioritizing them. This paper proposes a 3-phase approach to solve test case prioritization. In the first phase, we are removing redundant test cases by simple matrix operation. In the second phase, test cases are selected from the test suite such that selected test cases represent the minimal set which covers all faults and also at the minimum execution time. For this phase, we are using multi objective particle swarm optimization (MOPSO) which optimizes fault coverage and execution time. In the third phase, we allocate priority to test cases obtained from the second phase. Priority is obtained by calculating the ratio of fault coverage to the execution time of test cases, higher the value of the ratio higher will be the priority and the test cases which are not selected in phase 2 are added to the test suite in sequential order. We have also performed experimental analysis based on maximum fault coverage and minimum execution time. The proposed MOPSO approach is compared with other prioritization techniques such as No Ordering, Reverse Ordering and Random Ordering by calculating Average Percentage of fault detected (APFD) for each technique and it can be concluded that the proposed approach outperformed all techniques mentioned above.**

***Keywords*— Regression Testing, Test case selection, Test case prioritization, Multi objective Particle Swarm Optimization**

1. INTRODUCTION

Regression testing is a type of software testing that intends to validate modified software and it confirms that modifications made to the software have no adverse side effects. It is an expensive maintenance process, and is usually performed by running some or all of the test cases created to test modifications in previous versions of the software used to revalidate the modified software. The simplest strategy of regression testing is to re-run all existing test cases, it is easy to implement the strategy, but it is expensive. Running all test cases in an existing test suite, however, can consume an inordinate amount of time and resources. Thus, it is necessary to select the minimum set of test cases from existing test suite with the ability to cover all the faults in minimum execution time. When analyzing large test suits, redundancies are identified in test cases, hence, it is necessary to reduce these suites, in order to fit the available resources, without severely compromising the coverage of the test adequacy criterion being observed. The process of minimizing a test suite based on some selection criterion is known as "Test Case selection" [1]. Another issue is test case prioritization. Test case prioritization techniques [18] intend to arrange test cases of a test suite in a way, with the goal of maximizing some objective function. There are various classical techniques of test case prioritization, which are no prioritization, reverse prioritization and random prioritization etc. In no prioritization, test cases are prioritized in the same way they are generated. In reverse prioritization, test cases are prioritized in the opposite way they are generated. In random prioritization, we randomly order the test cases in a test suite.

The paper is organized as follows: section 2 discusses the literature survey, section 3 describes problem definition, section 4 explains the proposed work which is done using multi objective particle swarm optimization, section 5 presents an experimental analysis and results, and section 6 concludes the paper.

2. LITERATURE REVIEW

To solve the problem of test case prioritization, various algorithms such as search algorithms and meta-heuristics algorithms are used. Li et. al. [9] applied various algorithms such as greedy algorithm, additional greedy algorithm, 2-optimal algorithm, hill climbing and genetic algorithm to prioritize test cases. Singh et. al. [11] prioritized test cases using Ant Colony Optimization (ACO) algorithm, in a time constraint environment. Wang [12] designed an approach for test case prioritization based on genetic algorithm and improved the genetic algorithm proposed test case prioritization algorithm. Sabharwal et. al. [13] proposed an approach based on genetic algorithm to prioritize test cases scenarios in static testing.

The test case selection process can be considered as an optimization problem, which aims to find a subset of test cases from the existing test suite, according to one or more objective functions. Tallam [14] proposed a new greedy heuristic algorithm for choosing a subset of test cases from test suite T, which have the ability to cover all requirements as covered by T. In the year 2009, Lin and Huang [2] used Greedy Search

978-1-4799-3140-8/14/$31.00 ©2014 IEEE

for selection of structural test cases. In the context of multi-objective function, Yoo and Harman [4] applied Genetic Algorithm for structural test case selection. On the other hand, some authors focus on deterministic software engineering solutions [1], they consider only a single criterion for test case selection, although their results are good, it is computationally expensive when dealing with large test suites. Heuristic search based techniques seem to be more feasible to solve test case selection problem. In the context of this approach, the HGS algorithm [5], the GRE algorithm [6], Greedy Search techniques for set-covering [7], Genetic Algorithms [4] and Particle swarm optimization PSO [3] are identified.

Hla et. al. [10] solved the problem of test case prioritization by ordering test cases based on altered software parts, by using PSO. For the test case prioritization problem, the proposed algorithm finds the best positions of test cases on the basis of altered software parts, and prioritized test cases, according to new best order. Souza [3] designed a constrained PSO algorithm to solve the problem of test case selection, Binary Constrained PSO (BCPSO) and BCPSO integrated with forward selection (FS), BCPSO-FS were implemented, and both of these algorithm outperformed random search approach. Souza [19] proposed to use multi objective PSO for functional test case selection based on two objectives: requirement coverage, which is to be maximized and execution effort which is to be minimized. Binary Multi-Objective PSO (BMOPSO) and BMOPSO using Crowding Distance and Roulette Wheel (BMOPSO-CDR) were implemented and (BMOPSO-CDR) outperformed BMOPSO.

We present a 3-phase approach to solve the test case prioritization problem. First phase consist of redundant test case removal, it is necessary to remove redundant test cases from test suite, in order to save the resources and time. Second phase consist of using MOPSO for selecting test cases from test suite based on two objective functions: fault coverage and execution time such that selected test cases have the ability to cover all faults and also at the minimum execution time. In the third phase, test case prioritization is performed, priority is assigned to the test cases which are obtained from phase 2, which is calculated by equation (8), so that the prioritized subset of these test cases detects faults rapidly. The rest of test cases are added to the test suite in a sequential way. In our approach, we are prioritizing test cases considering both test suite reduction in phase 1 and test case selection in phase 2, based on fault coverage and execution time of test cases.

3. Problem Definition

As we have already discussed that in our approach, we consider two criteria to prioritize test cases. The two issues are discussed as follows-

A. Fault coverage

Fault coverage should be maximized. Let $T = \{T_1, T_2 .. T_n\}$ indicates a set of n test cases and $F = \{F_1, .. ,F_m\}$ be a given set of m faults of the existing test suite T. Let $F(T_j)$ be a function that returns the subset of faults in F covered by the individual test case T_j. Then, the fault coverage of a solution $t = \{t_1, .. ,t_n\}$ is given as-

$$Fault\_Coverage(t) = 100 * \frac{\left| U_{t_j=1} \{F(T_j)\} \right|}{m} \quad (1)$$

In eq. (1),$U_{tj} =1 \{F(T_j)\}$ represents the union of faults' subsets covered by the selected test cases (i.e.,T_j for which $t_j =1$).

B. Execution time

The execution time is to be minimized and it can be defined as the amount of time needed to manually execute the selected test cases. Formally, each test case $T_j \in T$ has an execution time score e_j. The total execution time of a solution t is defined as:

$$Execution\_time(t) = \sum_{t_j=1} e_j \quad (2)$$

4. Proposed Approach

In our proposed approach, first we have removed the redundancy among the test cases using simple matrix operations and in our second step, we have used MOPSO algorithm to select test cases and later in third step of prioritization we have calculated the order of test cases on the basis of ratio of fault coverage to the execution time to arrange the test cases for final execution. The diagrammatic representation of proposed approach is shown in figure (1).

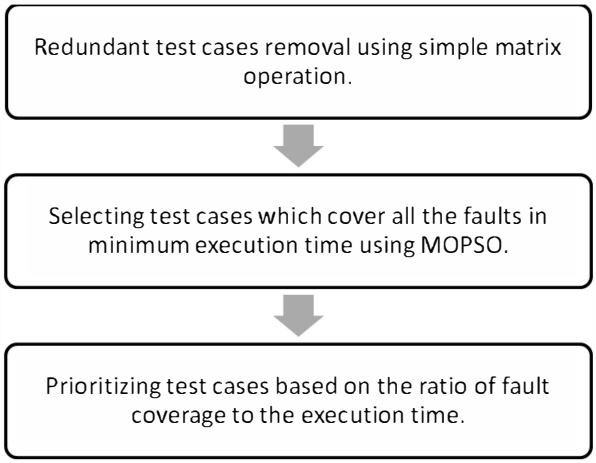

Fig. (1) Fundamental steps in proposed approach

A. Redundant Test Case Removal

It is assumed that F = {F_1, F_2, .. , F_m} and T = {T_1, T_2, .. , T_n} is the set of m faults and n test cases respectively. A relationship between m faults and n test cases are represented in matrix K of n × m and for $\forall$ $k_i \in$ K, if each "1" in k_i can be completely contained by other $k_j \in$ K for j ≠ i, then k_i can be eliminated from the chart without affecting the completeness of the test suite. It means that k_i can be eliminated from the test suite if following condition holds [20].

$$\overline{k_i} + \sum_{j=1} k_j = (111..11) \tag{3}$$

B. Selection of test cases

We are using Multi-Objective PSO in which Fault_Coverage and Execution_time are considered as objective functions. In this approach, the positions of particles are defined as decimal vectors indicating candidate subsets of test cases to be applied in the software testing process. Let T= { T_1, .. ,T_n} be a test suite with n number of test cases, a particle's position is defined pt = (t_1, .. ,t_n), in which $t_i \in$ {0,1}indicates the presence (1) or absence (0) of the test case T_i within the subset of selected test cases.

Particle swarm optimization is a stochastic and global optimization method [8], proposed by Eberhart and Kennedy in 1995. It is based on the behavior of birds flock when they search for food. They search the food by communicating among neighbor birds and from their own experience about searching. PSO uses the similar concept of the bird's flock for the optimal solution of the problem. Each particle goes through the problem search space towards the global optimal solution by the learning from its own experience which is cognitive component and learning from their neighbors which is social component. Each particle also keeps track of its best previous location, which is called pbest and calculated by the fitness function. Best location among all particles which is the optimal solution is denoted by gbest. On each generation of the algorithm, every particle chooses where to move next, considering its own experience, which is the memory of its best past position and the experience of its most successful particle in the swarm. So, the velocity of each particle is updated by the following equation:

$$v_i(g+1) = w \times v_i(g) + c_1 \times r_{1i} \times \left(pb_i(g) - x_i(g)\right) + c_2 \times r_{2i} \times \left(gb_i(g) - x_i(g)\right) \tag{4}$$

In the above equation (4), $v_i(g)$ velocity of i^{th} particle at g^{th} generation, previous best value $pb_i(g)$ of i^{th} particle at g^{th} generation and global best value gb of the population, c_1 denotes the cognitive component, c_2 is coefficient of the social component, w is known as the inertia factor, r_{1i} and r_{2i} are the random numbers uniformly distributed in the interval [0, 1].

$$x_i(g+1) = x_i(g) + v_i(g+1) \tag{5}$$

In equation (5), the current position of each particle is determined by previous position and current velocity of the particle. Suppose that a goal maximizing an objective function f is aimed, the new past best position of each particle is updated by following equation-

$$pb_i(g+1) = \begin{cases} pb_i(g), if\ f\left(x_i(g+1) \le f(pb_i(g))\right) \\ x_i(g+1), otherwise \end{cases} \tag{6}$$

It implies that particles will individually remember their best position where they have been. And then, the global best position, gb is updated by following equation-

$$gb(g+1) = \arg\max f_{pb_i} pb_i(g+1) \tag{7}$$

Multi-Objective Particle swarm Optimization (MOPSO) aims to optimize more than one objective at the same time. A Multi-Objective problem considers a set of k objective functions {$f_1(x)$, $f_2(x)$, .. , $f_k(x)$} in which x is an individual solution for the problem being solved. The objective functions to be optimized in our work are the fault coverage and the execution time of the selected test cases so fault coverage should be maximized which is shown in equation (1) and the execution time is to be minimized, which is shown in equation (2).

C. Test Case Prioritization

From above phase, we obtain $T_{selected}$ = {T_1, T_2, … ,T_c} which represents the minimal set of test cases. To generate priority for each test case obtained from previous step, we use the ratio of fault coverage and execution time, which is defined as follows –

$$priority(t) = \frac{Fault\_Coverage(t)}{Execution\_time(t)} \tag{8}$$

Where Fault_Coverage(t) and Execution_time(t) are defined in equation (1) and equation (2) respectively. Test cases are sorted in descending order according to priority(t) as represented in the above equation (8) and rest of test cases which are not selected in phase 2, are now added to the test suite in sequential order.

5. Experimental Analysis

Experiments are conducted at system configuration of PC with Intel Core i5 processor, system memory 4 GB and MATLAB R2009b is used as simulation software. We have considered two test suites as shown in table (2) and table (7). For particle swarm optimization, the maximum number of iterations used are 20 and particle size is also 20. Lower bound and upper bound for particles are defined as 1 and n, respectively, where n represents the number of test cases. We have compared proposed approach with no ordering, reverse ordering and random ordering. As MOPSO involves some random parameter so we are using average result over 100 runs of algorithm in case. For measuring performance, we are using fault coverage, execution time and APFD (Average Percentage of Fault Detected) which is calculated by using equation (9).

Table (1): Test case along with faults in test suite 1.where 'X' represents particular fault is detected by the test case

Test Cases	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
T1	X		X			X			X	
T2		X						X		
T3		X			X		X			
T4				X		X		X	X	
T5	X					X				X
T6				X	X			X		
T7							X	X		
T8			X							X

Table (2): Test suite 1 in which, test cases are represented in binary form, where '1' represents particular fault is detected by test case and execution time of test cases.

Test Case	Binary Form	Execution Time
T1	1010010010	7
T2	0100000100	3
T3	0100101000	3
T4	0001010110	2
T5	1000010001	5
T6	0001100100	6
T7	1000001100	3
T8	0010000001	5

Table (3): Minimum test case set in binary form along with its execution time in No ordering technique.

Test Cases	Binary Form	Execution Time
T1	1010010010	7
T2	0100000100	3
T3	0100101000	3
T4	0001010110	2
T5	1000010001	5

For test suite 1, various experimental details are given in table (1-6). Table (1) represents test cases along with faults in test suite 1.where 'X' represents particular fault is detected by the test case. In table (2), these test cases are represented in binary form along with their execution time.

Table (4): Minimum test case set in binary form along with its execution time in Reverse ordering technique

Test Cases	Binary Form	Execution Time
T8	0010000001	5
T7	1000001100	3
T6	0001100100	6
T5	1000010001	5
T4	0001010110	2
T3	0100101000	3

.

Table (5): Minimum test case set in binary form along with its execution time in Random ordering technique.

Test Cases	Binary Form	Execution Time
T1	1010010010	7
T5	1000010001	5
T3	0100101000	3
T8	0010000001	5
T4	0001010110	2

Table (6): Minimum test case set in binary form along with its execution time according to proposed MOPSO technique and priority of test cases calculated by using equation (8).

Test Cases	Binary Form	Execution Time	Priority(t)
T4	0001010110	2	2.0
T3	0100101000	3	1.0
T5	1000010001	5	0.6
T8	0010000001	5	0.4

Table (7): Test suite 2 in which, test cases are represented in binary form, where '1' represents particular fault is detected by test case and execution time of test cases.

Test Cases	Binary Form	Execution Time
T1	1011111000	9
T2	0111000100	3
T3	0100101000	5
T4	0001010110	5
T5	1101010001	3
T6	0001100100	6
T7	1000001100	3
T8	0010000001	2
T9	0010000001	2
T10	1011110001	2

The minimal test case set obtained by No ordering, Reverse ordering, Random orderings and proposed MOPSO is shown in Table (3-6) respectively. For test suite 2, table (7) represents test cases in binary where '1' represents particular fault is detected by test case form along with their execution time.

Table (8): The execution time obtained by No ordering, Reverse ordering, Random ordering and proposed MOPSO.

Test Sites	No Ordering	Reverse Ordering	Random ordering	Proposed MOPSO
Test Suite 1	20	24	22	15.42
Test Suite 2	25	23	17	13.67

The Proposed approach is compare with other prioritization techniques such as no ordering, reverse ordering, random ordering, by calculating the average percentage of fault detected (APFD)for each technique.To quantify the aim to increase the rate of fault detection of the test suite, an APFD (average percentage of fault detected) metric is used [15, 16, 17]. Let T be test suite with n test cases, F be the set of m faults covered by T. TF_i be the first test case in ordering T′ of T that expose fault i. The formula of APFD as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + .. + TF_m}{nm} + \frac{1}{2n} \qquad (9)$$

The Proposed approach outperformed No ordering, Reverse ordering and Random ordering as it obtained the minimum execution time and also covered all faults as represented in the table (8) and APFD value of each technique as represented in

Fig. (2-5). Table (9) represents the prioritized order of test cases in test suite 1 for different prioritization techniques.

Table (9): Prioritized order of test cases in test suite1 for no order, reverse order, random order and proposed order.

No order	Reverse order	Random order	Proposed order
T1	T8	T1	T4
T2	T7	T5	T3
T3	T6	T3	T5
T4	T5	T8	T8
T5	T4	T4	T1
T6	T3	T2	T2
T7	T2	T6	T6
T8	T1	T7	T7

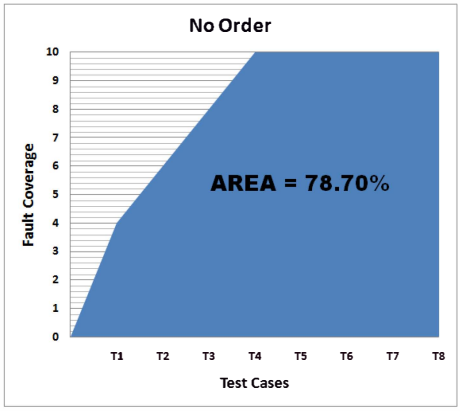

Fig. (2) Percentage of fault detected by No Ordering (Area = 78.70 %) on test suite 1.

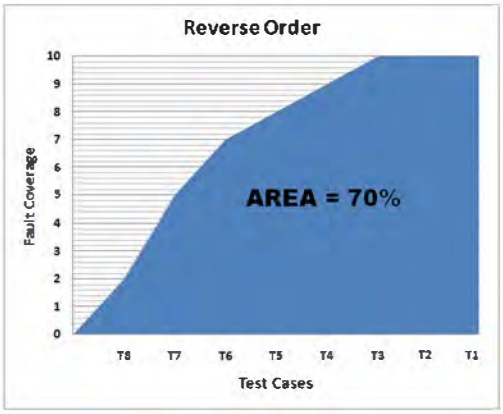

Fig. (3) Percentage of fault detected by Reverse Ordering (Area = 70%) on test suite 1.

Although APFD doesn't consider execution time, but we consider the execution time of Test suite 1 and 2 for selecting and prioritizing test cases, and later we used APFD to compare our results with other prioritization techniques as shown in Fig. (2-5), which also clearly indicates the superiority of the proposed approach over no prioritization, reverse prioritization and random prioritization, as proposed MOPSO obtains highest APFD. Fig. (6) shows the execution time of test suite 1 and 2 for no ordering, reverse ordering, random ordering and proposed MOPSO techniques and it also shows that the time needed in case of proposed approach is lesser then the others.

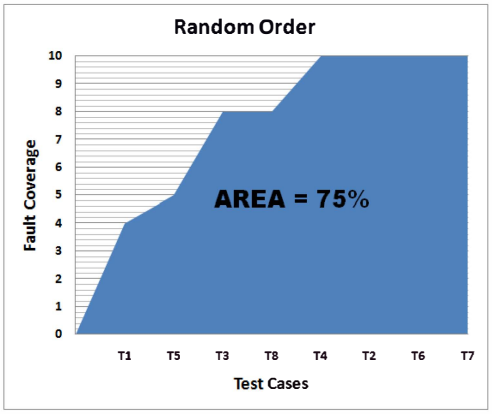

Fig. (4) Percentage of fault detected by Random ordering (Area = 75 %) on test suite 1

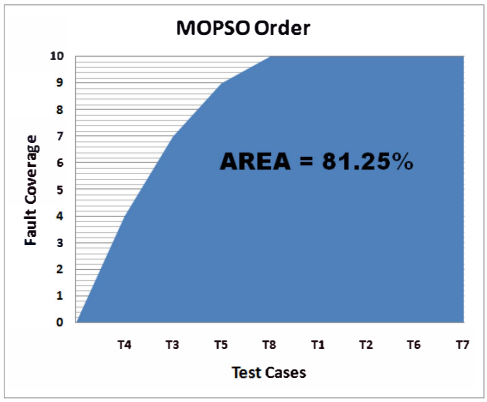

Fig. (5) Percentage of fault detected by proposed MOPSO (Area = 81.25 %), on test suite 1.

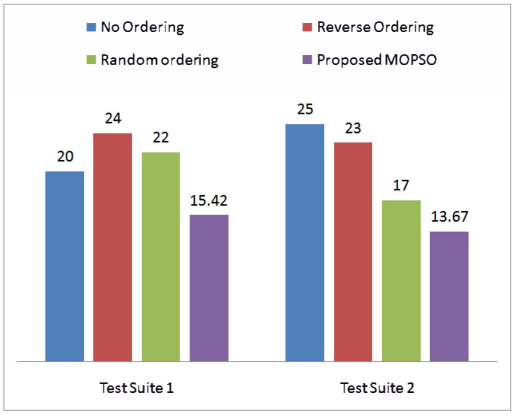

Fig. (6) The execution time obtained by No Ordering, Reverse Ordering, Random ordering and proposed MOPSO on test suite 1 and 2.

Another important aspect is stability of MOPSO. The average execution time of algorithm on 100 runs in test suite1 is 15.42, whereas minimum possible execution time is 15 so there is less deviation in the result. Hence proposed MOPSO is also stable.

6. CONCLUSION

Regression testing is one of the most important testing as it ensures that changes made to the software do not have adverse side effects. This paper proposes 3-steps approach to perform regression testing. In the first phase, redundant test cases are removed by simple matrix operation. In the second phase, test cases are selected from test suite such that selected test cases represent the minimal set which covers all faults and also at minimum execution time. For this phase, we are using multi objective particle swarm optimization (MOPSO) which optimizes fault coverage and execution time. In the third phase, we allocate priority to test cases obtained from the second phase. Priority is obtained by calculating the ratio of fault coverage to the execution time of test cases, and rest of test cases which are not selected in phase 2, are now added in the test suite in sequential order. The proposed MOPSO outperformed other approaches like No Ordering, Reverse Ordering and Random Ordering as MOPSO achieves maximum fault coverage, maximum value of APFD and minimum execution time.

REFERENCES

[1] P. Borba, A. Cavalcanti, A. Sampaio, and J. Woodcock, Eds., "Testing Techniques in Software Engineering", *Second Pernambuco Summer School on Software Engineering, PSSE2007, Recife, Brazil, December 3-7, 2007, Revised Lectures*, ser. Lecture Notes in Computer Science, vol. 6153. Springer, 2010.

[2] J.-W. Lin and C.-Y. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques," *Inf. Softw. Technol.*, vol. 51, no. 4, pp. 679–690, 2009.

[3] L. S. Souza, R. B. C. Prudencio, and F. d. A. Barros, "A constrained particle swarm optimization approach for test case selection," in *In Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering(SEKE 2010)*, Redwood City, CA, USA, 2010.

[4] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, 2007, pp. 140– 150.

[5] M. J. Harold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, pp. 270–285, 1993.

[6] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction." *Information & Software Technology*, vol. 40, no. 5-6, pp. 347–354, 1998.

[7] V. Chvatal, "A greedy heuristic for the set covering problem," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 233– 235, 1979

[8] Y. Shi, R.C. Eberhart, "A modified particle swarm optimizer," Proceedings of the IEEE World Congress on Computational Intelligence pp. 69–73, 1998

[9] Z. Li, M. Harman, and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," IEEE Trans. Software Eng., pp.225-237Apr.2007

[10] K. H. S. Hla, Y. Choi, J. S. Park, "Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting, "*Proceedings of the IEEE 8th International Conference on Computer and Information Technology Workshops*, pp. 527-532. 2008

[11] Y. Singh, A. Kaur, B.Suri,"Test case prioritization using ant colony optimization," *ACM SIGSOFT Software Engineering Notes*, Vol.35 No.4, pages 1-7, July 2010.

[12] J. Wang, Y. Zhuang, C. Jianyun, "Test Case Prioritization Technique based on Genetic Algorithm", *International Conference on Internet Computing and Information Services*, Hong Kong , pp. 173 – 175. 2011

[13] S. Sabharwal, R. Sibal, C. Sharma, "A genetic algorithm based approach For prioritization of test case scenarios in static testing," *Proceedings of the 2nd International Conference on computer and Communication Technology*, IEEE Xplore Press, Allahabad, pp: 304-309. Sept. 15-17, 2011

[14] S. Tallam, N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization, "*Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, Lisbon, Portugal, 2005.

[15] G Rothermel, R Untch, C Chu and M Harrold, "Test case prioritization: An empirical study", *In Software Maintenance, 1999. (ICSM' 99) proceedings. IEEE International conference*, on pages 179-188 IEEE, 1999.

[16] S. Elbaum, A. Malishevsky, and G Rothermel "Prioritizing test cases for regression testing", *Proc. The 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis, Portland, Oregon, USA, Aug-2000,102-112.*

[17] S. Elbaum, A. Malishevsky and G. Rothermel, "Test case prioritization: A family of empirical studies", *IEEE Transactions on Software Engineering*, vol. 28(2), 2002, pp. 159–182.

[18] W. Wong, J W. Wong, J. Horgan, S. London and H. Agrawal, "A Study of Effective Regression Testing in Practice", In *Proc. of the Eighth Intl. Symp. on Softw. Intl. Symp. onSoftw. Rel. Engr.*, pages 230–238, Nov. 1997.

[19] L. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio, Flavia de A. Barros, "A Multi-Objective Particle Swarm Optimization for Test Case Selection Based on Functional Requirements Coverage and Execution Effort",*23rd IEEE International Conference on Tools with Artificial Intelligence*, Boca Raton, FL, pp. 245 – 252. 2011.

[20] A.P. Mudgal, "A Proposed Model for Minimization of test suite" *Journal of nature inspired computing*, vol. 1, No. 2, PP. 34-37, 2013.