Improving Test Efficiency through Multiple Criteria Coverage Based Test Case Prioritization Using Modified Heuristic Algorithm

Abraham Kiran Joseph[1], G. Radhamani[2] and Vish kallimani[3]
[2]School of IT & Science,
[1,2] Dr. GRD College of Science,
Coimbatore, India.
[3]Universiti Teknologi PETRONAS,
Perak, Malaysia.
[1]abrahamkiran@ymail.com

***Abstract*— Test case prioritization involves reordering the test cases in an order that helps in attaining certain performance goals. The rate of fault detection is one of the prime goals that we tend to achieve while doing prioritization. Test cases should run in an order to increase the possibility of fault detection and it should be achieved early during the test life cycle. To reduce the cost and time of regression testing, test case prioritization should be done with the intention of periodically modifying the test suite. The humongous set of test cases makes it redundant and cumbersome for the testers who ensure quality for an end application. The fault detection capability of a prioritized test suite is improved up to 15% using Modified PSO which forms the base algorithms for prioritization. The algorithm illustrated detects serious errors at earlier phases of testing process and effectiveness between prioritized and unprioritized test cases.**

Keywords*— *Test case prioritization, Heuristic Algorithm, Modified Particle Swarm Optimization, Regression testing

I. INTRODUCTION

Due to its inherent complexity, most of the software modules developed in an Object Oriented Software has minute to complex bugs. Exhaustive testing of an application is close to impossible due to money and time constraints. Testing, by itself is a complex process and it requires new methods to keep the process of testing within manageable proportions in terms of resources allocated and cost. Test case prioritization schemes assist in organizing the test cases in a test suite and shows ways to increase the efficiency in testing [1]. At the same time as frequent reconstruction and regression testing achieves perfection, the requirement for a time constraint aware prioritization technique is mandatory. New software development processes like extreme programming also encourage a small improvement in the testing cycle and repeated implementation of faster execution of test cases [2]. As a result, there is an obvious requirement for a prioritization scheme that has the capability for more effectiveness when a test suite's permitted execution time is recognized, predominantly when that execution time is comparatively lesser.

This paper proves that when the maximum time allocated for implementation of the test cases is known earlier, an extremely efficient prioritization method can be generated. The time limited test case prioritization complication can be diminished to the NP-complete zero/one knapsack problem [3,4,5] which can often be effectively estimated using a Genetic Algorithm (GA) heuristic search technique. At the same time as GA's have been efficiently employed in many software engineering and programming language complications like test generation [6], program transformation [7] and software maintenance resource allocation [8] are present. A technique which prioritizes the regression test suites, in order that the new ordering (i) will constantly execute inside a specified time limit and (ii) will have the maximum potential for the purpose of defect detection based on gained coverage details, is followed here.

The main intent here is to reduce the overall number of test cases by prioritizing them and then using a modified version of Particle Swarm Optimization (MPSO) to further reduce the size of the test suite. A two-step process elucidates the entire work flow of reducing the optimized test suite

1. A MPSO based technique for the purpose of prioritizing a regression test suite that will execute inside a time limited execution setting.
2. An experimental assessment of the efficiency of the resultant prioritizations corresponding to (i) MPSO-produced prioritizations using different parameters.

II. RELATED WORK

In [9] conducted an empirical study to prioritize the test cases for various prioritization techniques. The results of the study indicate that the test case prioritization scheme considerably enhances the rate of fault discovery. An experimental study by Li et al [11] Li et al [12] prioritize the test cases using four algorithms namely, greedy algorithm, 2-optimal greedy algorithm, additional greedy algorithm and genetic algorithm.

Yoo et al [13] carried out a survey and comprehensive investigation of methods in regression test case selection,

 978-1-5090-2549-7/16/$31.00 ©2016 IEEE

reduction and prioritization. The survey report recommends the subject of test case prioritization is of increasing significance.

The techniques prioritize tests based on the dependency structure of the test suite itself. Wei et al [16] compare the effectiveness of two single objective Test Case Prioritization strategies, one based on statement coverage and the other one based on event coverage. An Average Percentage of Faults Detected (APFD) metric is used to determine the effectiveness of the Test Case Prioritization..

Major drawbacks of the different optimization algorithms are given below:

TABLE I. LIST OF THE DRAWBACKS OF PRIORITIZATION TECHNIQUE

Greedy algorithm	**Cost reduction is still not significant.**
Clustering approach	Provides less accuracy due to code complexity
Search Algorithm	Still cannot solve large number of test cases, possibly produces different result when number of test cases are large.
Fault Localization	The subsequent fault localization may suffer

In the literature, several research works have been carried out to prioritize the test cases based on single coverage criteria or multiple objectives. At present regression testing is conducted based on a single coverage criteria. To ensure quality, more than one prioritization technique is required to be done extensively. But it is time consuming and more expensive compared to an un-prioritized approach for optimized regression test suite. To overcome these problems a weighted method for coverage based test case prioritization is proposed here and discussed extensively.

III. PROPOSED METHODOLOGY FOR TEST CASE PRIORITIZATION TECHNIQUE IN REGRESSION TEST CASES

By using prioritization techniques test cases can be reordered to attain fault detection at an increased rate. A prioritized test suite is more effective if complete coverage has to be attained, provided that it is accomplished with the assistance of a random ordering. On the other hand, when the required time for the completion of the test case is known in advance, then an improved test case ordering might be possible.

Here a new regression test suite prioritization algorithm called the Modified Particle Swarm Optimization (MPSO) algorithm has been introduced, which prioritizes the test suite with the intention of increasing the quantity of faults, at some stage in a time constrained execution environment. The newly used modified prioritization algorithm is more than a selection algorithm where the nodes (test cases) that uncover the maximum bugs in minimum execution time are determined.

An experiment is done to analyze the MPSO algorithm with regard to its time/space overhead. The recording of MPSO for the processing of test cases is done at a stabilized condition.. These are employed for the purpose of identifying the fitness of the entire feasible ordering of the test cases. The overall proposed process for reordering the test cases is given in figure 1.

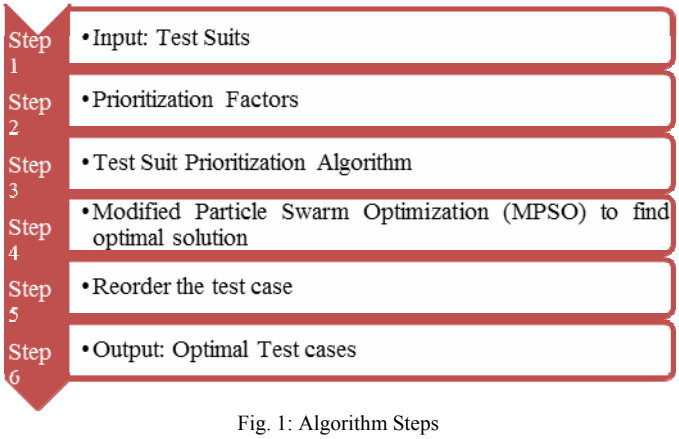

Fig. 1: Algorithm Steps

A. *Prioritization Factors*

Finding of few factors like (1) customer allocated priority of requirements, (2) working complexity, (3) transformations in requirements, (4) fault impact of requirements, (5) comprehensiveness (6) traceability and (7) execution time, is indispensable for the effective prioritization of the test cases, since they are employed in the prioritization algorithmCustomer-Assigned Priority (CP). CP indicates the amount of significance that is given to the customer's requirement. Customer's needs vary from 1 to 10 and assigned by the customer itself, where 10 is employed for the purpose of identifying the uppermost customer priority. It is indispensable to enhance the customer perceived value for satisfying the customer. Software tasks never/uncommonly employed is 45%, occasionally used is 19% and always used is 36%, which is a rough estimate. Maximum effort must be exploited for the purpose of effective faults identification which occurs on normal interval, if not these faults results in constant malfunctions

Implementation Complexity (IC)

It is the subjective determination of the complexity expected by the development members in executing the requirement. IC is estimated at the beginning of the project. Value from 0 to 10 allocated by the developer on the strength of its application complexity and by using bigger value higher complexity is indicated. Quantity of faults is higher since the requirement happens to be high in implementation complexity.

a) Changes in Requirements (RC)

Developer allocates a measure and the limit fluctuates from 1 to 10 which are used to point out the number of occasions a requirement is transformed in the development cycle by considering its origin date as a reference. The requirement transforms to 10 times when the volatility values for the entire needs are indicated on a 10 point scale. The amount of transformations for any requirement i divided to the maximum number of transformations for any requirement among the entire project requirements provides the change in requirement R_i of that requirement i. When the ith requirement is

transformed M occasions, N indicates the maximum number of requirements, subsequently the requirement change of I, R_i can be computed as follows,

$$R_i = \left(\frac{M}{N}\right) \times 100 \quad (1)$$

More or less 50% of error commences during the requirement stage of all faults noticed in the project. The most significant factor for the breakdown of the project is the transformation in the requirement stage.

b) Fault Impact of Requirements (FI)

FI permits the development team members to recognize the requirement which the customer indicates as breakdowns. At the same time as a system progresses to several versions, the developer can make use of the earlier data which is gathered from versions to recognize needs that are expected to cause the error. FI depends on the amount of field failure and in-house breakdowns.

c) Completeness (CT)

Completeness conveys a requirement as determined by a function to be performed, the level of success, the limits under which the function is to be implemented, and several restrictions, which have an effect on the expected solution, for instance an interface constraint. Customer allocates a value in the range between 0 and 10, when the requirement is chosen for reuse subsequent to analyzing the CT of every requirement. Customer satisfaction like quick response time of the software reaction to the user demand can be enhanced, by considering the CT.

d) Traceability (TR)

Traceability computes the mapping that is present between requirements and test. It assists in deciding whether a requirement is tested adequately, which is extremely tricky for the testers when the test cases are not applicable to that specific user requirements. A complication which is extremely common is lack of traceability. When traceability is not adequate or poor that possibly will cause project over runs and breakdowns. Instead of systematic process this process is carried out by following a specialized method. It is familiar that unsuitable traceability is somewhere a source for quite a lot of software breakdowns.

Execution Time (ET)

The time essential for performing the regression testing is based on that particular test suite. On the other hand, with the assistance of efficient prioritization technique, test cases can be reorganized by the testers in order to obtain fault detection at an increased rate in the system. It must be made sure that the software has been tested very cautiously. A prioritized test is more efficient when execution requires to be terminated after certain period of time, it can also be accomplished by means of a random ordering. On the other hand, if the anticipated time for the execution of the test case which is recognized previously, at that moment a better test case ordering might be possible. The first preference of this research is to prioritize the regression testing cases.

B. Proposed Prioritization Algorithm

Values for the entire six factors are allocated at some point in the design analysis stage and change constantly all through the software development process as the project develops. Requirement factor value for every requirement i, Rfv_i is computed as given below:

$$Rfv_i = \frac{\sum_{j=1}^{\tau} factor_j}{\tau} \quad (2)$$

Significantly, it corresponds to the requirement factor value for requirement i, which indicates the mean of factor value. RFV indicates a measure of significance of testing a requirement and it is employed in the calculation of Test Case Weight (TCW). Following the traceability, mapping the TCW of a test case is taken as the product of two constituents as follows: (i) The average RFV of the requirements the test case maps to; (ii). Considering there is a total of n requirements, when test case t maps to i number of requirements, subsequently the test case weight Tcw_t is computed as given below.

$$Tcw_t = \left(\frac{\sum_{k=1}^{i} R_k^i v_k}{\sum_{k=1}^{n} R_k^i v_k}\right) * \frac{i}{n} \quad (3)$$

The test cases are arranged for the purpose of effective execution in accordance with the descending order of TCW, in order that the test case with the maximum TCW runs at the beginning. The TCW and RFV are provided as the input to the MPSO. The fitness can be computed with the help of the factor values and weight age allocated. The test case with highest fitness value will be chosen as the high priority test case. Here, the MPSO reorganized the test cases by means of the established criteria without information of the faults that exist in the system. Proposed Modified PSO for prioritization of test cases

Particle swarm optimization algorithm, is for optimizing difficult numerical functions in accordance with the metaphor of human social communication, which is able of mimicking the ability of human societies capability to process knowledge [17]. The key constituents are artificial life and evolutionary computation. As it is reported in [18], this optimization approach can be utilized to solve many optimization problems and is works better than GA. The process starts with a compilation of random particles, M. The ith particle is designated through its position as a point in S-dimensional space, in which S denotes the number of variables. Throughout the progression, every particle i gets hold of three values, that is to say its current position (X_i), the best position it entered in previous cycles (P_i), its flying velocity (V_i). All these three values are given as follows:

$$\begin{aligned} \text{Current position } X_i &= (x_{i1}, x_{i2}, \ldots, x_{iS}) \\ \text{Best previous position } P_i &= (p_{i1}, p_{i2}, \ldots, p_{iS}) \\ \text{Flying velocity } V_i &= (v_{i1}, v_{i2}, \ldots, v_{iS}) \end{aligned} \quad (4)$$

In every time interval (cycle), the position (P_g) of the best particle (g) is found as the best fitness of the entire particles.

As a result, each particle revises its velocity V_i in order to get nearer to the best particle (g), as given below:

$$\text{New } V_i = \omega \times \text{current } V_i + c_1 \times \text{rand}() \times (P_i - X \quad (5)$$

As like that, by means of the new velocity V_i, the particle's revised position is converted into:

$$\text{New position } X_i = \text{current position } X_i + \text{New } V_i$$

$$V_{max} \geq V_i \geq -V_{max} \quad (6)$$

Where c_1 and c_2 indicate two positive constants, that is learning factors (as a rule $c_1 = c_2 = 2$); rand ( ) and Rand ( ) denotes two random functions in the range [0, 1], V_{max} is kept as an upper limit denoting the maximum change of particle velocity), and ω represents an inertia weight. It is used as an improvement to direct the effect of the earlier history of velocities on the existing velocity. Global search and the local search can be balanced by the operator ω which was begins to diminish linearly in accordance with time from a value of 1.4–0.5.

PSO initiates with a population of random solutions ''particles'' in a D-dimension space. The ith particle is indicated as $Xi = (x_{i1}, x_{i2}, \ldots, x_{iD})$. Every particle maintains its coordinates in hyperspace, which are connected with the fittest solution. Value of the fitness for the particular particle i (pbest) is also accumulated as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$. The global description of the PSO tests the overall best value (gbest), in addition to its location, attained is the best in the entire population. At each step, PSO includes transforming the velocity of every particle in the direction of its pbest and gbest in accordance with Eq. (7). The velocity of that particular particle i is indicated as $Vi = (v_{i1}, v_{i2}, \ldots v_{iD})$. Acceleration is weighted through a random factor, with separate random numbers being produced for the purpose of acceleration in the direction of pbest and gbest. The position of the ith particle is then revised based on equation (8)

$$V_{id}(t+1) = w \times V_{id}(t) + c_1 r_1 (P_{id} - x_{id}(t)) \quad (7)$$

$$x_{id}(t+1) = x_{id}(t) + c v_{id}(t+1) \quad (8)$$

Where, P_{id} and P_{gd} denote pbest and gbest respectively. A number of modifications have been proposed to improve the performance of PSO algorithm in terms of its speed and convergence toward the global minimum. Consider a local-oriented concept (lbest) with several neighborhoods is introduced. gbest version executes most excellent in terms of median number of iterations for effective convergence. But, Pbest version with neighborhoods of two is very resistant to local minima. It is observed from the thorough investigation of PSO that ω was not taken into account during the early stage of PSO algorithm..

In case of equation (7), term of $C_1 r_1 (P_{id} - X_{id}(t))$ represents the individual movement and term of $C_2 r_2 \left(P_{gd} \quad X_{id}(t)\right)$ denotes the social behavior in recognizing the global best solution. Here, in order to attain accurate solution and quick convergence of algorithm, parameters are used in IPSO algorithm has been initialized based on Table 2.

TABLE II. RATE OF PARAMETERS FOR IPSO ALGORITHM

Parameter	Rate
Problem Dimension	11
Number of Particles	100
Number of Iteration	100
Mutation Probability P_m	0.1
Inertia Weight Factor	$\omega_{min} = 0.4, \omega_{max} = 0.9$
r_1, r_2	Selected Randomly in (0,1)
c_1	1
c_2	1.5
C	0.9

In proportion to equation (7), the velocity update of the particle includes three segments. The initial term is its individual existing velocity of particles. The second term represents cognitive segment which indicate the particles individual experiences and the last term is social component which signifies the social communication among the particles. Concerning the equation (7), it is recognized that best position of particles occurs proportional to pbest_i. It can be observed that: if a particle's existing position overlaps with the global best position (gbesti), the particle will only go away with this point when the inertia weight and its existing velocity are diverse from zero. When the particles' current velocities in swarm are near to zero, subsequently these particles will not progress once arrive at the global best particle, as a result the entire particles will converge to the best position (gbest) recognized until now by the swarm.

At this point, if this positions fitness is not the expected global optimal, then it results in the premature convergence phenomenon. In order to overcome this limitation, an Improved Particle Swarm Optimization (IPSO) is proposed by introducing the mutation operator is often used in genetic algorithm. This process can make certain particles to jump out local optima and explore in nearby region of the solution space. Here, the mutation probability (PM) is dynamically adjusted based on the diversity in the swarm. The aim with mutation probability is to avoid the premature converge of PSO to local minima. In this study, PM is taken as 0.1.

Pseudo Code of Modified PSO

Step 1: int[] similarCount = new int[m]; // at initialization stage // Next code is employed to replace velocity Eqn (3.1) // in standard PSO process

Step 2: FOR ($i = 0; i < m; i++$) { // for each particle

Step 3: If($ll = g \&\& |\Delta F_i| < \varepsilon$

Step 4: THEN similarCount[i] ++; // add 1

Step 5: IF (similarCount[i] > T_s) //predefined count

Step 6: **THEN** replace (the i^{th} particle);

Step 7: **ELSE** execute Equation (3.1) in standard PSO);

}

The optimal solution is searched in MPSO on the origin of needed population which additionally can be substituted with the new set of population. In accordance with the complication, the creation and initialization of test cases (population) is carried out. RFV and TCW are elected as the fitness criterion.

C. *Prioritized Test Suite Effectiveness*

In order to estimate the performance of the prioritization scheme employed in this work, it is necessary to assess efficiency of the series of the test suite. Efficiency will be assessed through the rate of faults identified. Certain metrics were employed to compute the range of efficiency, they are given below.

APFD Metric

In order to compute the objective of increasing a subset of the test suite's rate of fault detection, here employed a metric known as APFD that computes the rate of fault detection per percentage of test suite execution. The APFD is computed by considering the weighted average of the amount of faults identified at some point in the run of the test suite. APFD can be computed as given below:

$$Apfd = 1 - \left(\frac{Tf_1 + Tf_2 + \cdots + Tf_m}{nm}\right) + \frac{1}{2n} \quad (9)$$

Where, n represents the number of test cases, m indicates the number of faults and $(Tf_1 \ldots Tf_m)$ represents the position of initial test T that exposes the fault.

IV. Results and Discussion

In order to measure the performance of the prioritization technique employed in this research work, it is essential to measure the effectiveness of the ordering of the test suite. Efficiency will be assessed with the assistance of rate of faults detected. The following metric is employed for the purpose of calculating the level of effectiveness.

Performance is assessed with the help of the APFD metric. The APFD value is a measure demonstrates how promptly the faults are recognized for a specific test suite set. Consider T indicate the test suite under assessment, F indicates the amount of faults enclosed in the program under test P, n indicates the total number of test cases and $reveal\ (i, T)$ is the spot of the initial test in T exposes fault i. Following is the formula for computing the APFD metric.

$$Apfd(T, P) = 1 - \left(\frac{\sum_{i=1}^{f} reveal(i,T)}{nf}\right) + \frac{1}{2n} \quad (10)$$

Consider the quantity of test cases $n = 8$ and the quantity of faults $f = 6$. This can be given in the table 3 below:

Based on the Table 2 and Fig 2, it is found that the prioritized test cases recognize the faults during the early phase. The APFD measure of prioritized test cases is beyond the non prioritized order for both projects. Fig 3 & Fig 4 shows the fault recognized at some stage in each test case is listed. It is clear from Fig 3 and Fig 4, that the proposed method identifies the rigorous fault during the early stage. As a result it will diminish the computational time.

TABLE III. The Faults Detected by the Test Suites

Faults Covered	Test Cases							
	T1	T2	T3	T4	T5	T6	T7	T8
F1	X							
F2	X	X						
F3			X		X	X		
F4				X	X	X		
F5					X	X		
F6						X		
No. of faults	2	1	1	1	3	3	0	0
Time Taken	5	7	8	4	9	10	12	6

Number of test cases is 8, i.e., $\{T1, T2, T3, T4, T5, T6, T7, T8\}$ and the number of faults occur during the regression testing is 6, i.e., $\{F1, F2, F3, F4, F5, F6\}$. The prioritized test suits with test sequence $\{T6, T5, T1, T4, T2, T3, T7, T8\}$, then the APFD metric after prioritization is

$$Apfd(T, P) = 1 - \left(\frac{3+3+2+1+1+1}{8*6}\right) + \frac{1}{2*8} \quad (11)$$

$$Apfd(T, P) = 1 - \left(\frac{11}{48}\right) + \frac{1}{16} \quad (12)$$

$$Apfd(T, P) = 1 - (0.2291) + 0.0625 \quad (13)$$

$$Apfd(T, P) = 0.7084 \quad (14)$$

The APFD metric before prioritization is

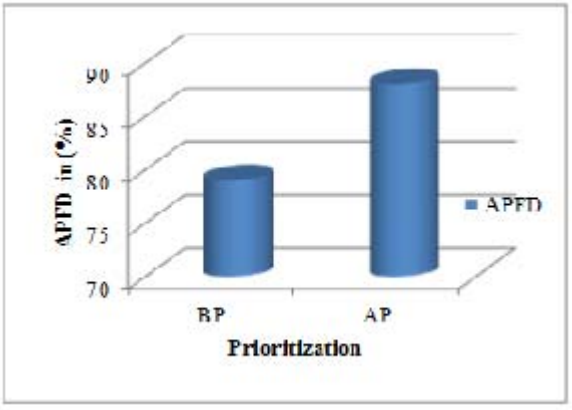

Fig. 2: APFD metric for the proposed application

From Fig 3, it is revealed that the test case 5 recognizes the most number of rigorous faults. The test case 5 is top in the final prioritized order and it is carried out first. As a result the proposed prioritization technique will recognize most number of rigorous faults during an early phase. The evaluation is drawn among prioritized and non prioritized test case, which demonstrates the quantity of test cases essential to discover the entire faults are considerably less in the scenario of prioritized test case compared against non prioritized test case. It is clear from Fig 4 that the new prioritization technique requires only 30% of test cases to discover the entire faults. APFD is the

segment of area underneath the curve in Fig 4, plotting percentage of test cases implemented against percentage of faults detected.

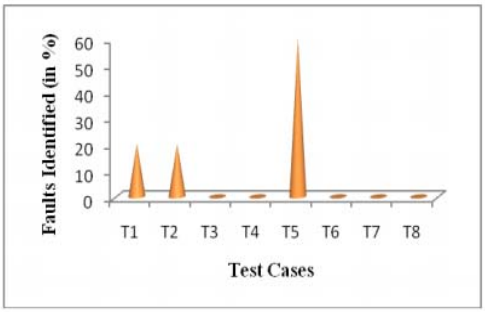

Fig. 3: Fault detection performance of each test case

The Fig 4 illustrates the Prioritized order of test cases identify the entire rigorous faults during the early phase when compared to the non prioritized order of test cases. Accordingly, the proposed method of test case prioritization process will considerably diminish the re-execution time of the project through prioritizing the most significant test cases.

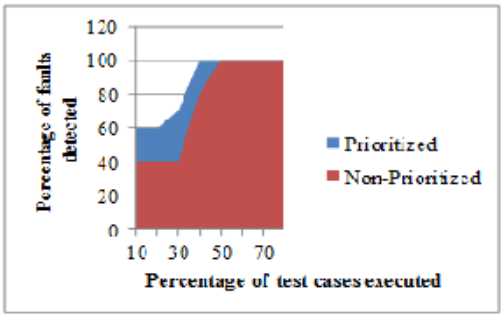

Fig.4: Comparison of Prioritized and Non-Prioritized test cases

Conclusion

This paper identifies and measures the challenges in the development of regression testing and test case prioritization. The proposed method uses seven factors that are discussed elaborately to prioritize the test cases. The novelty here is particularly the trace events process, which is one of the factors discussed in the methodology. Here, the effectiveness of the proposed technique is evaluated with the help of the APFD metric. The proposed method gives the better rate for severe faults detection which is shown through results. In future, more factors can be added to increase the efficiency of the MPSO approach and give more prioritized and optimized test suite

REFERENCES

[1] Smith, Temple F., and Michael S. Waterman. Identification of common molecular subsequences. Journal of molecular biology147.1, 1981, pp. 195-197.

[2] Elbaum, Sebastian, Alexey G. Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. Software Engineering, IEEE Transactions on 28.2, 2002, pp. 159-182.

[3] Poole, Charles, and Jan Willem Huisman. Using extreme programming in a maintenance environment. IEEE Software 18.6, 2001, pp. 42-50.

[4] Michael, R. Garey, and S. Johnson David. Computers and intractability: a guide to the theory of NP-completeness. WH Free. Co., San Fr, 1979, pp.90-91.

[5] Rothermel G, Untch RJ, Chu C, Prioritizing test cases for regression testing. IEEE Transaction on Software. Eng., vol.27(10), 2001, pp. 929-948.

[6] Chu P, Beasley J. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics, vol.4(1), 1998, pp.63-86.

[7] Pargas RP, Harrold MJ, Peck RR. Test-data generation using genetic algorithms. Software Testing, Verification and Reliability, vol.9(4), 1999, pp.263-282.

[8] Fatiregun D, Harman M, Hierons RM. Evolving transformation sequences using genetic algorithms. In Proc. Of 4th SCAM, 2004, pp. 66-75.

[9] Antoniol G, Penta MD, Harman M. Search-based techniques applied to optimization of project planning for a massive maintenance project. In Proc. of the 21st ICSM, Washington, DC, 2005.pp. 240-249.

[10] Hyunsook D, Siavash M, Ladan T, Gregg R. The Effects of Time Constraints on Test Case Prioritization: A Series of Controlled Experiments. IEEE Trans on Software Engineering, vol.36(54), 2010, pp.593- 617.

[11] Li Z, Harman M, Hierons RM. Search Algorithms for Regression Test Case Prioritization. IEEE Transactions On Software Engineering, vol.33(4), 2007, pp.225 – 237.

[12] Li S, Bian N, Chen Z, You D, He Y. A Simulation Study on Some Search Algorithms for Regression Test Case Prioritization. In proceedings of IEEE International Conference on Software Quality, 2010.pp.72-81.

[13] Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey. Journal of Software: Testing, Verification and Reliability, vol.22(2), 2012, pp.67-120.

[14] Krishnamoorthi R, Sahaaya Arul Mary, SA. Regression Test Suite Prioritization using Genetic Algorithms, International Journal of Hybrid Information Technology, vol.2(3), 2009, pp.35-52.

[15] Shifa-e-Zehra Haidry and Tim Miller "Using Dependency Structures for Prioritization of Functional Test Suites" IEEE Transactions On Software Engineering, Vol. 39(2), 2013, pp.258-275.

[16] Sun, W., Gao, Z., Yang, W., Fang, C., & Chen, Z. Multi-Objective Test Case Prioritization for GUI Applications, SAC'13, 2013, pp.1074-1079.

[17] Kennedy J, Eberhart R. Particle swarm optimization. Proceedings of the 1995 IEEE International Conference on Neural Networks. 1995, pp.1942–1948.

[18] Kennedy J. The particle swarm: social adaptation of knowledge. Proceedings of the 1997 International Conference on Evolutionary Computation. Indianapolis 1997, pp.303–308.