International Journal of Research and Development in Applied Science and Engineering (IJRDASE)
ISSN: 2454-6844

PSO based Optimized Software Testing Technique

Uzma Jafri
Department of CSE
Integral University, Lucknow
jafriuzma82@gmail.com

Halima Sadia
Department of CSE
Integral University, Lucknow
halima@iul.ac.in

Jameel Ahmad
Department of CSE
Integral University, Lucknow
jameel@iul.ac.in

Abstract: Regression testing is an expensive, but important, process. Unfortunately, there may be insufficient resources to allow for the re–execution of all test cases during regression testing. In this situation, test case prioritisation techniques aim to improve the effectiveness of regression testing, by ordering the test cases so that the most beneficial are executed first. Previous work on regression test case prioritisation has focused on Greedy Algorithms. However, it is known that these algorithms may produce sub–optimal results, because they may construct results that denote only local minima within the search space. By contrast, meta–heuristic and evolutionary search algorithms aim to avoid such problems. This paper presents results from an empirical study of the application of several greedy, meta–heuristic and evolutionary search algorithms to six programs, ranging from 374 to 11,148 lines of code for 3 choices of fitness metric. The paper addresses the problems of choice of fitness metric, characterisation of landscape and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning Greedy Algorithms. They shed light on the nature of the regression testing search space, indicating that it is multi–modal. The results also show that Genetic Algorithms perform well, although Greedy approaches are surprisingly effective, given the multi–modal nature of the landscape.

Keywords: APFD, APBC, APDC, Genetic Algorithm, and PSO.

1. Introduction:

Regression testing is a frequently applied but expensive maintenance process that aims to (re-)verify modified software. Many approaches for improving the regression testing processes have been investigated. Test case prioritisation [17] is one of these approaches, which orders test cases so that the test cases with highest priority, according to some criterion (a 'fitness metric'), are executed first. Rothermel et al. [11] define the test case prioritisation problem and describe several issues relevant to its solution. The test case prioritisation problem is defined (by Rothermel et al.) as follows: *The Test Case Prioritisation Problem:*

Given: T,a test suite;PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find T'ϵPT such that ($\forall T''(T''\in PT)(T''\neq T')[f(T')\geq f(T'')]$

Here, PT represents the set of all possible prioritisations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering. Test case prioritisation can address a wide variety of objectives [11]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing, to reach 100% coverage soonest or to ensure that the maximum possible coverage is achieved by some pre–determined cut–off point. Of course, the ideal order would reveal faults soonest, but this cannot be determined in advance, so coverage often has to serve as the most readily available surrogate. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [10]. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process. In previous work, many techniques for regression test case prioritisation have been described. Most of the proposed techniques were code–based, relying on information relating test cases to coverage of code elements. In [11], Rothermel et al. investigated several prioritising techniques such as total statement (or branch) coverage prioritisation and additional statement (or branch) coverage prioritisation, that can improve the rate of fault detection. In [13], Wong et al. prioritised test cases according to the criterion of 'increasing cost per additional coverage'. In [12], Srivastava and Thiagarajan studied a prioritisation technique that was based on the changes that have been made to the program and focused on the objective function of "impacted block coverage". Other non–coverage based techniques in the lit-erature include fault–exposing–potential (FEP) prioritisation [11], history–based test prioritisation [11], and the incorporation of varying test costs and fault severities into test case prioritisation [5, 6]. Greedy Algorithms have been widely employed for test case prioritisation. Greedy Algorithms incrementally add test cases to an initially empty sequence. The choice of which test case to add next issimple: it is that which achieves the maximum value for the desired metric (e.g., some measure of coverage).

2. Related Work:

This work aims to find optimization techniques for test suite optimization (TSO) in Regression testing(R/T). **[1]. Zheng Li**, have studied various optimization techniques and algorithms in order to find the technique that should be used for test suite optimization (TSO) in regression testing. The techniques like hill climbing, greedy algorithm, additional greedy algorithm,2-optimal greedy algorithm, genetic algorithm, particle swarm optimization(PSO) are studied and compared using different parameters. These techniques are compared on the basis of their code coverage area, computational effort and execution time.

Where the creation, understanding, and assessment of software testing and regression testing techniques are concerned, controlled experimentation is an indispensable research methodology. Obtaining the infrastructure necessary to support such experimentation, however, is difficult and expensive. As a result, progress in experimentation with testing techniques has been slow, and empirical data on the costs and effectiveness of techniques remains relatively scarce. To help address this problem, **[2] V. Basili,** have been designing and constructing infrastructure to support controlled experimentation with testing and regression testing techniques. This work reports on the challenges faced by researchers experimenting with testing techniques, including those that inform the design of our infrastructure. The work then describes the infrastructure that we are creating in response to these challenges, and that we are now making available to other researchers, and discusses the impact that this infrastructure has and can be expected to have.

This survey has presented by **[3]. Agrawal G**. on empirical work involving program slicing of relatively large scale programs (ranging from a few thousand lines of code to tens and even hundreds of thousands of lines of code). The results concern questions about the efficacy of slicing, its applications, the nature of the slices themselves, and the interplay between slicing and other source code analyses. Results from over thirty works, that contain empirical results are included. Some general trends in the results reported are evident and also some directions for future work emerge.

[4]. T. Abdel-Hamid presented a search{based approach for planning resource allocation in large software projects, which aims to find an optimal or near optimal order in which to allocate work packages to programming teams, in order to minimize the project duration. The approach is validated by an empirical study of a large, commercial Y2K massive maintenance project, comparing random scheduling, hill climbing, simulating annealing and genetic algorithms, applied to two different problem encodings. Results show that a genome encoding the work package ordering, and alertness function obtained by queuing simulation constitute the best choice, both in terms of quality of results and number of fitness evaluations required to achieve them.

Regression testing assures changed programs against unintended amendments. Rearranging the execution order of test cases is a key idea to improve their effectiveness. Paradoxically, many test case prioritization techniques resolve tie cases using the random selection approach, and yet random ordering of test cases has been considered as ineffective. Existing unit testing research unveils that adaptive random testing (ART) is a promising candidate that may replace random testing (RT). In this work, **[5]. K. P. Chan**, not only propose a new family of coverage-based ART techniques, but also show empirically that they are statistically superior to the RT-based technique in detecting faults. Furthermore, one of the ART prioritization techniques is consistently comparable to some of the best coverage-based prioritization techniques (namely, the “additional” techniques) and yet involves much less time cost**.** Test case prioritization is a means to achieve target objectives in software testing by reordering the execution sequences of test suites. Many existing test case prioritization techniques use random selection to resolve tie cases. Paradoxically, the random approach for test case prioritization has a long tradition to be deemed as ineffective. Adaptive random testing (ART), which aims to spread test cases as evenly and early as possible over the input domain, has been proposed for test case generation. Empirical results have shown that ART can be 40 to 50% more effective than random testing in revealing the first failure of a program, which is close to the theoretical limit. In regression testing, however, further refinements of ART may be feasible in the presence of coverage information.

3. Methodology:

Particle Swarm Optimization (PSO) is an evolutionary computation technique, which is inspired by flocks of birds and shoals of fish. In PSO, a number of simple entities (the particles) are placed in the space of some problem and each evaluates its fitness as its current location. Each particle determines its movement through the space by considering the particle which had the best fitness and the history of its own, then it moves with a velocity. At last, the swarm is liable to move near the best area. The speed and position of every molecule is balanced by the accompanying formulas:

$$V_{id} = WXV_{id} + c_1Xrand()X(P_{id}-X_{id})+C_2XRand()X(P_{ad}-X_{id})$$
$$X_{id} = X_{id} + V_{id}$$

where c1 and c2 are termed the cognitive and social learning rates. These two parameters control the relative importance of the memory of the particle itself to the memory of the neighborhood. The variable rand() and Rand() are two random functions that is uniformly distributed in the range [0,1]. Xi = (Xi1, Xi2, … , XiD) represents the ith particle. Pi = (Pi1, Pi2, …, PiD) represents the best previous position of the ith particle. The symbol g represents the index of the best particle among all the particles. Vi = (Vi1, Vi2, … , ViD) represents the velocity of the ith particle. Variable is the inertia weight. The general process of PSO is as follows.

```
Do
Calculate fitness of particle
Update pbest if the current fitness is better than pbest
Determine nbest for each particle: choose the particle with the best fitness value of all the neighbors as the nbest
For each particle Calculate particle velocity according to (1)
Update particle position according to (2)
While maximum iterations or minimum criteria is not attained
```

Since the introduction of the PSO algorithm, several improvements have been suggested. In 1998, inertia weight was first proposed by Shi and Eberhart [9]. The function of inertia weight is to balance global exploration and local exploitation. In the following year, Clerc proposed the

constriction factors to ensure the convergence of PSO [13]. Eberhart and Shi compared inertia weight with constriction factors and found that the constriction factors was better convergence than inertia weight [10].
The particle swarm optimization was proposed by Kennedy and Eberhart [8, 9]. The PSO is able to solve multidimensional optimization problems. It is based on simulating the social behavior of swarm of bird flocking, bees, and fish schooling. By randomly initializing the algorithm with candidate solutions, the PSO successfully leads to a global optimum. This is achieved by an iterative procedure based on the processes of movement and intelligence in an evolutionary system.
In PSO, each potential solution is represented as a particle. A position and a velocity are associated with each particle. The position and velocity of the ith particle are given by

$$\vec{x_i} = (x_{i.1}, x_{i.2}, \dots\dots x_{i.N}),$$
$$\vec{v_i} = (v_{i.1}, v_{i.2}, \dots\dots v_{i.N})$$

Experimental Design
To improve the generality of the results reported in this paper, the basic search experiment was instantiated with several values of the three primary parameters that govern the nature and outcomes of the search, namely:
1. The program to which regression testing is applied. Six programs were studied, ranging from 374 to 11,148 lines of code and including real as well as 'laboratory' programs.
2. The coverage criterion to be optimised.The three choices studied were block, decision and statement coverage for each program.
3. The size of the test suite. Two granularities of test suite size were used: small suites of size $8 - 155$ and large suites of size $228 - 4,350$. Of course, there is a connection between the first and third category since a larger program will typically require more test cases.

3.1 Subjects
In our study we used two groups of programs, called 'small programs' and 'large programs'1, from a total of six C programs and their corresponding test suites (see Table 1). The programs and test suites were from an infrastructure [4], that is designed to support controlled experimentation with software testing and regression testing techniques. Print tokens2 and Schedule2 are variations of the programs Print tokens and Schedule, respectively. These programs were assembled by researchers at Siemens Corporate Research for experiments with control–flow and data–flow test adequacy criteria [10]; Space and Sed are large programs.

Table 1: Experimental Subjects

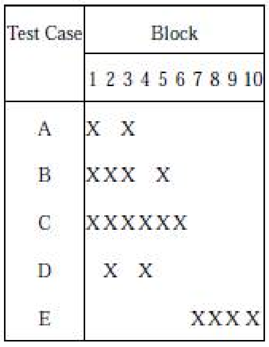
Subject	LoC	Blocks	Decision	Test Pool size	Average Small test Suite size	Average Large test Suite size
Print_tokens	726	126	123	4.130	16	318
Print_tokens2	570	103	154	4.115	12	388
Schedule	412	46	56	2.650	19	228
Schedule2	374	53	74	2.710	8	230
Space	9.564	869	1.068	13.585	155	4.350
Sed	11.148	951	2.331	1.293		1.293

Space was developed for the European Space Agency, and Sed is the Unix stream editor string processing utility. For the first four 'small' programs, the researchers at Siemens created a test pool containing possible test cases for each program [10]; Rothermel and his colleagues constructed test pools for the two 'large' programs following the same approach.
The infrastructure provided by Rothermel et al. [18] was used to obtain test suites in the following manner: in order to produce small test suites a test case is selected at random from the test pool and is added to the suite only if it adds to the cumulative branch coverage. This is repeated until the test suite achieves full branch coverage. In order to produce large test suites, test cases are randomly selected from the test pool and added to the test suite until full branch coverage is achieved.

3.2 Effectiveness Measure:
The fitness metrics studied are based upon APFD (Average of the Percentage of Faults Detected) [18], which measures the weighted average of the percentage of faults detected over the life of the suite. However, it is not (normally) possible to know the faults exposed by a test case in advance and so this value cannot be estimated before testing has taken place. Therefore, coverage is used as a surrogate measure. It is not the purpose of this paper to enter into the discussion of whether or not coverage is a suitable surrogate for faults found. Coverage is also an important concern in its own right, due to the way in which it has become a part of 'coverage mandates' in various testing standards (for example, avionics standards [14]). The presence of these mandates means that testers must achieve the mandated levels of coverage irrespective of whether or not they may believe in coverage *per se*. Depending on the coverage criterion considered, three metrics were used in the paper:
1. APBC (Average Percentage Block Coverage) This measures the rate at which a prioritised test suite covers the blocks.
2. APDC (Average Percentage Decision Coverage) This measures the rate at which a prioritised test suite covers the decisions (branches).
3. APSC (Average Percentage Statement Coverage) This measures the rate at which a prioritised test suite covers the statements.
Consider the APBC metric as an example and a test suite T containing n test cases that covers a set B of m blocks. Let TBi be the first test case in the order T′ of T that covers block i. The APBC for order T′ is given by the equation:

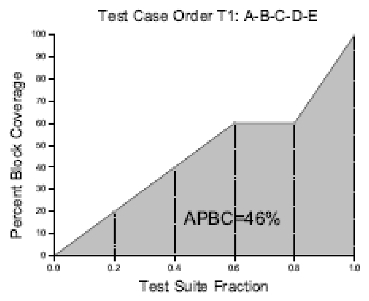
$$APBC = 1 - \frac{TB_1 + TB_2 + \cdots TB_m}{nm} + \frac{1}{2n}$$

APBC measures the weighted average of the percentage of block coverage over the life of the suite. APBC values range from 0 to 100; higher numbers imply faster (better) coverage rates.

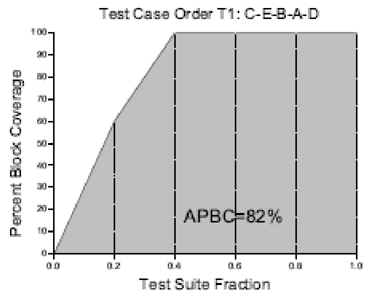


Test Case	Block 1	2	3	4	5	6	7	8	9	10
A	X		X							
B	X	X	X		X					
C	X	X	X	X	X	X				
D		X		X						
E							X	X	X	X

(a) Test Suites and Block Coverage



(b) APBC for order T1



(c) APBC for Order T2

Fig 1: Example Illustrating the APBC Measurer

To illustrate this measure, consider an example program with 10 blocks and a test suite of 5 test cases, A through E, each with block coverage characteristic. Consider two orders of these test cases, order T1: A–B–C–D–E, and order T2: C–E–B–A–D. Figure 3.4(b) and Figure 1(c) show the percentages of block coverage as a function of the fraction of the test suite used, for the two orders T1 and T2 respectively. The area under the curve represents the average of the percentage of block coverage over the life of the test suite. APDC and APSC are defined in a similar manner to APBC, except that they measure rate of coverage of decisions and of statements respectively.

4. Result and Discussion:

Software testing is a frequently applied but expensive maintenance process that aims to re–verify modified software. Many approaches for improving the testing processes have been investigated. Test case prioritisation [1] is one of these approaches, which orders test cases so that the test cases with highest priority, according to some criterion (a 'fitness metric'), are executed first. Rothermel et al. define the test case prioritisation problem and describe several issues relevant to its solution. The test case prioritisation problem is defined (by Rothermel et al.) as follows:

Every time software is modified it is required to be tested. Test case prioritisation is a black box technique that is used to identify bugs that have emerged at the time of software modification. This testing depends upon quality of test suites. A good quality test suite leads to early fault detection and hence improves quality of testing. Test suite optimization techniques are used for prioritization, selection and minimization of test cases in a test suite. Hill climbing, greedy algorithm, additional greedy algorithm,2-optimal greedy algorithm, genetic algorithm, particle swarm optimization(PSO) are optimization algorithm. Particle swarm optimization is a optimization Technique inspired by social behaviour of bird flocking. In case of particle swarm optimization the next position of a particle depends on its local best position and the global best position of the swarm.

PSO includes following steps:

1. Create random population of particle.
2. Find fitness value of each particle in the population.
3. If particles current position is better than its previous position make it personal best (pbest). For first iteration the initial position of the particle will be particles best position.
4. Find the global best (gbest) particle .
5. Find the new position of the particle with the help of personal best and global best position of the particle.

Change in position of particle is represented as velocity update.

- Vi(t+1)= w*Vi(t)+c1*rand*(pbest-p)+c2*rand*(gbest-p)
- newP(t+1)=P+Vi(t+1)

Vi(t+1)-update velocity
C1-Weight of local information
C2-Weight of global information
Rand- 0<=rand<=1
pbest- Particle personal best position
gbest- Particle global best position

6. Move particle to their new position.
7. Follow the step 3 to 6 .
8. We stop when no more better particles are found in next 50(or according to the problem space) iteration.

Test case prioritization:

Table 2: Test case coverage record

Test Cases➔	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Line of															

Code:															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
13	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1
14	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1
15	1	1	1	1	0	1	1	1	0	1	1	1	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	0	1	1	1	0	1	1	1	0	0	1
19	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
20	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1
21	1	1	0	0	0	1	1	1	0	1	1	1	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
23	0	0	0	0	1	0	0	0	1	0	0	0	1	1	0
24	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0
25	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 2 displays the test case coverage with respect to line of codes during block coverage. There are 15 test cases are applied on 26 code lines. For example column 1 represents that test case 1 covers 1to 15,17 ,18 ,20 and 21st line of code indicated by '1' and uncovered lines are 16, 19, 22 to 25 are uncovered and indicated by '0'.

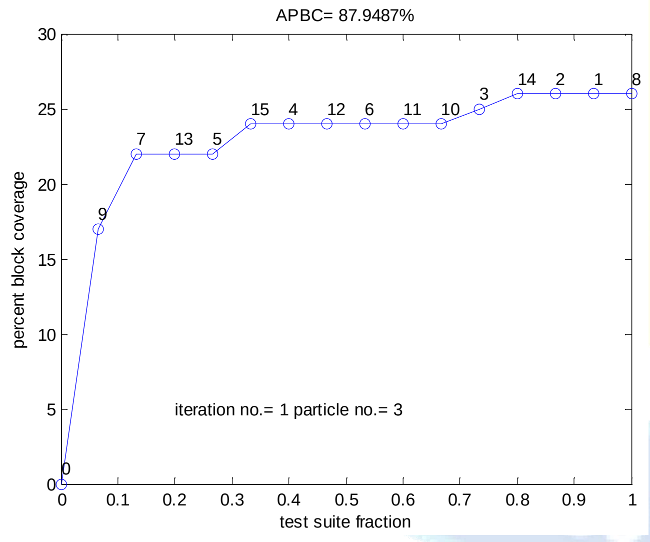


Fig. 2: Percentage block coverage and APBC value at 1st iteration for test suit represented by particle number 3 using PSO algorithm.

Test case selection:

It is the process of selecting subset of the test suite. The subset of test suite is generated depending on the programs that are affected by the software modification. Test case selection reduces the number of test cases to be executed and thus reduces the execution time of the test suite.

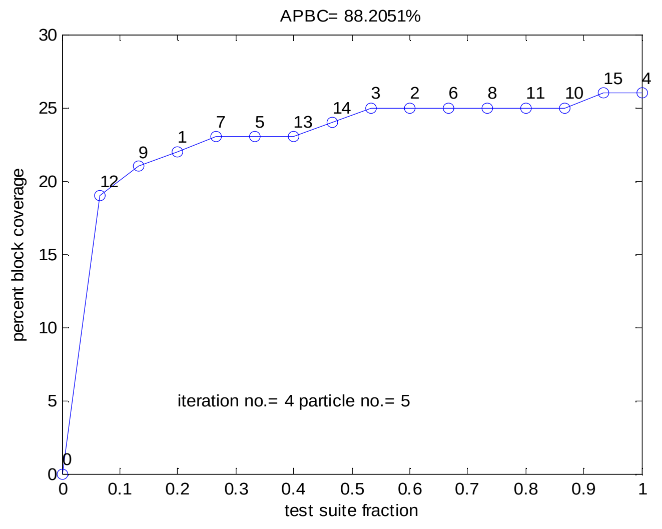


Fig. 3. Percentage block coverage and APBC value at 4th iteration for test suit represented by particle number 5 using PSO algorithm.

5. Conclusion:

This work described five algorithms for the sequencing problem in test case prioritisation for regression testing. It presented the results of an empirical study that investigated their relative effectiveness. The data and analysis indicate that the Greedy Algorithm performs much worse than Additional Greedy, 2–Optimal and Genetic Algorithms overall. Also, the 2–Optimal Algorithm overcomes the weakness of the Greedy

Algorithm and Additional Greedy Algorithm (see Table 1) referred to by previous authors. However, the experiments indicate that, in terms of effectiveness, there is no significant difference between the performance of the 2–Optimal and Additional Greedy Algorithms. This suggests that, where applicable the cheaper–to–implement–and–execute Additional Greedy Algorithm should be used. The choice of coverage criterion does not affect the efficiency of algorithms for the test case prioritization problem.

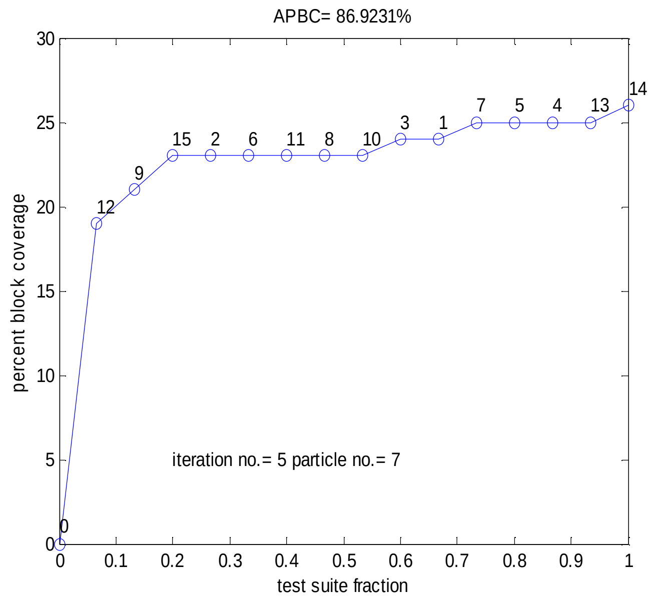


Fig. 4. Percentage block coverage and APBC value at 5th iteration for test suit represented by particle number 7 using PSO algorithm.

The size of the test suite determines the size of the search space, therefore affecting the 30 complexity of test case prioritisation problem. The size of the program does not have a direct effect, but increases the difficulty of computing fitness values. Studies regarding the performance of meta–heuristic algorithms led to several conclusions that have practical ramifications. The results produced by Hill Climbing show that the nature of the fitness landscape is multi–modal. The results produced by the Genetic Algorithm indicate that it is not the best of the five considered in all cases, but that in most cases the differences between the performance and that of the Greedy approach is not significant. However, an analysis of the fitness function shows that there are situations in which it is important to consider the entire ordering and, for such cases, Greedy Algorithms are unlikely to be appropriate. Given their generality, the fact that Genetic Algorithms perform so well is cause for encouragement.

References:

[1]. Zheng Li, Mark Harman and Robert M. Hierons, "Search Algorithms for Regression Test Case Prioritization", IEEE Transaction Paper on Software Engineering ,Vol. 33 Issue 4, pp.225-237, 2007

[2]. V. Basili, R. Selby, E. Heinz, and D. Hutchens. Experimentation in software engineering. *IEEE Trans. Softw. Eng.*, 12(7):733–743, July 1986.

[3]. Agrawal G., Guo L., "Evaluating explicitly context-sensitive program slicing", in: *Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Snowbird, Utah,* 2001, pp. 6-12.

[4]. T. Abdel-Hamid. The dynamics of software project sta_ng: a system dynamics based simulation approach. IEEE Transactions on Software Engineering, 15(2):109{119, 1989.

[5]. K. P. Chan, T. Y. Chen, and D. P. Towey. Restricted random testing. In *Proceedings of the 7th European Conference on Software Quality* (*ECSQ 2002*), volume 2349 of Lecture Notes in Computer Science, pages 321–330. Springer, Berlin, Germany, 2002.

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

[7] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of data flow and control-flow-based test adequacy criteria. In Proceedings of the 16th International Conference on Software Engineering, pages 191–200. IEEE Computer Society Press, May 1994.

[8] J.-M. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In ICSE '02: Proceedings of the 24th International Conference on Software Engineering, pages 119–129, New York, NY, USA, 2002. ACM Press.

[9] S. Lin. Computer Solutions of the Travelling Salesman Problem. Bell System Tech. Journal, 44:2245–2269, 1965.

[10] Microsoft Corporation. Regression testing. http://msdn.microsoft.com/library/default.asp?url=/library/enus/ vsent7/html/vxconregressiontesting.asp.

[11] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 27(10):929–948, Oct. 2001.

[12] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis, pages 97–106, New York, NY, USA, 2002. ACM Press.

[13] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In Proceedings of the Eighth International Symposium on Software Reliability Engineering, pages 230–238. IEEE Computer Society, Nov. 1997.