International Journal of Pure and Applied Mathematics
Volume 117 No. 20 2017, 809-828
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

Adaptive Swallow Swarm Optimization (ASSO) based Hybrid Criteria for Regression Testing

[1]V. Keerthika and [2]P.G. Sapna

[1]Department of R & D, Bharathiar University,

Coimbatore, Tamil Nadu.

Department of Computing,

Coimbatore Institute of Technology,

Coimbatore, Tamil Nadu.

[2]Department of Master of Computer Applications,

Coimbatore Institute of Technology,

Coimbatore, Tamil Nadu.

Abstract

Regression Testing (RT) is an essential and costly activity, which is carry out each time a program is changed in order to ensure that the changes do not bring in new bugs into the code that was validated earlier. RT tasks such as test suite reduction /minimization, test case prioritization and regression test selection that is concentrated around conditions that are based upon code coverage, test execution expenses, and modifications in code. Recently introduces new individual criteria, which combines the already existing criteria in various ways to create new function called as hybrid criteria. However, still in the present work, there are no standard representations are provided to researchers in order to create and formulate their novel multi-criteria techniques. With the purpose of solving this challenge, this research aim to creating a uniform representation of such unions such that they can be formulated without any ambiguity and could be shared among researchers. In the proposed work, a hybrid criteria function is optimized employing Adaptive Swallow Swarm Optimization (ASSO) Algorithm for RT along with test cases function values. ASSO algorithm begins with the number of swallows to be the test cases characteristics. Generally three hybrid combinations, namely Rank, Merge, and Choice are optimized, and their resourcefulness is demonstrated. It is also demonstrated that such kind of sharing will let the researchers to realize, examine, extend, and assess the hybrids exploiting a generic set of strategies and tools.

Index Terms: Regression Testing (RT), Adaptive Swallow Swarm Optimization (ASSO), hybrid combinations, software regression testing, test suite reduction /minimization, test case prioritization and regression test selection.

1. Introduction

Software regression testing is a crucial activity during the maintenance stage of developing software. But, it needs enormous number of test cases in order to test any fresh or changed functionality in the program [1].Re-run of all the existing test cases along with the new ones is frequently expensive and also impractical owing to time and resource limitations. In order to deal with this issue, the research community introduced methods for optimizing the regression testing [2]. Re-run of the test cases, which do not use any modified or impacted portions of the program leads to additional expense and there is no advantage in it. One efficient strategy is the permanent elimination of such repetitive or outdated test cases and hold on to the most efficient ones in order to minimize the additional expense of regression testing [3]. Such a technique tries to get a minimal subset of test cases that meets each one of the testing requirements just like the original set. This subset can be got while the test case is generated or once the test suite is created. Obviously, lesser the number of test cases, the lesser time would be taken for testing the program. Consequently, this enhances the efficiency of the test process. In the literature, this method is generally called as test suite reduction or test suite minimization and the resultant suite is referred to as representative set.

In [4] it is stated that Gregg Rothermel has shown that prioritization and scheduling test cases are one among the most crucial task in the process of software testing as it is seen in an example about industrial collaborators reports that depicts that approximately 20,000 lines of code exists, and running the whole suite of test cases needs seven weeks. In this scenario, prioritization on test cases has a vital role in saving the time. Test case prioritization methods carry out the prioritization and scheduling of test cases in an order to maximize some of the objective functions. For instance, software test engineers may want to schedule the test cases in a sequence, which accomplishes code coverage at the quickest possible, utilizing the feature in the order of expected frequency of usage, or exercising the subsystems in a sequence which is a reflection of their historical tendency of failure. If the time necessary for executing each of the test cases in a test suite is not long, then test case prioritization might not be cost efficient - it might be almost the measure to just schedule the test cases in any sequence [5]. If the time necessary for running all the test cases in the test suite is adequately longer, then the advantages provided by the test case prioritization techniques tend to become more important.

Software testing has been extensively employed as a means to assist the engineers in developing high-quality systems. It is a distinguished process, which is carried out to aid in quality assurance by collecting information regarding the nature of the software that is being examined [6]. These activities comprises of developing the test cases, running the software with those test cases, and then analysing the results generated by those executions. It shows that greater than fifty percent of the expense involved in software development

is performed to testing and the percentage for testing important software is even greater. Since software tends become more extensively used and is employed more frequently to carry out crucial tasks, the significance of its quality will stay high. If the engineers are not quite successful in finding effective means of performing efficient testing, then the percentage of development expenses dedicated to testing may see a significant increase.

Kadry et al [7] explained about software regression testing employing an efficient method. New automation test depending on decision tree and test selection process is introduced in order to decreasing the testing cost. The decision tree is applied to a practical case and the result show its improvement automation test based on decision tree. The test engineers prioritize the regression test cases based on different vital factors like aggregate efforts, cost-benefit compromises, execution time, deployment time etc., so that the rate of fault detection of test suites is increased. The method does the reordering of the test cases, scheduling of test cases with the greatest priority based on some criterion before in the testing process. This method can be divided into:

- General Prioritization: Choose an efficient sequence of test case for the next subsequent versions of software.
- And version specific prioritization: Related with specific version of the software.

Sebastian Elbaum et al. [8] introduced the usage of test case prioritization methods in regression testing and highlights on the goals of increasing the fault detection rate, and answered various extra questions that were raised by that research: 1) The prioritization methodologies be efficient if designed at particular modified versions, 2) What trade-offs present among fine granularity and coarse granularity prioritization methods; 3) The inclusion of the measures of fault susceptibility into test case prioritization methods enhances their test efficiency. In order to answer these questions, a variety of novel controlled experiments and case studies were performed. The data and analysis show that version-specific test case prioritization can generate statistically important enhancements in the test suites' rate of fault detection. With adding further complication to matters, both their controlled studies and case studies show that the comparative effectiveness of prioritization methods could differ across programs.

Li et al. [9], provided the focus on test case prioritization methods for code coverage, inclusive of block coverage, decision coverage, and statement coverage. Five search methods are analysed: two meta-heuristic search methods like Hill Climbing and Genetic Algorithms, along with three greedy algorithms like Greedy, Additional Greedy, and two-optimal Greedy. An empirical study has been carried out and the result obtained of the study provided the performance comparison of the five search algorithms used in six programs, ranging between 374 and 11,148 lines of code. For deciding the efficiency of the various methodologies, numerous metrics are brought into use. The metrics

offers greater performance in terms of average percentage block coverage, average percentage decision coverage and average percentage statement coverage.

Luciano et al [10] introduced Particle Swarm Optimization (PSO), a simple and effective algorithm in comparison with other popular search mechanisms. A Binary Constrained PSO (BCPSO) for functional TC selection, along with two hybrid algorithms merging BCPSO with local search techniques are introduced, with the purpose of refining the solutions given by BCPSO. These algorithms were assessed employing two diverse practical-world test suites of functional TCs corresponding to the mobile devices domain. The BCPSO received interesting results in the experiments for the optimization tasks taken into consideration. Moreover, the hybrid algorithms achieved better results statistically compared to the individual search mechanisms.

Sabharwal et al [11] presented a method for prioritization test case scenarios obtained from the activity diagram employing the concept of fundamental information flow metric and Genetic Algorithm (GA). It generates prioritized test cases in static testing utilizing GA. An identical approach is applied so as to prioritize the test case scenarios obtained from source code in static testing. Andrews [12] used GA for the purpose of randomized unit testing in order to find the test cases with the best suitability. Recently novel criteria is introduced which integrates the available criteria in various means of creating a new function named to as hybrid criteria. The proposed work is motivated from this work by optimizing hybrid criteria function. Three means of integrating criteria are described: Rank, Merge, and Choice. In case of Rank, the condition, which are united get ranked in the direction of importance; at first, the primary condition is employed; a user-supplied function chooses when the second and all the successive condition, in an order, have to be applied. Based on the scenario, Rank may utilize different criteria but they have to be used one after another, in the order of significance. A more essential criterion has to be used before the application of one with less importance. On the other hand, merge permits multiple criteria to be taken into consideration at the same time; all the criteria are integrated first and then the combination is brought into use. At last, choice chooses merely one from a group of similarly vital condition applying a user-supplied selection function. Moreover, in the newly introduced work, hybrid criteria function gets optimized employing Adaptive Swallow Swarm Optimization (ASSO) Algorithm for resolving Regression Testing (RT) with test cases function values.

2. Proposed Methodology

In the work carried out recently, various techniques have been investigated in detail and the advantages of employing multiple criteria in comparison with a single criterion for use in regression testing has been also mentioned. Nonetheless, every work makes use of its individual notation, if exists, during the formulation of the problem of multiple criteria regression testing. Moreover,

very less have performed the empirical comparison of their innovative approach with any earlier approaches. Even though there are several causes as to why such comparisons are not performed by researchers, it is felt that one huge reason for that is the absence of standard representations that can be made available to researchers to design and formulate their innovative multi-criteria techniques.

This work introduces a new way to representing multi-criteria approaches. This work is motivated from earlier work [13], a hybrid approach is used for test case prioritization employing two criteria: 1) frequency based criteria and 2) a combinatorial criterion, two means. The test cases are ordered by one condition and after that modified to a second condition when the Average Percent of Faults Detected (APFD) doesn't enhance one time a specific amount of test cases are attained. This valuation utilizes call center web application in the company of seeded faults. In the proposed hybrid criteria function is optimized by using Adaptive Swallow Swarm Optimization (ASSO) Algorithm for RT along with test cases function values.

This section formally defines the three hybrid criteria for carrying out RT tasks. Since hybrid criteria are got by integrating individual criteria in different means, the hybrid criteria are formulated exploiting the functional paradigm. This lets us to make use of the function composition, and its popular semantics, in order to combine various criteria together, providing a great flexibility on how the criteria can be combined and used. Let us consider that the test suite comprises of five test cases examples (See Table 1), each one of which is actually an ordered sequence of events; different measured features of these test cases, inclusive of the events(ε), statements(S), and branches(B) covered by them, their execution time, length(L), and faults (F) detected by them. With no loss of generality, it is assumed that these are represented as matrices (for instance, coverage matrix S for the statements covered) and in the vectors of values (that is to say, L is denoted as the test caseslength).

Table 1. Five Example Test Cases, Their Coverage and Other Metrics, and Faults Detected

Test cases	Events covered			Statements covered					Branches		Exec	Length	Faults detected				
ID	e_1	e_2	e_3	s_1	s_2	s_3	s_4	s_5	b_1	b_2	(sec)	(#events)	f_1	f_2	f_3	f_4	f_5
T_1	0	1	1	1	0	1	1	1	0	0	0.5	5	0	0	1	0	1
T_2	1	0	1	0	1	0	1	0	0	0	0.1	3	0	1	0	1	0
T_3	1	1	0	1	0	1	0	1	1	1	0.8	2	1	0	1	0	0
T_4	0	0	1	0	1	0	1	0	1	0	0.1	5	0	1	0	0	0
T_5	1	1	0	1	0	0	0	0	0	1	0.9	6	0	0	0	0	0
Definition	ε			S					B		χ	$\mathcal{L}$	$\mathcal{F}$				

Objective Function

1. Representing Stand-Alone Criteria

It is started by specifying the available classical, stand-alone (non-hybrid) criteria in the form of functions. Since the attention is on test case prioritization, the essential components of formulation is principally a function next () that takes three parameters: 1) the test case order selected till now, 2) the complete test suite, 3) a matrix, which encrypts the relationship amid the entire test cases as well as the metric, which is being applied to compute the ranked order for instance, statement coverage united with a function g() for additional coverage prioritization in order to be used for the computation. The outcome of next() is a group of test case that come subsequent in the priority order. A general usage of next() will start with a blank series in the form of the first parameter; next() would retrieve the test cases with the greatest priority; these would be utilized in the second invocation to act as the first parameter to get the next most significant test cases; and the next subsequent iterative invocations would be passed as the first parameter to all the ordered test cases gotso far in order to get to the next significant test cases; the iterations will proceed till an order is reached with the tests. If several test cases provide the coverage of the same number of statements that are not covered yet, a randomized selection of one test case is done. A function g() is defined for extra coverage prioritization.

Functional formulation permits the quick realization of alternate test case prioritization methods by developing new invocations and g() function that could “plug into” the complete prioritization structure. The g()

function is restricted in order to function on an individual coverage matrix/vector. This lets to have the same g() function reused in several places. Also, the invocation function yields a high-level control over the usage of the next () function [14]. Here, all of the coverage elements get covered by the test cases, which are chosen previously. So, the last suite, stored in y, turns out to be the reduced form of Suite since it precisely covers the same coverage elements. Instead, Test suite reduction, needs a somewhat more involved modification. It is necessary that a novel, non-greedy g() function is defined such that a reduced suite is got.

2. Representing Second-Order Criteria

The formulation is extended to take multiple criteria into consideration, and the developing term in the form of second-order criteria. As it can be deduced, the criteria may be merged in numerous means to create a hybrid. At this point, stooping back is necessary and the philosophy behind the formation of hybrids is to be considered for the case of test case prioritization. The fundamental notion behind and the hybrid is that it incorporates diverse condition, giving, at every choice that test case should be chosen subsequent to the prioritization technique, the powers of every basic individual condition [14]. A possible hybrid is the ordering of each of the individual condition (for instance primary, secondary, tertiary, and etcetera), start with the primary, and utilize the

secondary simply while the primary produces ties in test case selection, that is to say multitude of test cases meet the primary criterion. This leads to a ranking kind of hybrid. One more thing, which is possible, is with regard to mathematical combining the individual criteria into one single criterion. For example, consider that it is wished to order the tests by their capability in covering the summation of statement and branch coverage, that is to say that the most significant test case covers the most number of statements and branches unified. This devises hybrid criteria, which combines the criteria together and the combination is applied at the same time. At last, a third kind of hybrid must permit a clear choice between multiple criteria on the basis of a selection function.

An application was presented by Sampath et al [15] containing Merge as well as Rank formalizations. Rank: By intuition, the criteria that are being integrated are ranked by the order of importance and then applied serially. Two sets of criteria are applied in series. Rank based criteria that are obtained for web applications, such as name-value pairs, base requests, and sequences comprising base requests with size 2, and sequences comprising base requests and name of size 2 are utilized. The primary invocation of the next () function is stated here

$$next([], T_1 \dots T_i, \dots T_n\},$$
$$Rank\left(\left(M, g_{mod-hgs}()\right), \left(base, g_{mod-hgs}()\right)\right)$$

here the literals M and base specify the mapping amid the test cases and methods, and base requests, respectively. The function $g_{mod-hgs}()$stands for their realization of the modified Harrold, Gupta, Soffa (HGS) algorithm [15]. Even though the "HGS algorithm" has generally been utilized with a single criterion, their actual work explicitly defines the means in which it might be utilized with multiple numbers of criteria.

Same kind of invocations can be formulated when the method coverage is incorporated with the remaining usage-based condition. After that a consequent invocation of next () will utilize the test cases that are noted in connection with method coverage and assess them to break the ties corresponding to base requests coverage. Lin et al [16]'s technique is generally an application of Rank hybrid formalization. In those experiments, branch coverage was used as the primary criterion and resolve pair coverage is used as the secondary condition. The primary invocation of the next () function is stated here

$$next([], T_1 \dots T_i, \dots T_n\},$$
$$Rank\left(\left(B, g_{m-hgs}()\right), \left(DU, g_{m-hgs}()\right)\right)$$

here the B and DU specify the mappings amid the test cases and the branches, defuse pairs that are covered, respectively. For instance, one realization of Rank could make use of the first criterion for prioritizing the test cases; on condition that that criterion is unsuccessful as agreed by function g(), after that the second criterion is used; when that too fails, the third, and each one of the subsequent

criteria are applied in order.

Merge: By intuition, this is used to integrate a huge number of criteria considering all the criteria at the same time in order to have the tests ordered. For instance, one realization of Merge could make use of an operator to integrate all of the matrices into a single compound matrix and then that single compound matrix is used for ordering the test cases. This can be accomplished, for instance, by the operator Array Flatten in mathematical which, among the rest of the operations, horizontally combines two or additional matrices to make a novel matrix. Actually, the number of rows existent in all matrices, which are being combined must be similar for a resultant matrix.

Choice: By intuition, the third and last means of combining the criteria chooses one among a set of criteria employing a selection function. It is to be noted that dissimilar to Rank, every criterion has the chance for selection on the basis of coverage criteria; also, an earlier chosen criterion may be selected again by the selection function. At that point, the invocation function that understands the selection uses each matrix (vector) that are specified in selection concurrently in diverse next() invocations; the distinct outcomes of all of these next() get logged. An additional step computes the "goodness" of each outcome. The selection containing the greatest goodness is chosen to be the output. The process gets repeated for the rest of the unordered tests till all the tests become ordered.

3. Representing Higher Order Criteria

Rank, Choice, and Merge are united to make higher order condition. Primarily, the need for higher order criteria is an enthused one. Taking the example of Merge with S and B; it is to be recalled that T_3 is selected first, then a tie existed between T_2 and T_4; at last, the rest of the tests are added in any sequence. And then remained a solution which is less than satisfactory, in which greater than half of the test cases were ordered randomly. In order to assist in solve this situation, suppose that a secondary criterion has to be added, E, which would be employed as a tie breaker and also for ordering the rest of the test cases. In order to execute the higher order criterion the $Rank_{H(C_x,C_y,..C_Z)}$ invocation function is to be used. In this research, hybrid criteria function is employed for testcase prioritization along with test cases function values, which are optimized employing the Adaptive Swallow Swarm Optimization (ASSO) Algorithm.

Adaptive Swallow Swarm Optimization (ASSO)

The Swallow Swarm Optimization (SSO) is a novel swarm optimization technique [17]. One more research that is indicative of the effectiveness of SSO is actually a combination of Particle Swarm Optimization (PSO) [18]. The SSO algorithm, influenced by the unified movement of swallows and the interaction seen between flock members has produced better results. This algorithm has introduced a meta-heuristic technique depending on the special characteristics of swallows, inclusive of quick flight, hunting skills, and smart social

relationships. In this technical work, the hybrid criteria function is optimized employing ASSO algorithm. In ASSO algorithm, the number of swallows is regarded to be the test cases characteristic. Basically, with a test suite $SW = (T_1, \ldots . T_n) and C = (c_1, \ldots c_m)$. The operating procedure of SSO algorithm is just like PSO though it has few more characteristics that include the usage of three kinds of particles:

- Explorer Particles (e_i)
- Aimless Particles (o_i)
- Leader Particles (l_i)

Each one of them has specific duties in the group. The e_i particles are accountable for finding the best optimal hybrid criteria function space. They carry out this searching for an optimal hybrid criterion based on the impact of different parameters [17]:

1. Position of the local leader.
2. Position of the global leader.
3. The best individual experience along the path.
4. The earlier path.

The particles utilize the equations below for carrying out a search for an optimal hybrid criterion and proceeding with the test suite process:

$$V_{HL_{i+1}} = V_{HL_i} + \alpha_{HL} rand(\quad)(e_{best} - e_i) + \beta_{HL} rand()(HL_i - e_i) \quad (1)$$

Eq. (1) indicates the velocity vector variable in the test case prioritization of the global leader

$$\alpha_{HL} = \{ if(e_i = 0 || e_{best} = 0) \longrightarrow 1.5\} \quad (2)$$

Eqs. (2) and (3) computes the acceleration coefficient variable (α_{HL}) that directly influences the individual experiences of every particle

$$\alpha_{HL} = \begin{cases} if(e_i < e_{best})(e_i < HL_i) \rightarrow \dfrac{rand(\quad).e_i}{e_i.e_{best}} e_i.e_{best} \neq 0 \\ if(e_i < e_{best})(e > HL_i) \rightarrow \dfrac{2rand(\quad).e_i}{1/2(e_i)} e_i \neq 0 \\ if(e_i > e_{best}) \rightarrow \dfrac{e_{best}}{\frac{1}{2.rand(\quad)}} otherwise \end{cases} \quad (3)$$

$$\beta_{HL} = \{if(e_i = 0 || e_{best} = 0) \longrightarrow 1.5\} \quad (4)$$

$$\beta_{HL} = \begin{cases} if(e_i < e_{best})(e_i < HL_i) \rightarrow \dfrac{rand(\ \ ).e_i}{e_i.e_{best}} e_i.HL_i \neq 0 \\ if(e_i < e_{best})(e > HL_i) \rightarrow \dfrac{2rand(\ \ ).e_i}{1/2(e_i)} e_i \neq 0 \\ if(e_i > e_{best}) \rightarrow \dfrac{HL_i}{\frac{1}{(2.rand(\ \ ))}} otherwise \end{cases} \quad (5)$$

Eqs. (4) and (5) compute the acceleration coefficient variable (β_{HL}) that directly influences the value of test case prioritization. Actually, these two acceleration coefficients are quantified taking the position of every test-case present in the test suit T (particle) relative to the best individual experience and the global leader. The $o_i$particles is carried out on the basis of the arbitrary selection of the test cases in the test case prioritization scenario and traverses through the test case prioritization without achieving a particular goal and shares the results to the rest of the flock members in the technique. Factually, depending on the arbitrarily chosen test cases, the particles improve the possibility of finding the best test cases that belongs to the particles, not highlighted by the e_i particles. These particles employ the following Eq. (6) for arbitrary movements.

$$o_{i+1} = o_i + \left[rand(\{-1,1\}) * \frac{rand\left(\min_s \max_s\right)}{1 + rand(\ \ )} \right] \quad (6)$$

Adaptive Inertia Weight

In the case of SSO, the particles' movement is essentially dependent on a variety of parameters: the best fitness value, the position of particle present in local groups, and the evaluation of the best position of every particle among all the groups. Once a suitable position is found, then particles try to converge around it. One of the most essential parameters here is inertia weight, which determines the increase convergence speed of the algorithm. In the newly introduced SSO algorithm, an adaptive inertia weight is employed for increasing the speed of test case prioritization process and thus produces the best optimal fault detection results. Eq. (7) expresses this inertia coefficient.

$$W_{i+1} = \left\{ (W_t - W_{min}).\left[\frac{t_{max} - t}{t_{max}} \right] + W_{min} \right\}.e^{-\left[\frac{t}{\left(\frac{t_{max}}{4}\right)} \right]^2} \quad (7)$$

The findings from the results show that the earlier reported work of others are explained in regard to two of the novel operators: Rank and Merge. This offers a decent amount of confidence, which these create the dynamic operators for hybrid developments. It is also observed that the performance of the formulated hybrid criteria was better than their constituent individual criteria most of the times.

3. Results and Discussion

This section does the empirical evaluation of the efficiency of hybrid-criteria prioritization by doing the same on the new criteria developed for test case prioritization. These hybrid criteria functions over test suites are tested employing http test protocol, Call Centre GUI application. In the call centre GUI application form, the testing of the valid and non valid functions are done employing http test protocol and written with Java. For example, Figure 1 depicts the results as below. The valid mark represents that the particular function operate correctly. The invalid mark indicates that the specific function will not operate correctly. Those valid and non-valid comments are employed for usage in test case prioritization. The size of the applications range between 1,000 and 13,000 lines of code. They contain a large amount of windows as well as parameter values. The test cases for the GUI applications (amid 10 and 50 tests) are generated from a GUI model.

protocol	/CallCentre/addcust1.jsp	valid
protocol	/CallCentre/adddept.jsp	valid
protocol	/CallCentre/adddept1.jsp	valid
protocol	/CallCentre/adddesign.jsp	valid
protocol	/CallCentre/adddesign1.jsp	valid
protocol	/CallCentre/addemb.jsp	valid

Figure 1: Test Case Results for Call Centre Application

Web application usage logs are modified into test cases for use in the web applications [19]. In order to have the effectiveness of the measure used for a test order, the average percent of faults detected metric is used [20]. Though various metrics are available for evaluating the prioritized test orders, Average Percent of Faults Detected (APFD) is the most commonly applied metric. According to Sampath et al [20], specified a testsuite T encompassing n test cases, while F signifies a group of m faults identified by T, at that point, consider TF_i denote the place of the initial test case t in T', here T' represents an ordering of T that identifies fault i. Consequently, the APFD metric for T' is stated as

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{mn} + \frac{1}{2n} \quad (8)$$

Usually, APFD performs the measurement of the area underneath the curve that designs the test suite fraction and the amount of faults, which are identified by means of the prioritized test case order.

The results are reported employing bar chart depicted in Figure 2,3, 4, 5, 6 and 7. The bar plots are shown with the prioritization criterion plotted along on the x-axis, and their APFD values along the y-axis.

Sensitivity

It is also referred to as the True Positive Rate (TPR), the recall, or probability of detection. It provides the measure of the ratio of positives or the percentage of test errors, which are rightly detected to have the error condition.

$$\text{Sensitivity}=TP/(TP+FN) \quad (9)$$

Where TP=True Positive, FN=False Negative, TN=True Negative and FP=False Positive.

Table 2: Performance Analysis Results vs. Methods

Methods	**Results(%)**					
	Recall or Sensitivity	**Specificity**	**Precision**	**F-Measure**	**Accuracy**	**Error rate**
Merge-one-way-rank	77.50	48.00	82.67	80.00	70.48	29.52
Merge-two-way-rank	84.21	41.67	82.05	83.12	74.00	26.00
1-way	86.84	41.67	82.50	84.62	76.00	24.00
2-way	85.37	44.44	87.50	86.42	86.42	22.00
Rank	54.76	64.10	67.11	60.935	69.72	30.28
Merge	28.47	24.00	33.73	31.1	30.35	69.65
Choice	66.421	71.06	78.51	72.4655	77.501	22.499
ASSO	86.10	80.12	88.59	87.345	88.50	11.5

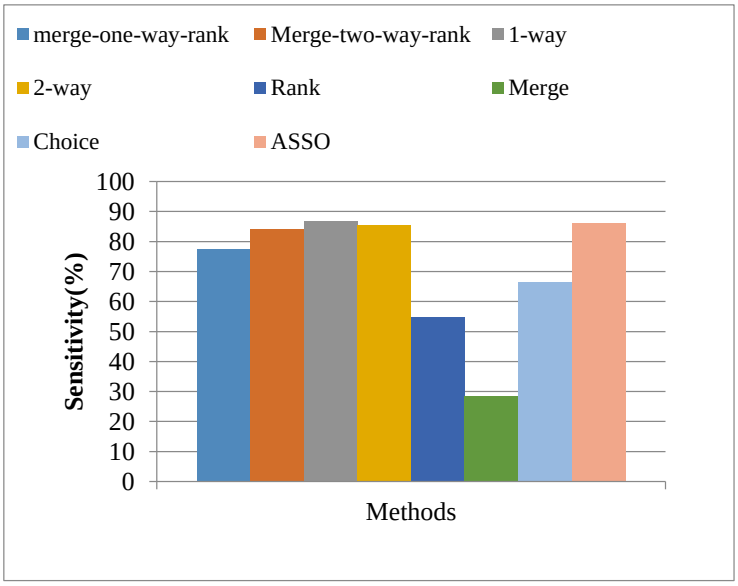

Figure 2: Sensitivity Comparison vs. Methods

Figure 2 illustrates the results of the sensitivity comparison of the proposed

ASSO algorithm and the other available criteria techniques. Therefore it is proved that the proposed ASSO algorithm yields more accurate fault detection ratio compared to the existing techniques. The proposed ASSO algorithm generates 86.1% of sensitivity results, while other criteria function like Merge-one-way-rank , Merge-two-way-rank, 1-way, 2-way, Rank, Merge and Choice generate sensitivity results of 77.50%, 84.21%, 86.84%, 85.37%, 54.76%, 28.47% and 66.421% correspondingly.

Specificity

Specificity also referred to as the True Negative Rate (TNR) provides the measure of the ratio of negatives onthe percentage of error which is not rightly detected as error.

$$\text{Specificity}=TN/(TN+FP) \quad (10)$$

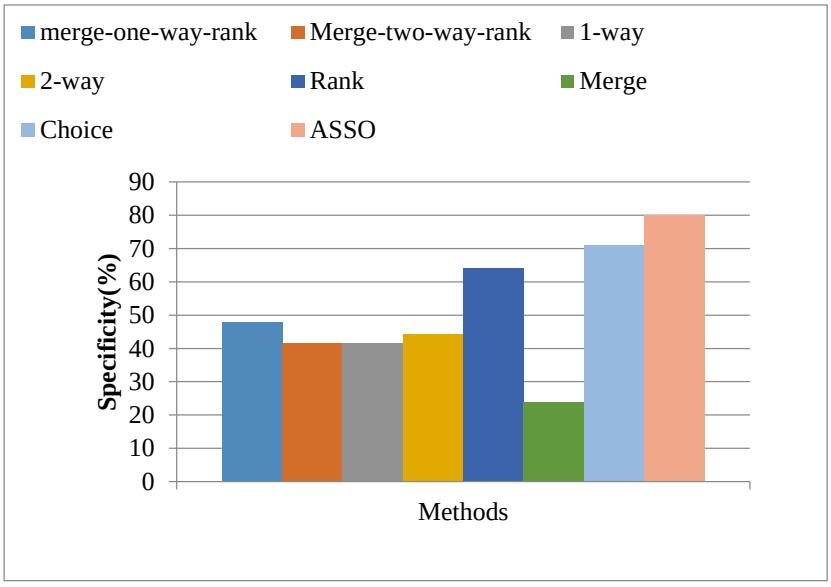

Figure 3: Specificity Comparison vs. Methods

Figure 3 illustrates the comparison made of specificity between the available and proposed techniques. The proposed ASSO approach yields greater specificity values in comparison with the other existing techniques. The proposed ASSO algorithm generates about 80.12% of specificity results, while the other criteria function like rank, merge, choice, two way, one way, merge one way rank and merge two way rank generates specificity results of 64.1%, 24%, 71.06%, 44.44%, 41.67%, 41.67%, and 48.00% correspondingly.

Precision

Precision is also called as Positive Predictive Value (PPV) and is formulated as below:

$$\text{Precision}=TP/(TP+FP) \quad (11)$$

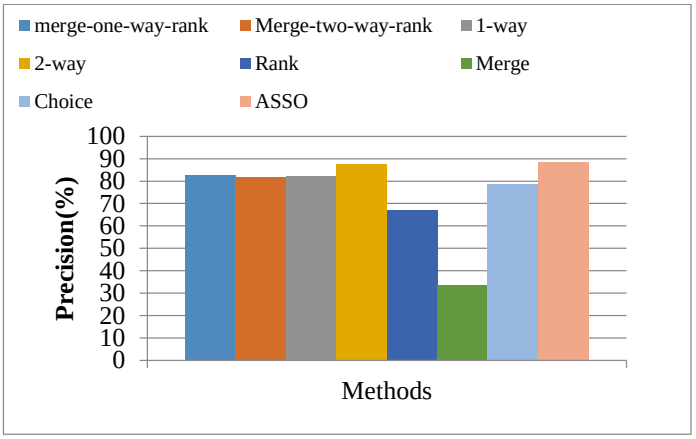

Figure 4: Precision Comparison vs. Methods

Figure 4 shows the performance comparison result of precision metric for the test cases between proposed and existing methods. The methods are plotted along the x-axis and the precision value is plotted along the y-axis. This way, the result shows that the newly introduced ASSO techniques have exhibited significant improvement compared to the other existing algorithms. The proposed ASSO algorithm generates 88.59% of precision results, while the rest of the criteria function like choice, merge ,rank, Two Way, one way, merge one way rank and merge two way rank generates precision results of 78.51%, 33.73%, 67.11%, 87.50%, 82.50%, 82.05%, and 82.67% correspondingly.

F-measure

The conventional F-measure or balanced F-score is defined to be the harmonic mean of precision and recall

$$F\text{-measure}=2.\ P.R/\ (P+R) \quad (12)$$

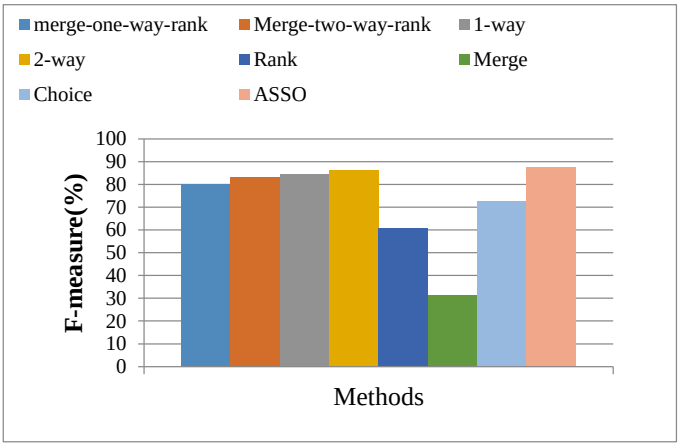

Figure 5: F-Measure Comparison vs. Methods

Figure 5 shows the performance comparison result of F-measure metric for the test cases between proposed and existing methods. The methods are plotted

along the x-axis and the precision value is plotted along the y-axis. So the result suggests that the proposed ASSO technique provides significant improvement when compared to the other existing algorithms. The proposed ASSO algorithm generates 87.345% of F-measure results, while the other criteria function like choice, merge, Rank, Two Way, one way, merge one way rank and merge two way rank generates f-measure results of 72.4655%, 31.1%, 60.935%, 86.42%, 84.62%, 83.12%, and 80.00% correspondingly.

Accuracy

Accuracy is defined to be the total accuracy of the model and is assessed as the sum of definite classification factors (TP+TN) divided by the total number of classification factors

$$Accuray = \frac{TP + TN}{TP + TN + FP + FP} \quad (13)$$

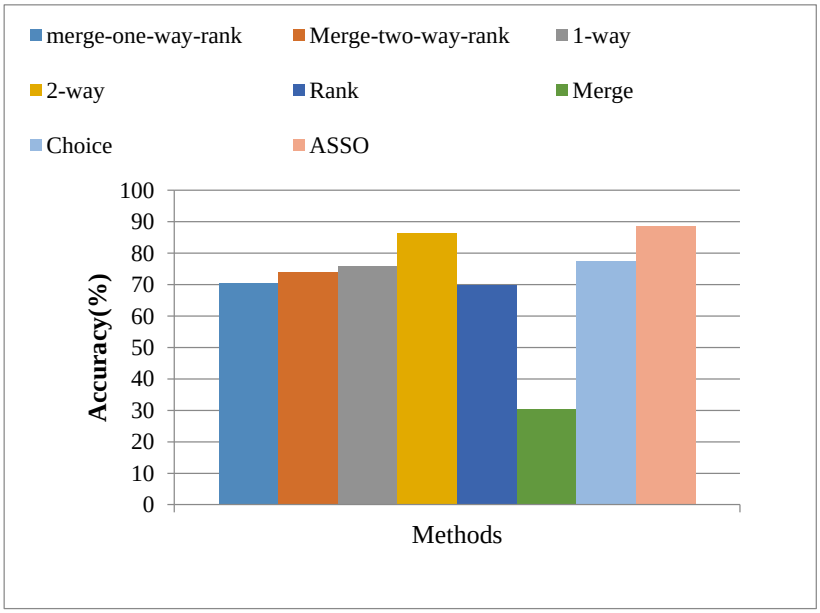

Figure 6: Accuracy comparison vs. methods

Figure 6 shows the performance comparison result of accuracy metric for the test cases between proposed and existing methods. The proposed ASSO algorithm generates 88.5% of accuracy results, while other criteria function like choice, Merge, Rank, Two Way, one way, merge one way rank and merge two way rank generates accuracy results of 77.501%, 30.35%, 69.72%, 86.42%, 76.00%, 74.00%, and 70.48% correspondingly. The result indicates that the proposed ASSO technique yields greater accuracy metric in comparison with the other existing techniques.

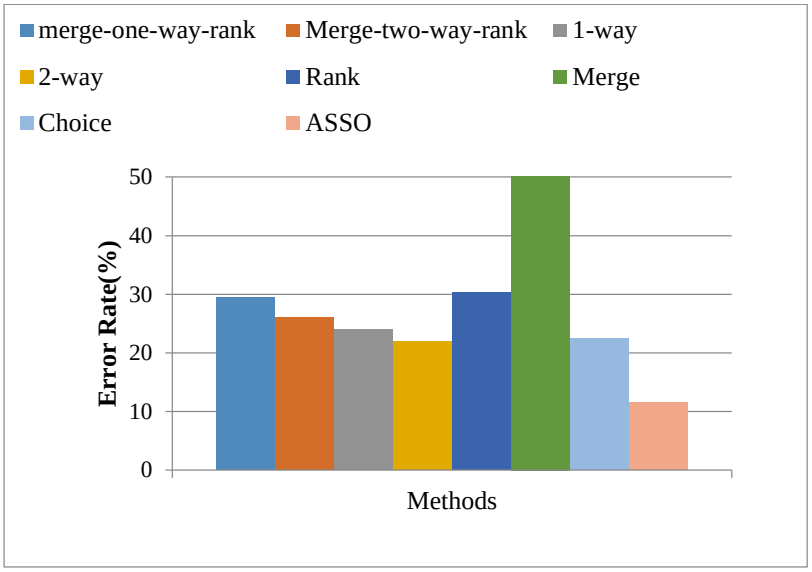

Figure 7: Error Rate Comparison vs. Methods

Figure 7 shows the performance comparison result of error rate for the test cases between proposed and existing methods. The proposed ASSO technique generates 11.5% of error rate results, while the other criteria function like choice, merge, Rank, Two Way, one way, merge one way rank and merge two way rank generates error results of 22.499%, 69.65%, 30.28%, 22.00%, 24.00%, 26.00%, and 29.52% correspondingly. It concludes that the proposed ASSO technique yields greater accuracy metric and minimal error rate in comparison with the other existing techniques.

4. Conclusion and Future Work

This paper formalized the idea of "hybrid criteria" by generating a standard representation in order to describe them precisely. Three hybrids are defined: 1) Rank, which provides primary, secondary, and n-order precedence to various criteria, 2) Merge, which uses multiple criteria at the same time, and 3) Choice, which chooses between several numbers of criteria. In the proposed work, hybrid criteria function is optimized by using Adaptive Swallow Swarm Optimization (ASSO) Algorithm for resolving RT along with test cases function values. Generally, it has been found that the performance of no specific hybrid criterion can be deemed to be the best across all the subject applications. It is found from the experiments, an ASSO with hybrid criterion, which integrates the various individual criteria functions better compared to one single criterion. In future work, it is expected that several approaches, which make use of a sole condition would be reconsidered with the purpose of evaluating the application relating to numerous conditions. Likewise, upcoming research work might as well examine the relationship of numerous condition with several techniques (for instance test case generation, test suite reduction, regression test selection and test case prioritization) over applications containing various features with

the intension that a good understanding regarding the relationship of numerous criteria in numerous situations is possible.

References

[1] Rothermel G., Harrold M.J., Von Ronne J., Hong C., Empirical studies of test-suite reduction, Software Testing, Verification and Reliability 12 (4) (2002), 219-249.

[2] Jones J.A., Harrold M.J., Test-suite reduction and prioritization for modified condition/decision coverage, IEEE Transactions on software Engineering 29(3) (2003), 195-209.

[3] McMaster S., Memon A., Call-stack coverage for GUI test suite reduction, IEEE Transactions on Software Engineering 34 (1) (2008), 99-115.

[4] Sripong R., Jirapun D., Test Case Prioritization Techniques, At JATIT & LLS (2010).

[5] Jeffrey D., Gupta N., Test case prioritization using relevant slices, 30th Annual International Computer Software and Applications Conference, 2006(COMPSAC'06) (2006), 411-420.

[6] Rothermel G., Harrold M.J., Von Ronne J., Hong C., Empirical studies of test-suite reduction, Software Testing, Verification and Reliability 12 (4) (2002), 219-249.

[7] Kadry S., A new proposed technique to improve software regression testing cost (2011).

[8] Elbaum S., Malishevsky A.G., Rothermel G., Test case prioritization A family of empirical studies. IEEE transactions on software engineering 28 (2) (2002), 159-182.

[9] Li Z., Harman M., Hierons R.M., Search algorithms for regression test case prioritization, IEEE Transactions on software engineering 33 (4) (2007).

[10] De Souza L.S., Prudêncio R.B., Barros F.D.A., Aranha E.H.D.S., Search based constrained test case selection using execution effort, Expert systems with applications 40 (12) (2013), 4887-4896

[11] Sabharwal S., Sibal R., Sharma C., A genetic algorithm based approach for prioritization of test case scenarios in static testing, 2nd International Conference on Computer and Communication Technology (ICCCT) (2011), 304-309.

[12] Andrews J.H., Li F.C., Menzies T., Nighthawk: A two-level genetic-random unit test data generator, In Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering (2007), 144-153.

[13] Bryce R.C., Sampath S., Memon A.M., Developing a single model and test prioritization strategies for event-driven software, IEEE Transactions on Software Engineering 37(1) (2011), 48-64.

[14] Sampath S., Bryce R., Memon A.M., A uniform representation of hybrid criteria for regression testing, IEEE transactions on software engineering 39 (10) (2013), 1326-1344.

[15] Harrold M.J., Gupta R., Soffa M.L., A methodology for controlling the size of a test suite, ACM Transactions on Software Engineering and Methodology (TOSEM) 2(3) (1993), 270-285.

[16] Lin J.W., Huang C.Y., Analysis of test suite reduction with enhanced tie-breaking techniques, Information and Software Technology 51 (4) (2009), 679-690.

[17] Neshat M., Sepidnam G., Sargolzaei M., Swallow swarm optimization algorithm: a new method to optimization, Neural Computing and Applications 23 (2) (2013), 429-454.

[18] Kaveh A., Bakhshpoori T., Afshari E., An efficient hybrid particle swarm and swallow swarm optimization algorithm, Computers & Structures (2014), 40-59.

[19] Sampath S., Sprenkle S., Gibson E., Pollock L., Greenwald A.S., Applying concept analysis to user-session-based testing of web applications, IEEE Transactions on Software Engineering 33 (10) (2007), 643-658.

[20] Sprenkle S., Gibson E., Sampath S., Pollock L., Automated replay and failure detection for web applications, In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (2005), 253-262.