

A Bee Colony Optimization Algorithm for Fault Coverage Based Regression Test Suite Prioritization

Dr. Arvinder Kaur

Associate Professor

University School of Information
Technology
Guru Gobind Singh Indraprastha
University, Delhi

arvinderkaurtakkar@yahoo.com

Shivangi Goyal

Research Student

University School of Information
Technology
Guru Gobind Singh Indraprastha
University, Delhi

goyal_shivangi@yahoo.co.in

Abstract

The process of verifying the modified software in the maintenance phase is called Regression Testing. The size of the regression test suite and its selection process is a complex task for regression testers because of time and budget constraints. In this research paper, the Bee Colony Optimization (BCO) algorithm for the fault coverage regression test suite prioritization has been presented. In the natural bee colony, there are of two types of worker bees; Scout bees and forager bee, who are responsible for the development and maintenance of the colony. The BCO algorithm developed for the fault coverage regression test suite is based on the behavior of these two bees. The BCO algorithm has been formulated for fault coverage to attain maximum fault coverage in minimal units of execution time of each test case, using two examples whose results are comparable to optimal solution. Average Percentage of Fault Detection (APFD) metrics and charts has been used to show the effectiveness of proposed algorithm.

Keywords: Bee Colony Optimization, Regression Testing.

1. Introduction

Due to the time and cost constraints, software maintenance is one of the most time consuming and crucial activity for the software development life cycle. Software maintenance phase can be extended for ten to fifteen years of duration in which regression testing is performed to check and confirm the correctness and accuracy of the modified software. Exhaustive regression testing is not possible due to the time and cost constraints associated with it. Thus effective selection and prioritization methods are required to make regression testing handy and worthy. Various techniques have been proposed for the same which have been mentioned in next section of related work.

In this paper the optimization phenomenon used is the Bee Colony Optimization phenomenon where bee's behavior has been observed, studied and mapped to find prioritization of the modified software's test suite.

Automated software testing has been an area of concern for big software development companies. Automated testing allows development and execution of test cases unattended. Also, comparison of actual test results to expected test results can be determined easily. Automation of the manual testing process is done to improves accuracy; increase test coverage, save time and money and help developers and testers to develop test cases quickly and efficiently. This algorithm automates the test suite prioritization process as per the bee colony optimization criteria.

Organization of the paper is as follow. In section 2 related work is presented, BCO method has been explained in section 3. In Section 4 the formalized BCO algorithm for regression test suite prioritization has been presented. Section 5 presents the prioritization of test cases based on total fault coverage with examples. Section 6 gives the comparison of various prioritization approaches to BCO and their APFD values and charts. And finally in the succeeding sections threats to validity, conclusion and future work are presented.

2. Related Work

This section presents an overview of the related work of test case selection and prioritization and the areas where BCO technique has been used to find solutions to problems.

The selection problem has been addressed by many researchers and they have proposed several techniques for solving it. In the selection process we choose test cases from the already available test suite according to some criteria as per the required solution to the problem in hand. The selection process is based on the approach being used to prioritize test cases. “Control Flow based approach” [1], which is based on incremental testing approach. Another approach could be based on “Automated Selection Revalidation” proposed by Fisher et. al. in 1981 [2] and was later extended and implemented for Fortran projects by [3] in 1990. In this a revalidation strategy based on zero-one integer programming model and test criteria based on mathematical optimization algorithm. Next, there was “A Safe, Efficient algorithm for Regression Test Selection technique” given by Rothermal in 1997. In this approach graph theory has been used to identify tests with the potential to identify fault revealing data. Another approach was based on “Intermediate code for virtual machines” given by Rothermal in 2000 [5]. In this the analysis of source code is done to select the candidate test cases. It was designed for C++ projects by Rothermal in 2000[5], redesigned for Java software by Harrold Jones in 2001[6] and then generalized by Koju 2003 [7] for virtual machines and languages like C/C++, Java, .Net etc.

There are a variety of prioritization techniques too. Some of them are “Modification Based Approach” [8], [9], [10],[11] in which prioritization is made after selection and performed based on sorting the test cases selected using Modification based techniques.“Total fault detection technique” [12] in which test cases are prioritized according to the increasing order of the number of faults detected by a test case. “Coverage Based Approach” [8], [12], [13] in which concept like code coverage, branch coverage, statement coverage etc are used for deciding the order of regression test suite. “History based prioritization” [14], [15], [16] in which reordering of test cases is based on the historical execution data. Another technique is “Control flow based technique”[17] in which the test cases are prioritizes using modified condition/decision coverage pairs. Many more techniques are there like “Data Flow based”[18], “Requirement Based”[19],“Cost based”[20],[21] “Slicing Based”[22],[9] “Genetic Based”[23], “Algorithm Based”[4], [24],[25],[26], [27], [28], [29]etc.

Bee Colony Optimization is an emerging field for researchers. It has been applied to areas like “Travelling Salesman Problem” [30] which is a NP-Hard combinatorial problem where an optimal path is to be searched from source to destination for a travelling salesman. “Optimization of problems” [31] where the optimal solution is to be searched on the basis of some known function. “Scheduling problems” [32] like job shop scheduling problems, task scheduling problems, project scheduling problems, process scheduling problems etc. For solving other combinatorial problems like “Generation of pair-wise test sets” [33] where automated test cases are to be generated. The concept of BCO is used for solving “Sudoku Puzzles” [34]. Routing problems where optimized path for better routing is searched using “Routing Algorithm” [35]. Moreover, BCO has also been applied in the area of “Web Search” [36] for effective search engine mechanism. The bee colony optimization has also been used for understanding the concept of software test suite optimization [37]. The bee’s concept has also been used for optimization approaches in the field of engineering [38].

3. Bee Colony Optimization (Natural Phenomenon of Bee Colony)

Bee Colony Optimization is the name given to the colony formed from the mutual understanding and team work of the natural bees in the process of foraging food. It is a fact that all the creatures in this earth follow one or the other mechanism/process to find food sources that suite them.

The build-up of honey bee comb and its management is a classic example of teamwork, synchronization experience and co-ordination. The way the bees find, build and maintain their comb and hive is remarkable. These are the factors which have given rise to interest of researchers to study this system and find solutions to their problems.

In a natural honey bee hive there are a variety of bees with specific role(s) to play. There are three types of bees:

1. Female Queen Bee

There is only one Queen of a bee colony. She is responsible for laying eggs which are used to build new colonies.

2. Male drone bees

There are many drone bees of a colony. They mate with the Queen to build new colonies.

3. Worker bees

There are thousands and thousands of worker bees. These perform all the maintenance and management jobs in the hive. There are two types of worker bees, namely Scout bees and Forager bees, which perform the Scout scenario and Forager scenario respectively, which are collectively responsible for the development of the honey bee colonies.

3.1. The Scout Scenario

In the Scout Scenario there are following steps:

- i. The Scout bees start from the hive randomly in search of the available food sources.
- ii. They continue with this exploration process and return back to the hive until they are out of energy/tired.
- iii. When they return back to the hive, they share their knowledge and experience with the fellow Forager bees by performing the mechanism called "Waggle Dance".
- iv. Waggle Dance is a form of dance in which the bees follow in circular direction in the shape of digit 8. It is repeated again and again. Its intensity and direction gives the idea of food source location and food source quality respectively to other bees. It is the means by which the bees communicate with each other. It is used to convey the parameters like foods Source Quality, distance of food source from hive, Location of food source w.r.t. sun, to guide the path to the available forager bees.
- v. These steps of the scout bees constitute the first step of the BCO process called the "Path Exploration".

3.2. The Forager Scenario

In the Forager Scenario, the Forager bees perform the following:

- i. Upon the arrival of the Scout bee to the hive, the Forager bees observe and learn the steps done by the scout bees so as to ease their journey.
- ii. Then these Forager bees go to the food sources as guided by the Scout bees to exploit Best Food Source Quality food sources.
- iii. These forms the second step of the BCO process called the "Path Exploration".

4. Proposed BCO algorithm

In [19] Artificial bee colony system has been modeled in [29]. This paper presents a new bee colony optimization algorithm that maps the food foraging behavior of natural worker bees as an algorithm to prioritize regression test suite's test cases based on maximum fault coverage. This algorithm takes the test cases as food sources, number of test cases as the number of scout bees, number of forager bees which make the initial population and the ratio of number of faults detected by each test case to the test case's execution time as the food source quality of that test case.

Following is the basic flow of the proposed BCO Algorithm:

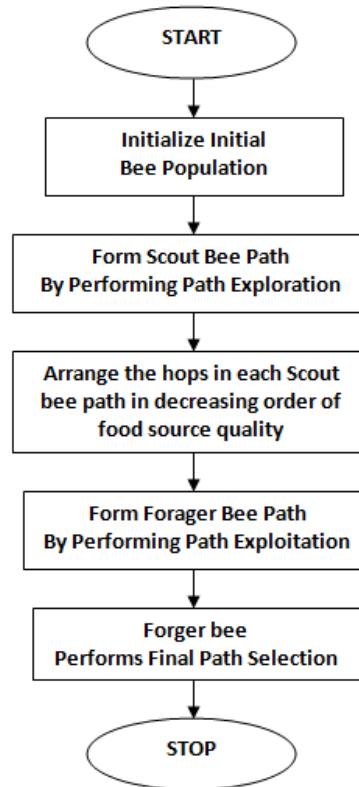


Figure 1. Basic Flow of the Proposed BCO Algorithm.

Following is the list of notations that are being used in the proposed algorithm:

1. N_b : Total number of bees (Initial Population)
2. N_s : Number of scout bees
3. N_f : Number of forager bee
4. FSQ_i : Quality of i^{th} Food Source
5. S_i : i^{th} Scout Bee
6. F_i : i^{th} Forager Bee
7. P_{SBi} : Path of SB_i
8. P_{FBi} : Path of FB_i
9. ET : Execution time of test case
10. NF : Total number of faults
11. FC : Number of Faults covered by a test case

Following two tables will be formed in execution of the algorithm:

1. P_{SB} Table

Path of Scout Bee Table, as seen in Table 2 and 5 contains all the paths explored (visited) by the S_i. It is used for the path exploration phase of S_i. This table will act as input for the P_{FB} table formed by B_F.

2. P_{FB} Table

Path of Forager Bee Table, as seen in Table 3 and 6 contains all the paths exploited (selected) by the F_i. In this table an account of the total ET of each path is also kept. This table helps in final path selection.

4.1. BCO Algorithm

Following are the proposed algorithm's main steps:

1) Assumptions

1. All the food sources have been identified initially.
2. The distance between hive to food source, food source to food source, food source to hive is equal that is unity.
3. A bee will visit any food source only once.
4. The input consists of a test suite T, with following information
 - (1) T={t₁,t₂...t_n} test cases,
 - (2) Faults Covered, (FC) by each test case and
 - (3) Execution Time, (ET) of each test case.
5. The number of bees N_b (N_S + N_F) to search and form the solution is 2n (n for scout bee and n for forager bee).
6. Each S_i will start the exploration process randomly till it develops such a path in which all the faults are covered.
7. Each F_i will exploit only S_i path.
8. Stopping criteria for both the processes is total number fault coverage.
9. No two scout bees will have same random path in the exploration process.
10. FSQ of a test case = (Number of Faults covered by a test case/Execution Time) = (FC/ET)

2) Initializations

- i. N_b = N_S + N_F
- ii. N_S = N_F = number of test cases in the test suite under test.

3) BCO Module

START

(1) For each Scout Bee

(i)Do

{Perform exploration process randomly to develop P_{SB} Table}
until FC of S_i = NF.

(ii)Sort the hops in scout bee path in decreasing ratio of FSQ of each test case in that path.

(2) For each Forager Bee

{Perform Exploitation process to develop P_{FB} Table}
until FC of F_i = NF.

(3)Select Path of bee from P_{FB} Table with minimum ET.

STOP

4.2. Explanation of the BCO Algorithm

1. Path Exploration

This is the first step in the execution of the proposed BCO algorithm. This step is executed by the Scout bees, N_S times, in order to explore all the food sources (Test Case) available. The S_i starts its exploration process randomly. After this they continue the exploration of test cases until the stopping criterion is met. Each path P_{SBi} constructed by the S_i is entered into the P_{SB} Table.

2. Path Exploitation

In the next step the forager bees exploit the P_{SB} . This step uses the P_{SB} table as input. Each F_i starts exploiting the P_{SBi} to form the P_{FBi} . Here each F_i chooses the test case (food source) with best FSQ_i value. The succeeding test cases are chosen such that which detects faults those are not detected by the already existing test cases. This condition makes sure that only those test cases are selected which are effective in fault detection and total fault coverage (stopping criteria). The summation of the ET of all the test cases present in a particular P_{FBi} is done to get that P_{FBi} 's total ET. All these entries are maintained in P_{FB} Table.

3. Path Selection

This is the final step of the BCO algorithm done by the F_i . From the P_{FB} table select the P_{FBi} with minimum Total ET.

4.3. Analysis of the BCO Algorithm

The running time of the proposed BCO Algorithm is bounded by time required to explore path and exploit path. For the input population of size 'n', the path exploration requires $O(n)$ operations . All 'n' particles in the population will go through 'n' iterations for completing the process of path exploration in the proposed BCO algorithm.

For the sorting of the P_{SBi} step $O(n)$ operations will be required.

For the path exploitation step all the 'n' individuals of the population will go for 'n' iterations to develop the path. Hence, taking $O(n^2)$ operation complexity. For the final path selection step only minimization process will take place which takes $O(n)$ operations.

Therefore, the best running time of proposed algorithm is $O(n^2) + O(n) + O(n^2) + O(n)$, which makes the final running time as $O(n^2)$.

4.4. Implementation of Proposed BCO Algorithm

The proposed BCO algorithm has been implemented in CPP compiler comprising of about 300 LOC. There are two structures in this code namely the test_case structure and the bee structure. Since, till now the input table is entered manually the code has been executed on small examples as shown in section 5.1. and 5.2. Work of incorporating file handling into the code is in progress so that this code will be able to take the input automatically and hence it could be used to solve test suites of larger size and it will then require minimum human interface.

5. Prioritization of Test Cases Based on Maximum Fault Coverage

In the proposed BCO technique the test cases are prioritized in such a manner so as to achieve maximum fault coverage within minimum execution time. To ensure that all the faults are covered by the test cases selected in the prioritized ordering we use mutation testing. In this we deliberately alter a program's code, then re-run a suite of valid test cases against the mutated program. A good test will detect the change (fault) in the program and fail accordingly. Thus, the

proposed algorithm's effectiveness is measured using mutation testing. This is illustrated by two examples in section 5.1. and 5.2. in which mutants are created.

5.1. Example 1

The problem taken is “college program for admission in courses”. The problem specification is available at website <http://www.planet-source-code.com>. In this example a test suite has been developed which consisted of 35 test cases. For simplified explanation of the working of the algorithm, a test suite with 9 test cases is considered in it, covering a total of 5 faults.

The input test suite contains 9 test cases with default ordering {T1, T2, T3, T4, T5, T6, T7, T8, T9}, the faults covered (FC) by each test case, the execution time (ET) required by each test case in finding faults and food source quality (FSQ) are as shown in Table 1. Equal priority is given to the number of faults covered and minimum execution time in selecting the test cases.

Table 1. Test Cases with the Faults Covered, Execution Time and Food Source Quality.

Test Case	F1	F2	F3	F4	F5	FC	ET	FSQ _i
T1	X	X	X		X	4	11.5	.34
T2	X	X				2	11.5	.17
T3	X		X		X	3	12.33	.24
T4	X			X	X	3	10.66	.28
T5	X					1	15	.06
T6	X		X		X	3	8.33	.36
T7	X					1	15	.06
T8	X	X		X		3	10	.30
T9			X			1	11	.09

Step 1. Scout Bee Performs Path Exploration and creates P_{SB} Table, Table 2.

Table 2. Path Explored by Scout Bee. (P_{SB} Table).

S _i	P _{SBi}					
S ₁	T1	T6	T7	T4		
S ₂	T2	T1	T9	T4		
S ₃	T3	T1	T6	T8		
S ₄	T4	T3	T5	T7	T2	
S ₅	T5	T1	T4			
S ₆	T6	T3	T5	T7	T2	T4
S ₇	T7	T6	T3	T9	T1	T8
S ₈	T8	T1				
S ₉	T9	T5	T7	T8	T3	

Step 2. Forger Bee Performs Path Exploitation and creates P_{FB} Table, Table 3.

Table 3. Path Exploited by Forager Bee. (P_{FB} Table).

F _i	P _{FBi}			Total ET
F ₁	T6	T1	T4	30.49
F ₂	T1	T4		22.16
F ₃	T6	T1	T8	29.83
F ₄	T4	T3	T2	34.49
F ₅	T1	T4		22.16
F ₆	T6	T4	T2	30.49
F ₇	T6	T1	T8	29.83
F ₈	T1	T8		21.5
F ₉	T8	T3		22.33

Step 3. Forger bee performs final path selection.

Hence PFB₈ with test cases T1 T8 forms the prioritized test suite for the given problem.

5.2. Example 2

Another problem specification for “Hotel Reservation” which reserves the rooms in hotel and maintains the record. The complete problem specification is available at the website “<http://www.planet-source-code.com>”. In this example a test suite has been developed which consisted of 40 test cases. For simplified explanation of the working of the algorithm, a test suite with 5 test cases is considered here, covering a total of 5 faults.

The input test suite contains 5 test cases with default ordering {T1, T2, T3, T4, T5}, the faults covered (FC) by each test case, the execution time (ET) required by each test case in finding faults and food source quality (FSQ) are as shown in Table 4. Equal priority is given to the number of faults covered and minimum execution time in selecting test cases.

Table 4. Test Cases with the Faults Covered, Execution Time and Food Source Quality.

Test Case	F1	F2	F3	F4	F5	FC	ET	FSQ _i
T1	x	x		x		3	12.2	.2459
T2				x		1	10	.10
T3	x			x	x	3	10.67	.28
T4					x	1	7	.14
T5		x	x	x	x	4	9.97	.40

Step 1. Scout Bee Performs Path Exploration and creates P_{SB} Table, Table 5.

Table 5. Path Explored by Scout Bee. (P_{SB} Table).

S _i	P _{SBi}			
	T1	T2	T3	T4
S ₁	T2	T3	T5	
S ₂	T3	T1	T2	T5
S ₃	T4	T3	T2	T5
S ₄	T5	T3		

Step 2. Forger Bee Performs Path Exploitation and creates P_{FB} Table, Table 6.

Table 6. Path Exploited by Forager Bee. (P_{FB} Table).

F _i	P _{FBi}		Total ET
	T3	T1	
F ₁	T3	T1	22.87
F ₂	T5	T3	20.64
F ₃	T5	T3	20.64
F ₄	T5	T3	20.64
F ₅	T5	T3	20.64

Step 3. Forger bee performs final path selection.

Hence PFB₂ with test cases T5 T3 forms the prioritized test suite for the given problem.

6. Comparison

The examples mentioned in sections 5.1 and 5.2 are compared with BCO order w.r.t. the following orderings: No order, Reverse order, Random order, Optimal order of the test cases. The orderings with respect to these approaches for examples in sections 5.1 and 5.2 are listed in Table 7 and 9. These approaches are compared by calculating Average Percentage of Faults Detected (APFD) [40] in Tables 8 and 10, for the maximum fault coverage concept.

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Where, T - The test suite under evaluation

m - The number of faults contained in the program under test P

n - The total number of test cases in and

TF_i - The position of the first test in T that exposes i^{th} fault.

The results obtained for the examples explained above have been plotted in APFD charts in Figure. 2 and Figure. 3. These show that the APFD values for the prioritization achieved using the proposed BCO algorithm are comparable to that obtained with the optimum ordering.

Table 7. Order of Test Cases for Various Prioritization Approaches for Example 1 of Maximum Fault Coverage.

No Order	Reverse Order	Random Order	Optimum Order	BCO Order
T1	T9	T3	T8	T1
T2	T8	T7	T6	T8
T3	T7	T1	T5	T6
T4	T6	T5	T4	T3
T5	T5	T4	T1	T5
T6	T4	T2	T7	T2
T7	T3	T6	T3	T4
T8	T2	T9	T2	T9
T9	T1	T8	T9	T7

Table 8. APFD Values of Various Prioritization Approaches for Example 1.

Technique	APFD %
No Order	77
Random Order	71
Reverse Order	71
Optimal Order	82
BCO Order	82

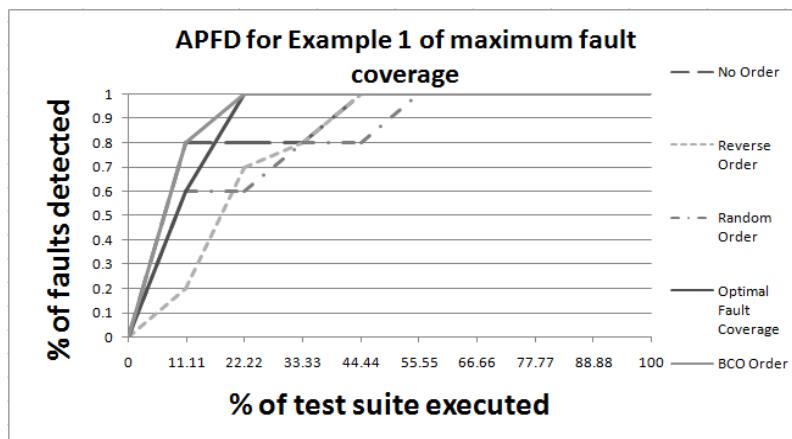


Figure 2. APFD Chart for Example 1 of Total Fault Coverage.

Table 9. Order of Test Cases for Various Prioritization Approaches for Example 2 of Maximum Fault Coverage.

No Order	Random Order	Reverse Order	Optimum Order	BCO Order
T1	T2	T5	T5	T5
T2	T1	T4	T3	T3
T3	T5	T3	T4	T2
T4	T3	T2	T1	T1
T5	T4	T1	T2	T4

Table 10. APFD Values of Various Prioritization Approaches for Example 2 of Maximum Fault Coverage.

Technique	APFD %
No Order	46
Random Order	46
Reverse Order	62
Optimal Order	66
BCO Order	62

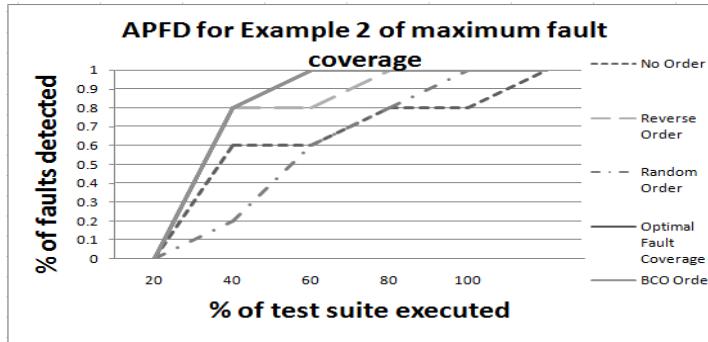


Figure 3. APFD Chart for Example 2 of Maximum Fault Coverage.

7. Threats to Validity

The proposed BCO algorithm which has been presented here has certain short comings which are as follows:

1. In the natural BCO the number of scout bees is 5-10% of forager bees which are assumed to be equal in the algorithm proposed here.
2. The concept of waggle dance has not been incorporated in the algorithm, which is used as a means of communication in the natural BCO because it was not required.
3. Since the paths are developed randomly it is possible that we may sometimes not get optimal/required solution.
4. Although the algorithm has been automated but it requires manual input of test suite information such as test case number, faults covered by each test case and its execution time, which also needs to be automated.

8. Conclusion and Future Work

The Bee Colony Optimization algorithm for the fault coverage regression test suite prioritization has been formulated and presented. This has been done by studying the natural food foraging behavior of bees. This algorithm makes effective use of the path exploration and path exploitation phenomenon of Scout bees and Forager bees for the

prioritization of the fault coverage test suite of the modified code. The proposed BCO algorithm has been explained using examples on maximum fault coverage. The effectiveness of this algorithm has well been demonstrated using APFD metric and chart. The results of the fault based test case prioritization are comparable to the optimal ordering values. The BCO algorithm has been implemented in CPP compiler.

Although the algorithm has been implemented, but it requires manual interface for the input test suite data which makes the utility of the implemented part restricted to small sized test suite. So to apply it to larger programs, there is a need to improve its automation concept to minimize the human interface requirement. Therefore a complete automation tool for the complete usage of the algorithm is being developed. It will also be analyzed on larger projects with large number of test cases and faults.

References

- [1] M.J. Harrold, M.L. Souffia, "An incremental approach to unit testing during Maintenance", Proceedings of the Conference on Software Maintenance (IEEE Cat. No. 88CH2615-3), IEEE Comput. Soc. Press, MA-USA, 1988, pp. 362–367.
- [2] K. Fischer, F. Raji, and A. Chruscicki, "A methodology for retesting modified software", Proceedings of the National Telecommunications Conference, Innovative Telecommunications- Key to the future, IEEE,NY-USA, Nov 1981 pp. 1-6.
- [3] J. Hartmann, DJ Robson, "Techniques for selective revalidation", IEEE Software, Vol. 7, No 1, Jan1990, pp. 31-36.
- [4] Y. Singh, A. Kaur, and B. Suri, "Test Case Prioritization Using Ant Colony optimization", Association in Computing Machinery ,Newsletter ACM SIGSOFT Software Engineering Notes, New York, USA, 2010, pp 1-7.
- [5] G. Rothermel, M. J. Harrold, and J. Dedhia, "Regression test selection for C++ programs", Software Testing, Verification and Reliability, Vol. 10, no. 2, 2000, pp.77-109.
- [6] M.J.Harrold, D. Rosenblum, G.Rothermel and E.Weyuker, "Empirical Studies of a Prediction Model for Regression Test Selection", IEEE Transactions on Software Engineering, Vol.27, No.3, March 2001, pp. 248-263.
- [7] T. Koju, S. Takada, N. Doi, "Regression test selection based on intermediate code for virtual machines", Conference on Software Maintenance, IEEE Computer Society Washington, DC, USA, 2003, pp. 420–429.
- [8] W.E. Wong, J.R.Horgan, S.London, and A.Agarwal. "A study of effective regression testing in practice", Proceedings of the Eighth International Symposium Software Rel. Eng., Albuquerque, New Mexico ,Nov. 1997, pp. 230-238.
- [9] D.Jeffrey, N.Gupta. "Test-case Prioritization using relevant slices", Proceedings of the 30th annual International Computer Software and Applications (COMPSAC), Chicago, USA, September 2006, pp.18-21.
- [10] S. Mirarab and L. Tahvildari, "A Prioritization Approach for Software Test Cases on Bayesian Networks", FASE, LNCS 4422-0276, pp: 276-290, 2007.
- [11] B. Korel, G. Koutsogiannakis, L.H. Talat, "Model-based test suite prioritization Heuristic Methods and Their Evaluation", Proceedings of 3rd workshop on Advances in model based testing (A – MOST), London, UK, 2007, pp. 34-43.
- [12] B. Korel, G. Koutsogiannakis, L.H. Talat, "Model-based test suite prioritization Heuristic Methods and Their Evaluation", Proceedings of 3rd workshop on Advances in model based testing (A – MOST), London, UK, 2007, pp. 34-43.
- [13] S. Elbaum, A.Malishevsky, and G.Rothermel, "Prioritizing test cases for regression testing", Proceedings of the International Symposium on Software Testing and Analysis, Aug 2000, pp. 102-112.
- [14] J.M.Kim, A.Porter. "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environment", Proceedings 24th International Conference Software Engineering, Orlando FL, May 2002, pp.119-129.
- [15] H. Park, H.Ryu, J.Baik, "Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing", Proceedings of second International Conference on Secure System Integration and reliability improvement, IEEE Computer Society, Washington, DC, USA 2008, pp. 39-46.
- [16] Y.Fazlalizadeh, A.Khalilian, H.A.Azgomi, S.Parsa, "Incorporating historical test case performance data and resource constraints into test case prioritization", Lecture notes in Computer Science, Springer, vol. 5668, 2009, pp. 43-57.
- [17] J.A. Jones, and M.J. Harrold, "Test suite reduction and prioritization for modified condition/decision coverage", Proceedings of the IEEE Transactions on Software Engineering, Vol.29, No.3, IEEE Press NJ, USA March, 2003, pp. 195-209.

- [18] M.J. Rummel, G.M. Kapfhammer, A. Thall, "Towards the Prioritization of Regression Test Suites with Data Flow Information", Proceedings of the 2005 ACM Symposium on Applied Computing, Santa Fe, New Mexico March 13-17, 2005, pp. 1499-1504,
- [19] H. Srikanth, L. Williams, J. Osborne, "System Test Case Prioritization of New and Regression Test Cases", Proceedings of International Symposium on Empirical Software Engineering (ISESE), Nov 2005, pp. 64-73.
- [20] A.G. Malishevsky, J. Ruthruff, G. Rothermel and S. Elbaum, "Cost-cognizant test case prioritization", Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln, 2006.
- [21] X. Zhang, C. Nie, B. Xu, B. Qu, "Test Case Prioritization based on Varying Testing Requirement Priorities and Test Case Costs", Proceedings of the 7th International Conference on Quality Software, IEEE Computer Society , Washington, DC, USA, 2007, pp. 15-24.
- [22] D. Jeffrey, N. Gupta, "Experiments with Test Case Prioritization using Relevant Slices", Journal of Systems and Software, Elsevier Science Inc. New York, NY, USA Vol. 81, No. 2, Feb 2008, pp. 196-221.
- [23] K.R. Walcott, M.L. Soffa, G.M. Kapfhammer and R.S. Roos, "Time aware test suite Prioritization", Proceedings of International Symposium on software Testing and Analysis (ISSTA), ACM, New York, NY, USA, July 2006.
- [24] S. Alspaugh, K.R. Walcott, M. Belanich, G.M. Kapfhammer, M.L. Soffa, "Efficient time aware prioritization with knapsack solvers", Proceedings of the 1st ACM international workshop on empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Symposium on Automated Software Engineering(ASE) New York, NY, USA , Nov 2007, pp.13-18.
- [25] Z. Li, M. Harman and R.M. Hierons, "Search Algorithm for Regression test case prioritization", IEEE TSE, Vol. 33, No. 4, IEEE Press Piscataway, NJ, USA, 2007, pp. 225-237
- [26] Z. Li, M. Harman, and R.M. Hierons, "Search algorithms for regression test case prioritization," IEEE Transactions on Software Engineering, San Francisco, CA, USA, 2007, pp. 225-237.
- [27] N.M.A. AL-Salami "Evolutionary Algorithm Definition.", American J. of Engineering and Applied, Science Publications, 2009, pp. 789-795.
- [28] N. Byson, "A goal programming method for generating priorities vectors." Journal of Operational Research Society, Palgrave Macmillan Ltd., Basingstoke, Hampshire, RG21 6XS, England, 1995, pp. 641-648.
- [29] G. Crawford, and C. Williams, "A note on the analysis of subjective judgment matrices." Journal of Mathematical Psychology, The Rand Corporation, USA, 1985, pp. 387-405.
- [30] S. M. Saab, N. K. T. El-Omari, and H. H. Owaied, "Developing Optimization Algorithm Using Artificial Bee Colony System."
- [31] L.P. Wong, M. Y. H. Low, and C. S. Chong, "A Bee Colony Optimization Algorithm for Travelling Salesman Problem", Second Asia International Conference on Modeling & Simulation, IEEE Computer Society, Washington, DC, USA, 2008, pp. 818-823.
- [32] C. S. Chong, and A. I. Sivakumar, M. Y. H. Low, and K. L. Gay, "A Bee Colony Optimization Algorithm to Job Shop Scheduling", Journal Winter Simulation Conference, Monterey, CA, 2006, pp. 1954-1961.
- [33] J. D. McCaffrey, "Generation of pair wise test sets using a simulated Bee Colony Algorithm", 10th IEEE International Conference, IEEE Press Piscataway, NJ, USA, 2009, pp. 115-119.
- [34] J. A. Pacurib, G.M.M. Seno, and J.P.T. Yusiong, "Solving Sudoku Puzzles Using Improved Artificial Bee Colony Algorithm," Fourth International Conference on Innovative Computing, Information and Control, Kaohsiung, Taiwan, 2009, pp. 885-888.
- [35] Sh. Rahmatizadeh, H. Shah-Hosseini, and H. Torkaman, "The ant-bee routing algorithm: A new Agent based Nature-Inspired Routing Algorithm.", Journal of Applied Sciences, Addison Wesley Publishers, Harlow. Esses. UK, 2009, pp. 983-987.
- [36] P. Navrat, T. Jelinek, and L. Jastrzembska, "Bee hive at work: A problem solving, optimizing mechanism", In Proceedings of Nature & Biologically Inspired Computing, IEEE Conferences, Coimbatore, 2009, pp. 122-127.
- [37] D. J. Mala, and V. Mohan "ABC Tester - Artificial Bee Colony Based Software Test Suite Optimization Approach", International Journal of Software Engineering, Springer Global Publication, 2009, pp. 15-43.
- [38] X.S. Yang, "Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms". Artificial Intelligence and Knowledge Engineering Applications: A Bio Inspired Approach, Lecture notes in computer science, Springer, Berlin / Heidelberg, 2005, pp. 317-323.
- [39] H. Owaied, and S. Saab, "Modeling Artificial Bees Colony System". ICAI, Proceedings of the 2008 International Conference on Artificial Intelligence, Las Vegas, Nevada, USA, 2008, pp. 443-446.
- [40] K. R. Walcott, G. M. Kapfhammer, M. L. Soffa, and R. S. Roos, "Time-aware test suite prioritization", International Symposium on Software Testing and Analysis, Association in Computing Machinery, Portland, Maine USA, 2006, pp. 1-19.

Authors



Dr. ArvinderKaur is an Associate Professor with the University School of Information Technology, Guru Gobind Singh Indraprastha University, India. She obtained her doctorate from Guru Gobind Singh Indraprastha University and her master's degree in computer science from Thapar Institute of Engineering and Technology. Prior to joining the school, she worked with B.R. Ambedkar Regional Engineering College, Jalandhar and Thapar Institute of Engineering and Technology. She is a recipient of the Career Award for Young Teachers from the All India Council of Technical Education, India. Her research interests include software engineering, object-oriented software engineering, software metrics, software quality, software project management, and software testing. She also is a lifetime member of ISTE and CSI. Kaur has published 60 research papers in national and international journals and conferences.



Shivangi Goyal received B. Tech in IT from Guru Gobind Singh Indraprastha University in 2009. She is currently pursuing M.Tech from Guru Gobind Singh Indraprastha University. Her area of interest is Software Engineering.

