Optimizing Automated Programming Contracts with Modified Ant Colony Optimization

S.V.Gayetri Devi, T.Nalini

Research Scholar, Department of Computer Science and Engineering, Bharath Institute of Higher Education and Research, Chennai, India
Professor Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute, Chennai, India
Gayetri.venkhatraman@gmail.com, drnalinichidambaram@gmail.com

***Abstract* – Development of sufficiently optimized software coding Contracts with insignificant association from developers directs at lowering the setbacks in abiding by the input specifications of the software, delivering systematic testing of several real life software. The associated framework put forward primarily speculates dependency aspects signifying both behavior and semantic attributes well-defined as conditions in a Decision tree as Automated Contracts for conformance with input specs. The contracts are perfected based on their feasibility with a modified form of Ant Colony Optimization algorithm motivated by pursuing behavior of ant colonies of real world, with considerable comprehensiveness in the Fault identification potential. This research focuses on the decision tree classifiers are** J48, RandomForest classifiers have same precision value which is 0.96 of precision value. RandomTree and REPTree classifiers have same precision value which is 0.95 of precision value, HoeffdingTree classifier is 0.82 of precision, DecisionStump is zero precision value. The J48, RandomForest classifiers have same recall value which is 0.96 of recall value. RandomTree and REPTree classifiers have same recall value which is producing 0.95 of recall value, HoeffdingTree classifier is having 0.76 of recall value, DecisionStump is having zero precision value.The RandomForest is having highest ROC value whichi is 1. The J48 classifier , REPTree classifier have 0.99 ROC value. HoeffdingTree classifier is 0.98 of recall value, RandomTree classifier has 0.97 ROC value and finally the lowest ROC value is produced by DecisionStump Decision classifier.The RandomForest, RandomTree, REPTree classifiers are having highest PRC Area value which is 0.99. The J48 classifier is having 0.92 PRC Area, HoeffdingTree classifier is 0.85 of PRC Area, and DecisioStump classifier is having 0.23 PRC Area value. Among the set of decision classifiers, the J48 decision classifier is producing the best results based on the accuracy, precision, recall, ROC and PRC area values which are compare with other models.

Keywords – Contracts, Meta-Heuristic, Pheromone paths, Static-Dynamic code analysis, Decision Tree

I. INTRODUCTION

The vital aspects such as Reliability and Measurability are assessed with efficient testing activities [1, 2]. To ease off the challenges in observing the input specifications and design with multifarious software levels and higher number of contracts, a framework to build those contracts from the software modules post analysis with negligible association from programmers, is desirable [4, 8]. Hence, the assertions developed must be fortified to keep hold of only feasible contracts in a time subjugated verification background.

This paper puts forth Ant Colony Optimization (ACO), a Bio- motivated metaheuristic algorithm in order to automatically create and optimize software coding contracts. The insight behind ACO aspires on the seeking activities of real world colonies of ants. When watching out for foodstuff, the ants chiefly discover the area adjoining their nest at random. As they pass through, ants deposit a pheromone specimen trail across the ground. As soon as an ant recognises a source of food, it judges the quality as well as the size of the food and takes a fragment of it to the nest. While coming back, the pheromone portion that an ant dropped on the base might be reliant on the food quantity and characteristic. These trails guide further ants to the source. It is specified that message passing amongst the ants across these pheromone marks aids them in unearthing the shortest routes concerning the food sources and nests. The shortest path discovery skills of ant colonies in reality are applied in simulated ant colonies in landing at optimization solutions.

The optimization problem could be resolved by ACO continually in two stages:

1. Engaging probability distribution across the investigation space to find candidate solutions in a probabilistic way
2. Alter probability distribution by means of candidate solutions

The presented work accentuates on optimizing the neurons connection (numerical) weights on the Feed Forward Neural networks [9].

This paper is structured as follows: First, the review of literature presented in section 2. In section 3, materials and methods are described. In section 4, the impelentation and discussions have given. Finally, the conclusion is given in section 5.

II. RELATED WORKS

Programming contracts endeavor to achieve the correctness and dependability intentions of verification tools to uncover potential software downsides. Embracing increasing number of contracts in the software expends significant efforts and thus one main difficult step is the appropriate development of programming contracts and judicious selection of those dependable contracts and equivalent test cases to advance with software testing [11].

Table 1 provides a summary of the various related works in prevailing optimization algorithms.

Table 1. Related works on prevalent optimization algorithms

Paper & Author	Optimization method	Merits	Future working lines
Regression test optimization and prioritization using Honey Bee optimization algorithm with fuzzy rule base [13] S.Kumar et.al., (2020)	1. Honey Bee Optimization algorithm is used to improve the error detection rate during Test case prioritization in minimal time 2. Average percentage of fault detection (APFD) metric is used to quantify the prioritization outcomes	1. The volume of the test cases is lessened by the Bee algorithm as well as fuzzy rule base. 2. Achieved greatest APFD outcomes amongst the various algorithms compared 3. Consumed the lowest count of tests to detect all the software errors	Future work to verify algorithm on real world repositories data
Optimized test suites for automated testing using different optimization techniques [3] Manju Khari et.al, (2020).	1. Assesses the performance of 6 meta-heuristic optimization techniques - particle swarm optimization (PSO), hill-climbing algorithm (HCA), cuckoo search algorithm (CS), firefly algorithm (FA), artificial bee colony algorithm (ABC) and bat algorithm (BA) for optimizing the path and branch coverage created by the test data. 2. Algorithms verified against 5 programs in Java implementation for average, best and worst time, best time, worst time, path coverage and fitness function values of the created test suites	1. Artificial Bee Colony algorithm resulted in the most optimum test suites in acceptable time 2. Bat algorithm gave least optimal results in quicker time 3. Slowest was the Firefly algorithm Particle swarm, Cuckoo search and hill-climbing algorithms performed in avaerage	Scope to improvise the best accomplishment algorithms for path coverage-oriented optimization techniques
Automatic Contract Insertion with CCBot [15] Carr e.al., (2017)	1. Instrumenting C# program using Contracts as method presuppositions, post conditions and object invariants with CodeContracts function following the Static Analysis 2. Automatically translating the programs source wise with contracts included as annotations and not as warnings. 3. The instrumented assertions identify segment errors, overflows of integers and buffers, and floating point accuracy discrepancies	1. CodeContractsBot tool (CCBot) realization, the tool aiding error identification and program code quality improvement tool.	Deadlock and Data race issues not captured
A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection [5] De Souza et.al, (2015).	1. Employ Multi-Objective optimization techniques to select test cases with test case criterion greater than 1. 2. BMOPSO-CDR as well as BMOPSO-CDRHS techniques of MOO are applied and compared against two baseline methods – NSGA-II and MBHS covering functional and structural testing scenarios.	BMOPSO-CDRHS algorithm performed better and was not impacted by the kind of testing being employed	1. Scope to perform a more exhaustive parameter study with further settings and specific features of the PSO. 2. Scope to apply the algorithms on higher number of programs to verify if equivalent results are obtained.
An Ant Colony Optimization Algorithm Based Automated Generation of Software Test Cases [16] Saju Sankar S et.al, (2020)	Optimal test cases are developed by employing an altered form of ACO as Comprehensive Improved Ant Colony Optimization (ACIACO) to determine the efficient optimization route and by establishing transformation relationship.	Attained maximum coverage with reduced iterations generating feasible test cases covering all transitions once at least.	Recommends agent as well as swarm oriented algorithms for developing effective automated test suites

III. METHODOLOGY FOR CONTRACTS OPTIMIZATION USING ANT COLONY OPTIMIZATION ALGORITHM

A central phase of the method presented in this paper for developing coding contracts is the application of an altered variant of Ant colony optimization algorithm to separate out the feasible as well as non-feasible contracts. The precursory phase of extracting programming contracts and ensuing optimization [7] is completed by semantically abstracting the code constitution and the behavior accompanying characteristics using applicable dynamic and statically scrutinized knowledge [14, 17]. This knowledge is then outlined as standards as a Decision Tree. Decision Tree algorithms are implemented in Python.3.8.0. Specifications linked criteria for the testing sections on the Decision tree are stated as contracts for developing software programs [12].

The necessary steps for the proposed work for contracts optimization is along the following lines. To commence with ants start from the decision list number same as that of ants' count and the recent contract is stored in a set 'C'. The succeeding list to be traversed by is established randomly and probabilistically on the grounds of pheromone. Passing through the preferred decision list of assertions and taking it inside the set 'C' represents the next step. The purpose is to inspect if all the contracts are evaluated. Otherwise, the ensuing node to be travelled into is handled in an akin way. The time for processing the total path of every single ant is detailed and set 'C' is dissipated accordingly. The optimal path in this distinct iteration is identified and the pheromone on the given path is revised. Likewise, the time restriction to deal with the decision list is confirmed and if not gratified, the equivalent algorithm is restated in the succeeding iteration.

For a specified decision list of assertions, the difficulty to pick and optimize the contracts can be established as follows:

Given the exclusive set 'C' (Decision list of assertions) of 'n' assertions, find the subset 'S', consisting of 'm' number of contracts (m<n), in a way that contracts are chosen and ordered on based on utmost error coverage capability of contracts. The optimization question is symbolized by an undirected graph 'G' (V, E) where V≡C expresses the set of assertions (vertices) and E representing graph edges. Contracts are characterized along graph vertices. w_i signifies weight of distinctive 'i^{th}' graph edge representative of pheromone footpath associated with edge e_i.

The stated model deals with only the contracts' decision list with two objectives in mind. The first objective is intended to consider all contracts from decision list and the second is to expend slighter time to locate flaws minimalizing costs and efforts.

The over-all framework of the improved ACO to optimize contracts is depicted in Figure 1 as given below:

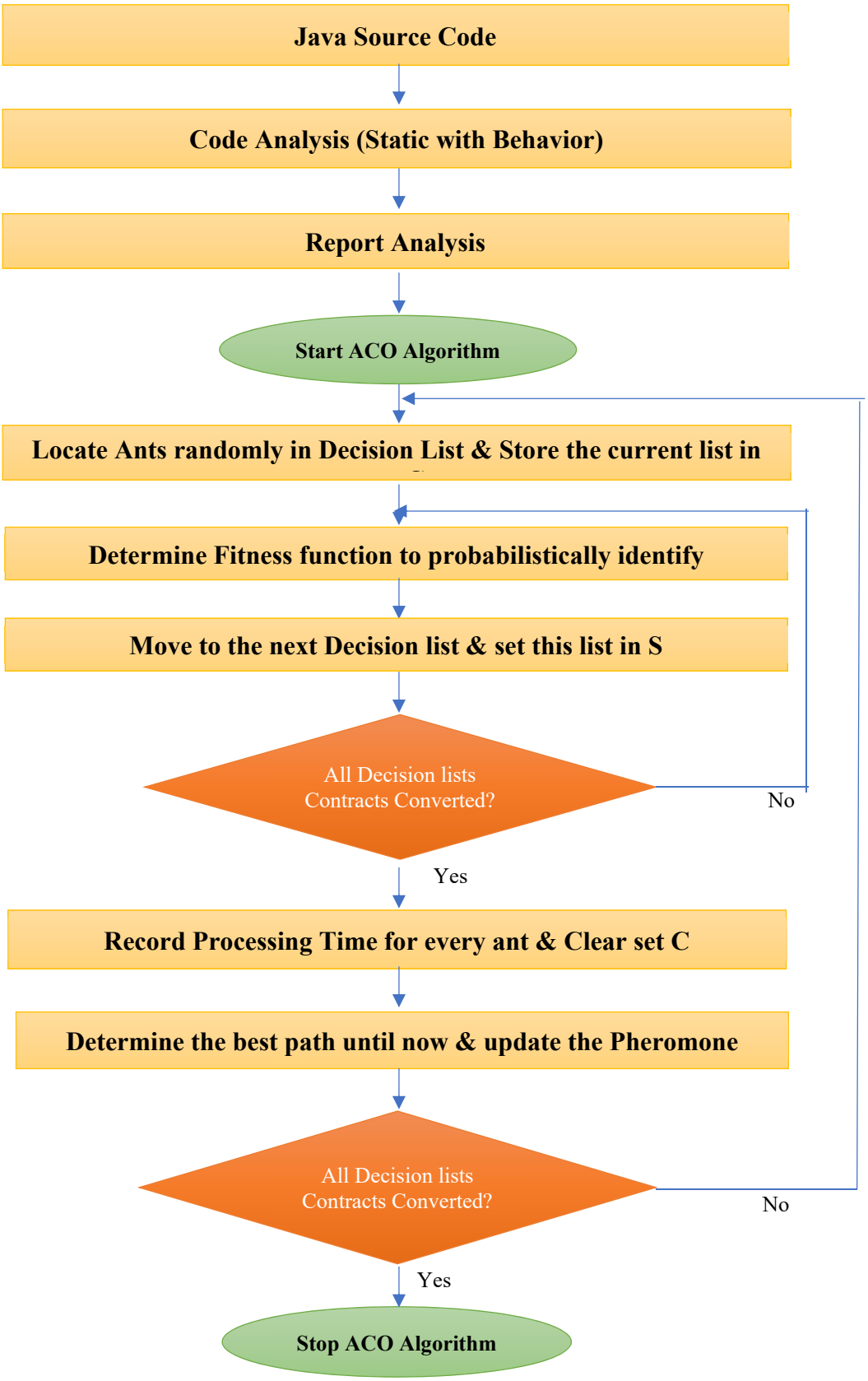

Figure 1.Proposed Architecture

The dataset C is termed as a tuple of assertions (contracts) C_i from i=1 to n as ($C_1, C_2 \ldots C_n$). The feasibility estimation of C is indicated by C'. The greatest time to prioritize the assertions is the greatest ability of knapsack in which knapsack elements are articulated by contracts, execution time of individual contract is designated by its weight.

The following figure 2, thus points out the ACO technique for automated contracts to be optimized.

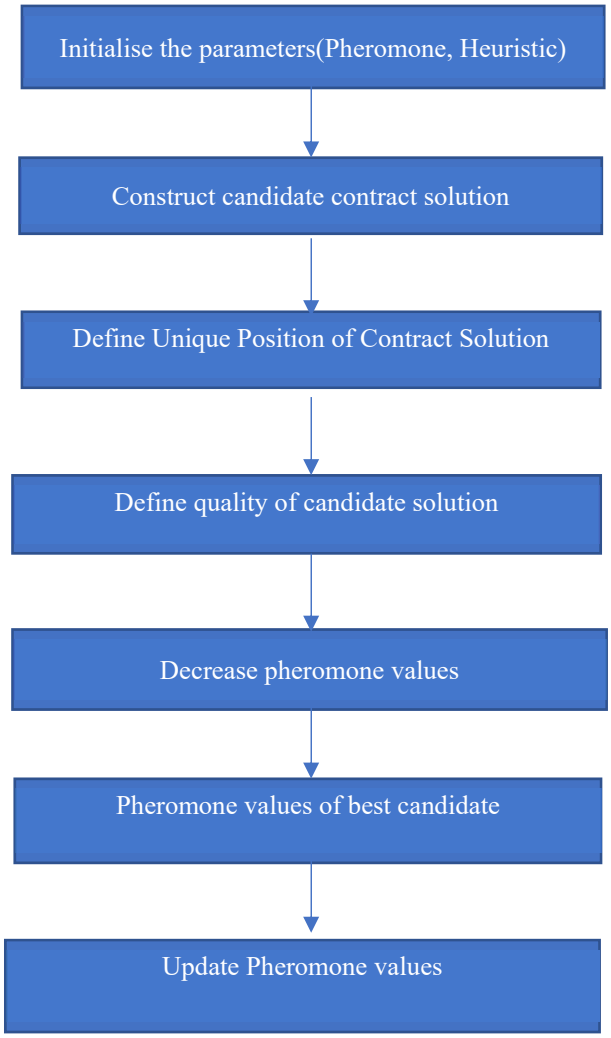

Figure 2. Steps for modified ACO algorithm for Contracts optimization

Being population focused algorithms, 'n' ants arise candidate elucidations to optimization problem of decision list of assertions, indicative of colony of ants and solutions are materialized by a probabilistic technique, pheromone as well as heuristic data to choose optimal solutions. At every single iteration, the pheromone values are revised using quality pointers of candidate assertion solutions.

At every point of the tree building process, an ant randomly chooses vertex to call on due to pheromone amount as well as the heuristic data, like Equation - 1. An ant while building a candidate tree, has the leaf and internal nodes denoted by the agreeing class objects. In order to exclude nodes directing to improvement in the tree's condition, a tree shortening class is engaged. Each internal node is made use of for incrementing pheromone values of the related depth of tree nodes. This increase is interrelated to the importance of the decision tree.

The following stages are vital for optimizing the contracts making:

1. Establish node coverage 'NC' for every decision list containing contracts.
   Assume NC (contract) = s_1, s_2 … s_n, wherein s_1, s_2, s_n represent the transitions from source to destination nodes.

2. The number of contracts becoming 0 indicates the attainment of optimized contracts set.

Node coverage is calculated for each decision list and identify which contracts are covered by other contracts. The contracts are valid if the code coverage is empty. Others are deemed invalid and can be disregarded.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

A multiple-layered software application is developed for the purpose of Concurrent Java software verification [10] and this embodies supplying inferred behavior allied dependencies and semantic data from source under test [6] and then subjecting the contracts generating process to optimization. Java web application using Servlet - JSP units, versioned - 1.8, with Eclipse platform in Windows OS 10 is used to realize the software programming optimized contracts. The source code to carry out tests can be gained access at github: Banking software application. The Dataset for optimized Contracts formation can be obtained from https://github.com/nardevar/Banking.

J48 classifier based out of Decision tree is utilized in the presented ACO adapted algorithm to assess the fitness of the derived Decision list enclosing the contracts and also to revise the pheromone values.

The Decision Classifiers like

Table 2, below, shows the output of code analyzer stage that extracts the semantic information of the input source files.

Table 2. Code Analyser Output

S.No	File name	File Class Count	File Method Count	File Main Method Line No#	File Total Lines	File Iteration Count
1	EmployeeServlet.java	1	19	No Main Method	341	29
2	MainControllerServlet.java	1	63	No Main Method	788	55
3	Now.java	1	5	No Main Method	58	2
4	ContextListener.java	1	5	No Main Method	46	0
5	AccountNotFoundException.java	1	1	No Main Method	25	0
6	AccountsNotFoundException.java	1	1	No Main Method	25	0
7	BillPaymentException.java	1	1	No Main Method	25	0
8	DateException.java	1	1	No Main Method	14	0
9	PayeeNotFoundException.java	1	1	No Main Method	25	0
10	TransferAccountNotFoundException.java	1	1	No Main Method	25	0
11	TransferException.java	1	1	No Main Method	25	0
12	UserNotFoundException.java	1	1	No Main Method	25	0
13	adminlogin.java	1	7	No Main Method	65	0
14	BankDB.java	1	29	No Main Method	536	10
15	DAOFactory.java	1	3	No Main Method	28	1
16	DBConnection.java	1	8	No Main Method	100	1
17	EmployeeDB.java	1	16	No Main Method	356	8
18	ErrorHandling.java	1	10	No Main Method	95	19
19	Authentication.java	1	3	No Main Method	59	0
20	AuthenticationDetails.java	1	8	No Main Method	101	0
21	AccountActivityDetails.java	1	8	No Main Method	107	0
22	AccountDetails.java	1	4	No Main Method	62	0
23	AccountSummaryDetails.java	1	4	No Main Method	62	0
24	BillAccountDetails.java	1	8	No Main Method	134	3
25	CustomerDetails.java	1	19	No Main Method	213	0
26	EmployeeDetails.java	1	6	No Main Method	83	1
27	PayeeDetails.java	1	3	No Main Method	52	0
28	TransferAccountDetails.java	1	5	No Main Method	85	1

Table 3, as shown below, portrays the snapshot of Dynamic analysis stage to extract the behavioral dependent classes in the source files.

Table 3. Behavioural Analysis Output

S.No	Class Name	Dependent Class Name
1	BankDB	DBConnection
2	EmployeeDB	DBConnection
3	AccountNotFoundException	Exception
4	AccountsNotFoundException	Exception
5	BillPaymentException	Exception
6	DateException	Exception
7	PayeeNotFoundException	Exception
8	TransferAccountNotFoundException	Exception
9	TransferException	Exception
10	UserNotFoundException	Exception
11	EmployeeServlet	HttpServlet
12	MainControllerServlet	HttpServlet
13	adminlogin	HttpServlet

Table 4 shows the Background metrics of the input source files as below.

Table 4. Background Metrics

S.No	Metric	Count		S.No	Metric	Count
1	Coupling between object classes (CBO):	28		**16**	Number of Class(NC):	28
2	Number of Children (NOC):	13		**17**	Number of Lines of Source Codes(NLSC):	3560
3	Number of Attributes (NOA):	377		**18**	Number of Iteration(NOI):	130
4	Number of Instance Variable (NIV):	35		**19**	Number of called methods(NCM):	181
5	Depth of Inheritance Tree (DIT):	8		**20**	Rate of called methods(RCM):	8
6	Number of Methods per Class(Min) (NOMMIN):	1		**21**	Number of method invocations on a class(NMI):	181
7	Number of Methods per Class(Max) (NOMMAX):	63		**22**	Number of internal method invocations on a class(NIMI):	5
8	Number of Instance Method (NIM):	107		**23**	Number of external method invocations on a class(NEMI):	102
9	Number of Local Methods (NLM):	241		**24**	Number of called class (static) methods(NCCM):	2
10	Response For a Class (RFC):	33		**25**	Number of class (static) method calls on a class(NCMI):	1
11	Number of Static Methods(NOSM):	2		**26**	Number of created instances (NCI):	41
12	Number of Private Methods (NPRM):	1		**27**	Number of created objects by the class instances(NCO):	22
13	Number of Protected Methods (NPROM):	4		**28**	Total number of calls(TI):	181
14	Number of Public Methods (NPM):	100		**29**	Number of calls by Owner(ITI):	5
15	Lack of Cohesion amongst methods (LCOM):	7		**30**	Number of calls by Foreign(ETI):	102

Table 5 shows the automated contracts generated for input source files using the modified ACO.

Table 5. Automated Contracts Generation per Source File

S.No	Class List	Decisions Count	Sample Contract
1	EmployeeServlet.java	27	assert(module.equals("ADD_CUSTOMER")):"Check module Value:"+module;
2	MainControllerServlet.java	27	assert(session != null):"Check session Value:"+session;
3	Now.java	3	assert(day >= day2):"Check day Value:"+day;
4	ContextListener.java	2	assert(context != null):"Check context Value:"+context;
5	adminlogin.java	1	assert(rs.next())):"Check rs Value:"+rs;
6	BankDB.java	8	assert(rows > 0):"Check rows Value:"+rows;
7	EmployeeDB.java	6	assert(accountType == 0):"Check accountType Value:"+accountType;
8	ErrorHandling.java	4	assert(ERR_CNT == 0):"Check ERR_CNT Value:"+ERR_CNT;
9	Authentication.java	1	assert(result != null):"Check result Value:"+result;

Table 6, below depicts the Performance parameters of ACO oriented automated coding contracts generation.

Table 6. Contracts generation Performance analysis after and prior to ACO algorithm

S.No	Performance parameter	Before ACO	After ACO
1	Time for processing in secs	7.3	6.4
2	CPU Utilization	74.2	67.3
3	Memory Utilization	82.1	78.2
4	Number of files processed for Contracts generation	28	9

The graphical results of the optimization performance in terms of files considered for contracts formation and the CPU-Memory consumption along with processing time is shown in Figure 3 and Figure 4 depicts the contracts generated in the 9 source files as illustrated below.

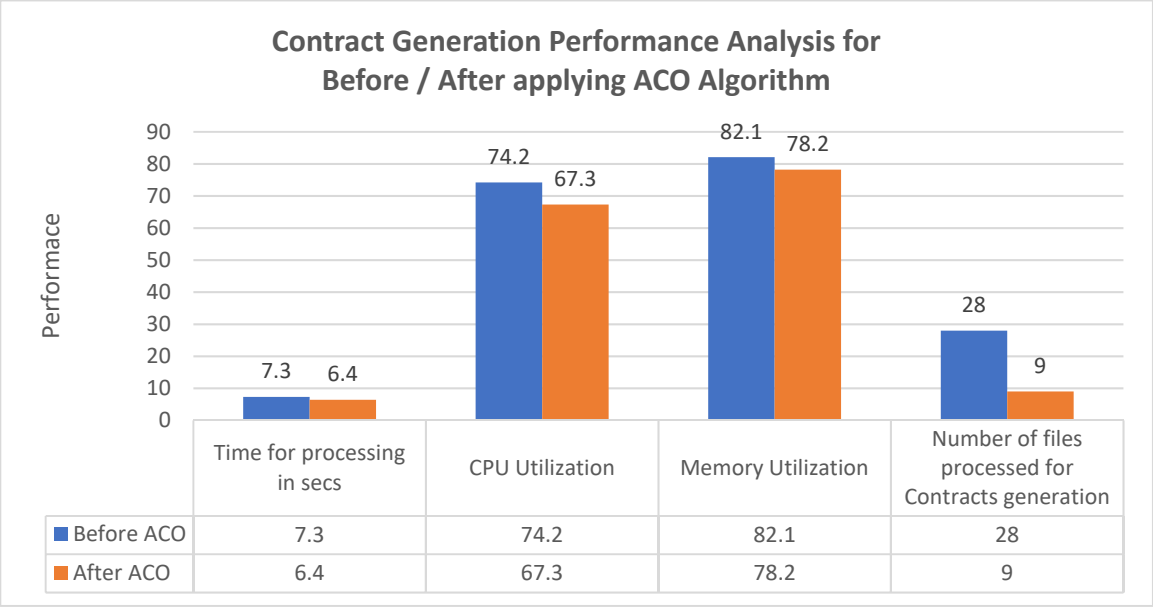

Figure 3.Automated Contracts optimization performance of ACO

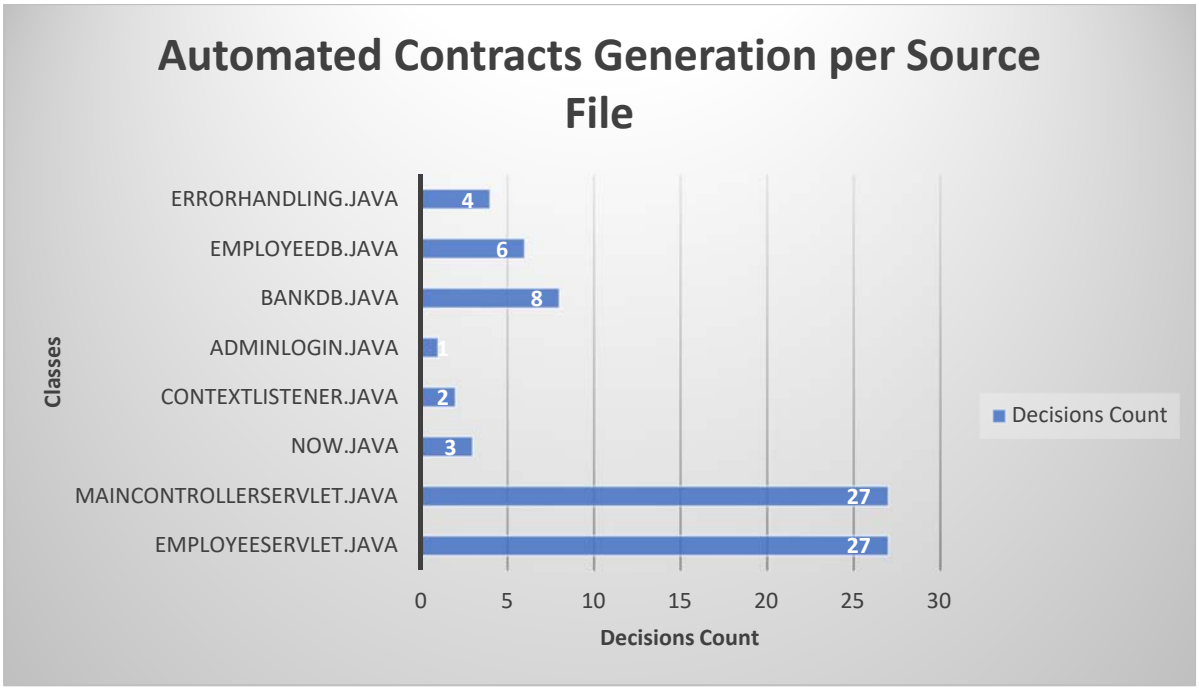

Figure 4 Automated Contracts generation with ACO

28 input Java files are taken for contracts derivation stage ahead of being subjected to optimization and shown to have Processing time of 7.4 secs, CPU utilization of 74.2% and Memory usage of 82.1% without optimization. With ACO in place, an improved Processing time of 6.4 secs, CPU usage of 67.3% as well as Memory consumption of 78.2% accomplished. It is shown that with ACO applied, the model processes and generates contracts only for 9 source files since valid behavioral dependencies exist among the classes of these source files alone.

Table 7: Decision Tree Classifiers

Classifiers	Accuracy	Precision	Recall	ROC	PRC Area
J48	99.53%	0.96	0.96	0.99	0.92
DecisionStump	28.09%	0	0	0.78	0.23
HoeffdingTree	76.02%	0.82	0.76	0.98	0.85
RandomForest	99.05%	0.96	0.96	1	0.99
RandomTree	99.36%	0.95	0.95	0.97	0.99
REPTree	99.42%	0.95	0.95	0.99	0.99

The above table, the J48 classifier holds 99.53% of accuracy, DecisionStump classifier holds 28.09% of accuracy, HoeffdingTree holds 76.02% of accuracy level, RandomForest holds 99.05%, RandomTree holds 99.36% of accuracy level,and REPTree holds 99.42% of accuracy.

The J48 classifier has 0.96 of precision value, DecisionStump classifier has 0 of precision value, HoeffdingTree has 0.82 of precision value, RandomForest has 0.96 of precision value, RandomTree 0.95 of precision value, and REPTree has 0.95 of precision value.

The J48 classifier has 0.96 of recall value, DecisionStump classifier has 0 of recall value, HoeffdingTree has 0.76 of recall value, RandomForest has 0.96 of recall value, RandomTree 0.95 of recall value, and REPTree has 0.95 of recall value.

The J48 classifier has 0.99 of ROC value, DecisionStump classifier has 0.78 of ROC value, HoeffdingTree has 0.98 of ROC value, RandomForest has 1 of ROC value, RandomTree 0.97 of ROC value, and REPTree has 0.99 of ROC value.

The J48 classifier has 0.92 of PRC Area value, DecisionStump classifier has 0.23 of PRC Area value, HoeffdingTree has 0.85 of PRC Area value, RandomForest has 0.99 of PRC Area value, RandomTree 0.99 of PRC Area value, and REPTree has 0.99 of Area value.

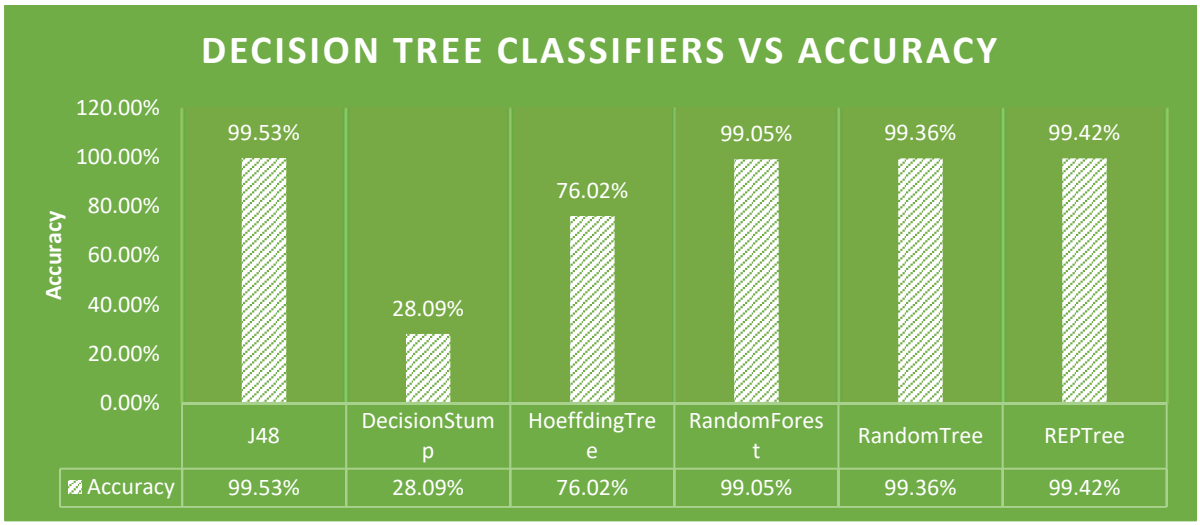

Figure 5: Decision Tree Classifiers Vs Accuracy

The J48, RandomForest, RandomTree and REPTree classifiers have above 99% of accuracy level but HoeffdingTree classifier is 76.02% of accuracy level, DecisionStump is having low accuracy level which is 28.09%. J48 is having the 99.53% of accuracy level.

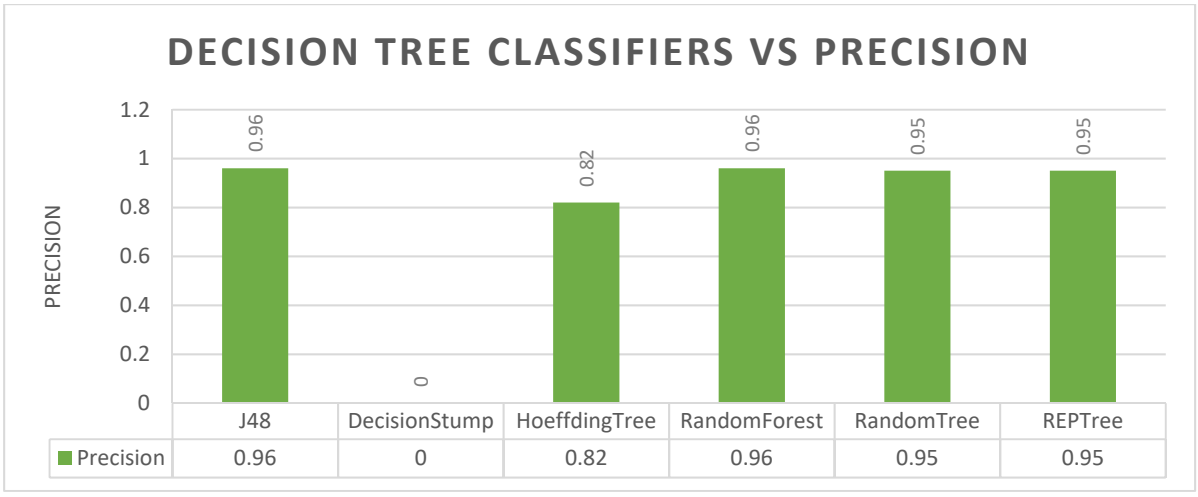

Figure 6: Decision Tree Classifiers Vs Precision

The above diagram presents that the J48, RandomForest classifiers have same precision value which is 0.96 of precision value. RandomTree and REPTree classifiers have same precision value which is 0.95 of precision value, HoeffdingTree classifier is 0.82 of precision, DecisionStump is zero precision value.

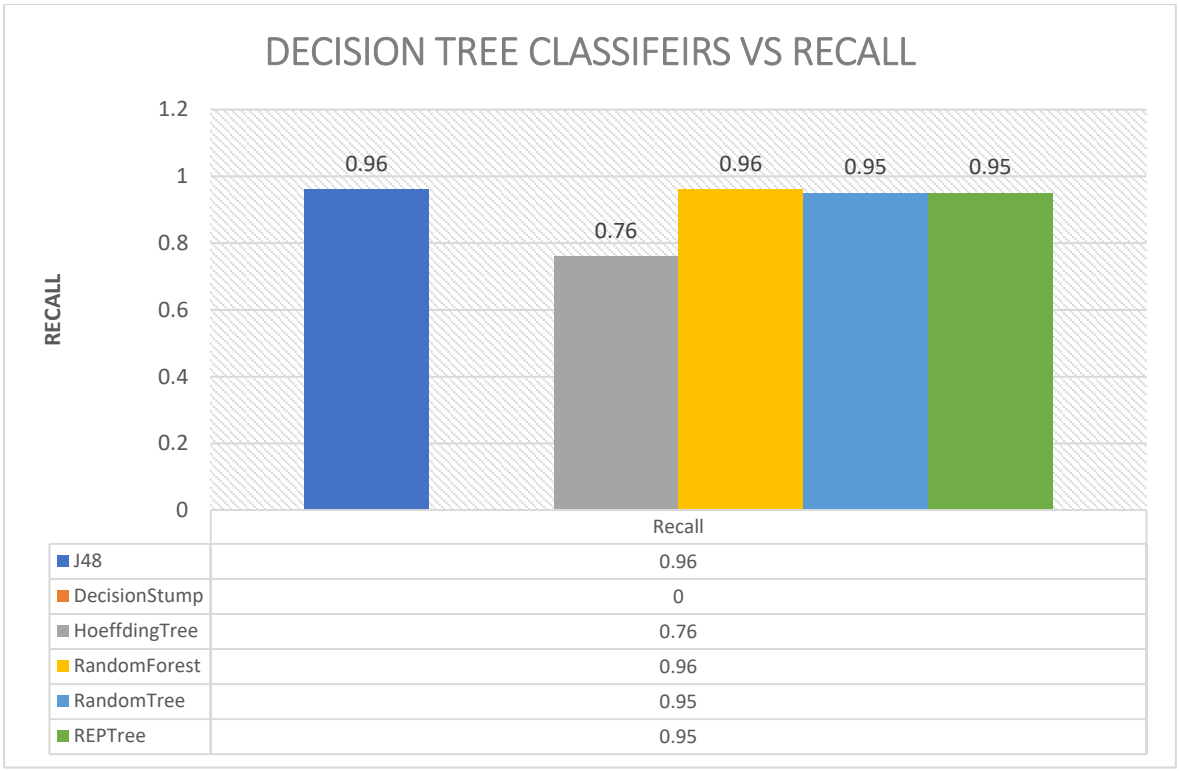

Figure 7: Decision Tree Classifiers Vs Precision

The above diagram presents that the J48, RandomForest classifiers have same recall value which is 0.96 of recall value. RandomTree and REPTree classifiers have same recall value which is producing 0.95 of recall value, HoeffdingTree classifier is having 0.76 of recall value, DecisionStump is having zero precision value.

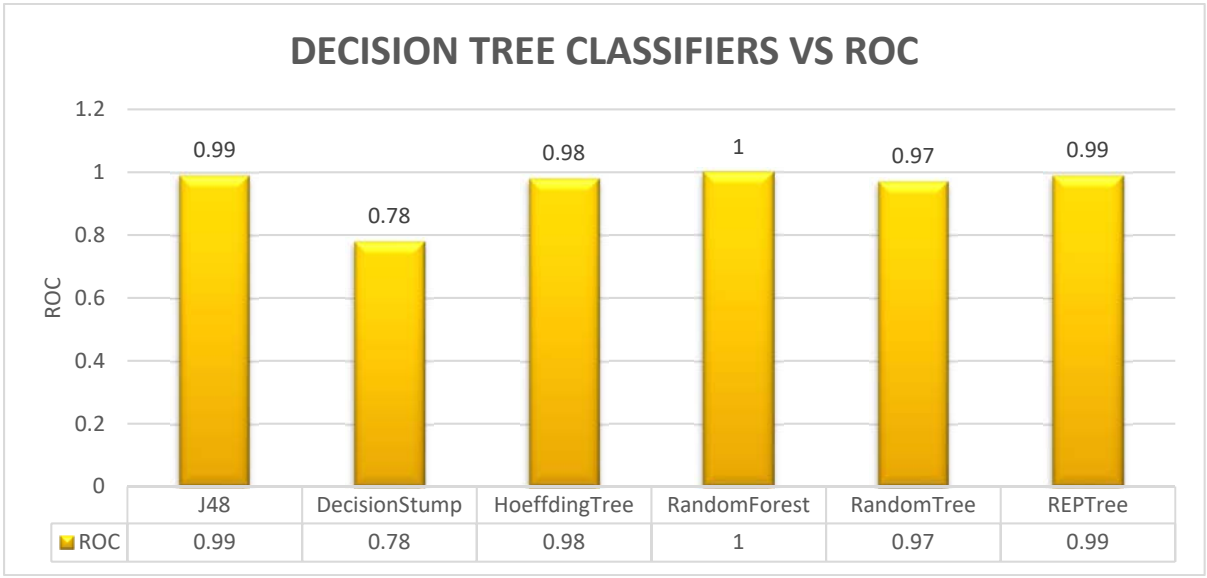

Figure 8: Decision Tree Classifiers Vs Precision

The above diagram presents that the RandomForest is having highest ROC value whichi is 1. The J48 classifier , REPTree classifier have 0.99 ROC value. HoeffdingTree classifier is 0.98 of recall value, RandomTree classifier has 0.97 ROC value and finally the lowest ROC value is produced by DecisionStump Decision classifier.

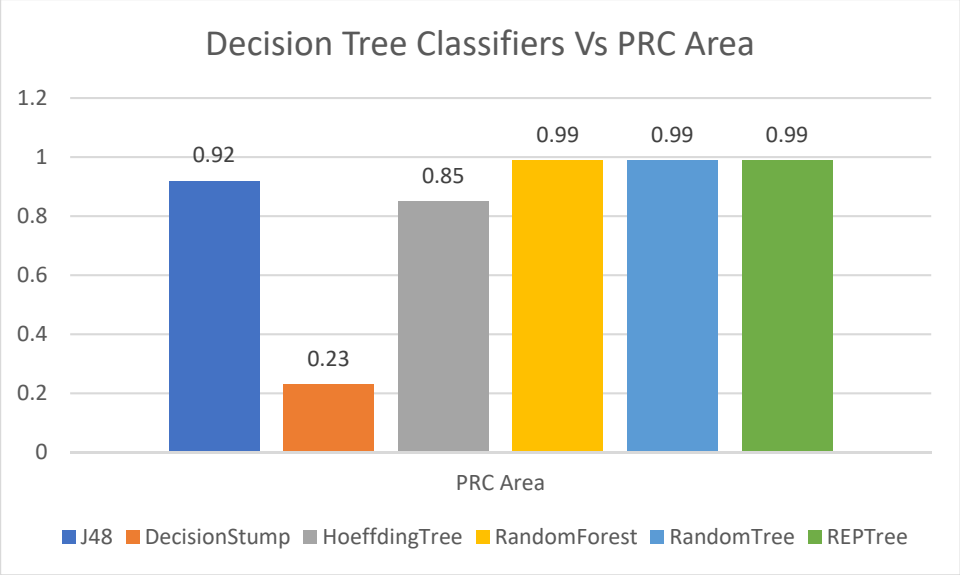

Figure 9: Decision Tree Classifiers Vs Precision

The above diagram presents that the RandomForest, RandomTree, REPTree classifiers are having highest PRC Area value which is 0.99. The J48 classifier is having 0.92 PRC Area, HoeffdingTree classifier is 0.85 of PRC Area, and DecisioStump classifier is having 0.23 PRC Area value.

V. CONCLUSION

For systematic Verification and Validation of diversified software, it is indispensable to give emphasis on the dependencies analyses unveiled in the Source code essentials. This paper brings forth a wider framework to articulate contracts with developers' association at basic minimum in the contracts derivation process by appropriate analyses of software and then adopt modified Ant Colony Optimization to bring about the apt programming contracts as per input specifications. The magnitude of the optimized contracts can be substantiated by comparing across various metaheuristic optimization methods for prominent software verification reassurance.

Among the set of decision classifiers, the J48 decision classifier is producing the best results based on the accuracy, precision, recall, ROC and PRC area values which are compare with other models.

REFERENCES

[1] Ian Sommerville. –, (2010). - "Software Engineering" (9th edition.), Addison-Wesley Publishing Company, USA.
[2] C. D. Knutson and S. Carmichael, - "Verification and Validation for Embedded Software" - , Embedded. System. Program. Spring, - (2001)
[3] Khari, M., Kumar, P., Burgos, D. et al. (2018),"Optimized test suites for automated testing using different optimization techniques". Soft Comput 22, 8341–8352, (2018). https://doi.org/10.1007/s00500-017-2780
[4] S.V.Gayetri Devi, and C. Nalini., - 2020, "Prioritized Automated Generation of Contracts with Modified Swarm Optimization". International Journal of Advanced Science and Technology, 29(8s), 2432 - 2439. - Retrieved from http://sersc.org/journals/index.php/IJAST/article/view/14731
[5] De Souza, L.S., Cavalcante Prudêncio, R.B. & de Barros, F.A. (2015) "A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection". J Braz Comput Soc 21, 19 - (2015). https://doi.org/10.1186/s13173-015-0038-8
[6] S.V.Gayetri Devi &, C.Nalini. – 2020, - "Optimization of Automated Software Contracts Generation by Modified Particle Swarm Optimization", - International Journal of Future Generation Communication and NetworkingVol. 13, No. 1, (2020), pp.629-637.
[7] Chang C., Wu X. - (2020) "An Improved Particle Swarm Optimization Algorithm". In: Xu Z., Choo KK., Dehghantanha A., Parizi R., Hammoudeh M. (eds) Cyber Security Intelligence and Analytics. CSIA 2020, Advances in Intelligent Systems and Computing, vol 928. Springer, Cham. https://doi.org/10.1007/978-3-030-15235-2_195
[8] S. V. Gayetri Devi and C. Nalini. – july 2019, "A Systematic Judgment to Automated Programming Contracts Generation". International Journal of Recent Technology and Engineering (IJRTE), ISSN: 2277-3878, Volume-8 Issue-2, (July 2019).
[9] Xian-Da Zhang. (2020), Neural Networks. In: A Matrix Algebra Approach to Artificial Intelligence. Springer, Singapore., 2020: https://doi.org/10.1007/978-981-15-2770-8_7
[10] Gayetri Devi, S.V, Chidambaram, Nalini and Narayanan, Kumar, (2018) - "An efficient software verification using multi-layered software verification tool". International Journal of Engineering & Technology. 7. 454. 10.14419/ijet.v7i2.21.12465.
[11] Arialdis Japa, Daniel Brown, Yong Shi, 2019: "Towards Optimizing Data Analysis for Multi-Dimensional Data Sets", IEEE Future of Information and Communication Conference (FICC), San Francisco, March 14-15, (2019)
[12] S. V. G. Devi and C. Nalini, - "Enhanced K-Means Clustering Algorithm for Feasibility Assessment of ACC". 2020, - Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India. (2020), pp. 340-345, doi: 10.1109/ICIRCA48905.2020.9182934.

[13] Nayak, S., Kumar, C., Tripathi, S. (2020), – "Regression test optimization and prioritization using Honey Bee optimization algorithm with fuzzy rule base" (2020). Soft Computing (2020). https://doi.org/10.1007/s00500-020-05428-z
[14] Julien Signoles, Nikolai Kosmatov, and Kostyantyn Vorobyov, "E-ACSL, a Runtime Verification Tool for Safety and Security of C Programs". (2017), Tool Paper. In International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES), September , 2017
[15] S. A. Carr, F. Logozzo and M. Payer, "Automatic Contract Insertion with CCBot," (2017), in IEEE Transactions on Software Engineering, vol. 43, no. 8, pp. 701-714, 1 August 2017, doi: 10.1109/TSE.2016.2625248.
[16] S S.S., S V.C. 2020, An Ant Colony Optimization Algorithm Based Automated Generation of Software Test Cases. In: Tan Y., Shi Y., Tuba M. (eds) Advances in Swarm Intelligence. ICSI 2020. Lecture Notes in Computer Science, volume 12145. Springer, Cham. https://doi.org/10.1007/978-3-030-53956-6_21
[17] S. E. Bondarev, M. A. Chudinov and A. S. Prokhorov (2019), "The Analysis of Existing Methods of Software Verification," 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia, 2019, pp. 191-193, doi: 10.1109/EIConRus.2019.8657169.