Modified artificial ant colony optimization based approach for number of fault prediction to assist bug report reverse engineering

Dhayashankar J M[1], Dr. A.V. Ramani[2]
[1]*Department of Computer Applications, Sri Ramakrishna Mission Vidyalaya College of Arts and Science*
[2]*Department of Computer Science, Sri Ramakrishna Mission Vidyalaya College of Arts and Science*
[1]*(E-mail: dayal.jm@gmail.com)*

Abstract— In current information technology era most of the business applications are depending on the software and their efficiency and quality are main factors. The software maintenance plays a vital role because there may be a chance of upgrade or error occurrence during the course of use. In such cases the details of documentation relevant to the concern software is very essential for the maintenance group but if it is not properly maintained then the process undergoes reverse engineering to recover the design by finding the faults occurred in the software. Presence of fault not only affects the software quality in addition it increases the development cost also. Due to the poor documentation about the software under investigation often leads to difficult in prediction of faults. This paper aims at developing a Meta heuristic model using ant colony optimization for finding faults in the given software systems. The proposed approach has been validated using experimental investigation on six software projects which are available in the tera-PROMISE repository. The performance of the proposed method is evaluated using different evaluation metrics and the result shows that modified Ant Colony Optimizaiton (ACO) model have produced more significant results in faults prediction accuracy

Keywords—*Ant Colony Optimization (ACO), Software fault prediction, software metrics, tera-PROMISE repository, T-test analysis*

I. INTRODUCTION

Software bug is a major bug in coding implementation since without correct code it is not possible to produce correct result. The software engineering team have bug reports which explains the type of bugs occurred in each module during the software development phase. If those documents are not perfectly handled or if those bug reports are missing then a need arise to develop a prediction system to find the occurrence of the fault in software to correct the bugs occurred in the software. The software bug report is known as problem report which can be identified using the software fault prediction models. This helps in allocation testing resources efficiently and economically. According to a survey [1] detection and removal of software's faults cover around 50% of the total project budget. Abundant research has been carried out in the earlier works but there are still issues that prevent them from becoming widely adopted in practice because most of the earlier software fault prediction studies have predicted the fault proneness of the software modules in terms of fault and non-faulty (binary class classification). There are several issues with this binary class classification even if the performance of the prediction model was reported excellent the interpretation of the finding is hard to put into the proper usability context i.e. identification of the number of faults per module.

In order to explain the practical use of the software fault prediction model this work present an approach to predict the number of faults in a given software system using Ant Colony Optimization (ACO). The contribution of this paper is to explore the abilities of the ant colony optimization for the number of software fault prediction. This proposed work developed fault prediction models using modified ACO and the performance of the constructed model is evaluated using Error Rate , Recall and completeness measures. This research work indicates that ACO based fault prediction approach produces significant results to predict the number of faults in software system. he rest of the paper is organizes as follows, Section 2 contains related work, Section 2 consist of overview of proposed approach, section 4 presents a brief description of the used software fault datasets, software metrics considered and dependent variables. In section 5, a detailed explanation of experimental setup along with the discussion of the evaluation metrics are considered. The result of the investigation is presented in section 6 and finally draw the conclusion about the performance of the proposed method is discussed in section 7.

II. RELATED WORK

There have been many existing works relevant to predict fault proneness of software modules in terms of the modules being faulty or non-faulty. But studies reporting the fault proneness of software modules in terms of predicting the fault density or the number of faults in a software module are very few. Jun Zheng [2] described that the software fault prediction model can be built with the help of threshold-moving technique. The motive of the software developer is to develop the better quality software on time and inside the financial plan. Software fault prediction model classifies the modules into two classes: faulty modules and non – faulty modules. R.Shatnawi [3] states that the majority of the modules for finding the prediction performance are correct whereas some modules are defective. They applied technique to find the number of faults in the particular module. This

technique is called Eclipse. This technique works well on real world objects called Object Oriented systems. In this Object Oriented System, they used the existing defected data for eliminating the defective modules.

Shepperd, Schofield and Kitchenham [4] discussed that need of cost estimation for management and software development organizations and give the idea of prediction and discuss the methods for estimation. Alsmadi and Magel [5] discussed that how data mining provide facility in new software project its quality, cost and complexity also build a channel between data mining and software engineering. Runeson and Nyholm [6] discussed that code duplication is a problem which is language independent. It is appearing again and again another problem report in software development and duplication arise using neural language with data mining. Lovedeep and Arti [7] data mining provide a specific platform for software engineering in which many task run easily with best quality and reduce the cost and high profile problems. Ostrand et al [8] have used negative binomial regression analysis to predict the fault proneness in software modules. In their study a NBR model was developed and used to predict the expected number of faults and the fault density in every module of the next release of the system. In another study Afzal et al [9] performed an empirical study to predict the fault count using the genetic programming. They performed their experiments over the three industrial projects and evaluated their results using some goodness of fit and predictive accuracy parameters. They concluded that GP is statistically significant and accurate to predict fault counts.

Arvinder and Inderpreet [10] in their work used six machine learning models for software quality prediction on five open source software. Varieties of metrics have been evaluated for the software including C & K, Henderson & Sellers, McCabe etc. Results show that Random Forest and Bagging produce good results while Naive Bayes is least preferable for prediction. Dhyan and Saurabh [11] in their paper classified and detect software bug by J48, ID3 and Naïve Bayes data mining algorithms. Comparison of these algorithms is done to detect accuracy and time taken to build the model. Rohit et al [12] devised a Bayesian Regularization (BR) technique has been used for finding the software faults before the testing process. This technique helps to reduce the cost of software testing which reduces the cost of the software project.

This proposed work differs from the previously mentioned studies in several ways. Firstly, most of the earlier studies have used industrial datasets. But this work used six open source datasets. Secondly the existing methods used change history and LOC metric of the files to determine number of faults, whereas this work used 20 object oriented metrics. Most of the prior works have used some hypothesis testing or goodness of fit parameters to evaluate their results. While this proposed work used Error Rate, Recall and completeness measures to perform fair evaluation of the proposed method results and remained consistent with these three set of measure for all the used dataset. In comparing the proposed model with the existing methods, it is found that the proposed fault prediction model achieved the higher recall value. The error rate analysis also confirmed the effectiveness of the proposed modified ACO prediction model.

2.1 EXISTING ANT COLONY OPTIMIZATION

It is a Meta heuristic technique which is based upon the natural phenomena. ACO is a probalistic technique which gives solution by using previous results. In this process each ant follow different path to reach to the destination and secrete pheromone liquid on the way to destination. The path which has the highest liquid pheromone is considered as the shortest path and all other ants follow the same path. So pheromone liquid is used to attract the other ant and update the latest information about the path [17. 18]

III. PROPOSED METHODOLOGY OF MODIFIED ANT COLONY OPTIMIZATION BASED FAULT PREDICTION MODEL

Ant colony optimization algorithm is a heuristic algorithm that mimics the behavior of ants to find the best food sources in this work best possible solution to perform a user specified task. It is a search based method which search for optimal solution to perform a given computation task. Ant Colony optimization algorithm starts with a randomly generated population of potential space. Subsequently, it iteratively transforms initial population of potential solutions into a new generation of the population by applying the fitness function. These selected individual are close to the goal and the whole process is continued in this manner until the termination condition is met, which is generally bound to the maximum number of generations. After the termination criterion is satisfied, the single best individual in the population is chosen as resulting solution.

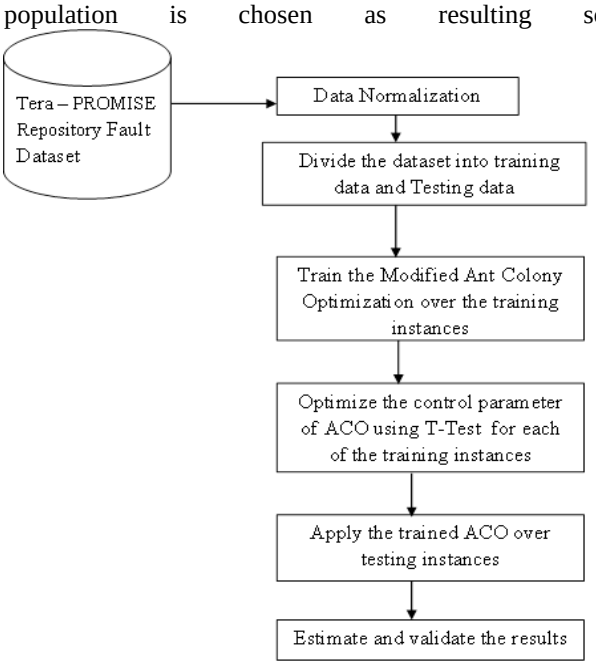


Figure Overview of the Modified ACO based Fault Prediction Model

Figure 1 depicts an overview of the proposed modified Ant Colony Optimization fault prediction model. First requisite fault data of 6 projects listed in Table 1 are taken from tera-PROMISE Repository. The data set containing fault information and twenty software source code metrics are also included. All source code metrics are normalized over the range between 0 to 1 using Min-Max normalization technique.

The normalization of source code metrics is required to adjust the defined range of metrics. Hence, before applying the machine learning algorithm and subsequent t-test analysis, the input metrics are normalized or standardized using min-max scaling. The conventional ACO algorithm [17] is modified by using T-test analysis to determine the significance among the software metrics and those metrics are considered as the best software metrics which have high impact on determination of number of faults occurred in each module of the given dataset. The characteristics of features to select significant sets of features without involving a learning algorithm T-test has been employed to remove insignificant features. The objective of this step is to test the relationship between each source code metric and fault proneness. In this study, t-test analysis is used to test the statistical significance between faulty and non-faulty group metrics. In 2-class problems (faulty class and non-faulty classes), test of null hypothesis (H0) means that the two populations are not equal; on other words, there is a significant difference between their mean value and both features are different. It further implies that the metrics affect the fault prediction result. Hence, these metrics have been considered and those having no significant difference between their mean values are rejected. Therefore, it is necessary to accept the null hypothesis (H0) and P −value for each metric as a measure of how effectively it separates the groups. Software Metrics which have P - value smaller than 0.05 consist of strong discrimination powers. Based on the highest significance of each software metrics the artificial ants determine the best solution in terms of number of faults prediction.

IV. ALGORITHM FOR PROPOSED METHOD OF MODIFIED ANT COLONY OPTIMIZATION FOR FAULT PREDICTION

Input: Instance of Tera – PROMISE Repository Fault Dataset
Output: Number of faults predicted in each module

1. Begin Initialize code_mod = cmf1,cmf2, cmf3... cmfn
2. Initialize fault_time = E1, E1, E1,….En
3. Assign start_post pos(p) = rand(x),rand(y)
4. Repeat
5. For k = 1 to m do /* for the m ants*/
6. Ant k randomly selects a node s 1
7. for i = 1 to n-1 do

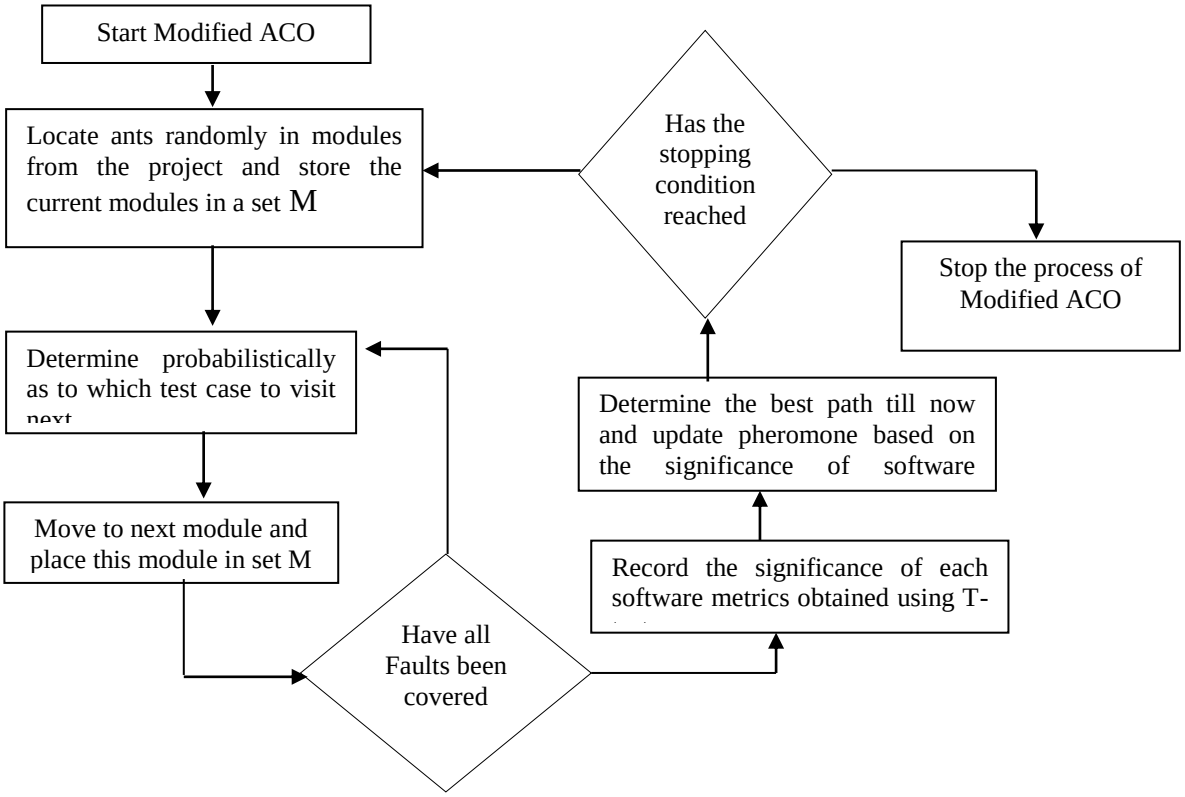


Figure Flow chart of the proposed Modified ACO Model for fault prediction

8. Ant k selects the next node according to the equation as follows $P_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k=1}^{n} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}$ Where, τij is the pheromone on the edge between nodes $v i$ and $v j$, ηij is a heuristic function which is defined as the visibility of the edge ($v i$,$v j$). Parameters α, β determine the relative influence of the pheromone and the heuristic information.
9. End for i
10. Calculate the fitness of the path formed by ant k according to the equation

$$Q(S)=c * \frac{1}{n} \sum_{i=1}^{n} d(s_i)$$

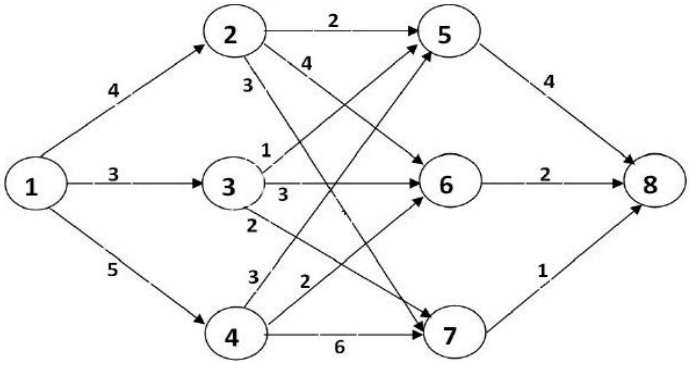
Here, Q(S) is quality of path, $d(s i)$is the degree of the node $s i$, C is a positive constant

11. End for k
12. Update the pheromone values according to

$$\tau ij(t+1)=\rho \cdot \tau ij(t)+\Delta \tau ij(t)$$

13. Until max1 ≤ i,j ≤ n $|\tau ij (t+1)-\tau ij (t)| \leq \varepsilon$ or t > Nc;
14. Score = τ;
15. Output the score matrix Score;
16. Stop (fault cover)
17. End loop
18. End

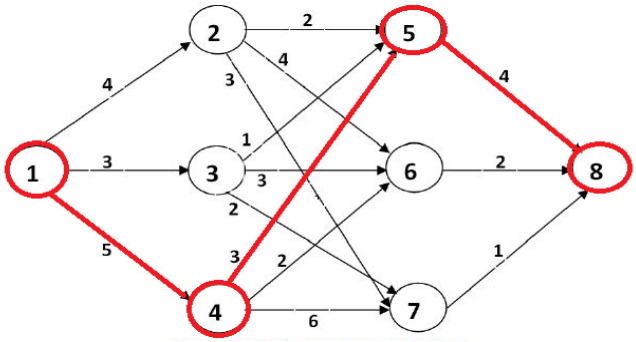
Working Example of proposed Modified Ant colony Optimization in number of Software fault prediction



The above figure depicts the process of feature selection using ACO algorithm in a graph representation. For assumption let us consider that the dataset consist of 8 attributes the goal of the ants is to select optimal features to predict the number of faults in a software. In this graph each vertex represents the feature or attribute of the fault prediction dataset. The attributes are connected using edges and the weight of the edges signifies the correlation among attributes or vertex's which are obtained using T-test significance. In this example the attribute1 is connected with three attributes 2, 3 and 4. Now this graph consists of three artificial ants where each ant selects different edges. Let us assume that ant 1 select the edge between the attributes 1-2. Ant 2 selects the edge for traversal is 1-3. The ant 3 chooses the edge between the attributes 1-4. Now the pheromone trail is calculated based on t-test between the attributes of the selected edges and they are assigned as weight

	Iteration 1	Iteration 2	Iteration 3	Total Attributes selected	Weight Calculation
Ant 1	1-2	2-5,	5-8	1-2-5-8	4+2+4 =10
		2-6,	6-8	1-2-6-8	4+4+2=10
		2-7	7-8	1-2-7-8	4+3+1=8
Ant 2	1-3	3-5,	5-8	1-3-5-8	3+1+4=8
		3-6,	6-8	1-3-6-8	3+3+2=8
		3-7	7-8	1-3-7-8	3+2+1=6
Ant 3	1-4	4-5,	5-8	1-4-5-8	5+3+4=12
		4-6,	6-8	1-4-6-8	5+2+2=9
		4-7	7-8	1-4-7-8	5+1+1=7

The table above shows the complete calculation of the graph shown in the figure. Here each ant traverses in three different directions. The Ant 1 has the probability of choosing 3 different paths of attribute combination 1-2-5-8, 1-2-6-8 and 1-2-7-8. The Ant 2 has the probability of selecting 3 different paths of attributes 1-3-5-8, 1-3-6-8 and 1-3-7-8. The Ant 3 also has the probability of picking 3 different paths of attributes 1-4-5-8, 1-4-6-8 and 1-4-7-8. To choose best combination of attributes the corresponding weights of each path is calculated as shown in the above table. According to this example ant 3 produces the optimal features selection with the highest weight value it means the amount of pheromone deposited on this path is higher and expose that the potential prediction of number of faults in a software can be predicted more optimal using this modified ACO.



V. EXPERIMENTAL METHODOLOGY

In this section a brief overview of the software fault dataset used for simulation comparison, with the information of independent and dependent variable are discussed. Six different dataset are used in this simulation in which two of them belong to PROP project, two are form xerces project and remaining two are from camel project. PROP dataset is one of the largest dataset that are available in tera-PROMISE Repository data repository [13, 14]. This dataset is collect for a software project that was developed in a commercial organization and it is written in the java language. Similarly Xerces is an Apache collection of software libraries used for parsing, validating and manipulating XML. Camel is rule based routing and mediation engine that offers the interfaces for the Enterprise Integration Patterns (EIPS). The Xerces and Camel both are open source projects. Each of the six fault dataset used in this work consist of same 20 object oriented metrics and number of faults information for each software module. The percentage of faulty modules varies between 9% - 55% (approx.), which makes them the good candidate for the experimental study.

VI. SOFTWARE FAULTS DATASET DESCRIPTION

6.1 Dependent and Independent Variables

In this work, the objective is to develop a fault prediction model using Modified Ant Colony Optimization for the number of faults prediction in a given software systems. Therefore in this work source code metrics considered as dependent variables. The fault proneness of a class is the probability that a class contains fault, given the metrics for that class. Since this proposed prediction model assigned an expected number of faults to each module of software. Therefore, the number of faults is selected as depended variable instead of dividing them into two categories of fault and non-faulty.

Table 1: Detail of the fault Datasets used for the Study

OO Metric	Description
Weighted methods per class (WMC)	Sum of the complexities of methods defined in class
Depth of inheritance tree (DIT)	Maximum height of the class hierarchy
Number of children (NOC)	Number of immediate descendants of the class
Coupling between object classes (CBO)	Number of classes coupled to a given class
Response for a Class (RFC)	Number of different methods that can be executed when an object of that class receives a message
Lack of cohesion in methods (LCOM)	Number of sets of methods in a class that are not related through the sharing of some of the class's fields
Afferent coupling (Ca)	Number of other classes use the specific class
Efferent coupling (Ce)	Number of classes used by the specific class
Number of public methods (NPM)	Number of methods in a class that are declared as public.
LCOM3	Lack of cohesion in methods Henderson-Sellers version
Lines of code (LOC)	Number of lines in the text of the source code
Data access metric (DAM)	Ratio of the number of private (protected) attributes to the total number of attributes declared in the class
Measure of aggregation (MOA)	Number of data declarations, whose types are user defined classes
Measure of functional abstraction (MFA)	Ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class
Cohesion among methods (CAM)	Sum of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods
Inheritance coupling (IC)	Number of parent classes to which a given class is coupled

Coupling between methods (CBM)	Number of methods to which all the inherited methods are coupled
Average method complexity (AMC)	Average method size for each class.
Maximum McCabe's cyclomatic complexity (M ax − CC)	Maximum cyclomatic complexity of methods defined in a class
Average McCabe's cyclomatic complexity (Avg–CC)	Average cyclomatic complexity of methods defined in a class

TABLE 2: List of projects used in our experiments, number of modules and percentage of faulty classes

Project	No. of modules	No. of Faulty modules	Faulty (%)
prop-3	10274	1180	11.49
prop-4	8718	840	9.64
xerces-1.2	440	71	16.14
xerces-1.3	453	69	15.23
camel-1.0	339	13	3.83
camel-1.2	608	216	35.53

From the table 2 it is observed that the project prop consist is the largest dataset whose prop-3 consist of 10274 classes and prop 4 consist of 8718 classes with faulty modules 1108 and 840 respectively. Xerces project with release of 1.2 and 1.3 consist of 440 and 453 modules respectively and they contain fault modules of 71 and 69. The Camel project with the version 1.0 and 1.2 has 339 and 608 modules respectively with 13 and 216 faulty modules.

For each module its percentage of fault is calculated as follows:

$$\text{Faulty (\%)} = \frac{\text{No.of.faulty modules}}{\text{total number of modules}} * 100$$

6.2 Evaluation Measure

To find the performance of the proposed Modified ACO based fault prediction model this work used three types of performance evaluation measure. To measure the deviation between the predicted and actual fault values this work used Error rate parameters. The prediction accuracy is measure by using Recall and Completeness of the model is measure by using completeness measure. The description of each of the aforementioned measures is as follows:

6.3 Error Rate

The difference among the actual fault values and predicted fault values is measured using the average relative error (ARE) is defined as

$$\text{ARE} = \left(\frac{1}{n}\right) \sum_{i=1}^{n} |\bar{yi} - yi| / (yi + 1)$$

Where $\bar{yi}$ is the predicted value of the number of faults in a software model and Yi is the corresponding actual value. N is the number of modules. In the case of ARE, as the actual value of the faults may be zero, one is added to the denominator to make the definition always well-defined [15].

6.4 Recall

Recall shows the fraction of relevant instances that are retrieved. It measures the ability of the classifier to identify a condition correctly. A classifier with high recall value insures that a high number of positive examples will be identified. It is defined as

Recall = True Positive / (True Positive + False Negative)

The used fault data is imbalance in nature. It contains a large number of non-fault modules compared to faulty modules. The main goal of the proposed Modified ACO based fault prediction model is to identify faulty modules as much as possible. So that the processes for software reverse engineering to recover the correctness of the software can be done more optimally. IF a fault remained unidentified, then it will require lots of resources and efforts for identification at later stages or may be remain unidentified and cause the sever effect on software performance later. For this reason, this proposed work select recall value as the measure to assess the performance of the proposed fault prediction model.

6.5 Completeness

Completeness measure is defined as the number of faults found in the modules classified as fault-prone divided by the total number of faults in the software system [16]. This parameter tells how complete the proposed fault prediction model is.

$$\text{Completeness} = \frac{\text{observed Total no of faults found by classifier}}{\text{Actual Total number of faults in the software}}$$

6.6 Results

This section discusses about the result of the various evaluation parameters and assesses the accuracy and completeness of the fault models. For each of the dataset this work used 10 fold cross validation in which 90% of the involved training and remaining 10% is used for testing the model. Likewise ten iterations are performed for the whole dataset to train and test.

Table 3 Error rates Produced by existing Genetic Algorithm, Decision Tree, Ant colony Optimization and proposed Modified Ant colony optimization for all six Datasets

Data set	Genetic Algorithm	Decision Tree	Ant Colony Optimization	Modified ACO
PROP-3	0.11	0.27	0.19	0.09

PROP-4	0.35	0.48	0.37	0.25
xerces-1.2	0.39	0.52	0.4	0.24
xerces-1.3	0.32	0.43	0.38	0.26
camel-1.0	0.19	0.27	0.23	0.12
camel-1.2	0.45	0.59	0.48	0.32

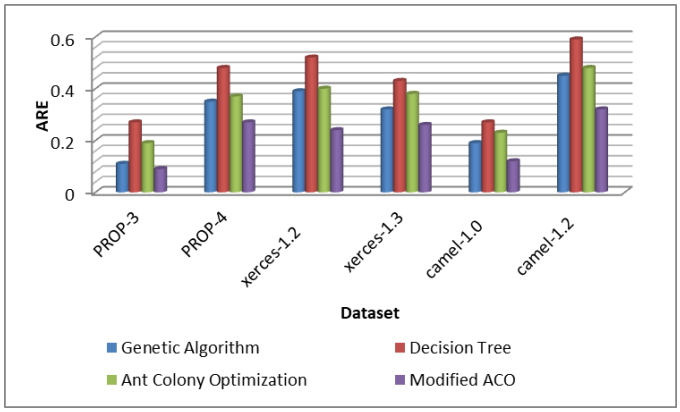


Figure Average Error Rate Analysis for six different Datasets

Table 3 and figure shows the values of Average Error Rate (AER) errors for each of the used dataset. Naturally, the mean absolute error has been widely used in the past to report the errors. However, if the difference between the absolute value and predicted values is large, the relative error rate provides the much useful information. Therefore, this work reported the results in terms of relative error measurement unit. And from the obtained result it is observed that among the existing approaches the proposed modified ACO has produced low error results. For PROP 3 and prop 4 datasets, Modified ACO based fault perdition produced low error rate 0.09 and 0.27 respectively. But for camel 1.2 and Xerces 1.3 error rate is much higher which is 0.32 and 0.26 respectively. These result confirmed the predictive capability of the fault models based on the modified ACO System.

Table 4 Recall Value analysis of existing Genetic Algorithm, Decision Tree, Ant colony Optimization with proposed Modified Ant colony optimization for all six Datasets based

Data set	Genetic Algorithm	Decision Tree	Ant Colony Optimization	Modified ACO
PROP-3	65	58	60	73
PROP-4	46	32	42	58
xerces-1.2	28	20	25	45
xerces-1.3	40	29	36	62
camel-1.0	32	28	30	45
camel-1.2	30	25	28	37

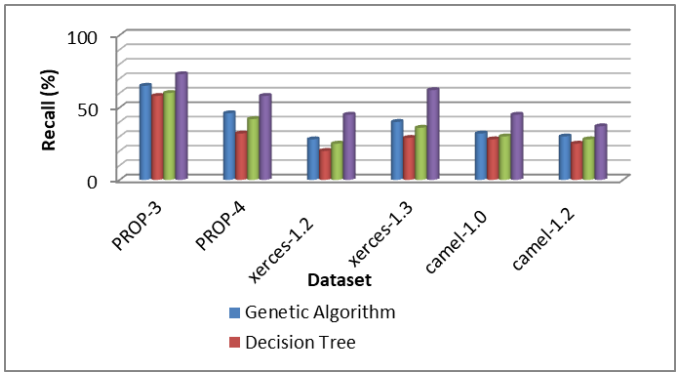


Figure Recall value Analysis of Six Dataset with four different fault prediction models

Table 4 and figure illustrates the obtained recall value for all the four fault prediction models applied on six different datasets. Since proposed fault prediction model assigned an expected number of faults to each module of the given software. It is also observed that lowest recall value is obtained by Xerces 1.2 varies within the range of 28% to 45% approximately. These results also confirmed the proposed model predictive accuracy is higher than the other existing methods.

Table 5 Completeness analysis of Existing Genetic Algorithm, Decision Tree, Ant colony Optimization and proposed model for all six Datasets.

Data set	Genetic Algorithm	Decision Tree	Ant Colony Optimization	Modified Ant Colony Optimization
PROP-3	75	60	68	96
PROP-4	90	78	86	98
xerces-1.2	96	78	85	97
xerces-1.3	86	72	80	99
camel-1.0	92	80	86	100
camel-1.2	89	80	85	99

Figure Completeness Analysis of Six Dataset with four different fault prediction models

Table 5 and figure depicts the completeness values of the four fault models for all the six datasets used in this work. It is perceived that for most of the used datasets, the proposed modified ACO algorithm achieved the completeness close to 100%. The proposed model achieves more completeness than the other existing approaches.

Running Example

Table 6 Example of T-test based Significance Determination among Software Metrics

Attribute	Coefficient	Std Error	Std Coefficient	Tolerance	t-stat	p-Value	Code
wmc	0.179	0.924	0.208	0.967	0.194	0.848	
cbo	13.942	2.849	25.169	0.983	4.894	0.000	****
lcom	-3.162	0.435	-13.766	0.917	-7.263	0.000	****
ca	-13.640	2.749	-48.166	0.983	-4.962	0	****
ce	-3.252	0.513	-3.358	0.993	-6.336	0.000	****
npm	1.602	0.792	2.065	0.961	2.024	0.051	*
loc	-0.714	0.235	-0.951	0.940	-3.039	0.003	***
mfa	0.050	0.035	0.051	0.997	1.400	0.212	
max_cc	1.012	0.213	1.142	0.997	4.743	0.00	****

Table 8: RESULTS AFTER SAMPLE RUN

From the table 6 it is observed t − test on each software metric is applied and compared with their corresponding P -value for each metric as a measure of how effectively it separates the groups. In this Software metrics which are having P - value smaller than 0.05 have strong discrimination powers.

TABLE 7: Sample Modules with software metrics selected to predict the number of faults.

Modules/ Software Metrics	S 1	S 2	S 3	S 4	S5	S 6	S 7	N0.of Faults determined
M1		X	X		X	X	X	5
M2	X		X	X	X	X	X	6
M3		X	X					2
M4			X	X		X		3
M5	X		X	X	X	X	X	6
M6		X	X	X	X	X		5
M7	X	X	X		X		X	5
M8		X			X	X		3

From the table 7 depicts the input to the modified ACO assumes a priori knowledge of the number of faults detected for each module based on the software metrics selected

In this table 8, for each run the best path of all iterations are reported. Also the final weight on the edges and the path found in that run is shown. Though different paths were explored by artificial ants in all he runs, still they could converge to the optimal path.

RUN	Iteration	ANT	Best Path	Weight on edges after all the iterations
1	1	A1	1,5,7,3	1,5 : 0.421441 3,4 : 5.12349 3,7 : 1.121931 4,5 : 5.12349 5,7: 0.421441 Rest all edges have 0 weight. Best Path is found to be : 3,4,5
	2	A8	7,3,4,5	
	3	A5	3,4,5	
	4	A3	3,4,5	
2	1	A5	5,1,3	Weight on edges after all the iterations 1,3 : 6.801621 1,5 : 6.801621 Best Path is found to be : 3,1,5
	2	A3	3,1,5	
	3	A5	5,1,3	
	4	A3	3,1,5	

VII. CONCLUSION AND FUTURE WORK

This paper shows the results of modified ACO based fault prediction model to predict the number of faults in given software. The experimental study was applied over six software fault datasets taken from the tera-PROMISE repository. The modified ACO determines the significance among the software metrics using T-test to choose the optimal metrics for prediction number of faults in the selected project. The results have been evaluated in terms of average error rates, recall and completeness measure. The resulting statistics shows that proposed fault prediction model is able to predict the number of faults with significant accuracy. In future, the work will be extended by applying Different methodologies will also be considered for further detailed investigation.

VIII. REFERENCES

1. Glenford J, Myers, Corey Sandler, The art of software testing, John Wiley and sons, 3rd edition 2011, 240 pages.
2. Zheng, J. Cost-sensitive boosting neural networks for software defect prediction. J. Expert Systems with Applications: An international Journal, ACM digital Library, proceedings of sixth International symposium on Software Metrics, 1999; 242-249.
3. Shatnawi, R. Improving software fault-prediction for imbalanced data. IEEE Proceedings of International Conference on Innovations in Information Technology, 2012; 54-59.
4. M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in of the 18th International Conference On Software Engineering, pp.170- 178. Berlin, Germany, 1996.
5. Alsmadi and Magel, "Open source evolution Analysis," in proceeding of the 22nd IEEE International Conference on Software Maintenance (ICMS'06), Philadelphia, USA, 2006.
6. Runeson and Nyholm, "Detection of duplicate Defect report uses neural network processing", in Proceeding of the 29th international conference on Software engineering 2007.
7. Lovedeep and Varinder Kaur Arti, "Application of Data mining techniques in software engineering" International journal of electrical, electronics and computer system(IJEECS) Volume-2 issue-5, 6. 2014.
8. T.J Ostrand, E.J Weyuker, R.M Bell, Prediction the location and number of faults in large software systems. IEEE Transactions on software engineering 2005, 31(4) p 340-355.
9. W. Afzal, R. Torkar, R.Feldt, Prediction of fault count data using genetic programming, in proceedings of the 12th IEEE International Multitopic conference, 2008, p 23-24.
10. Arvinder Kaur, Inderpreet Kaur, An empirical evaluation of classification algorithms for fault prediction in open source projects Journal of King Saud University – Computer and Information Sciences (2018) 30, 2–17
11. Dhyan Chandra Yadav , Saurabh Pal , Software Bug Detection using Data Mining, International Journal of Computer Applications (0975 – 8887) Volume 115 – No. 15, April 2015
12. Rohit Mahajana, Sunil Kumar Gupta, Rajeev Kumar Bedi, Design Of Software Fault Prediction Model Using BR Technique, International Conference on Information and Communication Technologies (ICICT 2014), Science Direct, Procedia Computer Science 46 (2015) 849 – 858
13. http://openscience.us/repo/defect/
14. http://openscience.us/repo/
15. T.M. Khoshgoftaar ; J.C. Munson ; B.B. Bhattacharya ; G.D. Richardson, Predictive modeling techniques of software quality from software measures, IEEE Transactions on Software Engineering (Volume: 18, Issue: 11, Nov 1992)
16. L Briand, J. Wust, Empirical studies of quality models in object oriented systems. Advances in computers journal, 2002, 56(1), p 97-166
17. M. Dorigo and T. Stutzle. The ant colony optimization metaheuristic: Algorithms, applications, and advances, 2002
18. Blum C. Theoretical and practical aspects of Ant colony optimization. Dissertations in Artificial Intelligence. Berlin: Akademische Verlagsgesellschaft Aka GmbH; 2004

AUTHOR PROFILE

J. M. Dhayashankar, Assistant Professor, Department of BCA, Sri Ramakrishna Mission Vidyalaya College of Arts and Science,

Dr. A. V Ramani, Head and Associate Professor, Department of Computer Science, Sri Ramakrishna Mission Vidyalaya College of Arts and Science.