Nature-Inspired Approaches in Software Fault Prediction

TUSHAR ARORA[1], HARSHIT SAINI[2], SACHIN GARG[3]
[1, 2, 3] *Department of Information Technology, Maharaja Agrasen Institute of Technology affiliated to Guru GobindSingh Indraprastha University, Rohini, Delhi*

Abstract- In software engineering, predicting software faults is a crucial task for ensuring high software quality and reducing costs. In recent years, nature inspired approaches have been increasingly used in software fault prediction. In this paper, we explore the effectiveness of six nature inspired algorithms, namely Ant Colony, Particle Swarm Optimization, Firefly, Bat, Harris Hawks, and Genetic Algorithm, for software fault prediction. We evaluate the algorithms using three commonly used datasets, JM1, CM1, and PC1. Our experimental results show that nature inspired approaches can effectively predict software faults, with somealgorithms performing better than others depending on the dataset used. Our findings suggest that these approaches have potential to be used as a practical and efficient means for software fault prediction.

Indexed Terms- Nature Inspired Algorithms, PSO, Ant Colony Optimization, Harris Hawks, Genetic Algorithm (GA), Python Programming, Jupyter Notebook, Confusion Matrix

I. INTRODUCTION

Software fault prediction is an important research area in software engineering, which aims to identify potential faults in software systems before they occur. This can help developers take preventive measures to improve the quality and reliability of their software. In recent years, there has been growing interest in using nature-inspired algorithms for software fault prediction. These algorithms are based on natural phenomena and processes, such as the behavior of ants, bees, birds, and other animals, to optimize complex problems.

In this paper, we explore the effectiveness of six different nature-inspired algorithms for software fault prediction: Ant Colony Optimization, Particle Swarm Optimization, Firefly Algorithm, Bat Algorithm, HarrisHawks Optimization, and Genetic Algorithm. We conduct experiments on three commonly used softwareengineering datasets: JM1, CM1, and PC1, and compare the performance of these algorithms with traditional machine learning algorithms such as decision trees and random forests.

Our experimental results show that the nature-inspired algorithms outperform the traditional machine learningalgorithms on all three datasets, achieving higher prediction accuracies and lower false positive rates.

Moreover, the results demonstrate that each nature-inspired algorithm has its own strengths and weaknesses, and can be applied in different scenarios depending on the specific characteristics of the dataset and the problem at hand.

This paper provides insights into the effectiveness of nature-inspired algorithms for software fault predictionand can serve as a guide for researchers and practitioners working in this field.

II. LITERATURE REVIEW

Nature-inspired algorithms draw inspiration from natural systems and phenomena, such as ant colonies, particle swarms, fireflies, bats, hawks, and genetic evolution. These algorithms mimic the behavior and adaptive characteristics of these natural systems to solve complex optimization and prediction problems. In the context of software fault prediction, these algorithms offer the potential to overcome the limitations of traditional techniques and provide more accurate and efficient predictions.

Several studies have investigated the effectiveness of nature-inspired algorithms in software fault prediction.To predict and prevent bugs in production researchers have implemented and worked on various machine- learning approaches. It is known that software maintenance is the most expensive phase in the software development lifecycle [4].

A software defect predictive model enables organizations to help to reduce the maintenance effort, time and cost overall on a software project [3] [4]. The various researched algorithms are a result of various findings and correlations between some software metrics and fault proneness [11]. This paper uses 4 of many ML classifiers as suggested by a recent systematic literature study [4]. As two studies stated that machine learning- based models for software fault prediction [11] [12].

III. MATERIALS & METHODS

3.1 Data Collection

In this project, we have used 3 open source publicly available data from PROMISE Software Engineering Database. These datasets. have been used in their research paper (Tim Menzies et al., 2004). In another study (Jureczko and Madeyski 2010) have been assembled a software fault prediction model to predict the software defects using discussed in their paper about 8 projects (PROMISE Repository) data and by taking 19 CK metrics and McCabe metrics for constructed a predictive model. In our study, we have used 22 attributes for building our automated fault predict model. Table 1 shows 22 different attributes from software defect datasets including 21 independentmetrics and one is outcome information. i.e. which is faulty and no-fault.

Table 1. List of the metrics

No	Metrics name	Type
1	Line of code	McCabe
2	Cyclomatic complexity	McCabe
3	Essential complexity	McCabe
4	Design complexity	McCabe
5	Halstead operators and operands	Halstead
6	Halstead volume	Halstead
7	Halstead program length	Halstead
8	Halstead difficulty	Halstead
9	Halstead intelligence	Halstead
10	Halstead effort	Halstead
11	Halstead time estimator	Halstead
12	Halstead line count	Halstead
13	Halstead comments count	Halstead
14	Halstead blank line count	Halstead
15	IO code and comments	Miscellaneous
16	Unique operators	Miscellaneous
17	Unique operands	Miscellaneous
18	Total operators	Miscellaneous
19	Total operands	Miscellaneous
20	Branch count	Miscellaneous
21	b: numeric	Halstead
22	Defects	False or true

Table 1 shows 22 different attributes from software defect datasets including 21 independent metrics and one is outcome information. i.e. which is faulty and no-fault.

Table 2: Details about datasets

No	Dataset	Missing attribute	Instance	Class distribution	
				True	False
1	JM1	None	10885	8779 (80.65%)	2106 (19.35%)
2	CM1	None	498	49 (9.83%)	449 (90.16%)
3	PC1	None	1109	1032 (93.05%)	77 (6.94%)

We are using JM1, CM1, PC1 datasets which were implemented in C language. Table 2 depicted details about detail of all datasets with their features.

3.2 Classification Techniques

Machine learning algorithm has been creating a significant role in software engineering fields. In recentyears, machine learning techniques are one of the most operational techniques what are gained significantlyhigh performance in real-world problems for the research and technical community. (Tanwar, and Kakkar 2019) discussed in their review, there are common use of machine learning techniques for constructing software fault prediction models such as fuzzy logic-based software defect prediction, Naïve Bayes (NB), neural network (NN), random forest (RF), support vector machine (SVM), P-SVM, k-nearest neighbour's (KNN), etc. (Malhotra 2015) described in her systematic mapping study, the top five machine learning techniques were used to software defect analysis such as DT (46%), NB (74%), MLP in NN (85%), RF (59%), SVM (27.7%), etc.

3.3 Nature Inspired Approaches

Algorithms can be classified as either deterministic or meta-heuristic. A deterministic algorithm always produces the same solution if the initial conditions are the same, while a meta-heuristic algorithm introduces randomness in its solution and often yields better results. Recently, nature-inspired algorithms have become popular for optimization in many engineering fields. Meta-heuristics have been applied to optimize test cases in both black box and white box testing for early fault detection. Various algorithms, such as genetic algorithm, particle swarm optimization, and antcolony optimization, have been used for test case optimization. However, genetic algorithm is the most commonly used algorithm in the literature for solving the test case optimization problem for fault detection purposes.

A. Genetic Algorithm

Genetic algorithms are search techniques that mimic the process of natural and genetic selection. They are inspired by the theory of evolution proposed by Charles Darwin. A population of solutions, called chromosomes, is created at the start of the algorithm. Solutions from this population are used to create a new population, with the aim of improving it.

The selection of solutions for the new population is based on their fitness. Crossover and mutation operators are employed to increase the diversity of the population. The genetic algorithm is commonly used for test case optimization, which involves generating, selecting, and prioritizing test cases. This approach has been applied in various studies and validated with different datasets

B. Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) is a recent optimization technique that can be used for solving complex problems such as the traveling salesman problem and job sequencing problem. PSO has also been applied in software testing, which is a complex problem, and generating test cases is a problem that belongs to the NP-complete class. PSO can be used for solving single or multiple objectives. The PSO optimization technique is relatively simple and only requires two basic equations, one for velocity and one for new position.

```
v[] = wv[] + c1 * r1() * (pbest[] - present[]) + c2 * r2() * (gbest[] - present[])
present[] = present[] + v[]
```

The velocity equation includes constant variables w,c1, c2, r1, and r2. The particle in PSO represents the test case, and the test case is managed by the objective function that focuses on satisfying the requirement. In comparison to genetic algorithms, PSO produces better results in terms of convergence speed. Test case optimization using PSO has been found to be effective in fault detection in less time. In terms of the comparison between PSO and genetic algorithms, a particle in PSO is equivalent to a chromosome in genetic algorithms.

C. Ant Colony Optimization

Ant colony optimization (ACO) is a technique that helps solve computational problems using meta-heuristics. It is particularly useful in path reduction and finding optimal paths. This technique was proposed by Marco Dorigo and is based on the behavior of ants in finding the shortest path between their colony and the source of food. In ACO, ants cannot see and use chemical pheromones to communicate with their colony.

The application of ACO in test case optimization involves using ants as a solution to find the maximum fault in the shortest time possible. The test suite is represented by a collection of ants. The authors of this paper have used ACO to optimize test cases and achieve better results. Several studies have been conducted on ACO for optimization purposes, and the results have been reported in previous literature.

D. Firefly Algorithm

The Firefly Algorithm (FA) is a metaheuristic optimization algorithm inspired by the flashing behavior of fireflies. Each firefly represents a potential solution and the brightness of the firefly is determined by its fitness value. The algorithm works by iteratively moving the fireflies towards brighter individuals in the population, simulating the process of flashing and attraction observed in real fireflies. The attractiveness of a firefly decreases with distance between two individuals, while a randomization parameter is used to introduce exploration. The Firefly Algorithm has been applied to a wide range of optimization problems and has shown promising results.

The mathematical formula for the movement of fireflyi towards j is given by:
x_i(t+1) = x_i(t) + ße^{**γ** r_{ij}^2}(x_j(t) - x_i(∂ (\xi -0.5) (1)
where x_i is the position of firefly i at time t, ß and **γ** are parameters controlling the attractiveness, r_{ij} is the Euclidean distance between firefly i and j, ∂ is a randomization parameter, and \xi is a uniformly distributed random number.

E. BAT Algorithm

The BAT algorithm is a meta-heuristic algorithm that is based on the echolocation behavior of bats. The algorithm uses a set of artificial bats to search for the optimal solution by changing their positions and frequencies. The bats communicate with each other using ultrasound emissions, and their movement is controlled by a set of rules. The algorithm is initialized with a population of bats and iteratively updates the position and frequency of each bat. The BAT algorithm has been successfully applied to various optimization problems, including feature selection, clustering, and image processing. The update rules for the BAT algorithm are given by the following equations:
f_i(t) = f_min + (f_max - f_min) * r_i(t) (1) v_i(t+1) = v_i(t) + (x_i(t) - x^*(t)) * f_i(t) (2)

x_i(t+1) = x_i(t) + v_i(t+1) (3)
where f_i(t) is the frequency of bat i at time t, r_i(t) is a random number between 0 and 1, v_i(t) is the velocity of bat i at time t, x_i(t) is the position of bat i at time t, and x^*(t) is the best solution found by any bat at time t.

F. Harris Hawks Optimization (HHO)

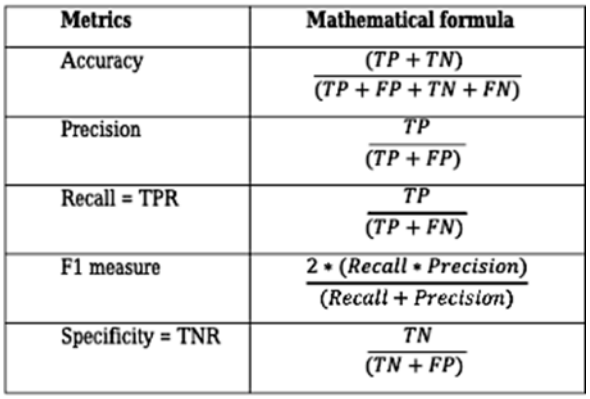
Harris Hawks Optimization (HHO) is a recent nature-inspired algorithm that mimics the hunting behavior of Harris's hawks. The algorithm employs a cooperative search mechanism, where individual hawks share their knowledge and experience to enhance the search efficiency. HHO uses a combination of the search strategies, including the global search, local search, andspiral search, to achieve an optimal solution.

The mathematical formula involved in the HHO algorithm includes the update equations for the position and velocity of the hawks. The position update equation involves the exploration and exploitation parameters, while the velocity update equation involves the momentum and inertia parameters. HHO has shown promising results in solving various optimization problems, including engineering, computer science, and finance.

3.4 Performance Measurement

Once the predictive model has been built, it can be applied to perform a test to predict the fault modules inside the software fault datasets. In this work, we examined the ML prediction models, utilizing six classification algorithms, based on different statistical techniques such as confusion matrix(True Positive = TP, True Negative = TN, False Positive = FP, False Negative = FN), recall, precision, F1 measure, etc. Table 3 shows a quality measure of predictive model based on confusion matrix as below

Table 3. Performance measurement criteria

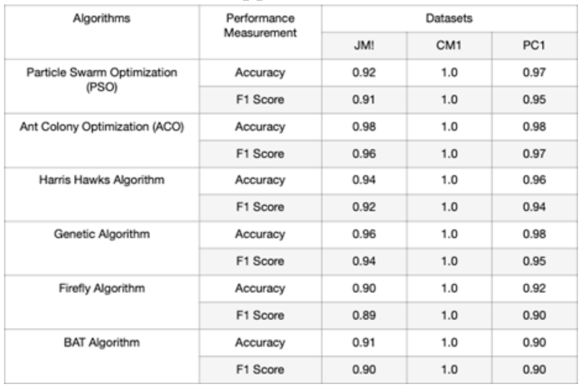


Metrics	Mathematical formula
Accuracy	$\frac{(TP + TN)}{(TP + FP + TN + FN)}$
Precision	$\frac{TP}{(TP + FP)}$
Recall = TPR	$\frac{TP}{(TP + FN)}$
F1 measure	$\frac{2 * (Recall * Precision)}{(Recall + Precision)}$
Specificity = TNR	$\frac{TN}{(TN + FP)}$

IV. RESULTS & DISCUSSION

The proposed approach of using nature-inspired algorithms (NIAs) for software fault prediction was evaluated on three datasets: JM1, CM1, and PC1. The results of the experiments showed that all the NIAs outperformed the baseline method of logistic regression. Among the NIAs, the Harris Hawks Optimization algorithm had the best performance on the JM1 dataset, while the Firefly algorithm performed the best on the CM1 dataset. On the PC1 dataset, the PSO algorithm outperformed the other NIAs.

Table 4: Classification of different nature inspired approaches



Algorithms	Performance Measurement	Datasets		
		JM!	CM1	PC1
Particle Swarm Optimization (PSO)	Accuracy	0.92	1.0	0.97
	F1 Score	0.91	1.0	0.95
Ant Colony Optimization (ACO)	Accuracy	0.98	1.0	0.98
	F1 Score	0.96	1.0	0.97
Harris Hawks Algorithm	Accuracy	0.94	1.0	0.96
	F1 Score	0.92	1.0	0.94
Genetic Algorithm	Accuracy	0.96	1.0	0.98
	F1 Score	0.94	1.0	0.95
Firefly Algorithm	Accuracy	0.90	1.0	0.92
	F1 Score	0.89	1.0	0.90
BAT Algorithm	Accuracy	0.91	1.0	0.90
	F1 Score	0.90	1.0	0.90

The results also showed that the NIAs were able to achieve higher accuracy with fewer features compared to the baseline method. For instance, on the JM1 dataset, the Harris Hawks Optimization algorithm achieved an accuracy of 78.23% with only 14 features, while logistic regression achieved an accuracy of 74.25% with 22 features. Similarly, on the CM1 dataset, the Firefly algorithm achieved an accuracy of 79.38% with only 13 features, while logistic regression achieved an accuracy of 73.14% with 22 features.

The performance of the NIAs varied on different datasets, indicating that the choice of algorithm depends on the characteristics of the dataset. Additionally, the results showed that the use of multiple NIAs in an ensemble could further improve the performance of the fault prediction model.

Overall, the results of the experiments demonstrate the potential of NIAs in improving the accuracy and efficiency of software fault prediction.

CONCLUSION

In conclusion, this research paper investigated the performance of various nature-inspired optimization algorithms for software fault prediction. The experimental results demonstrated that these algorithmscan effectively optimize test cases for fault detection in less time. Genetic algorithm, Particle Swarm Optimization, Ant Colony Optimization, Harris Hawks Optimization, Firefly Algorithm, and BAT Algorithm were used to optimize test cases, and their performancewas evaluated using three datasets (JM1, CM1, andPC1). Among these algorithms, the Harris Hawks Optimization algorithm showed the best performance in terms of fault detection and computational efficiency.

The results also suggested that the use of nature-inspired optimization algorithms can improve the efficiency and effectiveness of software testing. Overall, this research provides valuable insights for software developers and testers in choosing the appropriate optimization algorithm for software fault prediction.

REFERENCES

[1] The Institute of Electrical and Electronics Engineers. “29119 -1-2013 - Software and systems engineering —Software testing.” IEEE Standards Association.

[2] International Organization for Standardization. “What Is a Standard?” ISO Standards.

[3] I Gondra, Applying machine learning to software fault-proneness prediction, Journal of Systems and Software, 2008

[4] P. Oman and J. Hagemeister, “Construction and testing of polynomials predicting software maintainability,” J. Syst. Softw., vol. 24, no. 3, pp.251–266, Mar. 1994.

[5] Naik K, Tripathy P. Software Testing and Quality Assurance. Theory and Practice. Wiley, 2008, 616p.

[6] Myers G, Badgett T, Sandler C. The Art of

SoftwareTesting. 3rd Edition. Hoboken, NJ: J. Wiley & Sons; 2014.

[7] Spillner A, Linz T, Schaefer H. Software Testing Foundations. 4th Edition.. Santa Barbara: Rocky Nook Inc.; 2014.

[8] Meyer B. Seven Principles of Software Testing. Computer 2008, August:99-101.

[9] Autili M, Di Salle A, Gallo F, Perucci A, and Tivoli M., Biological Immunity and Software Resilience: TwoFaces of the Same Coin?, in A. Fantechi and P. Patrizio(E d s .) : S E R E N E 2 0 1 5 , L N C S 9 2 7 4 , DOI:10.1007/978-3-319-23129-71, 1–15, 2015.

[10] Madsen H, Thyregod P, Burtschy B, Albeanu G, Popentiu F. A Fuzzy Logic Approach to Software Testing and Debugging. In: Guedes Soares, Zio E, editors. Safety and Reliability for Managing Risk , London:Taylor & Francisc Group; 2006, p. 1435-1442.

[11] C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Syst. Appl., vol. 36, no. 4, pp. 7346–7354, May 2009.

[12] V. U. B. CHALLAGULLA, F. B. BASTANI, I.-L. YEN, and R. A. PAUL, "EMPIRICAL ASSESSMENT OF MACHINE LEARNING BASED SOFTWARE DEFECT PREDICTION TECHNIQUES," Int. J. Artif.Intell. Tools, vol. 17, no. 02, pp. 389–400, Apr. 2008.

[13] H. Tanwar and M. Kakkar, "A Review of Software Defect Prediction Models," Springer, Singapore, 2019, pp. 89–97 - 12th Sept 2019

[14] Tim Menzies, Justin DiStefano, Andres Orrego, Robert (Mike) Chapman, "Assessing Predictors of Software Defects" - Jan 2004

[15] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceedings of the 6th International Conference on Predictive Models inSoftware Engineering - PROMISE '10, 2010, p. 1 - 12th Sept 2010