Evaluating the Proposed Hybrid Whale Optimization Algorithm - Genetics Algorithm (WOA-GA) Approach for Software Fault Prediction

Joanna Victoria S. Pitao
College of Computing and Information Sciences - Caraga State University - Main Campus

Junrie B. Matias
College of Computing and Information Sciences - Caraga State University - Main Campus

Jayrhom R. Almonteros

https://doi.org/10.5109/7395573

出版情報：Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES). 11, pp.583-589, 2025-10-30. International Exchange and Innovation Conference on Engineering & Sciences
バージョン：

Evaluating the Proposed Hybrid Whale Optimization Algorithm – Genetics Algorithm (WOA-GA) Approach for Software Fault Prediction

Joanna Victoria S. Pitao[1], Junrie B. Matias[2], Jayrhom R. Almonteros[3]
[12]College of Computing and Information Sciences - Caraga State Univertisty-Main Campus, Butuan City, Philippines
jvspitao@carsu.edu.ph, jbmatias@carsu.edu.ph, jralmonteros@carsu.edu.ph

Abstract: *Software Fault Prediction (SFP) is a pivotal field in software engineering aimed at proactively identifying defect-prone software modules. Several studies have explored hybrid optimization approaches for SFP by combining strengths of different meta-heuristic techniques to address class imbalance using benchmark datasets, such as the PROMISE repository. In this study, the researchers proposed a hybrid Whale Optimization Algorithm – Genetic Algorithm (WOA-GA) model to tackle class imbalance across 14 projects from the PROMISE repository. Moreover, the study systematically evaluates the performance of five different models - the proposed WOA-GA approach, classical GA, classical WOA, Boosted WOA, and the SMOTE + AdaSS+GA (SAGA) when paired with various classifiers (RF, DT, and SVM). The performance of these models is assessed using metrics such as ROC AUC and MCC.*

Keywords: Software Fault Prediction, meta-heuristic algorithm, software metrics, PROMISE repository

1. INTRODUCTION

The increasing demands for high-quality software solutions has placed a significant focus on Software Fault Predictions (SFP) as an essential field of research [1], [2], [3], [4] . SFP is a technique used in software engineering that aims to analyze historical data and patterns to identify potential faults or defects within software systems before they occur.

Traditionally, SFP models often utilize well-established statistical and machine learning techniques such55 as Logistic Regression (LR), Decision Tree (DT), and basic ensemble methods, relying on available software metrics like lines of code (loc), complexity measurements (cm), or historical defect data. While these methods offer ease of implementation and easy interpretation, empirical studies show that they often struggle with issues such as complex metric interactions, data irregularities, and contextual nuances, particularly in benchmark datasets like those from the PROMISE repository [5], [6], [7]. Moreover, traditional approaches also face challenges with data dimensionality, class imbalance, and achieving optimal predictive accuracy [8], [9].

The advancement of machine learning techniques and meta-heuristic optimization algorithms, such as conventional machine learning methods like Random Forest (RF), K-nearest Neighbor (KNN), Support Vector Machines (SVM), Logistic Regression (LR), and ensemble learning methods provided opportunities to improve SFP models [10]. However, these methods encounter challenges when dealing with high-dimensional datasets [2], [11], [12]. Several studies have explored hybrid optimization approaches by combining the strengths of different meta-heuristic techniques to address these limitations. Nature-inspired algorithms, particularly the Whale Optimization Algorithm (WOA) [13] and Genetic Algorithm [14], have shown promise for handling data imbalance [15]. However, despite significant advancements in SFP using traditional machine learning and meta-heuristic optimization techniques, the challenges persist. Existing methods often struggle from the ability to manage class imbalance and limited adaptability to complex datasets like those in the PROMISE repository [9]. This study evaluates the performance of the proposed hybrid WOA-GA approach in addressing class imbalance and comparing its performance against established fault prediction approaches.

2. RELATED WORKS

Software Fault Prediction (SFP) focuses on forecasting defective modules before software deployment. It involves three critical components: software fault datasets, fault prediction techniques, and performance evaluation measures [16]. The values for various software metrics like lines of code (loc) and cyclomatic complexity (cc) are extracted from different software fault datasets, serving as independent variables. While the dependent variables in this process are the required fault information concerning fault prediction, such as the number of faults and faulty and non-faulty items. Statistical techniques and machine learning models are utilized to construct fault prediction models. Once the model is trained, the performance of the constructed fault prediction model is assessed using a range of performance evaluation measures such as the ACC, PRE, REC, F1-score, and ROC AUC.

A study by Sharma and Chandra [4] reviewed several prominent research papers from 1990 to 2017. Their study comprehensively surveyed SFP techniques, focusing on how modules are classified as faulty or non-faulty to reduce development costs and time. Based on their observations, they found that most of the studies are based on traditional techniques (statistical methods like LR and Bayesian Network Classifier, and other techniques like DT) with experimentation on public datasets (from different repositories including the PROMISE datasets) and have employed method-level and traditional metrics for performance evaluation. This is because conventional approaches are easy to understand and use. On the other hand, research with computational intelligence methods (e.g., SVM, GA, Swarm Intelligence Methods, and other techniques like Fuzzy Set Theory) are next to the most employed classification algorithms.

Conventional fault prediction methods that incorporated machine learning (ML) algorithms like Random Forest (RF), K-nearest Neighbors (KNN), Support Vector Machines (SVM), Logistic Regression (LR), and

ensemble methods or optimization algorithms like Genetics Algorithm (GA) or Whale Optimization Algorithm (WOA) have shown notable success; still, Hassouneh et al. and Tumar et al. [12], [17] emphasized that these methods encountered challenges when dealing with high-dimensional datasets. Researchers have also explored hybrid approaches by combining the strengths of different optimization algorithms to enhance fault prediction of the SFP models. Malhorta et al. [8] introduced a hybrid model called 'SAGA' which combines Synthetic Minority Over-sampling Technique (SMOTE) [6], [18], Adaptive Splitting and Selection (AdaSS), and GA to tackle class imbalance. Hassouneh et al. [12]introduced five (5) natural-selection schemes to binary WOA that aim to strengthen the exploration and exploitation capabilities of the WOA. However, the study [12] chose feature subsets directly from the imbalanced data, affecting the optimization process of the model.

2.1 The PROMISE Repository

The Predictor Models in Software Engineering (PROMISE) [19] repository is one of the sources of defect prediction datasets available for exploration. It was developed and curated by the PROMISE Software Engineering team at the University of Ottawa, led by Boetticher, Menzies, and Ostrand, aiming to promote reproducibility and facilitate direct model comparisons. Among other repositories like the Australian Enterprise Engineering Management (AEEM), National Aeronautics and Space Admission (NASA), RELink, and Software Laboratory (SOFTLAB) projects, the PROMISE repository consolidates diverse metric categories including Object Oriented (OO) metrics, CK metrics, and McCabe's cyclomatic metrics alongside historical defect labels [4] Each dataset in the PROMISE repository contains roughly 20 to 30 software metrics which include size, complexity, cohesion, and coupling measures that correlate with defect-proneness. The PROMISE repository's standardized structure provides a consistent basis for benchmarking and evaluating fault prediction techniques [9]. In this paper, 14 datasets from the PROMISE repository are utilized where each dataset has twenty (20) features.

Table 1. PROMISE Dataset Description

Project	No. of Instances	KLOC	Bug %
ant-1.7	742	208.653	22.37
camel-1.0	339	33.721	4.13
camel-1.6	965	113.055	51.81
data_ivy-2.0	352	87.769	11.36
jedit-3.2	272	128.883	140.44
jedit-4.2	376	170.683	28.19
log4j-1.1	109	19.938	78.90
lucene-2.0	195	50.596	137.44
poi-2.0	314	93.171	11.78
synapse-1.2	257	53.5	56.42
velocity-1.6	229	57.012	82.97
xalan-2.4	723	225.088	21.58
xerces-1.2	440	159.254	26.14
xerces-1.3	453	167.095	42.60

The total number of instances refers to the number of rows present, Kilo Lines of Code (KLOC), and bug percentage. KLOC is used to estimate defect density by measuring the number of defects per thousand lines of code. Bug percentage is used to estimate the percentage of buggy modules across the 14 projects. This provides insight into whether a class imbalance is present in the project and should be addressed. A bug percentage less than 30% indicates that the project is strongly imbalanced toward the clean class, and approximately 40 – 60% suggests a nearly balanced or mild skew, while a bug percentage higher than 60% indicates imbalanced toward the buggy class [20]. Table 1 clearly shows that from its bug percentage only 3 projects, namely, camel-1.6, synapse-1.2, and xerces-1.3 are nearly balanced or had mild skew.

2.2 The Whale Optimization Algorithm (WOA)

The Whale Optimization Algorithm (WOA) is a nature-inspired metaheuristic algorithm inspired by the hunting behavior of humpback whales. A population-based algorithm that mimics a group of whales moving toward a prey's location, which represents the optimal location, while each whale represents a solution. The paper of Mirjalili and Lewis [13] discussed the unique behavior of humpback whales – bubble net feeding, a characteristic where whales swim in a helical route that is generated by blowing a net of bubbles.

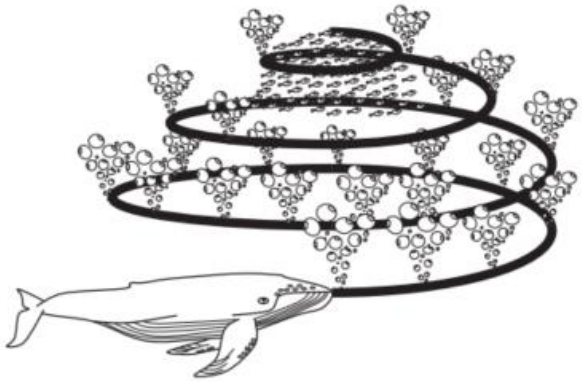

Fig. 1. Bubble net feeding of Humpback Whales [13].

WOA balances the search for new areas of the search space (exploration) and intensifies the search around promising regions (exploitation). This algorithm employs two primary mechanisms to simulate the whales' bubble net feeding – Encircling Prey and Spiral updating position. These mechanisms are presented in Figure 2 illustrating the overall design of the WOA. The boundary check will ensure that all whale positions remain within the valid limits and if it lies outside the search space the algorithm will amend it. Fitness is calculated for each whale (solution) based on the defined objective function, this phase will involve a classifier and a measuring metrics like ROC AUC. In terms of its movement decision logic, the algorithm chooses how a whale updates its position. Following the condition where, if p is greater than or equal to 0.5 it will perform its spiral updating, thus, mimicking the bubble-net spiral attack. Else, if $|A|$ is less than 1, it performs its encircling behavior in which a whale moves closer to the current best. And if $|A|$ *is* greater than or equal to 1, the algorithm will select a random whale, and update its position to maintain exploration.

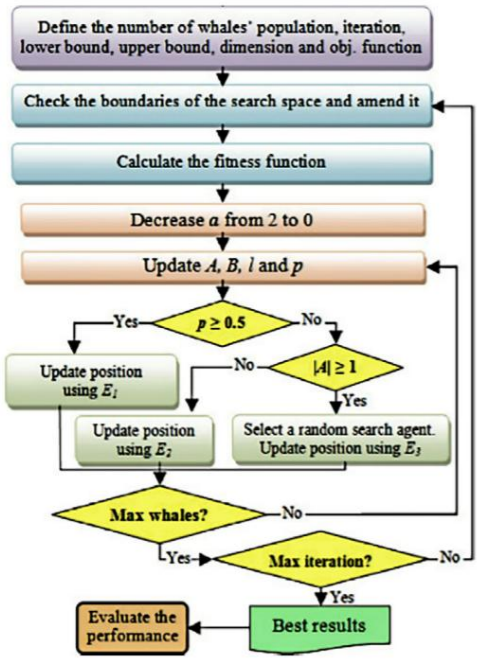

Fig. 2. The overall design of WOA [21]

2.3 The Genetic Algorithm (GA)

GA is another meta-heuristic population-based algorithm inspired by the principles of natural selection and genetics [14], [22]. The algorithm begins by generating an initial set of potential solutions, referred to as chromosomes. These solutions are utilized to form a new set and enhance its quality. Their effectiveness determines the process of choosing solutions for the new set. Techniques such as crossover and mutation are applied to improve the variety within the set.

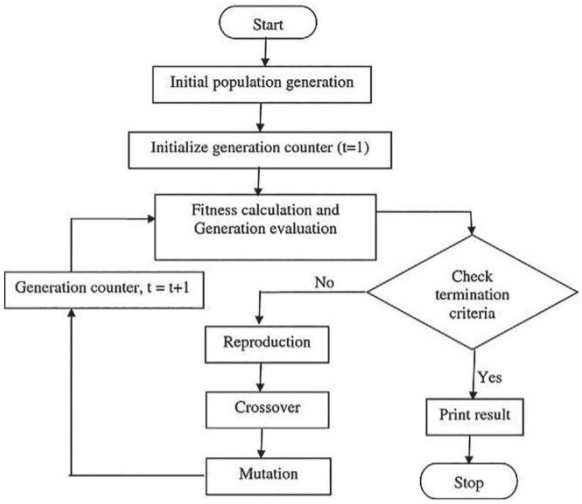

Fig. 3. The overall design of GA [21].

Figure 3 presents the overall design of the algorithm. In the context of SFP, GA offers the potential to overcome the limitations of existing techniques and provide more accurate and efficient predictions. With its crossover and mutation mechanism, GA can ensure diverse solutions across the search space, improve convergence and reduces the chance of getting stuck in local minima. The study of Alsghaier and Akour [21] performed GA with SVM and GA-WOA with SVM on the NASA datasets and obtained the best results when applied to large datasets, indicating the adequacy of the model for SFP.

2.4 Classifiers

The choice of classifiers plays a pivotal role in model complexity, interpretability, and performance. Researchers Ali et al.[2], Shamar and Chandra [4] and Zhang et al. [7] reviewed several SFP models and approaches. They found that traditional methods including DT, remain the most widely adopted in SFP studies due to their transparency, ease of implementation, and ability to work effectively with smaller, well-selected feature sets. Additionally, computationally intelligent approaches, like SVM and ensemble methods like RF, have captured the interest of researchers due to their robustness to noise and imbalanced classes. Further, these classifiers are also used in other fields due to their capabilities and robustness in identifying features from different datasets or samples [23], [24].

3. Methods

SFP often adopts Nature-based metaheuristic algorithms to address intricate optimization problems such as high-dimensional datasets, feature selection and identification, and class imbalance. WOA and GA have become increasingly popular in SFP [15]

Table 2. Comparison between GA and WOA [18]

	Genetics Algorithm	**Whale Algorithm**
Algorithm Type	**Evolutionary**	**Swarm Intelligent**
Main Criterion	Achieve the optimal solutions	Achieve the optimal solutions and worst solutions.
Operation Mechanism	A multi-way information sharing mechanism, where the entire population moves forward to the optimal Area.	One-way sharing mechanism, where only the first randomly selected search agent gives out the information to other search agents.
Optimization mechanism	There is a genetic recombination, fitness selection, and work with discrete or continuous features.	There is no deletion or criterion of individuals, and the input vectors need to be normalized.
Number of operators	Three operators (selection, crossover, and mutation)	Three operators (search prey, encircling-prey, and bubble-net foraging behavior)

Both algorithms have strengths in global search capabilities but sometimes struggle with finding an optimal balance between exploration and exploitation, resulting in the model converging early before exploring the entire search space. To overcome this limitation, a hybrid algorithm that combines the strengths of GA's exploitation with WOA's exploration process is proposed.

Figure 4 shows the proposed WOA-GA hybrid approach integrating the WOA and GA in two-phase sequence. The first phase performs the global exploration of WOA by updating real valued solution vectors into binary feature masks using the sigmoid function and thresholding. Each masks represents a subset of software metrics used to train classifiers and evaluate its predictive performance. Following this, the predefined number of

GA generations are executed. The binary population from WOA undergoes refinement through its tournament selection, single-point crossover, and bit-flip mutation.

Algorithm 1 Proposed WOA-GA Hybrid for Feature Selection

Require: Dataset $D = \{X, y\}$; population size N; max WOA iterations $T_{\max}$; GA generations T_{GA}; sigmoid threshold τ; crossover rate ρ; mutation rate μ

Ensure: Best binary feature subset b^* and fitness score f^*

Initialize population $\{X_i\}_{i=1}^{N}$ with real values
for $i = 1$ to N **do**
  Convert X_i to binary b_i using sigmoid and threshold τ
  Evaluate fitness $f(b_i)$
end for
Set $b^* \leftarrow \mathrm{argmin}_{b_i} f(b_i)$
Set $f^* \leftarrow f(b^*)$
for $t = 1$ to $T_{\max}$ **do**
  $a \leftarrow 2 - 2 \cdot (t/T_{\max})$
  for $i = 1$ to N **do**
    Generate A, C, p, l
    if $p < 0.5$ **then**
      if $|A| < 1$ **then**
        Update position using encircling mechanism (E_2)
      else
        Select random whale and update position using (E_3)
      end if
    else
      Update position using spiral mechanism (E_1)
    end if
    Convert updated X_i to b_i using sigmoid + threshold τ
    Evaluate fitness $f(b_i)$
    if $f(b_i) < f^*$ **then**
      $b^* \leftarrow b_i$
      $f^* \leftarrow f(b_i)$
    end if
  end for
end for
for $g = 1$ to T_{GA} **do**
  Select parent pairs using tournament selection
  Apply single-point crossover with probability ρ
  Apply bit-flip mutation with rate μ per bit
  Evaluate fitness of offspring
  if offspring fitness $< f^*$ **then**
    $b^* \leftarrow$ offspring
    $f^* \leftarrow$ offspring fitness
  end if
end for
return b^*, f^*

Fig. 4. Proposed WOA-GA Hybrid Algorithm

3.1 Classifier Mapping and Hyperparameters

The implementation supports five different classification algorithms. For each classifier, the researcher utilized its default parameter setting as provided by the scikit-learn library. Classifiers and default parameters are presented in Table 3.

Table 3. Classifiers and default parameters

Learner	Default Parameters
Random Forest	n_estimators=100, random_state=42, n_jobs=-1
Support Vector Machine	kernel='rbf', probability=True, random_state=42
Decision Tree	random_state=42
Logistic Regression	solver='liblinear', max_iter=100, random_state=42
K-Nearest Neighbors	n_neighbors=5, weights='uniform'

Table 4 presents the summary of hyperparameters used in this research.

Table 4. Summary of hyperparameters

Parameter	Default	Description
Population size	60	Number of whale solutions in the population
Max iterations	100	Total number of WOA generation steps
Sigmoid threshold	0.5	Threshold used to convert real-valued positions to binary masks
GA crossover rate	0.9	Probability of applying single-point crossover during the GA phase
GA mutation rate	0.1	Probability of flipping each bit in a binary mask during the GA mutation phase

3.2 Results

The implementation supports three different classification algorithms. Table 5, 6, and 7 shows the ROC AUC scores of the WOA-GA in comparison to the existing models - classical GA, classical WOA, SAGA, and BWOA across the fourteen PROMISE datasets.

Table 5. ROC AUC using RF

Project	GA	WOA	WOA-GA	SAGA	BWOA
ant-1.7	**0.8417**	0.8395	0.8350	0.8368	0.8103
camel-1.0	0.8897	0.8667	0.8590	**0.9705**	0.8026
camel-1.6	0.7436	0.7449	0.7004	**0.8237**	0.6292
ivy-2.0	0.8145	0.7976	0.8026	**0.9146**	0.7698
jedit-3.2	**0.8506**	0.8431	0.8026	0.8105	0.8033
jedit-4.2	0.8188	0.8016	0.7453	**0.9063**	0.8055
log4j-1.1	**1.0000**	0.9905	0.8000	0.8442	0.7714
lucene-2.0	**0.8148**	0.7447	0.7712	0.7937	0.7275
poi-2.0	0.8406	0.7449	0.7832	**0.9135**	0.7985
synapse-1.2	**0.8538**	0.8420	0.7639	0.7878	0.7529
velocity-1.6	0.7531	0.7438	0.6646	**0.7812**	0.6229
xalan-2.4	**0.8734**	0.8540	0.7975	0.8676	0.8178
xerces-1.2	**0.8750**	0.8181	0.8335	0.8474	0.8316
xerces-1.3	**0.9332**	0.9281	0.8599	0.8990	0.8952

GA and SAGA achieved the highest ROC AUC scores in most datasets, with GA leading in 6 and SAGA in 4 out of 14 projects. GA showed strong consistency, especially in jedit-3.2, lucene-2.0, and xerces-1.2/1.3, while SAGA excelled in camel-1.0 and poi-2.0. The WOA-GA approach performed moderately but did not outperform GA or SAGA in any case. BWOA had the lowest scores overall. These results suggest GA and SAGA are better suited for ROC-based fault prediction under the RF classifier.

Table 6 showed that the WOA-GA approach showed moderate to low ROC AUC scores and never ranked first. As with RF, BWOA had the weakest performance. While SAGA and GA outperformed other methods across most datasets. SAGA ranked highest in camel-1.0, camel-1.6, jedit-4.2, poi-2.0, and xalan-2.4, while GA led in ant-1.7, jedit-3.2, and xerces-1.3 (0.9545).

Table 6. ROC AUC using SVM

Project	GA	WOA	WOA-GA	SAGA	BWOA
ant-1.7	**0.8516**	0.8406	0.7743	0.8368	0.8216
camel-1.0	0.9487	0.8974	0.5179	**0.9705**	0.4564
camel-1.6	0.7361	0.7095	0.6133	**0.8237**	0.6584
ivy-2.0	0.7798	0.7937	0.5992	**0.9146**	0.7183
jedit-3.2	**0.8649**	0.8408	0.8213	0.8105	0.7763
jedit-4.2	0.8391	0.7766	0.7234	**0.9063**	0.6859
log4j-1.1	0.9714	**0.9905**	0.8190	0.8442	0.7619
lucene-2.0	0.7937	0.7857	0.7646	0.7937	0.6905
poi-2.0	0.8622	0.8431	0.4949	0.9135	0.4005
synapse-1.2	0.8471	0.8647	0.7176	0.7878	0.7580
velocity-1.6	0.7125	0.7083	0.6646	**0.7812**	0.6688
xalan-2.4	0.7982	0.7432	0.7273	**0.8676**	0.7306
xerces-1.2	0.7939	0.8123	0.7003	0.8474	0.6467
xerces-1.3	**0.9545**	0.9063	0.9026	0.8990	0.9212

Table 7. ROC AUC using DT

Project	GA	WOA	WOA-GA	SAGA	BWOA
ant-1.7	0.7968	0.7060	0.7881	**0.8368**	0.7146
camel-1.0	0.8026	0.6513	0.8256	**0.9705**	0.4769
camel-1.6	0.6965	0.6480	0.7025	**0.8237**	0.5683
ivy-2.0	0.8819	0.8194	0.8512	**0.9146**	0.5615
jedit-3.2	**0.8498**	0.8491	**0.8498**	0.8105	0.7680
jedit-4.2	0.7375	0.7609	0.8188	**0.9063**	0.4797
log4j-1.1	0.9952	0.9905	**1.0000**	0.8442	0.6190
lucene-2.0	0.7130	0.7063	**0.8413**	0.7937	0.4934
poi-2.0	0.7143	0.6224	0.8214	**0.9135**	0.5013
synapse-1.2	0.8252	0.6966	**0.8546**	0.7878	0.6807
velocity-1.6	0.7146	0.6813	0.7729	**0.7812**	0.6188
xalan-2.4	0.7629	0.7328	0.8108	**0.8676**	0.6768
xerces-1.2	0.8127	0.8320	**0.8958**	0.8474	0.8774
xerces-1.3	0.8896	0.7922	**0.9314**	0.8990	0.6688

Table 7 still showed that SAGA and GA are still best in performance when paired with DT. Notably, WOA-GA lightly improved in this setting, topping xerces-1.2 (0.8958) and xerces-1.3 (0.9314).

In terms of the models performance, it should be noted that the higher ROC AUC score means that the model is better at distinguishing between faulty and non-faulty modules across varying threshold values. Based on the results, GA and SAGA are the most effective across all classifiers, demonstrating high model separability and generalizability. The WOA-GA approach, while showing stable mid-range results, does not outperform GA or SAGA, highlighting the importance of stochastic strategies in hybrid metaheuristic models for software fault prediction.

To provide a more balanced and comprehensive evaluation of the classifiers' performance, the researchers include the Matthews Correlation Coefficient (MCC) as a key metric. Unlike traditional metrics such as accuracy or F1-score, MCC considers all four components of the confusion matrix—true positives, true negatives, false positives, and false negatives—making it particularly suitable for imbalanced datasets common in software fault prediction.

MCC yields a value between −1 and +1, where:

- +1 indicates perfect prediction,
- 0 implies performance no better than random guessing,
- −1 reflects total disagreement between prediction and actual outcomes.

Table 8. MCC Score using RF

Project	GA	WOA	WOA-GA	SAGA	BWOA
ant-1.7	0.4258	0.4745	0.4260	0.3708	**0.5213**
camel-1.0	-0.0262	-0.0262		-0.0201	
camel-1.6	0.1782	0.1995	0.2717	0.1114	**0.2139**
ivy-2.0	**0.4365**	0.4242	0.2501	0.2981	0.1466
jedit-3.2	**0.5397**	0.4394	**0.5397**	0.5264	0.3559
jedit-4.2	0.3660	0.4125	**0.4299**	0.3477	0.4125
log4j-1.1	0.7939	**0.9037**	0.5610	0.7217	0.5610
lucene-2.0	**0.4908**	0.3413	0.3879	0.3200	0.2735
poi-2.0	0.2699	0.0830	0.1581	**0.4052**	0.3592
synapse-1.2	0.5444	**0.5630**	0.4756	0.4877	0.3164
velocity-1.6	0.2791	0.3105	0.1466	**0.3528**	0.2512
xalan-2.4	0.3188	0.3947	**0.3983**	0.1094	0.3532
xerces-1.2	0.4028	0.3754	**0.5610**	0.3668	0.4462
xerces-1.3	**0.5893**	0.5729	0.4709	0.3455	0.4709

Table 9. MCC Scores using SVM

Project	GA	WOA	WOA-GA	SAGA	BWOA
ant-1.7	**0.4930**	0.4376	0.3737	0.3585	0.3173
camel-1.0	-0.0262	-0.0262	-0.0262	**0.2638**	
camel-1.6	0.0000	0.0780	0.0780	**0.1843**	0.1109
ivy-2.0	0.2085	0.3354	**0.3354**	0.1186	
jedit-3.2	**0.5771**	0.5709	0.4219	0.5673	0.3720
jedit-4.2	0.2961	0.0000	**0.3196**	0.2953	0.2961
log4j-1.1	0.7939	0.7939	0.6901	**0.8000**	0.0129
lucene-2.0	0.3413	0.3413	0.3539	0.3556	0.1197
poi-2.0	0.0000	0.0000		**0.1498**	
synapse-1.2	0.4982	0.4421	0.3881	**0.5468**	0.4397
velocity-1.6	0.2151	0.2791	0.2791	**0.3058**	0.1237
xalan-2.4	**0.1970**	0.0000		0.1215	
xerces-1.2	0.0000	0.0000		-0.0380	
xerces-1.3	**0.5029**	0.3543	0.3543	0.3474	0.4330

Table 8 shows that GA and WOA-GA achieved the highest MCC scores in several datasets, notably in log4j-1.1, xerces-1.3, and xerces-1.2, indicating strong reliability. SAGA also performed well, leading in synapse-1.2. While WOA and BWOA generally underperformed, BWOA topped ant-1.7. Meanwhile, Table 9 shows that GA consistently led in MCC scores, topping datasets like log4j-1.1 (0.7939), jedit-3.2 (0.5771), and xerces-1.3 (0.5029). SAGA also performed well in synapse-1.2 (0.5468) and velocity-1.6. The WOA-GA approach had modest scores, with a peak in ivy-2.0 (0.3354). Results confirm GA's robustness, with SAGA offering competitive adaptability under SVM.

As shown in Table 10, GA outperformed all other methods, ranking highest in 9 out of 14 datasets, including log4j-1.1 (0.8964), xerces-1.3 (0.7058), and jedit-3.2 (0.6590). WOA-GA showed moderate scores, with its best in jedit-4.2 (0.5015), while SAGA and BWOA generally trailed. These results highlight GA's dominance in MCC-based performance when using Decision Trees.

Table 10. MCC Scores using DT

Project	GA	WOA	WOA-GA	SAGA	BWOA
ant-1.7	**0.5813**	0.3925	0.3637	0.2568	0.4164
camel-1.0	0.4381	0.3026	-0.0374	-0.0459	-0.0462
camel-1.6	**0.3650**	0.2816	0.1643	0.1303	0.1473
ivy-2.0	**0.6071**	0.5223	0.1320	0.1903	0.1118
jedit-3.2	0.6590	0.6810	0.5379	0.4044	0.5038
jedit-4.2	0.4146	**0.5015**	0.2551	0.3833	-0.0406
log4j-1.1	0.8964	0.7905	0.4980	0.8018	0.2256
lucene-2.0	0.3879	0.3879	-0.0088	0.1094	-0.0495
poi-2.0	0.3430	0.1685	0.4568	0.1775	0.0169
synapse-1.2	**0.6504**	0.3546	0.2873	0.4340	0.3455
velocity-1.6	**0.4540**	0.3683	0.2512	0.3528	0.2512
xalan-2.4	**0.5361**	0.3904	0.2489	0.1283	0.3172
xerces-1.2	0.4167	0.4609	0.4316	0.3462	0.4932
xerces-1.3	**0.7058**	0.5092	0.5303	0.3561	0.3482

4. Conclusion

The results indicate that GA handles class imbalance effectively, achieving high MCC and ROC AUC scores across imbalanced datasets, which reflects its ability to maintain balanced predictions. SAGA, with its adaptive nature, also demonstrated strong performance on imbalanced data, particularly in synapse-1.2 and velocity-1.6. The WOA-GA approach showed moderate success, performing better than WOA and BWOA but lacking the consistency of GA and SAGA. Overall, GA and SAGA are more robust in managing class imbalance, while WOA-based approaches require further adaptation to improve balance-sensitive outcomes.

Based on the ROC AUC and MCC results, it is evident that while the WOA-GA approach shows moderate performance, it consistently trails behind GA and SAGA—especially in handling class imbalance. This underscores the limitation of the WOA-GA in effectively handling imbalance datasets which can hinder its fault prediction performance. To improve its effectiveness, the proposed WOA-GA model can be paired with techniques such as SMOTE or AdaSS to address imbalanced data. Moreover, researchers have also explored the concept of adaptive approaches [25] where models adjust their behavior or parameters based on data characteristics during training and prediction processes.

5. REFERENCES

[1] M. R. Ahmed, M. F. Bin Zamal, M. A. Ali, F. M. J. M. Shamrat, and N. Ahmed, "The impact of software fault prediction in real-world application: An automated approach for software engineering," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jan. 2020, pp. 247–251. doi: 10.1145/3379247.3379278.

[2] M. Ali *et al.*, "Analysis of Feature Selection Methods in Software Defect Prediction Models," *IEEE Access*, vol. 11, pp. 145954–145974, 2023, doi: 10.1109/ACCESS.2023.3343249.

[3] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering." [Online]. Available: http://www.informatik.uni-trier.de/~ley/db/

[4] D. Sharma and P. Chandra, "Towards recent developments in the methods, metrics and datasets of software fault prediction," *International Journal of Computational Systems Engineering*, vol. 6, no. 1, p. 14, 2020, doi: 10.1504/IJCSYSE.2020.109110.

[5] M. Meiliana, S. Karim, H. L. H. S. Warnars, F. L. Gaol, E. Abdurachman, and B. Soewito, "Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset," in *2017 IEEE International Conference on Cybernetics and Computational Intelligence, CyberneticsCOM 2017 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 19–23. doi: 10.1109/CYBERNETICSCOM.2017.8311708.

[6] T. Tahir *et al.*, "Early Software Defects Density Prediction: Training the International Software Benchmarking Cross Projects Data Using Supervised Learning," *IEEE Access*, vol. 11, pp. 141965–141986, 2023, doi: 10.1109/ACCESS.2023.3339994.

[7] W. H. Zhang, R. Y. He, L. J. Wu, Y. Jian, and X. Y. Han, "Comparison of software defect prediction models based on machine learning," *IOP Conf Ser Mater Sci Eng*, vol. 1043, no. 3, p. 032074, Jan. 2021, doi: 10.1088/1757-899X/1043/3/032074.

[8] R. Malhotra, R. Kapoor, P. Saxena, and P. Sharma, "SAGA: A Hybrid Technique to handle Imbalance Data in Software Defect Prediction," in *ISCAIE 2021 - IEEE 11th Symposium on Computer Applications and Industrial Electronics*, Institute of Electrical and Electronics Engineers Inc., Apr. 2021, pp. 331–336. doi: 10.1109/ISCAIE51753.2021.9431842.

[9] K. Bhandari, K. Kumar, and A. L. Sangal, "Data quality issues in software fault prediction: a systematic literature review," *Artif Intell Rev*, vol. 56, no. 8, pp. 7839–7908, Aug. 2023, doi: 10.1007/s10462-022-10371-6.

[10] A. Kumar, M. Khatoon, Mt. Student, and A. Professor, "Issue 4 www.jetir.org (ISSN-2349-5162)," 2020. [Online]. Available: www.jetir.org

[11] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?," *Software Quality Journal*, vol. 26, no. 2, pp. 525–552, Jun. 2018, doi: 10.1007/s11219-016-9353-3.

[12] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, "Boosted Whale Optimization Algorithm with Natural Selection Operators for Software Fault Prediction," *IEEE Access*, vol. 9, pp. 14239–14258, 2021, doi: 10.1109/ACCESS.2021.3052149.

[13] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, May 2016, doi: 10.1016/j.advengsoft.2016.01.008.

[14] D. E. Goldberg, "Genetic Algorithm in Search, Optimization and Machine Learning," *Addison. W esley Publishing Company, R eading, MA, 1(98), 9.*, 1989.

[15] H. Abubakar, K. Umar, R. Auwal, K. Muhammad, and L. Yusuf, "Developing a Machine Learning-Based Software Fault Prediction Model Using the Improved Whale Optimization Algorithm," MDPI AG, Mar. 2024, p. 334. doi: 10.3390/asec2023-16307.

[16] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artif Intell Rev*, vol. 51, no. 2, pp. 255–327, Feb. 2019, doi: 10.1007/s10462-017-9563-5.

[17] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, "Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction," *IEEE Access*, vol. 8, pp. 8041–8055, 2020, doi: 10.1109/ACCESS.2020.2964321.

[18] S. Yin, Q. Shi, Y. Wang, and C. Chen, "Summary of software reliability Research," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing Ltd, Feb. 2021. doi: 10.1088/1757-899X/1043/5/052039.

[19] L. Cheikhi and A. Abran, "Promise and ISBSG Software Engineering Data Repositories: A Survey," in *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, IEEE, Oct. 2013, pp. 17–24. doi: 10.1109/IWSM-Mensura.2013.13.

[20] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview," in *Data Mining and Knowledge Discovery Handbook*, Boston, MA: Springer US, 2009, pp. 875–886. doi: 10.1007/978-0-387-09823-4_45.

[21] H. Alsghaier and M. Akour, "Software fault prediction using Whale algorithm with genetics algorithm," *Softw Pract Exp*, vol. 51, no. 5, pp. 1121–1146, May 2021, doi: 10.1002/spe.2941.

[22] D. I. Merino, E. N. Reyes, and C. Steidley, "Genetic Algorithms: Theory and Application."

[23] M. Salem, N. Tsurusaki, A. Eissa, and T. Osman, "Detection of Slums from Very High-Resolution Satellite Images Using Machine Learning Algorithms: A Case Study of Fustat Area in Cairo, Egypt," *Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES)*, vol. 6, pp. 219–224, Oct. 2020, doi: 10.5109/4102491.

[24] Mst Alema Khatun, Taskin Noor Turna, B. Roy, and Md. E. Hossain, "Performance Analysis of Different Classifiers Used In Detecting Benign And Malignant Cells of Breast Cancer," *Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES)*, vol. 6, pp. 243–248, Oct. 2020, doi: 10.5109/4102498.

[25] H. Lu and B. Cukic, "An adaptive approach with active learning in software fault prediction," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, New York, NY, USA: ACM, Sep. 2012, pp. 79–88. doi: 10.1145/2365324.2365335.