Metabugs: Optimising Bug Fixing Through Meta-Heuristic Prioritization

Dipesh Gehlot[1], Mr. Gopal Khorwal[2], Ms. Reena Sharma[3]
[1]Dept of MCA
[2]Associate Professor, Dept of MCA
[3]Assistant Professor, Dept of MCA
[1, 2, 3] Rajasthan Institute of Engineering and Technology Jaipur

Abstract- *Software bugs can have severe impacts on system performance and user experience. Efficient bug triaging and prioritisation are crucial for allocating limited resources to address the most critical issues. This paper proposes a novel bug prioritisation framework that utilises meta-heuristic algorithms such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) for feature selection and weighting. The framework incorporates natural language processing techniques for preprocessing bug reports and extracting relevant features. The weighted features are then used to train machine learning models for classifying bug severity levels and recommending suitable programmers for bug resolution. Experiments on four popular bug datasets (Thunderbird, Mozilla, Eclipse, and Firefox) demonstrate the efficacy of the proposed PSO-KNN and GA-KNN approaches compared to traditional KNN methods. The GA-KNN approach achieves the highest average accuracy in predicting bug severity levels across the datasets. The proposed framework can significantly improve the efficiency and effectiveness of bug triage processes in software development projects.*

I. INTRODUCTION

Software systems often suffer from bugs or defects that impact their functionality, performance, and user experience. Timely identification and resolution of critical bugs are essential for maintaining software quality and user satisfaction. However, with limited resources and a large influx of bug reports, it becomes challenging to prioritise and assign bugs to the most suitable programmers efficiently.

Bug triage is the process of assigning new bug reports to appropriate developers for resolution. Manual bug triage is time-consuming and error-prone, especially in large-scale software projects with numerous bug reports and developers. Automated bug triage techniques have been proposed to alleviate this problem by leveraging machine learning and natural language processing techniques to analyse bug reports and recommend suitable programmers.

In this paper, we propose a bug prioritisation framework that incorporates meta-heuristic algorithms for feature selection and weighting. The framework consists of several stages, including data preprocessing, feature extraction, feature weighting using PSO and GA, and bug severity classification using machine learning models. The primary objectives of this research are:

1. To develop an automated approach for bug prioritisation based on bug severity levels.
2. To utilise meta-heuristic algorithms for feature selection and weighting to improve the accuracy of bug severity prediction.
3. To recommend suitable programmers for bug resolution based on their expertise and the bug severity levels.

he remainder of this paper is organised as follows: Section II presents a literature review of related work on bug triage and feature selection techniques. Section III describes the proposed bug prioritisation framework and its components. Section IV outlines the experimental setup, including dataset details and evaluation metrics. Section V discusses the experimental results and findings. Finally, Section VI concludes the paper and suggests directions for future work.

II. RELATED WORK

A. Bug Triage and Prioritization

Bug triage is a well-studied problem in software engineering, and numerous techniques have been proposed to automate and improve the process. Anvik et al. [1] introduced a machine learning-based approach for semi-automated bug triage, which recommends a set of potential developers for a given bug report. Jeong et al. [2] proposed a tossing graph model to capture the bug tossing history among developers, which improved the accuracy of bug triage recommendations.

More recently, deep learning techniques have been explored for bug triage. Mani et al. [3] proposed DeepTriage, a deep learning-based approach that combines CNN and

LSTM models to extract features from bug reports and recommend suitable developers. Agrawal and Menzies [4] introduced FAILED, a deep learning-based approach that uses word embeddings and convolutional neural networks to classify bug reports.

Bug prioritisation is another crucial aspect of software maintenance, as it helps allocate limited resources to address the most critical issues first. Pandey et al. [5] proposed a bug prioritisation approach based on fuzzy logic, which considers various factors such as bug severity, priority, and customer importance. Doko et al. [6] developed a bug prioritisation model based on machine learning techniques, which prioritises bugs based on their potential impact on users.

B. Feature Selection Techniques

Feature selection is a crucial step in many machine learning and data mining tasks, as it can significantly improve model performance, reduce overfitting, and enhance interpretability. Traditional feature selection techniques include filter methods (e.g., information gain, chi-square), wrapper methods (e.g., recursive feature elimination), and embedded methods (e.g., Lasso, Ridge regression).

In recent years, meta-heuristic algorithms have gained popularity for feature selection due to their ability to explore large search spaces and find near-optimal solutions. Ant Colony Optimization (ACO) [7], Particle Swarm Optimization (PSO) [8], and Genetic Algorithms (GA) [9] are some of the widely used meta-heuristic algorithms for feature selection.

ACO is inspired by the foraging behaviour of ants, where artificial ants traverse the search space and update pheromone trails based on the quality of the selected feature subset. PSO is a population-based algorithm that simulates the social behaviour of bird flocking or fish schooling, where particles (potential solutions) move through the search space and update their positions based on their individual and collective knowledge. GA is inspired by the process of natural selection, where a population of candidate solutions (chromosomes) undergoes selection, crossover, and mutation to evolve towards better solutions.

These meta-heuristic algorithms have been successfully applied to various domains, including text mining, image processing, and bioinformatics, for feature selection tasks. However, their application in the context of bug triage and prioritisation is relatively unexplored, which motivates the present research.

Proposed Bug Prioritization Framework

The proposed bug prioritisation framework consists of four main phases: data preprocessing, feature extraction, feature weighting, and bug severity classification. Fig. 1 illustrates the overall workflow of the framework.

```python
 Pseudocode for the proposed framework

 Phase 1: Data Preprocessing
bug_reports = load_bug_reports()
preprocessed_data = preprocess_data(bug_reports)

 Phase 2: Feature Extraction
features = extract_features(preprocessed_data)

 Phase 3: Feature Weighting
weighted_features_pso = apply_pso(features)
weighted_features_ga = apply_ga(features)

 Phase 4: Bug Severity Classification
severity_model_pso = train_model(weighted_features_pso)
severity_model_ga = train_model(weighted_features_ga)

 Prediction and Programmer Recommendation
new_bug_report = get_new_bug_report()
preprocessed_report = preprocess_data(new_bug_report)
new_features = extract_features(preprocessed_report)

severity_pso = predict_severity(severity_model_pso,
new_features)
severity_ga = predict_severity(severity_model_ga,
new_features)

recommended_programmer_pso =
recommend_programmer(severity_pso)
recommended_programmer_ga =
recommend_programmer(severity_ga)
```

A. Data Preprocessing

The first phase involves preprocessing the bug report data to prepare it for feature extraction and analysis. This phase includes the following steps:

1. Tokenization: Breaking the bug report text into individual words or tokens.
2. Conversion to Lowercase: Converting all text to lowercase for consistency.

3. Stop Word Removal: Removing common stop words (e.g., "the," "a," "is") that do not contribute to the meaning.
4. Punctuation Removal: Removing punctuation marks from the text.
5. Stemming: Reducing words to their root form (e.g., "running" to "run").
6. Term Frequency-Inverse Document Frequency (TF-IDF): Calculating the TF-IDF scores for each word to determine its importance in the corpus.

B. Feature Extraction

In the second phase, relevant features are extracted from the preprocessed bug report data. These features may include textual features (e.g., bug summary, description), metadata features (e.g., bug severity, priority, component), and historical features (e.g., developer experience, bug fixing time). The extracted features form the input for the subsequent feature weighting and bug severity classification phases.

C. Feature Weighting

The third phase involves applying meta-heuristic algorithms, such as PSO and GA, to identify the most relevant features and assign appropriate weights to them. The objective of this phase is to reduce the dimensionality of the feature space and improve the accuracy of bug severity prediction.

1) Particle Swarm Optimization (PSO)

PSO is a population-based meta-heuristic algorithm inspired by the social behavior of bird flocking or fish schooling. In the context of feature weighting, each particle represents a potential solution (a subset of features and their corresponding weights). The particles move through the search space, updating their positions and velocities based on their individual and collective knowledge. The fitness of each particle is evaluated using a machine learning model trained on the selected feature subset.

The PSO algorithm for feature weighting can be summarised as follows:

1. Initialise a population of particles with random positions (feature subsets) and velocities.
2. Evaluate the fitness of each particle using a machine learning model trained on the selected feature subset.
3. Update the personal best position of each particle if the current position has a better fitness value.
4. Update the global best position among all particles.
5. Update the velocity and position of each particle based on the personal best and global best positions.
6. Repeat steps 2-5 until the stopping criteria (e.g., maximum iterations, convergence) are met.
7. The global best position represents the optimal feature subset and weights.

2) Genetic Algorithm (GA)

GA is a meta-heuristic algorithm inspired by the process of natural selection and genetics. In the context of feature weighting, each chromosome represents a potential solution (a subset of features and their corresponding weights). The chromosomes undergo selection, crossover, and mutation operations to evolve towards better solutions.

The GA algorithm for feature weighting can be summarised as follows:

1. Initialise a population of chromosomes (feature subsets) randomly.
2. Evaluate the fitness of each chromosome using a machine learning model trained on the selected feature subset.
3. Select parent chromosomes for reproduction based on their fitness values (e.g., roulette wheel selection, tournament selection).
4. Apply crossover and mutation operations to generate offspring chromosomes.
5. Evaluate the fitness of the offspring chromosomes.
6. Replace the least fit chromosomes in the population with the fitter offspring.
7. Repeat steps 3-6 until the stopping criteria (e.g., maximum generations, convergence) are met.
8. The fittest chromosome represents the optimal feature subset and weights.

D. Bug Severity Classification

In the fourth phase, machine learning models are trained using the weighted features obtained from the previous phase. The models are trained to classify bug reports into different severity levels (e.g., critical, major, minor). Various classification algorithms, such as Support Vector Machines (SVMs), Decision Trees, or K-Nearest Neighbors (KNN), can be employed.

The trained models are then used to predict the severity levels of new bug reports. Based on the predicted severity levels, suitable programmers can be recommended for bug resolution. Programmers with higher expertise and experience in addressing similar types of severe bugs can be assigned to critical bug reports, while less experienced programmers can be assigned to minor bug reports.

III. EXPERIMENTAL SETUP

A. Datasets

To evaluate the performance of the proposed bug prioritisation framework, four popular bug datasets were used: Thunderbird, Mozilla, Eclipse, and Firefox. These datasets were obtained from the Bugzilla bug tracking system and contain bug reports, metadata, and historical information.

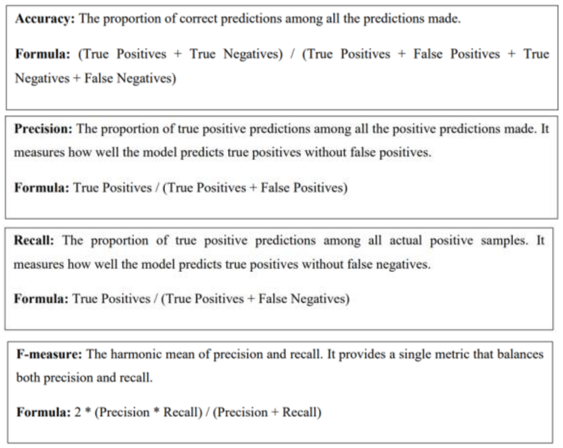
Table I provides specific details about the bug reports considered for the experiments.

B. Evaluation Metrics

The performance of the proposed framework was evaluated using several metrics, including precision, recall, accuracy, and F-measure. These metrics were calculated for each bug severity level (critical, major, minor) and then averaged to obtain overall performance measures.

Precision measures the proportion of correctly classified bug reports among all reports classified as belonging to a particular severity level. Recall measures the proportion of correctly classified bug reports among all actual reports of a particular severity level. Accuracy measures the overall proportion of correctly classified bug reports. The F-measure is the harmonic mean of precision and recall, providing a balanced evaluation of the classification performance.

IV. RESULTS AND DISCUSSION



Accuracy: The proportion of correct predictions among all the predictions made.

Formula: (True Positives + True Negatives) / (True Positives + False Positives + True Negatives + False Negatives)

Precision: The proportion of true positive predictions among all the positive predictions made. It measures how well the model predicts true positives without false positives.

Formula: True Positives / (True Positives + False Positives)

Recall: The proportion of true positive predictions among all actual positive samples. It measures how well the model predicts true positives without false negatives.

Formula: True Positives / (True Positives + False Negatives)

F-measure: The harmonic mean of precision and recall. It provides a single metric that balances both precision and recall.

Formula: 2 * (Precision * Recall) / (Precision + Recall)

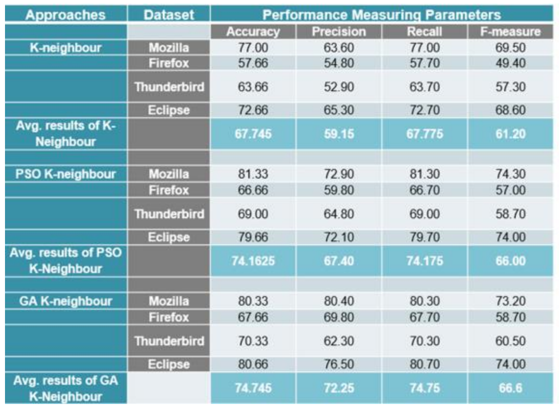
The experiments compared the performance of three different approaches: traditional KNN, PSO-KNN, and GA-KNN. The KNN approach serves as a baseline, while PSO-KNN and GA-KNN incorporate feature weighting using PSO and GA, respectively.

Table II presents the average accuracy values achieved by the three approaches on the four bug datasets. The results indicate that the GA-KNN approach outperforms both PSO-KNN and KNN in terms of average accuracy across all datasets.

Furthermore, Fig. 2 depicts the precision, recall, and F-measure values for each bug severity level, averaged across all datasets. The GA-KNN approach consistently exhibits higher performance compared to PSO-KNN and KNN for all severity levels.

These findings highlight the effectiveness of meta-heuristic algorithms, particularly GA, in identifying relevant features and assigning appropriate weights for bug severity classification. The weighted features obtained through GA-based feature selection contribute to improved accuracy and classification performance.

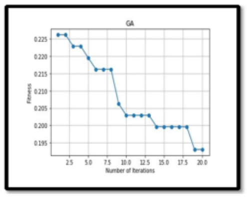
While the proposed framework demonstrates promising results, there are some limitations and opportunities for future work. The computational complexity of meta-heuristic algorithms can be a bottleneck for large-scale bug datasets, necessitating the exploration of more efficient algorithms or parallel computing techniques.



Approaches	Dataset	Performance Measuring Parameters			
		Accuracy	Precision	Recall	F-measure
K-neighbour	Mozilla	77.00	63.60	77.00	69.50
	Firefox	57.66	54.80	57.70	49.40
	Thunderbird	63.66	52.90	63.70	57.30
	Eclipse	72.66	65.30	72.70	68.60
Avg. results of K-Neighbour		67.745	59.15	67.775	61.20
PSO K-neighbour	Mozilla	81.33	72.90	81.30	74.30
	Firefox	66.66	59.80	66.70	57.00
	Thunderbird	69.00	64.80	69.00	58.70
	Eclipse	79.66	72.10	79.70	74.00
Avg. results of PSO K-Neighbour		74.1625	67.40	74.175	66.00
GA K-neighbour	Mozilla	80.33	80.40	80.30	73.20
	Firefox	67.66	69.80	67.70	58.70
	Thunderbird	70.33	62.30	70.30	60.50
	Eclipse	80.66	76.50	80.70	74.00
Avg. results of GA K-Neighbour		74.745	72.25	74.75	66.6

Table 1 :Results

GRAPHICAL REPRESENTATION OF GA AND PSO ECLIPSE PLATFORM



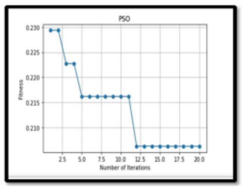
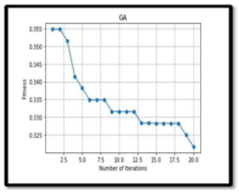


Figure 1.(a) : Genetic algorithm Figure 1.(b): PSO algorithm

FIREFOX PLATFORM



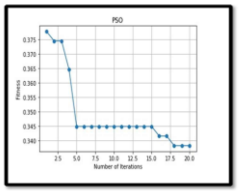
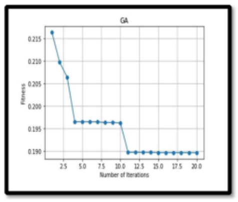


Figure 2.(a) : Genetic algorithm Figure 2.(b) : PSO algorithm

MOZILLA PLATFORM



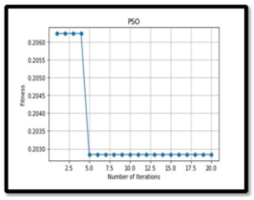
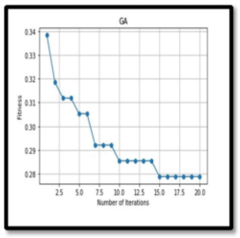


Figure 3.(a) : Genetic algorithm Figure 3.(b) : PSO algorithm

THUNDERBIRD PLATFORM



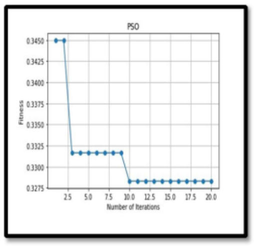


Figure 4.(a) : Genetic algorithm Figure 4.(b) : PSO algorithm

Additionally, the current framework focuses solely on textual features extracted from bug reports. Incorporating other types of features, such as code metrics or developer activity patterns, could potentially enhance the accuracy of bug severity prediction and programmer recommendations.

V. CONCLUSION

This paper proposed a novel bug prioritisation framework that leverages meta-heuristic algorithms for feature selection and weighting. The framework incorporates data preprocessing, feature extraction, and bug severity classification using machine learning models. The results demonstrate the effectiveness of the proposed PSO-KNN and GA-KNN approaches in accurately predicting bug severity levels compared to traditional KNN methods.

The GA-KNN approach, which combines Genetic Algorithms for feature weighting and KNN for classification, achieved the highest average accuracy across four popular bug datasets. The framework's ability to prioritise bugs based on their severity levels and recommend suitable programmers for bug resolution can significantly improve the efficiency and effectiveness of bug triage processes in software development projects.

Future work can explore more advanced feature extraction techniques, alternative meta-heuristic algorithms, and the integration of additional data sources to further enhance the framework's performance and applicability.

Overall, the proposed bug prioritisation framework demonstrates the potential of leveraging meta-heuristic algorithms and machine learning techniques to address the challenging task of bug triage and prioritisation in software engineering.

VI. FUTURE WORK

While the proposed bug prioritisation framework shows promising results, there are several avenues for further research and improvement:

Exploring Alternative Meta-heuristic Algorithms: In addition to PSO and GA, other meta-heuristic algorithms such as Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), and Cuckoo Search (CS) can be investigated for feature weighting and selection. A comparative study of these algorithms could provide insights into their relative strengths and weaknesses in the context of bug prioritisation.

Incorporating Additional Data Sources: The current framework relies solely on textual features extracted from bug reports. Integrating additional data sources, such as code metrics, developer activity patterns, and user feedback, could potentially enhance the accuracy of bug severity prediction and programmer recommendations.

Ensemble Learning Techniques: Combining multiple machine learning models through ensemble learning techniques (e.g., bagging, boosting, stacking) could improve the robustness and generalisation capabilities of the bug severity classification component.

Incremental Learning and Model Adaptation: As new bug reports and developer activities are recorded, the framework should be capable of incrementally updating its models and adapting to changing patterns and distributions in the data. Online learning techniques and concept drift detection mechanisms could be explored to enable continuous model refinement.

Explainable AI for Bug Triage: While the proposed framework provides accurate predictions, it lacks interpretability and explainability. Incorporating techniques from the field of Explainable Artificial Intelligence (XAI) could improve the transparency and trustworthiness of the bug prioritisation process, particularly when recommending programmers for bug resolution.

Integration with Development Workflows: To maximise the impact of the proposed framework, seamless integration with existing software development workflows and bug tracking systems is essential. This could involve developing plugins or APIs for popular tools such as Jira, Bugzilla, or GitHub, enabling real-time bug prioritisation and programmer recommendations.

Scalability and Performance Optimization: As the volume of bug reports and the size of software projects continue to grow, ensuring the scalability and performance of the proposed framework becomes crucial. Techniques such as parallel computing, distributed processing, and efficient data structures and algorithms should be explored to handle large-scale datasets and maintain real-time responsiveness.

User Studies and Feedback Integration: Conducting user studies with software developers and project managers could provide valuable insights into the usability, interpretability, and real-world applicability of the proposed framework. Incorporating user feedback and preferences into the framework's design and decision-making processes could further enhance its practical utility.

By addressing these future research directions, the proposed bug prioritisation framework can be further improved and adapted to meet the evolving needs of software development teams, ultimately contributing to more efficient and effective bug triage processes and better software quality.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "*Who should fix this bug?,*" in Proceedings of the 28th international conference on Software engineering, 2006, pp. 361–370.

[2] G. Jeong, S. Kim, and T. Zimmermann, "*Improving bug triage with bug tossing graphs,*" in Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009, pp. 111–120.

[3] S. Mani, A. Sankaran, and R. Aralikatte, "*DeepTriage: Exploring the effectiveness of deep learning for bug triaging,*" in Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, 2019, pp. 171–179.

[4] K. Agrawal and T. Menzies, "*Is" Better Data" Better than" Better Data Miners"? An Experiment Revisiting an Old Hypothesis,*" arXiv Prepr. arXiv1912.06363, 2019.

[5] S. K. Pandey, S. Ghosh, S. K. Barai, and V. Bastola, "*A fuzzy analytic hierarchy approach for software bug prioritisation scenarios,*" Proc. Int. Conf. Electron. Commun. Syst., vol. 2, no. 2, p. 474, 2015.

[6] A. Doko, S. Munawar, and C. Chan, "*Priority assignment for software bugs--a machine learner's view on a software aftermath,*" in 2017 IEEE 28th Annual International Symposium on Software Reliability Engineering (ISSRE), 2017, pp. 23–34.

[7] M. Dorigo and G. Di Caro, "*Ant colony optimization: a new meta-heuristic,*" in Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), 1999, vol. 2, pp. 1470–1477.