An Empirical Study on Optimal Solutions Selection Strategies for Effort-Aware Just-in-Time Software Defect Prediction

Xingguang Yang*†, Huiqun Yu*‡✉, Guisheng Fan*✉, Kang Yang*
*Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China
†Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China
‡Shanghai Engineering Research Center of Smart Energy, Shanghai, China

***Abstract*—Just-in-time software defect prediction (JIT-SDP) is an active topic in the filed of software engineering, and many methods have been proposed to solve this problem. State-of-the-art method MULTI applies multi-objective optimization algorithm to the effort-aware JIT-SDP problem, and obtains good average performance. Although the average performance of the MULTI method is high, there are many optimal solutions with poor performance. If an optimal solution is randomly selected, a poor prediction model may be obtained. In order to further improve the performance of the MULTI method, we propose three optimal solutions selection strategies: benefit priority (BP), cost priority (CP), and a compromise between cost and benefit (CCB). In order to compare and validate the effectiveness of the strategies, we conduct a large-scale empirical study on data sets of six open source projects. The experimental results show that, compared with the average performance of MULTI, the optimal solutions selection strategy based on BP has a significant improvement in ACC and P_{opt} indicators. Therefore, we recommend using the BP-based optimal solutions selection strategy to improve the performance of MULTI when using the MULTI method to solve the effort-aware JIT-SDP problem.**

***Index Terms*—software defect prediction, empirical software engineering, multi-objective optimization algorithm, just-in-time, effort-aware**

I. Introduction

Software defect prediction [1] [2] is an active research topic in the domain of software engineering. As software scales increase, it becomes very difficult to release a high-quality software system. In the case of limited software testing resources, it is critical for the enterprise to find and fix defects in the software as early as possible.

Software defect prediction is an effective method. By using defect prediction models, developers can accurately estimate whether a program module is likely to have defects, thereby allocating more test resources to modules that are more likely to have defects, which help to improve the quality of the software [3].

Traditional software defect predictions have shortcomings in practical applications [4]. First, because the prediction granularity is coarse (i.e., class, file, or package), it is time consuming to locate risky code regions. Second, a module is usually completed by many developers, so it is difficult to find the suitable developer to code checking for the defect-prone module. Finally, because the defect prediction is made in the later stages of software development, it is difficult for developers to come up with thoughts for software development.

Just-in-time software defect prediction (JIT-SDP) can well overcome the above deficiencies. JIT-SDP is made at change-level rather than module-level. Once the developer submits the modified code, the defect prediction will be executed. Developers can quickly know whether the change is defect-inducing or not.

Nowadays, many supervised and unsupervised methods have been proposed to solve the JIT-SDP problem. Yang et al. [5] compared the performance of supervised learning methods and unsupervised learning methods in the context of effort-aware JIT-SDP through a large-scale empirical study. The experimental results show that some unsupervised learning methods are even better than supervised learning methods, which contradicts people's experience, because supervised learning methods can learn knowledge from data sets and should have better performance than unsupervised learning methods.

Therefore, in order to improve the performance of supervised learning methods, Chen et al. [6] proposed a new method called MULTI, which formalized the effort-aware JIT-SDP problem into a multi-objective optimization problem. The optimal solution set is obtained by using the classical multi-objective optimization algorithm (MOA) NSGA-II. The experimental results show that the average performance of the MULTI method is better than the state-of-the-art supervised learning methods and unsupervised learning methods. However, Chen et al. [6] used the median of the performance of the optimal solution set to represent the average performance of the MULTI method. In the actual use of the MULTI method, if an optimal solution is randomly selected, a poor performance prediction model may be obtained. Therefore, in order to improve the stability and performance of the MULTI

Corresponding Authors: Huiqun Yu (yhq@ecust.edu.cn), Guisheng Fan (gsfan@ecust.edu.cn) DOI reference number: 10.18293/SEKE2019-174

method, we design three optimal solutions selection strategies: benefit priority (BP), cost priority (CP), and a compromise between cost and benefit (CCB). Empirical results show that the optimal solutions selection strategy based on BP can significantly improve the performance of MULTI. Therefore, we recommend using the BP strategy to obtain a higher performance prediction model when using the MULTI method to solve the effort-aware JIT-SDP problem.

The contributions of this paper are summarized as follows:

- In view of the shortcomings of the MULTI method, we propose three optimal solutions selection strategies BP, CP, and CCB, aiming to improve the effort-aware prediction performance of the MULTI method.
- Through large-scale empirical research, we compare the performance of three optimal solutions selection strategies and the average performance of MULTI on the data sets of six open source projects. Experimental results demonstrate that BP-based optimal solutions selection strategy can significantly improve the performance of the MULTI method in ACC and P_{opt} indicators.

The rest of this article is as follows. Section II introduces related work of software defect prediction. Section III introduces the principle of MULTI and our proposed optimal solutions selection strategies. Case study is introduced in section IV. Experimental results and discussion are presented in section V. Section VI describes the threats to validity. The conclusion and future work are presented in section VII.

II. Related Work

A. Just-in-time Software Defect Prediction

Mockus and Weiss [7] firstly applied JIT-SDP to 5ESS updates by designing a series of change metrics. Experimental results show that JIT-SDP can be effectively used in many commercial software projects. Kamei et al. [4] conducted a large-scale empirical study of 11 projects, including 6 open source projects and 5 commercial projects, and they shared data sets of 6 open source projects. Empirical results reveal that the model based on logistic regression can achieve 68% accuracy and an average recall rate of 64%.

Subsequently, many methods were proposed to improve the performance of the JIT-SDP models. Yang et al. [8] proposed a novel method called TLEL, which leverages decision tree and ensemble learning. Experimental results indicate that their method can significantly improve the performance of JIT-SDP. McIntosh et al. [3] explored the impact of data validity on the performance of the JIT-SDP models. Empirical results demonstrate that in order to ensure the performance of the prediction models, the defect data sets used should be within the last three months.

B. Effort-aware Defect Prediction

When building a defect prediction model, in addition to considering the precision, recall, accuracy, etc., it is also necessary to consider the effort required to code checking for defect-prone models. Kamei et al. [4] applied effort-aware defect prediction to JIT-SDT. They proposed a new method EALR, which can identify 35% buggy changes while 20% effort is used. Yang et al. [5] compared unsupervised models with supervised models for effort-aware JIT-SDP. Experimental results show that some unsupervised models outperform state-of-the-art supervised models for effort-aware JIT-SDP. Later, Fu and Menzies [9] repeated the experiment of Yang et al. [5] and proved that although supervised models are better than unsupervised models in project-by-project-based verification, supervised models are not superior to unsupervised models in general.

III. Optimal Solutions Selection Strategies

In order to improve the prediction performance of supervised methods in effort-aware JIT-SDP, Chen et al. [6] proposed a novel method MULTI, which applies multi-objective optimization algorithms (MOAs) to JIT-SDP. Their experimental results show that the average performance of MULTI is significantly better than the 43 state-of-the-art methods including 31 supervised methods and 12 unsupervised methods.

However, we find that the Pareto optimal set derived from MULTI has many optimal solutions with poor performance, which affects the performance of the MULTI method. Therefore, we propose three optimal solutions selection strategies designed to improve the performance of the MULTI method.

A. MULTI

Inspired by search based software engineering (SBSE) [10], Chen et al. [6] first formalized the effort-aware JIT-SDP problem into a multi-objective optimization problem. SBSE aims to solve complex problems with large-scale search space in software engineering by using search technology.

1) The Design of Objectives: MULTI uses logistic regression for defect prediction, which is widely used in previous studies [4] [11]. Assuming that a change $c =< m_1, m_2..., m_n >$ has n metrics, the prediction process of the logistic regression models can be denoted by the formula 1, where $w =< w_0, ..., w_n >$ is the coefficient vector of the logistic regression model. The output value of $y(c)$ represents the probability that a change c is buggy.

$$y(c) = \frac{1}{1 + e^{-(w_0 + w_1 m_1 + ... + w_n m_n)}} \tag{1}$$

Since JIT-SDP is a bi-classification problem, we convert the output value of $y(c)$. The rule is as in formula 2. When the y value is greater than 0.5, the change c is classified as buggy, otherwise it is classified as clean.

$$Y(c) = \begin{cases} 1 & if y(c) > 0.5 \\ 0 & if y(c) \leq 0.5 \end{cases} \tag{2}$$

For the effort-aware JIT-SDP problem, MULTI mainly considers two optimization objectives based on cost-benefit analysis [6]. The first objective is designed from the perspective of benefit and to identify as many buggy changes as possible. For a set of changes C, the benefit of a model can be calculated by formula 3, which represents the number of buggy changes identified by the model.

$$benefit(C) = \sum_{c_i \epsilon C} Y(c_i) \times buggy(c_i) \quad (3)$$

The return value of the function $buggy(c_i)$ indicates whether the change c_i is defect-inducing. When the change is buggy, the return value is 1 otherwise it returns 0.

The second objective is designed from the cost of the model and is to minimize the effort used for code checking. Once a change is predicted to be buggy by the defect prediction model, the software quality assurance (SQA) team will invest a lot of effort in code checking and test case design. For a set of changes C, the cost value can be calculated by formula 4. Here the function $SQA(c_i)$ represents the effort required to code checking for the change c_i. According to the suggestion of Kamei et al. [4], the value of $SQA(c_i)$ can be set to the lines of code(LOC) modified by the change c_i.

$$cost(C) = \sum_{c_i \epsilon C} Y(c_i) \times SQA(c_i) \quad (4)$$

2) The Generation of Optimal Solutions: Obviously, the above two objectives are usually conflicting. If a model wants to identify more buggy changes, it will lead to more effort. Conversely, if the model wants to reduce the effort for code checking, it will miss many buggy changes. MULTI is designed based on NSGA-II [12], which is one of classical MOAs. Before introducing the coding scheme of chromosomes of MULTI, we give some definitions of MOAs.

- **Pareto dominance**. Suppose w_i and w_j are two feasible solutions of the JIT-SDP problem. If and only if $benefit(w_i) > benefit(w_j)$ and $cost(w_i) \leqslant cost(w_j)$ or $benefit(w_i) \geqslant benefit(w_j)$ and $cost(w_i) < cost(w_j)$, w_i is Pareto dominance on w_j.
- **Pareto optimal solution**. For a feasible solution w, w is a Pareto optimal solution if and only if there is no feasible solution w^* which is dominance on w. Pareto optimal set is composed by all the Pareto optimal solutions.

The process of MULTI can be summarized into the following four steps.

1) **Population initialization**. For the effort-aware JIT-SDP problem, a chromosome can be encoded as a coefficient vector, denoting the coefficients of a logistic regression model. During the population initialization process, N chromosomes are randomly generated, and the values of the elements of each chromosome are randomly generated.
2) **Evolution**. After population initialization, classical evolutionary operations are performed to generate new chromosomes. General evolutionary operators include crossover operator, mutation operator, etc.
3) **Selection**. Choose the optimal chromosomes from the parent and offspring populations. The selection operator of NSGA-II is based on non-dominated sorting algorithm and the concept of crowding distance [12]. Repeat steps 2 and 3.
4) **Termination**. Once the evolutionary process satisfies the termination condition, the iteration process terminates and returns to the final optimal chromosomes.

B. Optimal Solutions Selection Strategies

MULTI designs two optimization objectives and generates a set of optimal solutions based on NSGA-II. In previous studies, Chen et al. [6] use MULTI-B to indicate the best performance of MULTI, and use MULTI-M to indicate the average performance of MULTI. Although the average performance of the MULTI method is good in the context of effort-aware JIT-SDP, randomly selecting an optimal solution from the optimal solution set may result in poor prediction performance.

Fig. 1 shows the values of ACC indicator generated from a run results on six subject systems based on 10 times 10-fold cross-validation. It can be seen from the Fig. 1 that although the median of indicator ACC in optimal solution set can obtain a high performance, there are a large number of poor performance solutions in the optimal solution set. Therefore, randomly selecting an optimal solution may result in obtaining a prediction model with poor performance. Therefore, it is necessary to design a suitable optimal solutions selection strategy to improve the stability of MULTI.

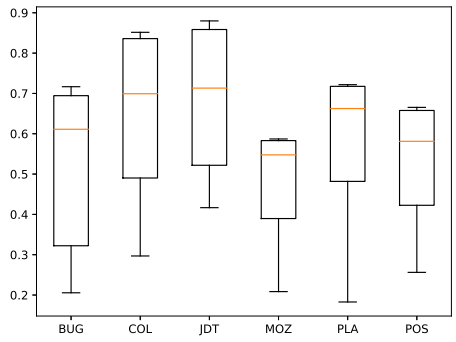

Fig. 1. values of ACC indicator for six subject systems

Since MULTI considers two optimization objectives, we design three optimal solutions selection strategies, as follows:

- Benefit priority (BP). BP strategy returns the optimal solution with maximum benefits in the optimal solution set.
- Cost priority (CP). CP strategy returns the optimal solution with minimal cost in the optimal solution set.
- A compromise between cost and benefit (CCB). CCB strategy considers cost and benefit simultaneously, and returns the optimal solution with middle cost or benefits in the optimal solution set.

To better describe the three strategies, we use the pseudo code to further describe them, as shown in Algorithm 1.

IV. Case Study

Our case study aims to solve following research question.

Algorithm 1: Three optimal solutions selection strategies

Input: training set: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2)..., (\mathbf{x}_n, y_n)\}$;
Output: optimal solutions:
$solution\_BP, solution\_CP, solution\_CCB$

begin
  // Generate optimal solution based on MULTI method
  $solutions = MULTI(D)$
  // Sort optimal solutions in ascending order based on benefit values
  $solutions = order\_by\_benefit(solutions)$
  // Return solutions according to three strategies
  $solution\_BP = solutions[solutions.length - 1]$
  $solution\_CP = solutions[0]$
  $solution\_CCB = solutions[(solutions.length - 1)/2]$
end

- Which optimal solutions selection strategy is appropriate for solving the effort-aware JIT-SDP problem?

The experimental hardware environment is *Intel(R) Core(TM) i7-7700 CPU@ 3.60GHz; RAM 8.00GB*. The experimental code is written in python.

This section introduces data sets, performance indicators, data analysis method, and experimental design.

A. Data Sets

The experiment considers the data sets of six open source projects shared by Kamei et al. [4], which have been widely used in previous studies [5] [6]. These six data sets include Bugzilla (BUG), Columba (COL), Eclipse JDT (JDT), Eclipse Platform (PLA), Mozilla (MOZ), and PostgreSQL (POS), and come from different domains with different scales. The basic information is shown in the Table I.

In order to better solve the JIT-SDP problem, 14 change metrics are designed for these data sets, as shown in Table II. These metrics can be divided into five dimensions: diffusion, size, purpose, history, and experience. More details can be found in reference [4].

TABLE I
THE BASIC INFORMATION OF DATA SETS

Project	Period	#defective changes	#changes	%defect rate
BUG	1998/08/26∼2006/12/16	1696	4620	36.71%
COL	2002/11/25∼2006/07/27	1361	4455	30.55%
JDT	2001/05/02∼2007/12/31	5089	35386	14.38%
MOZ	2000/01/01∼2006/12/17	5149	98275	5.24%
PLA	2001/05/02∼2007/12/31	9452	64250	14.71%
POS	1996/07/09∼2010/05/15	5119	20431	25.06%

B. Performance Indicators

Effort-aware JIT-SDP primarily considers effort for code inspection of defect-prone changes. According to the suggestion of Kamei et al. [4], the effort used to check changes can be obtained by calculating their code churn (i.e., the total number of lines added and deleted by the change). Similar to previous studies [4] [5] [6], our experiment use ACC and P_{opt} to evaluate the effort-aware prediction performance for prediction models. ACC indicates the recall of bug changes when using 20% of effort. P_{opt} is the normalized version of effort-aware performance indicator proposed by Mende and Koschke [13]. Specific information on these two performance indicators can be found in the literature [4] [5].

TABLE II
THE DESCRIPTION OF METRICS

Dimension	Metric	Description
Diffusion	NS	Number of modified subsystems
	ND	Number of modified directories
	NF	Number of modified files
	Entropy	Distribution of modified code across each file
Size	LA	Lines of code added
	LD	Lines of code deleted
	LT	Lines of code in a file before the change
Purpose	FIX	Whether or not the change is a defect fix
History	NDEV	Number of developers that changed the files
	AGE	Average time interval between the last and the current change
	NUC	Number of unique last changes to the files
Experience	EXP	Developer experience
	REXP	Recent developer experience
	SEXP	Developer experience on a subsystem

C. Data Analysis Method

The experiment uses 10 times 10-fold cross-validation technique to evaluate the performance of prediction models. 10 times 10-fold cross-validation technique is performed within the same project. First, the data set of one project is randomly divided into 10 sets of the same scale, nine of which are used to train the model and produce the optimal solution set, and the other one is to test the performance of the optimal solution set. This step will be repeated 10 times. In our case study, the optimal solutions selection strategies only select one solution from optimal solution set. Therefore, 10 times 10-fold cross-validation can ultimately return 100 solutions for each strategy.

In order to test the degree of performance difference between different optimal solutions selection strategies, the experiment uses Wilcoxon signed-rank test and Cliff's δ to further analyze the experimental results. Wilcoxon signed-rank test is a commonly used non-parametric statistical hypothesis test method. In particular, we use corresponding p-values to exam whether two optimal solutions selection strategies have significant difference at the significance level of 0.05. Meanwhile, we use Cliff's δ to determine the magnitude of difference in practical application [14]. Traditionally, the magnitude of the difference is considered trivial ($|\delta| < 0.147$), small ($0.147 \leq |\delta| < 0.33$), moderate ($0.33 \leq |\delta| < 0.474$), or large ($|\delta| \geq 0.474$).

D. Experimental Design

- **Data preprocessing**. In order to obtain a better prediction model, according to the suggestion of Kamei et al. [4], we preprocess the experimental data sets as following steps.

1) ND and REXP metrics are excluded since NF and ND, REXP and EXP are highly correlated. LA and LD metrics are normalized by dividing by LT metric since LA and LD are highly correlated. LT and NUC metrics are also normalized by dividing NF since LT and NUC have highly correlation with NF.
2) Each metric (except FIX) is executed with logarithmic transformation, since these metrics are highly skewed.
3) Due to class imbalance in the defect data sets, we perform random undersampling and keep the number of clean changes same as the number of buggy changes by deleting clean changes randomly.

Our optimal solutions selection strategies are based on the MULTI method, which uses NSGA-II, so some parameters need to be set.

- **Parameter settings**.

1) Population size: 200.
2) The range of coefficient vector: [-10000, 10000].
3) The interval of population initialization: [-10, 10].
4) Crossover operation: simulated binary crossover.
5) Mutation operation: polynomial mutation.
6) The number of generations: 800.

V. Experimental Results and Discussion

A. Experimental Results

The experiment uses ACC and P_{opt} to evaluate the performance of different optimal solutions selection strategies. The results are shown in the Table III and Table IV. In Table III and Table IV, the first column indicates the names of projects. The second column MULTI-M is the median value of all optimal solutions, representing the average performance of the MULTI method. The third to fifth columns represent the performance of the models based on strategies BP, CP, and CCB, respectively. Since the experiment uses 10 times 10-fold cross validation technology, each strategy will get 100 result values. All the values in the Table III and Table IV reflect the median of the 100 result values.

As we can see from Table III and Table IV, the performance of strategy BP is far superior to the performance of MULTI-M in the ACC and P_{opt} on six data sets. In addition, compared to MULTI-M, the performance of strategy CP is even worse, and the performance of strategy CCB is equivalent. Finally, we use Wilcoxon signed-rank test and Cliff's δ determine the significant difference between different optimal solutions selection strategies and MULTI-M. If and only if p-value is less than 0.5 and Cliff's δ is greater than or equal to 0.147, the performance of our strategy is significantly different from MULTI-M, otherwise the difference is negligible. We have bolded the values that have significant differences.

Conclusion. Compared with MULTI-M, the performance of strategy BP is significantly better than MULTI-M, the average performance can be increased by 12% on indicator ACC, and the average performance on indicator P_{opt} can be increased by 15%. In addition, the performance of strategy CCB is comparable to MULTI-M, and the performance of strategy CP is worse than MULTI-M. Therefore, we advise to use BP-based optimal solutions selection strategy to improve the performance of the MULTI method in the effort-aware JIT-SDP problem.

TABLE III
Comparison of three strategies and MULTI-M using ACC

project	MULTI-M	BP	CP	CCB
BUG	0.633	**0.774**	**0.251**	0.636
COL	0.723	**0.804**	**0.344**	0.721
JDT	0.658	**0.723**	**0.371**	0.653
MOZ	0.577	**0.606**	**0.213**	0.579
PLA	0.682	**0.746**	**0.179**	0.683
POS	0.612	**0.703**	**0.252**	0.613
Average	0.648	**0.726**	**0.268**	0.648

TABLE IV
Comparison of three strategies and MULTI-M using P_{opt}

project	MULTI-M	BP	CP	CCB
BUG	0.771	**0.930**	**0.492**	0.773
COL	0.842	**0.936**	**0.547**	0.841
JDT	0.764	**0.880**	**0.557**	0.767
MOZ	0.731	**0.812**	**0.473**	0.731
PLA	0.792	**0.887**	**0.528**	0.790
POS	0.747	**0.900**	**0.441**	0.749
Average	0.775	0.891	0.506	0.775

B. Discussion

The experimental results show that the performance of our optimal solutions selection strategies have the following characteristics.

$$BP > CCB > CP$$

In order to explain this phenomenon, we further analyze the experimental results. Take data sets BUG as an example, the experiment uses 10 times 10-fold cross validation to perform model evaluation, so a total of 100 runs will be produced. The results of one run of the experiment is shown in the Fig.2 and Fig.3. As we can see from Fig.2 and Fig.3, each graph contains 200 red dots, each representing an optimal solution. In the Fig.2 and Fig.3, the horizontal axis represents the value of the benefit, and the vertical axis represents the value of the performance indicators (i.e., ACC and P_{opt}).

It is obvious that in an optimal solution set, the benefit values of optimal solutions are positively correlated with the performance indicators including ACC and P_{opt}. Our experimental results show that this feature is also present on five other data sets. Therefore, BP-based optimal solutions selection strategy can significantly improve the performance of MULTI.

VI. Threats to Validity

External validity. Although the data sets used in the experiment are widely used in the field of JIT-SDP [6] [5] [4], we still cannot guarantee that the findings of the experiment will apply to all other defect data sets. Therefore, more data sets in different fields have yet to be mined and shared to verify the generalization of experimental conclusions.

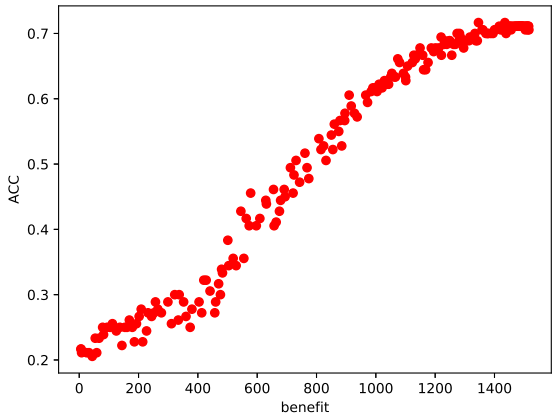

Fig. 2. ACC values in an optimal solution set

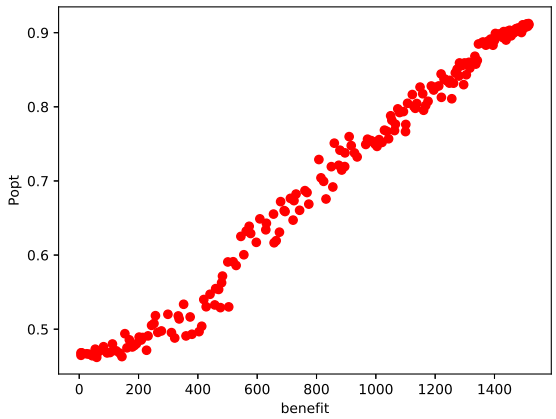

Fig. 3. P_{opt} values in an optimal solution set

Construct validity. Threats to construct validity are mainly considered whether the evaluation indicators can accurately reflect effort-aware prediction performance of models. Our experiment uses ACC and P_{opt} to evaluate effort-aware prediction performance of models, which are widely used in previous JIT-SDP studies [6] [5] [4].

Internal validity. The threats to internal valitidy are mainly about the accuracy of experimental code. Previous research code is mainly written in R language [5] [4], while our experimental code is written in python. In order to reduce the errors in the code, we check all the code and use the mature python libraries.

VII. Conclusions and Future Work

In this paper, in order to improve the performance of the MULTI method, we propose three optimal solutions selection strategies BP, CP, and CCB. In order to verify the effectiveness of the strategies, we conduct a large-scale empirical study on data sets of six open source projects. Experimental results show that compared with the average performance of MULTI, BP-based optimal solutions selection strategy can effectively improve the performance of MULTI.

In the future, we hope to further expand our work. First, since the experiment only uses data sets from open source projects, we will collect data sets from commercial projects to further verify the generalization of experimental conclusions. Secondly, we only consider the cross-validation scenario when evaluating the performance of prediction models. In the future, we will extend the work to cross-project-validation and timewise-cross-validation scenarios to further verify the generalization of the experimental conclusions.

Acknowledgment

This work is partially supported by the NSF of China under grants No.61772200 and 61702334, Shanghai Pujiang Talent Program under grants No. 17PJ1401900. Shanghai Municipal Natural Science Foundation under Grants No. 17ZR1406900 and 17ZR1429700. Educational Research Fund of ECUST under Grant No. ZH1726108. The Collaborative Innovation Foundation of Shanghai Institute of Technology under Grants No. XTCX2016-20.

References

[1] Z. Li, X. Jing, X. Zhu, Progress on approaches to software defect prediction, IET Software 12 (3) (2018) 161–175.
[2] X. Chen, Q. Gu, W. Liu, S. Liu, C. Ni, Survey of static software defect prediction, Journal of Software 27 (1).
[3] S. McIntosh, Y. Kamei, Are fix-inducing changes a moving target?: a longitudinal case study of just-in-time defect prediction, in: Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, 2018, p. 560.
[4] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, N. Ubayashi, A large-scale empirical study of just-in-time quality assurance, IEEE Transactions on Software Engineering 39 (6) (2013) 757–773.
[5] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, H. Leung, Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, 2016, pp. 157–168.
[6] X. Chen, Y. Zhao, Q. Wang, Z. Yuan, MULTI: multi-objective effort-aware just-in-time software defect prediction, Information & Software Technology 93 (2018) 1–13.
[7] A. Mockus, D. M. Weiss, Predicting risk of software changes, Bell Labs Technical Journal 5 (2) (2000) 169–180.
[8] X. Yang, D. Lo, X. Xia, J. Sun, TLEL: A two-layer ensemble learning approach for just-in-time defect prediction, Information & Software Technology 87 (2017) 206–220.
[9] W. Fu, T. Menzies, Revisiting unsupervised learning for defect prediction, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, 2017, pp. 72–83.
[10] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Computing Surveys 45 (1) (2012) 11:1–11:61.
[11] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, IEEE Transactions on Software Engineering 43 (1) (2017) 1–18.
[12] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.
[13] T. Mende, R. Koschke, Effort-aware defect prediction models, in: 14th European Conference on Software Maintenance and Reengineering, CSMR, 2010, pp. 107–116.
[14] E. Arisholm, L. C. Briand, E. B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, Journal of Systems and Software 83 (1) (2010) 2–17.