Dang and Wang
Journal of Engineering and Applied Science (2024) 71:189
https://doi.org/10.1186/s44147-024-00529-0

Journal of Engineering and Applied Science

REVIEWS Open Access

Leveraging meta-heuristic algorithms for effective software fault prediction: a comprehensive study

Zhizheng Dang[1] and Hui Wang[1*]

*Correspondence:
bingyuexiaxing@163.com

[1] Hebei Chemical & Pharmaceutical College, Shijiazhuang 050026, China

Abstract

In large-scale software development, the increasing complexity of software products poses a daunting challenge to maintaining software quality. Given this challenge, software fault prediction (SFP) is a critical endeavor for effective budgeting and refinement of the testing process. Quantitative insights into software quality gained through measurements are crucial in enabling accurate SFP. With the proliferation of software in various fields, ensuring software reliability throughout the software life cycle has become paramount. Anticipating software bugs, which have the potential to reduce software maintenance costs dramatically, is a key approach to improving software reliability. In this regard, using nature-inspired metaheuristic algorithms is promising because of their ability to predict future conditions and identify software anomalies. This study examines the potential of various meta-heuristic algorithms, particularly particle swarm optimization, genetic, ant colony optimization, cuckoo search, lion optimization, firefly, moth-flame, whale optimization, and artificial bee colony algorithms, in addressing the SFP challenge. The study outlines the challenging problems, compares approaches based on fundamental variables, and offers suggestions for future studies, providing a comprehensive and systematic analysis of these algorithms in the context of SFP.

Keyword: Software defect; Software reliability; Software quality; Meta-heuristic algorithms

Introduction

Context

Recent rapid growth in cloud computing [1], Internet of Things (IoT) [2], smart grids [3], and machine learning [4] has driven an explosion of data in almost every aspect of computer science and engineering. Testing software applications for quality has become increasingly important as the quality of software applications has grown in recent years [5]. A significant step towards changing the testing procedure is to assess software fault predictions (SFPs), determine the severity of fault in a product module, and then test it [6, 7]. A sensible estimation of programming issue inclination before testing enables programming groups to concentrate on the testing exercises and to estimate costs

accurately [8, 9]. After a testing session, estimating software faults provides insight into the testing process and contributes to defining delivery and maintenance procedures. Programming deficiency and the probability of nearness of issues in the product cannot be measured straightforwardly in programming [10, 11]. Nonetheless, blame inclination can be defined through quantifiable software properties [12, 13] whenever there is a correlation between these qualities and faults. This field has been the subject of three major research directions [14]: (1) formulation and specification of criteria to measure software complexity, (2) checking the thoroughness and accuracy of the measurement, and (3) determination and investigation of models that correlate software indicators to have fault prediction [15, 16].

Some software metrics describing software quality have been proposed as static and dynamic platforms. Characteristics of the code structure are used to measure the metrics in the static platforms [17]. Static measurements are used by several administrators [18, 19] and several bunches [20]. Dynamic platforms measure testing meticulousness. Auxiliary and information streams determine the basic element measurements [12]. Numerous creators have revealed a direct correlation between product measurements and blame orientation, as well as many quantifiable programming characteristics [21–25]. On the other hand, various types of research do not have any vision for verifying and validating software fault metrics. The traditional techniques, such as testing or simulation, and the proposed challenges cannot cope well with each other [26]. For example, high costs and overheads prevent testing.

Similarly, reenactment is not for supporting transient properties since it does not encompass all framework states. Formal approaches draw much attention based on mathematical logic. Formal specification and formal verification are the components of formal methods [27]. The formal specification specifies interactions among software fault-proneness, and formal verification can logically accommodate all system states [28]. They are complete and more reliable than testing and simulation to analyze and verify the interaction behaviors among software fault-proneness methods. All papers on SFP use simulation and experiments to evaluate the proposed method. So, it is conceivable that all the state space has not been evaluated well. Model checking, as an automatic technique for verifying software systems, is a suitable approach to solving problems [29].

Problem statement

Assurance of fault programming modules plays an integral role since it allows for identifying modules that need to be refactored or tested item by item [30]. This will enable the creation of high-quality software products. SFP is a model that estimates the fault inclination of upcoming modules by utilizing fundamental expectation measurements and authentic blame information. Considering the frameworks' defects, a venture timetable can be tested and support stages all productively [31]. Early programming SFP methodologies depend on measurements; however, the forecast execution of these methodologies is inappropriate. Machine learning algorithms, particularly data mining [32], support vector machine (SVM) [33], Naive Bayes (NB) [24], and artificial neural networks (ANNs) [34], have been introduced in most of the recent works. Although software faults have been considered using these strategies, there are still sufficient parts of flaws that stay vague. Currently, ANNs have been presented as a viable form of machine

learning and information-digging groups capable of handling order and relapse issues [10]. The upside of ANN is its capacity to be used as a discretionary capacity estimate instrument that learns from watched information. ANN can be received at displaying non-linear practical connections that are complicated to show with different methods, and hence, it is appealing for programming shortcoming inclination expectation demonstrating.

Nature-inspired meta-heuristic algorithms have attracted considerable attention over the past few decades regarding engineering optimization problems. They can avoid local optima by employing principles derived from natural processes and providing solutions to various challenges in diverse fields. Meta-heuristics algorithms are useful in solving the SFP problem. Typically, these algorithms fall under four major classes: physics-oriented, swarm-driven, human-based, and evolutionary. Table 1 summarizes these algorithms, which are further classified as single-objective or multi-objective algorithms according to the number of objectives simultaneously considered. Evolutionary algorithms draw inspiration from biological evolution and natural phenomena, including selection, reproduction, combination, and mutation. During this process, potential solutions are repeatedly improved until the conditions of termination are met. The

Table 1 Nature-inspired meta-heuristic algorithms

Group	Single-objective	Multi-objective
Physics-oriented	Gravitational Search Algorithm (GSA) [35]	Multi-objective GSA [36]
	Galaxy-based Search Algorithm (GbSA) [37]	Multi-objective GbSA [38]
	Multi-Verse Optimization (MVO) [39]	Multi-objective MVO [40]
	Simulated Annealing (SA) [41]	Multi-objective SA [42]
	Big-Bang Big-Crunch (BBBC) [43]	Multi-objective BBBC [44]
	Charged System Search (CSS) [45]	Multi-objective CSS [46]
	Black Hole (BH) [47]	Multi-objective BH [48]
Human-based	Harmony Search (HS) [49]	Multi-objective HS [50]
	Jaya Algorithm (JA) [51]	Multi-objective JA [52]
	Exchange Market Algorithm (EMA) [53]	Multi-objective EMA [54]
	Teaching–Learning-Based Optimization (TLBO) [55]	Multi-objective TLBO [56]
	League Championship Algorithm (LCA) [57]	Multi-objective LCA [58]
Swarm-driven	Ant Colony Optimization (ACO) [59]	Multi-objective ACO [60]
	Whale Optimization Algorithm (WOA) [61]	Multi-objective WOA [62]
	Ant-Lion Optimizer (ALO) [63]	Multi-objective ALO [64]
	Firefly Algorithm (FA) [65]	Multi-objective FA [66]
	Artificial Bee Colony (ABC) [67]	Multi-objective ABC [68]
	Particle Swarm Optimization (PSO) [69]	Multi-objective PSO [70]
	Cuckoo Search (CS) [71]	Multi-objective CS [72]
	Harris Hawks Optimization (HHO) [73]	Multi-objective HHO [74]
	Black Widow Optimization (BWO) [75]	Multi-objective BWO [76]
Evolutionary	Genetic Programming (GP) [77]	Multi-objective GP [78]
	Genetic Algorithm (GA) [79]	Multi-objective GA [80]
	Biogeography-based Optimizer (BBO) [81]	Multi-objective BBO [82]
	Differential Evolution (DE) [83]	Multi-objective DE [84]
	Evolutionary Strategy (ES) [85]	Multi-objective ES [86]
	Evolutionary Programming (EP) [87]	Multi-objective EP [88]
	Shuffled Frog Leaping Algorithm (SFLA) [89]	Multi-objective SFLA [90]

probability of achieving improved results near the global optimum increases by growing the number of generations or iterations. The global optimum cannot be accurately estimated.

Swarm-driven algorithms draw inspiration from naturally intelligent organisms. Instead of relying on evolutionary rules to determine the most optimal solution, common swarm-based algorithms rely only on genetic principles and continuously evaluate each possible solution within the search space. Human-based algorithms are derived from daily human activities, particularly interaction, competition, and training. These algorithms draw their inspiration from human behavior modeling. Physics-based optimization algorithms comprise meta-heuristic algorithms designed to exploit physical behaviors or laws. These algorithms are motivated by physics rules associated with electromagnetic, inertia, and gravitational forces.

Motivation

SFP is accomplished through software measurements, which measure the quality of a program quantitatively [91]. The fault-proneness of software can be correlated with some software metrics, as different reviews show [92–94]. The lines of code were very important for searching previous software metrics. Several approaches have been evaluated to develop accurate SFP models. Statistical strategies [95–99] have been recommended. Several efforts have been made to determine the best way to choose the software metrics that might indicate fault proneness. For instance, Jolliffe [100] has been used as a part of [101] and [102] to decrease the number of software performance indicators while holding the vast majority of the viewed variety. It is demonstrated that the representation provided by the prediction approach is an ideal mean-square regression [100]. Information is derived from several vectors, such as eigenvectors. The vectors represent the underlying elements, and the data depict linear aggregations of the elements. As a result, the projected data describe the largest difference between the primary component and the second highest variance on the next primary component. In this manner, data dimensionality can be minimized simply by wiping out the last principal components.

Catal and Diri [24] have studied 90 papers published between 1990 and 2009 on SFP. An essential commitment of the review has shown that guidelines on software metrics and the techniques used for SFP, datasets, and performance assessment indicators are all reviewed. Catal and Diri [103] have used machine learning algorithms to construct a middleware SFP approach based on the Eclipse platform. Predicting faults in software programs is one of the goals of the review. The Naive Bayes algorithm is used due to its excellent performance. The Model for Assisted Software Process (MASP) model can filter suitable measurements for specific fault types. In addition, Vandecruys and Martens [104] have been considered to proficiently detect software defects and facilitate software development by examining software repositories utilizing the Ant Miner algorithm.

Dejaeger and Verbraken [105] have discussed the Bayesian intelligent networks for fault estimation. For this purpose, 15 different Bayesian networks are employed. Model outputs are compared with similar model results. The outcome of Markov's principal effects on the estimation model shows that it does not significantly affect the feature selection technique. The advantage of this method is that it makes Bayesian network clustering for fault estimation, but the disadvantage is that it is not verified using formal

methods. Also, Rajaganapathy and Subramani [106] have studied a combination of an immune system and a random forest algorithm as machine learning methods. An adaptable neuro-fuzzy algorithm has been developed to address accuracy issues. This paper focuses on three criteria of dataset size, metrics and techniques set, and error estimation selectable algorithms for comparing. The strength of this approach is that it has effective in fault of measurement indexes, and the disadvantage is that it is not verified using formal methods.

De Carvalho and Pozo [107] have presented an approach for rules classification that can be used for predictive models. This study used a multi-objective PSO algorithm to extract data classification rules. For this reason, the authors have changed this algorithm and developed a new algorithm using a multi-objective PSO algorithm to extract this kind of rule. The advantage of this method is that the obtained results are better than the simple PSO method, and the disadvantage is the lack of formal verification methods. Finally, Kwok and Lu [108] have offered a fuzzy regression method to predict faults of modules based on fuzzy support vector regression. The fuzzification has also been applied to the regression algorithm of the support vector so that the algorithm can manage the unbalanced data set. The presented model in this study has been tested on the proprietary dataset of a firm, and the obtained results have shown that this model shows a better operation when modules have high lines. However, the verification of the proposed model is not discussed. The reviewed studies are summarized in Table 2.

Systematic literature review

A systematic literature review (SLR) is a process that formulates hypotheses and uses specific methodologies to collect, investigate, and synthesize relevant research related to a particular topic or trend in the literature [112]. The SLR process is rigorous and involves a comprehensive search of databases and other sources. SLRs explicitly explain the knowledge and the uncertainty associated with a practice-related question. SLRs provide insight into research trends in a particular area on a single platform. SLRs are useful for identifying gaps in knowledge and potential research directions. The results of the SLR are presented as evidence to support conclusions and inform decision-making [113].

Contribution

Various meta-heuristic algorithms have been proposed to solve SFP problems in recent years. Whenever it is necessary to find optimum solutions in infinite time, traditional algorithms perform better in parameter optimization and feature selection. On the other hand, these conventional algorithms cannot provide results for more challenging optimization issues, including NP-hard or global optimization, which take a long time to resolve. ACO, genetic, and PSO algorithms have been well-known in recent years for their success in predicting software faults. After reviewing various studies utilizing meta-heuristic algorithms to predict software faults, we observed that identifying which algorithms are useful for feature selection in addition to those useful for parameter optimization is very difficult.

Further, compared to standard estimation strategies, it is often difficult to determine their performance characteristics. Considering these algorithms' ability to predict

Table 2 Comparison of prediction-related works

Disadvantage	Advantage	Dataset	Approach	Paper
• Without analyzing formal approaches • Without evaluating cost and complexity	Training ANN using historical data attention	NASA dataset	ANN and SVM	[94]
• Without evaluating functional properties	Finding a relation between fault metrics	NASA dataset	SVM	[109]
• Without examining the complexity • Without analyzing the correctness of the approach	Reducing long-association rules	Dataset of Mylyn and Eclipse PDE	Association rules	[110]
• Without presenting a simulation environment • Without analyzing the complexity	Finding fault dependency between proneness metrics	Source codes	-	[111]
• Without analyzing using formal approaches • Without discussing the cost and complexity	Hybrid prediction approach to find proneness metrics without high cost and complexity	-	ANN	[10]
• Without evaluating with formal methods • Without analyzing the complexity	Presenting a tool for finding the fault metrics using data prediction	The metrics of McCabe	ANN	[15]
• Without evaluating with formal methods	Maker of Bayesian network clustering for fault estimation	NASA dataset	Bayesian smart network	[105]
• Without evaluating with formal methods	Effective in fault than measurement indexes	Promise dataset	Combination of random forest algorithm and body immune system	[106]
• Without evaluating with formal methods • Without analysis of the complexity	Its results are better than the simple particles swarm optimization method	Nasa data set	Extraction of classification rules by particles congestion method	[107]
• Without evaluating with formal methods	The model acts better when modules have upper lines	Data sets of multiple companies	Fuzzy support vector regression	[108]

Fig. 1 Steps of the research

software faults more accurately, deciding whether they should be used for parameter optimization or feature selection is also necessary. To cover these gaps, this paper comprehensively analyzes studies on the SFP problem involving different meta-heuristic algorithms. Particularly, we examine publications in digital libraries between 2010 and 2023. As depicted in Fig. 1, this study comprises four major stages. The authors begin by providing a background to the SFP problem. Digital libraries are searched to identify useful publications according to certain criteria. Then, the current SFP approaches are discussed. Finally, we compare the reviewed approaches and explore potential gaps and areas for future study. The present paper makes the following major contributions:

- Highlighting the major issues potentially encountered during the SFP process;
- Outlining the value of meta-heuristic algorithms to address the SFP issue and discussing the latest strategies for this problem;
- Providing a thorough evaluation of SFP methods according to key factors;
- Determining and identifying areas for upcoming research.

The paper is organized into four sections. The "Methods" section explains the article selection process and reviews current SFP approaches based on meta-heuristic algorithms. The "Results and discussion" section summarizes the study findings and discusses outstanding research topics and possible directions for future research. Finally, the paper concludes with the "Conclusions" section.

Methods

The systematic review conducted in this paper was planned, conducted, and reported based on the procedures given by Kitchenham and Brereton [114]. Figure 2 illustrates the process. We drafted the review protocol at the planning stage, including the six main stages: identification of research queries, design of the search procedure, selection of studies based on specific parameters, study assessment, data extraction, and data interpretation. We prepared the review procedure at the planning stage, including the six main stages: identification of research queries, design of the search procedure, selection of studies based on specific parameters, quality assessment, data extraction, and data interpretation. The initial stage was formulating the research questions to be explored in the SLR. In the second stage, the search process was explained, along with search phrases and selecting databases for identifying relevant studies. In the third stage, related studies are identified under the research questions. This stage involves establishing exclusion and inclusion criteria for every preliminary study. The next stage was identifying quality evaluation factors and creating a questionnaire to examine the studies. Data extraction worksheets are created to provide the necessary details needed to clarify research questions and develop data synthesis strategies. This SLR presents and assesses empirical evidence from studies using meta-heuristic algorithms for SFP. This SLR addresses six research questions listed in Table 3. The following search terms are applied to find primary studies.

("software fault prediction") AND ("nature-inspired" OR "meta-heuristic").

After determining keyword phrases, the most pertinent and useful online resources were chosen. There was no restriction on selection based on digital portal availability at home universities. Searches were conducted in the following seven electronic databases:

- SpringerLink[1]
- Google Scholar[2]
- Wiley Online Library[3]

[1] www.link.springer.com

[2] www.scholar.google.com

[3] www.onlinelibrary.wiley.com

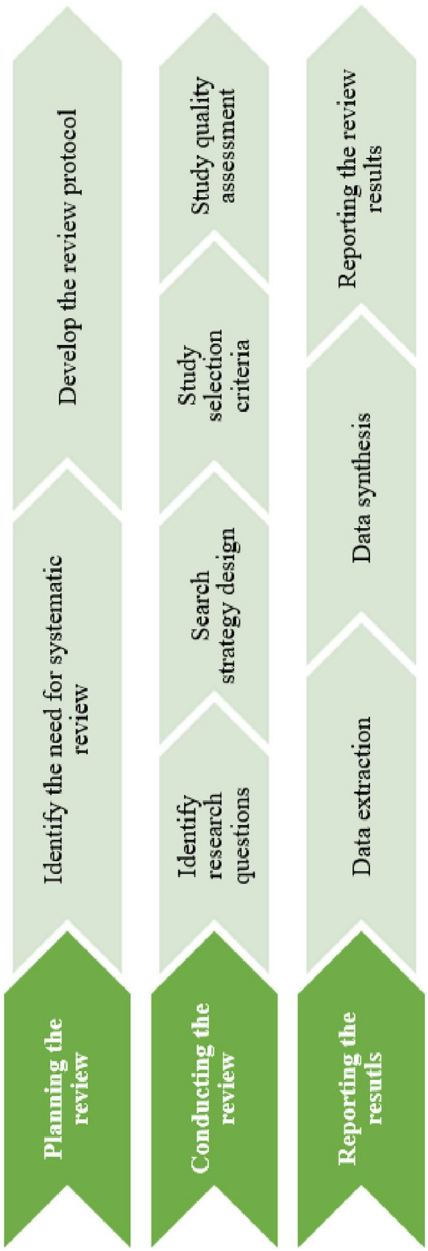

Fig. 2 Article selection process

Table 3 Adopted research questions

Index	Research questions	Aim
RQ1	Which meta-heuristic algorithms have been proposed for SFP?	Identifying the meta-heuristic algorithms widely employed in SFP
RQ2	What risks and issues are posed by testing based on meta-heuristic algorithms in the SFP area?	Need to examine different risks and issues
RQ3	What are the software indicators employed in SFP?	An overview of key performance indicators commonly used for SFP
RQ4	What are the various feature selection strategies implemented in SFP?	Identifying approaches ideal for the reduction and selection of software features
RQ5	What different datasets are available in the SFP field, and how can those datasets be accessed?	Identify the relevant dataset applicable to SFP and how to obtain that dataset
RQ6	Which meta-heuristic algorithms are better and outperform existing SFP?	Performance analysis of meta-heuristic algorithms

- ACM Digital Library[4]
- ScienceDirect[5]
- IEEE Xplore[6]

The search was restricted to 2010–2024. A preliminary search was conducted to pinpoint potential original research papers by determining which electronic databases to search. A list of relevant studies was then identified by reviewing the full-text articles according to predetermined criteria for including and excluding studies. A total of 25 main studies were chosen for inclusion in the SLR. Nine additional studies were included based on a review of the relevant research references. Therefore, 34 studies were considered for further analysis.

We incorporated specific inclusion and exclusion guidelines to ensure a focused and rigorous literature review. The inclusion criteria consist of research papers that utilize meta-heuristic algorithms for SFP, combine multiple meta-heuristic algorithms, and compare these approaches with traditional statistical methods. Conversely, the exclusion criteria eliminate studies that lack statistical analysis or assessment of meta-heuristic algorithms applied to SFP, those that use dependent variables other than fault proneness, and those that apply meta-heuristic algorithms in contexts unrelated to SFP. Through these criteria, we maintained a high standard of relevance and quality in our comprehensive review.

This section reviews the meta-heuristic algorithms employed in SFP and discusses their challenges, such as runtime, convergence, and performance evaluation. The meta-heuristic algorithms employed in SFP are based on evolutionary computation and swarm intelligence. They are used to identify global optima cost-effectively by creating and manipulating populations of feasible solutions iteratively. The results obtained from these algorithms are then compared to the optimal solution.

[4] www.dl.acm.org

[5] www.sciencedirect.com

[6] www.ieeexplore.ieee.org

ABC-based approaches

The ABC algorithm is an adaptive optimization method that divides the artificial bee colony into working bees, spectators, and observers. Colonies are divided into two halves, with one half engaged in beekeeping and the other half viewing the activity. Food source locations are potential solutions to the optimization problem. Each food source has unique nectar content that can determine physical fitness. Food sources equal the number of working bees. It is natural for a bee to become disappointed after being abandoned by its food source. The algorithm begins by distributing bees in different positions around a food source and then assigns bees, spectators, and observers to varying positions around the food source [115]. This section discusses the approaches used for SFP using ABC and their main characteristics.

Software defects continue to pose a significant problem. Software defect prediction ranks among the most critical issues in software quality research, as it can be used to plan, control, and execute software development efforts. Recently, computer researchers have studied social insects' behavior in neural networks to solve various prediction problems. Farshidpour and Keynia [116] investigate using the ABC algorithm to simulate the intelligent foraging behavior of honeybees. Multilayer Perceptron (MLP) usually uses computationally intensive training algorithms when trained with the standard backpropagation algorithm. Due to nearly constant local optima, the Backpropagation algorithm (BP) can sometimes result in networks with suboptimal weights. To overcome the complexity of predicting software defects from BP data, MLP-ABC efficiency is compared to MLP training with conventional BP. Results from the experiment indicate that MLP-ABC outperforms MLP-BP.

Most software development and maintenance costs are incurred when debugging software. Thus, it has become vital research in software engineering to develop approaches to automate the debugging process of software faults. Huang and Ai [117] propose a mechanism based on the ABC algorithm that can be integrated with other related methods. Initial instrumentation is performed on the source program following the analysis of its dependency information. After compiling and running the test cases in the program, the results are entered into the ABC algorithm. The iterative nature of this algorithm can be used to determine the best food source and the most significant fitness value among employed bees. Based on the most reliable test cases, the unit with the highest suspicion level is considered for the final fault localization. The TCAS program in the Siemens suite is used to conduct the experiments. The suggested fault localization approach provided accurate and efficient results. The ABC algorithm effectively avoids local optima and ensures greater fault location validity.

ACO-based approaches

The ACO algorithm attempts to determine the short route from colonies to food supplies, mimicking natural ant behavior. To determine the shortest route, the ants release pheromones as they move through the pathway [118]. Software defects or bugs may arise due to poor design or coding. In the case of a bug in a project, incorrect results are produced. Software bugs increase the estimated project cost. This cost can be minimized by predicting software bugs before product delivery. Kumar and Gyani [119] implemented the ACO algorithm on eight open-source datasets, comparing it to logistic regression

(LR), k-nearest neighbors, and SVM algorithms. As evident from the results, ACO offers better prediction performance than conventional approaches.

Manivasagam and Gunasundari [120] present an optimization technique that improves the ability to predict software defects. Fuzzy mutual information ACO searches for optimal features based on a meta-heuristic search. Datasets from NASA's metric data repository are used to determine the efficiency of the suggested feature selection process. According to simulation outcomes, it can be concluded that the developed method significantly enhances the prediction of routines for three distinct classifiers considered in this study. These classifiers were SVM, J48, and Naive Bayes. The results demonstrated that the suggested approach had better prediction accuracy than other methods.

Comprehensive models describe software quality by predicting low-quality components based on observable patterns. This model guides the programming and testing teams to concentrate on low-quality modules, thus ensuring that scarce resources available for software quality inspection are devoted to defects. An ACO-based learner could provide rules to describe software modules as defective or not defective. Singh and Verma [121] construct a rule-based SFP model with useful metrics by combining ACO-based mining and ROC-based rule quality update. The suggested approach was tested on public data sets of software faults. The efficiency of ACO-based learning was compared against three benchmark classifiers according to their receiver operating characteristic areas. Based on an assessment of performance measures, ACO-based learners outperform other benchmark methods.

Azar and Vybihal [122] presented a strategy to maximize the accuracy of software quality predictive models while classifying original data. A predictive model is adapted (in stages) to new data based on previously constructed models. The adaptation process relies on the ACO algorithm. The approach has been verified for class stability in object-oriented software systems and applies to other quality characteristics. It has the potential to be easily adapted to solve software quality predictive issues comprising multiple classification labels. The proposed approach outperformed both C4.5 and random guessing algorithms. Also, it maintains the clarity of the models, providing both classification labels and guidelines for achieving them.

Predicting software reliability at an early stage of development is challenging. Recently, numerous strategies have been suggested to quantify software reliability. However, it is difficult to create accurate prediction models because the software engineering domain undergoes recurrent data changes. Consequently, models developed on one dataset lose significant accuracy when applied to new datasets. An ACO-based approach developed by Mohanthy and Naik [123] enhances software reliability prediction accuracy when combined with raw data. To produce enhanced software reliability results with new data, the ACO algorithm and accompanying TSP algorithm have been modified by incorporating multiple algorithms and adding additional features. Using a colony of cooperating artificial ants, it has been shown that the framework's behavior results in promising results. NRMSE (Normalized Root Mean Square Error) validates the method on real datasets.

Mondal and Sahu [124] developed a supervised feature selection methodology leveraging the ACO algorithm for SFP purposes. Their approach involved the application of K-nearest neighbors (KNN), NB, and decision tree (DT) algorithms. A novel fitness

function and two-stage pheromone adjustment mechanism were devised to efficiently mitigate feature redundancy. Inspired by real ants' foraging patterns, which rely on pheromone trails to find optimal paths to food suppliers, the proposed algorithm explores the feature space. Comparative analysis across 12 diverse datasets was conducted, utilizing fitness plots to visualize and quantify the performance of the ACO algorithm in conjunction with the multiple classifiers.

CS-based approaches

The CS algorithm is a meta-heuristic algorithm inspired by cuckoo birds' behavioral patterns. Cuckoo birds place their eggs in other birds' nests. The CS algorithm uses this concept to discover optimal solutions by randomly selecting solutions and "laying eggs" in the nest of the best-performing solution. This procedure is repeated until an optimal solution is found. The CS algorithm combines exploitation and exploration to find the optimum solution. The exploration part randomly selects solutions, while the exploitation part evaluates the best-performing solutions. The fusion of exploitation and exploration enables the algorithm to find the best solution while preventing local optimum behavior [125].

Software engineering requires predictive defects. This is done using static analysis tools to identify and fix bugs before the code is released. The goal is to ensure the software is reliable, efficient, and secure. Continuous testing is also essential for uncovering software issues. Defect-prone modules should be checked thoroughly. It facilitates bug discovery more effectively and prioritizes the testing process. Intensive research has been carried out on this issue in recent years. Nevertheless, limited studies have examined prediction results concerning time factors. Thus, Song and Lv [126] have proposed an enhanced Elman neural network design adaptable to changes in characteristics over time. By embedding a variable step size into the CS algorithm, the underlying parameters and thresholds of the Elman neural network were optimized. The approach was evaluated by analyzing seven projects retrieved from the public PROMISE repository. The outcomes indicate that the enhanced CS algorithm contributes significantly to the Elman neural network formulation, and the improved prediction rate of the approach is superior to those of 5 benchmarks for F-measure and Cliff's Delta scores.

SVMs require both imbalanced datasets and appropriate parameters to predict software defects effectively. To solve this problem, Cai and Niu [127] have proposed a multi-objective CS algorithm for under-sampled software defect prediction. In the first step, a multi-objective CS algorithm and adaptive local search are employed to select non-defective sampling concurrently and improve SVM variables. Next, three under-sampling strategies are proposed to determine which modules are non-defective within the decision region range. Three indicators measure the efficiency of the proposed algorithm: G-mean, detection probability, and false positive rate. Also, eight datasets are selected from the Promise database to verify the proposed approach to predicting software defects. The suggested method performs better at identifying software defects than eight other prediction models based on comparing the results.

Accurate prediction of defect-prone software modules is a critical component of modern software development, as it enables efficient allocation of limited testing resources. While existing defect prediction methods are promising, enhanced performance is

needed. To address this challenge, Badvath and Miriyala [128] developed an ensemble-based approach to SFP, incorporating a hybrid technique that leverages the CS algorithm and principal component analysis (PCA) for feature extraction. By applying this methodology to five PROMISE datasets and evaluating performance using appropriate metrics, they aimed to identify the most effective classifier for defect prediction. The proposed model demonstrated better defect prediction compared to existing methods, contributing to improved software quality.

Rath and Mahato [129] conducted a systematic analysis of feature selection algorithms for enhancing the prediction of software defects. The study focused on evaluating the efficiency of the CS algorithm in identifying crucial software metrics for accurate prediction. A dataset comprising various software metrics was subjected to feature selection using CS, genetic algorithm feature selection (GAFS), differential evolution feature selection (DEFS), ant colony optimization feature selection (ACOFS), and particle swarm optimization feature selection (PSOFS). The empirical results demonstrated that CS significantly improved prediction accuracy while reducing model complexity compared to other optimization algorithms. These findings underscore the potential of CS for developing more precise and efficient software failure prediction models.

FA-based approaches

The FA emulates fireflies' behavior by flashing their lights to attract prey or mates. The intensity of light increases as it gets brighter. There is a tendency for the swarm of fireflies to congregate around the brighter fireflies. Fireflies move randomly if their intensities are equal. Flashing lights are indicative of objectives that need to be optimized. FA can be used to find the global optima for a specific problem. The algorithm is effective in non-linear, non-convex, and multimodal optimization problems. Moreover, FA requires no derivative information and is an efficient global optimization method [130].

The occurrence of software defects is a universal phenomenon. Preventing such defects at the earliest possible stage requires more attention as it requires less effort and costs. Predicting software defects is an essential component of determining software quality and reliability. Prediction of defects is a relatively new field in software quality engineering. Software quality can be identified by identifying the key predictors, the type of data to be collected, and the role of defect prediction models. Feature selection is an important preprocessing technique for applications that utilize large amounts of data. The process involves selecting the likely minimal attribute expected to appear in the set of actual attributes. Anbu and Anandha Mala [131] proposed an FA-based feature selection strategy and classifiers like SVM, NB, and KNN for classifying the selected features. The feature space can be searched quickly for a subset of features that minimizes a certain fitness function. The fitness function considers classification accuracy and size reduction. The experiment results revealed that selecting features using the FA provides better classification accuracy than the other methods.

Yenduri and Gadekallu [132] introduced a novel Maintainability Index (MI) constructed from a combination of software metrics to minimize prediction error. To optimize this index, they employed FA and subsequently compared the resulting base model against traditional counterparts: DE, ABC, PSO, and GA. Performance evaluation was conducted using differential ratio, correlation coefficient, and RMSE metrics.

Pemmada and Nayak [133] proposed a hybrid model combining a deep neural network (DNN) with a memetic firefly algorithm (MFA) for SFP. The DNN is employed for classification tasks, while the MFA optimizes its hyperparameters. A novel perturbation operator was incorporated into the MFA to boost its exploration potential and avoid local optima. The proposed method was evaluated against several hybrid alternatives, including DNN with Firefly Hill-Climbing, DNN with Firefly, DNN with PSO, and a standalone DNN. Experimental results demonstrated that the DNN-MFA model achieved a superior accuracy of 98.8%, outperforming the compared approaches. This research highlights the potential of the proposed model for effective software risk prediction in project environments.

GA-based approaches

GA is a probabilistic search algorithm that incorporates genetics and natural selection. GA begins with a pool of solutions known as a population. Chromosomes represent solutions. Chromosomes are evaluated at every generation and selected for the next generation based on their fitness scores. The fittest chromosomes are then recombined in a process known as crossover. Finally, a mutation process is initiated to the next generation of chromosomes, resulting in a new, improved population. This cycle is repeated until an optimal solution or a predetermined number of generations is reached. The result is a set of chromosomes optimized for the problem [134].

Software module defects can be predicted with the help of fuzzy classification. Jin and Dong [135] employed the fuzzy measure to enhance prediction accuracy and performance by obtaining the interaction among metrics by applying the Choquet Integral (CI) for classification in the n-dimensional area and determining the lowest misclassified object by distance automatically. The model uses GA on the training data to estimate unknown parameters. Four NASA software projects were examined to verify the proposed model. The proposed model performs better in predicting results than other prediction models.

Engineering machinery must be maintained and repaired when a defect or problem is identified and the error is detected and fixed. Reduced costs can be achieved through rapid troubleshooting, defect analysis, and repair. Consequently, the role of repair and troubleshooting in a repair and maintenance system is crucial. Nowadays, software plays an important role in performing system tasks, so it is essential to ensure the reliability of systems. Due to this, error-tolerant systems are necessary to increase reliability. Considering the increasing development and implementation of software across a wide range of domains, software reliability has an imperative role to play throughout the lifecycle of a piece of software. Software error prediction is one of the most crucial solutions for improving software reliability and decreasing maintenance costs. Software errors can be predicted in several ways. Genetic algorithms, due to their intelligence, have a high ability to predict, so Fazel [136] used them to predict software future conditions or predict software errors. This method aims to predict software errors accurately and rapidly. The results indicate that the approach effectively predicts the error and output rates for the given time. According to the results, the suggested method achieves a recognition rate of more than 95% in the best-case scenario.

Software defect prediction aims to identify potentially defective source code areas and minimize effort, time, and costs associated with software quality assurance. The prediction of software defects is achieved by implementing machine learning algorithms into the code and evaluating non-code metrics. Nalini and Krishna [137] explored using code profiles to substitute conventional measures for predicting software defects. An analysis of the proposed novel evolution algorithm reveals that it has more potential than any traditional machine learning approach. The goal is to develop an effective machine-learning algorithm for predicting the number of bugs a software project produces as it reaches the quality assurance stage.

Kaliraj and Reddy [138] addressed the challenges inherent in SFP, focusing on class imbalance, metric significance, and feature selection. They employed random oversampling to mitigate class imbalance, enhancing the model's ability to predict faulty and non-faulty software instances. A comprehensive analysis of software metric categories, including size, cohesion, complexity, coupling, and documentation, was conducted to identify influential predictors. A modified GA was applied to optimize feature selection and reduce dimensionality. Experimental results using a diverse open-source dataset demonstrated a significant improvement in prediction accuracy compared to traditional methods. This research introduces a robust framework for SFP, empowering practitioners to develop more accurate models by effectively handling class imbalance, selecting relevant metrics, and optimizing feature sets, ultimately contributing to enhanced software quality and reliability.

Gupta and Rajnish [139] proposed a novel approach to SFP that involves selecting optimal machine learning and deep learning techniques from a pool of high-performing algorithms. Mutual information was employed for feature selection to enhance model performance, while a hybrid SMOTE-Tomek oversampling technique addressed class imbalance. Subsequently, GA-based decision trees (GA-DT) and artificial neural network-based decision trees (ANN-DT) models were developed. The proposed approach was evaluated using the Eclipse dataset (versions 2.0, 2.1, and 3.0), with precision, recall, accuracy, and F1-score metrics employed for performance assessment. Experimental results demonstrated the effectiveness of both GA-DT and ANN-DT models in predicting software faults, with ANN-DT consistently achieving superior accuracy across all Eclipse dataset versions.

Hybrid approaches

Combining different algorithms is currently regarded as one of the most successful optimization techniques. Combining various algorithms makes achieving better and more efficient solutions possible than those obtained using a single algorithm. Furthermore, it is possible to exploit the strengths of each algorithm while mitigating the weaknesses. This technique is especially useful when dealing with complex optimization problems that are difficult to solve with a single algorithm.

Researchers have focused on obtaining a correlation between software metrics and a module's fault-proneness. Jin and Jin [10] discussed the application of hybrid ANN and quantum PSO (QPSO) in predicting software fault-proneness. ANN performs a fault-proneness classification, and QPSO achieves dimensionality reduction. Results from the experiments indicate that the proposed prediction approach can establish a correlation

between software metrics and modules' fault-proneness and that its implementation does not require expertise or additional costs. Software developers can use the proposed prediction approach to identify potential fault-prone software modules, so they only have to concentrate on these modules, which may reduce the effort and cost of software maintenance. Moussa and Azar [140] presented an algorithm that uses object-oriented metrics to classify software modules as fault-prone. It is a combination of PSO and genetic algorithms. It is experimentally verified using eight different data sets. It is evaluated against other widely used classification methods.

Software testing is one of the most critical and time-consuming tasks in the software development process. To enhance software quality assurance processes, researchers have proposed several approaches for predicting the fault-proneness of software modules. Ibrahim and Ghnemat [141] proposed a strategy for software defect prediction, combining two existing algorithms, the Bat-based search algorithm (BA) for feature selection and the random forest algorithm (RF) for prediction. Additionally, several feature selection classifiers and strategies were evaluated in this study to determine their effectiveness.

The low accuracy of SFP results in the late detection of some faulty modules, increasing the effort and cost of repairing abnormal faults. To increase the accuracy of SFP, it is necessary to solve the data dimensionality problem. Dimensionality reduction is accomplished by using feature selection algorithms. Feature selection algorithms fall into filter-based feature selection and wrapper-based feature selection. Prediction models based on wrapper-based algorithms are more accurate. These algorithms can use different methods to find the best solutions; meta-heuristic search is the best. Since each meta-heuristic algorithm has certain strengths and weaknesses, the researchers use a combination of algorithms to overcome these weaknesses [134] combined genetic, ACO, and WOA as the wrapper feature selection. Applying early SFP methods before the actual test is one of the most effective passive defense strategies for reducing the costs associated with the development of software systems. The proposed method is evaluated using 19 software projects. Results show that the proposed method performs better than other methods.

The quality of the software is reflected in software defects, and software failures can be predicted using software reliability models. Yang and Li [142] applied a hybrid algorithm for estimating model parameters to software defect prediction to address the difficulty of estimating the parameters of software reliability models. PSO is a typical swarm intelligence algorithm with fast convergence but low accuracy in its solution. Sparrow search algorithm (SSA) is known for its high search accuracy, fast convergence speed, and good stability and robustness. To accelerate the convergence before the individual updates of the SSA, Yang and Li [142] proposed a hybrid approach that combines the PSO with the SSA. Additionally, the authors constructed a new fitness function based on the maximum likelihood estimation of the parameters and used it to initialize the parameters. An analysis of five sets of actual data sets revealed that the hybrid algorithm performed better than a single algorithm in terms of convergence speed and accuracy than a single algorithm in terms of convergence speed and stability.

Alsghaier and Akour [143] incorporated genetic algorithms into SVM classifiers and WOA to predict software faults. This approach was applied to 24 datasets, in which NASA MDP is considered a large-scale dataset, and Java open-source projects are

considered a small-scale dataset. Results indicate that the proposed approach is effective at predicting software faults in large and small datasets and overcomes the limitations of previous studies.

Anju and Judith [144] proposed a novel deep-learning model for SFP. The model incorporates an adaptive recurrent neural network (ARNN) optimized by a Levy-Flight integrated cuckoo search optimization (LICSO) algorithm. Data was preprocessed using Box-Cox transformation to enhance model performance, and feature selection was performed using Quantum theory-particle swarm optimization (QPSO-FS). The model's effectiveness was evaluated using accuracy, precision, recall, F1-score, and processing time metrics. Experimental results demonstrated superior performance compared to existing approaches, with the proposed model achieving a peak accuracy of 96.4%.

Alsghaier and Akour [145] developed a novel approach to SFP by integrating GA with SVM and PSO. This hybrid model aimed to enhance prediction accuracy and address the limitations of previous studies. The proposed method was evaluated on a comprehensive dataset comprising 12 NASA Metrics Data Program (MDP) and 12 Java open-source projects representing large-scale and small-scale software systems. Experimental results demonstrated that integrating GA, SVM, and PSO significantly improved software fault prediction performance across both dataset types.

LOA-based approaches

Lion optimization algorithm (LOA) is derived from lions' special lifestyle and cooperation characteristics. An initial population is formed by a set of randomly generated solutions called lions. Some of the lions in the initial population are selected as nomad lions, and the rest of the population is randomly partitioned into subsets called prides. S percent of the pride's members are female, and the rest are male, while this rate in nomad lions is vice versa. For each lion, the best-obtained solution in past iterations is called the best-visited position and is updated progressively during optimization. SFP is extensively performed using machine learning-based classifiers. Classifiers' performance in predicting fault-prone software modules is threatened by the curse of dimensionality. It was discussed by Goyal and Bhatia [146] how to select optimal feature subsets from high-dimensional defect datasets using meta-heuristics. They proposed an LOA-based feature selection model and statistically compared it with state-of-the-art meta-heuristic models. Experiments are conducted with the NASA dataset. Based on the experiments, it can be concluded that the proposed algorithm performs better than the baseline techniques, with the highest AUC measure (90%) and accuracy measure (94%).

MFO-based approaches

In Moth-Flame optimization (MFO), moths are simulated to move around light sources in a spiral pattern at night. The MFO stands out from other meta-heuristic algorithms for its simplicity and low computational complexity. Consequently, the MFO can be applied to a variety of real-world problems, including feature selection and constraint engineering. MFO uses flames to preserve the best solutions. A global search strategy is also employed to explore the search space efficiently.

SFP is one of developers' most complex problems during software development. In real-life software development projects, the collection of data can be challenging; the

distribution of data collected may be imbalanced. To predict software faults, Tumar and Hassouneh [147] developed an enhanced binary MFO algorithm incorporating adaptive synthetic sampling. The MFO algorithm is used as a wrapper feature selection, while adaptive synthetic sampling is used to strengthen the input dataset and deal with the imbalance. This study investigates the conversion of MFO from a continuous representation to a binary one by combining two transfer functions (V-shape and S-shape). This study uses data from fifteen actual projects retrieved from the PROMISE repository. In this study, three distinct types of classifiers are employed: KNN, DT, and linear discriminant evaluation. The findings suggest that the presented approach improves the performance of classifiers and is superior to previous research, demonstrating the significance of TF when selecting features for classifiers.

Anjali and Dhas [148] proposed a hybrid model combining faster convolutional neural networks (FCNN) with MFO for predicting software bugs. The model leverages program-level metrics, such as code lines and method characteristics, as input features. MFO is employed to optimize FCNN's weight parameters. The proposed MFO-FCNN approach was compared against traditional machine learning methods, including AdaBoost, random forest, K-nearest neighbors, K-means clustering, SVM, and bagging classifier. Experimental results demonstrated the superior performance of the MFO-FCNN model in accurately predicting software bug counts.

PSO-based approaches

The PSO algorithm is an evolutionary computational methodology that focuses on particle social behavior. PSO is a meta-heuristic method for optimizing a candidate's solution based on quality indicators by repeating the process. Simulating social behavior was originally used to show the activity of birds and fish.

By accurately predicting defect-prone software modules, software testing efforts can be reduced, costs can be reduced, and the software testing process can be improved. Using static code attributes as defect predictors in software defect prediction research has become common practice due to their usefulness, generalizability, ease of use, and wide acceptance. However, class imbalance and noisy attributes are common data quality concerns that can impact software defect prediction accuracy. To improve software defect prediction accuracy, Wahono and Suryana [149] combined the PSO algorithm with the bagging method. The PSO algorithm handles feature selection, and class imbalance is addressed by the bagging method. A statistical evaluation of the proposed approach uses data sets provided by NASA's metric data repository. The proposed approach significantly enhances the prediction performance of most classifiers, according to experimental results.

Several strategies have been developed to reduce testing costs and efforts based on the fault-proneness of classes or methods. Machine learning algorithms have recently been employed to predict fault-proneness through design metrics. However, some of these algorithms cannot handle unbalanced data, which is common in fault datasets. Furthermore, the results produced by these algorithms are difficult for most programmers and testers to understand. The multi-objective PSO algorithm was used by De Carvalho and Pozo [107] to develop a novel fault-prediction approach. Pareto dominance concepts are used to generate a model containing rules with particular characteristics. The rules apply

to an unordered classification, which makes them more intuitive and understandable. Two experiments were conducted to determine whether classes and methods are fault-prone. The findings demonstrate meaningful correlations between fault prediction and the studied measures. Furthermore, the performance of the proposed approach is tested in comparison with other machine learning algorithms based on several criteria, such as the area under the ROC curve, one of the most important criteria for handling unbalanced data sets.

The importance of expeditious, efficient, and productive fine-tuning becomes increasingly paramount with time as something breaks in the application portfolio. Therefore, recognizing faults early in the software development lifecycle will decrease effort and money costs. Furthermore, it is crucial to identify redundant or highly correlated features, as this will have a significant impact on the learning process of the model. This study combines crossover ANN and binary PSO with Binary Cross-Entropy (BCE) loss as the fitness function. Malhotra and Shakya [150] explained the importance and potential of using BCE in binary PSO for the feature reduction scheme to reduce developer workload and maintenance expenses.

WOA-based approaches

The WOA was inspired by humpback whale hunting. The solutions are categorized as whales. The whale uses the best element of the group as a reference point when searching for a new location. Whales use two mechanisms: searching for prey locations and attacking them. The first approach involves encircling prey, while the second involves creating bubble nets.

SFP can be enhanced by using soft computing and machine learning methods. Since fault data is derived from mining software historical repositories, it is usually large in size. This data contains a wide range of features (metrics). Data dimensionality can be reduced by identifying the most valuable features. The WOA is enhanced by Hassouneh and Turabieh [151] by integrating it with a simple crossover approach. By strengthening the exploration process, the suggested modification allows the WOA to escape local optimum conditions. The selection procedure comprises five distinct procedures: tournaments, roulette wheels, linear ranks, stochastic universal sampling, and random selection. The proposed enhancement is evaluated by adopting 17 SFP datasets from the PROMISE repository. The detailed assessment indicates that the suggested method surpasses the standard WOA and the existing five existing approaches.

Results and discussion

Individual algorithm analyses

Meta-heuristic algorithms have demonstrated significant promise in SFP, offering diverse approaches to optimize the detection and prediction of software defects. Each algorithm has its unique strengths and weaknesses that contribute to its effectiveness in specific scenarios. As illustrated in Table 4, The ABC algorithm excels in avoiding local optima due to its adaptive swarm optimization and the division of bee roles, particularly when compared to PSO and genetic algorithms. It is particularly effective when integrated with other methods to enhance fault localization. However, it may require extensive tuning and is sensitive to initial parameters.

Table 4 Comparative analysis of meta-heuristic algorithms for SFP

Algorithm	Key features	Strengths	Limitations
ABC	Adaptive swarm optimization, division of bee roles	Effective in avoiding local optima, integrates with other methods	It may require extensive tuning, sensitivity to initial parameters
ACO	Mimics ant foraging, pheromone-based learning	High prediction accuracy, effective feature selection	Computationally intensive, may suffer from stagnation
CS	Inspired by cuckoo bird's brood parasitism, the combination of exploration and exploitation	Good exploration–exploitation balance, high accuracy in feature selection	May converge slowly, sensitive to parameter settings
FA	Inspired by firefly flashing behavior, non-linear optimization	Efficient global search, robust against non-convex problems	Performance may degrade with large datasets that require parameter tuning
GA	Evolutionary principles of selection, crossover, and mutation	High predictive accuracy, flexible for hybrid approaches	It can be computationally expensive, sensitive to population size
Hybrid	Combines multiple algorithms to leverage strengths	Improved accuracy, reduced dimensionality, efficient optimization	Complexity in implementation, the potential for overfitting
LOA	Mimics lion pride and nomad behavior	Effective feature subset selection, high performance	It may require complex tuning, computationally demanding
MFO	Simulates moth navigation around flames, spiral optimization	Low computational complexity, efficient feature selection	It may be less effective with highly imbalanced data
PSO	Social behavior of particles, iterative optimization	High convergence speed, good for unbalanced data	May suffer from premature convergence, sensitive to initial conditions
WOA	Inspired by humpback whale hunting, bubble-net strategy	Robust exploration capabilities avoid local optima	Computationally intensive, parameter sensitivity

Although PSO converges quickly, it is prone to premature convergence, whereas ABC's scout bees ensure broader exploration, reducing this risk. By avoiding local optima using dynamic role adaptation, ABC offers a more effective strategy than GA, which relies on mutations and crossovers. When ABC is integrated with other methods, such as PSO or GA, its strengths are enhanced, improving convergence speed and robustness. For applications such as software fault prediction, these hybrid approaches combine the best features of both algorithms to provide superior accuracy and reliability.

The ACO algorithm is noted for its high prediction accuracy and effective feature selection capabilities, driven by its pheromone-based learning mechanism. Despite these strengths, ACO can be computationally intensive and may suffer from stagnation issues, which can limit its scalability in large-scale applications. To address these challenges, parallel processing is employed to distribute computational loads, significantly speeding up convergence. Hybrid approaches, combining ACO with other algorithms, are used to mitigate stagnation by introducing diversity into the solution pool. For improved robustness and to avoid premature convergence, adaptive parameter tuning or further integration with other optimization techniques, such as PSO, could be explored in the future.

The CS algorithm combines exploration and exploitation based on cuckoo brood parasitism. It delivers an excellent balance between exploration and exploitation, making it highly accurate for feature selection. However, CS may converge slowly and is sensitive to parameter settings, which could affect its performance in some contexts. FA, inspired by firefly flashing behavior, is robust against non-convex problems and offers efficient

global search capabilities. Nevertheless, its performance may degrade with large datasets that require careful parameter tuning.

GA is based on evolutionary principles of selection, crossover, and mutation, providing high predictive accuracy and flexibility for hybrid approaches. However, it can be computationally expensive and sensitive to the population size used. LOA mimics lion pride and nomad behavior, offering effective feature subset selection and high performance. Its main limitation is the complex tuning required, which can be computationally demanding.

The MFO algorithm simulates moth navigation around flames, offering low computational complexity and efficient feature selection. However, MFO may be less effective with highly imbalanced data and requires careful parameter tuning to avoid performance degradation. The PSO algorithm is popular for its high convergence rate and effectiveness in handling unbalanced data. However, it may suffer from premature convergence and is sensitive to beginning conditions. WOA, derived from humpback whale bubblenet hunting, provides robust exploration capabilities and effectively avoids local optima. Its limitations include being computationally intensive and sensitive to parameters.

Hybrid approaches analyses

Hybrid approaches that combine multiple algorithms often employ the strengths of each to attain superior accuracy and efficiency in SFP. These approaches can significantly improve prediction accuracy and reduce dimensionality. They are prone to overfitting, especially when models become too complex or are tailored to specific data sets. For example, the combination of ANN and QPSO has been shown to significantly improve prediction accuracy while reducing dimensionality. While this hybrid approach improves prediction accuracy and reduces dimensionality, it is prone to overfitting, especially when models become too complex or are tailored to specific data sets.

Cross-validation can mitigate these risks by ensuring model generalization across different data subsets. Regularization techniques, including L1/L2, penalize overly complex models in order to prevent overfitting. Furthermore, ensemble techniques like boosting and bagging combine multiple models to enhance robustness and reduce variance. In software engineering, studies have successfully used boosting to balance accuracy with generalization in fault prediction. Using these strategies, hybrid models are able to achieve high performance without sacrificing reliability or becoming too dataset-specific.

Using multi-algorithm wrappers that include combinations of genetic, ACO, and WOA algorithms can optimize feature selection and improve fault prediction accuracy. These wrappers allow the strengths of different algorithms to be harnessed in a single framework, though they may increase implementation complexity and risk of overfitting.

While hybrid methods provide improved performance, they also pose challenges, such as increased complexity in implementation and the potential for overfitting. These challenges necessitate careful design and validation to ensure that the hybrid models generalize well to unseen data.

On the other hand, algorithms like MFO and LOA provide efficient feature selection mechanisms with low computational complexity, making them suitable for handling large datasets. The theoretical time complexity of MFO is typically $O(N \times T)$, where N

denotes the population size and T signifies the number of iterations. In contrast, the theoretical time complexity of LOA is $O(M \times T)$, where M represents the pride size. In high-dimensional spaces, MFO and LOA outperform more complex algorithms like genetic algorithms in execution time, with resource consumption reductions of up to 30%. Nevertheless, they may be less effective with highly imbalanced data and require careful parameter tuning to avoid performance degradation.

Feature reduction techniques in SFP

Various techniques have been developed over the years to reduce feature dimensions and generate models that provide more accurate predictions. There are two approaches to diminishing features: feature extraction and feature selection. The feature selection process entails determining relevant features, while feature extraction entails merging several useful features into a much smaller set of features. In the reviewed studies, Co-relation-based Feature Selection (CFS) was the most frequently employed feature selection approach. Using CFS, redundant and noisy features are eliminated, and only features closely correlated with the fault proneness characteristic are kept. Other common feature selection methods include wrapper attribute selection, correlation analysis, and the χ2-based filter. Most papers utilized a feature selection approach, but only a few studies utilized PCA to extract features. The studies use software metrics as independent variables to estimate fault proneness. Software engineering uses several metrics to quantify the attributes of software products. These studies are categorized according to the criteria applied to the selected research papers to predict fault proneness.

- Procedural metrics: According to [152, 153], these studies use a combination of static code metrics and size measures, including Lines of Code (LOC).
- Object-oriented metrics: Many metrics are used in these studies to evaluate different aspects of Object-Oriented (OO) software, including inheritance, coupling, and cohesion for an OO class.
- Hybrid metrics: In some studies, the prediction of fault proneness has been conducted using OO and procedural metrics.
- Miscellaneous metrics: Various metrics are included in some studies, including miscellaneous, elementary design evolution, file age, fault slip-through, churn, network metrics derived from dependency graphs, and requirements that cannot be classified as object-oriented or procedural metrics.

Several studies have reported that OO metrics are strongly related to fault proneness. OO metrics useful for SFP are presented in Fig. 3. This figure depicts LOC, response to a class (RFC), and coupling between objects (CBO) as highly useful metrics for SFP.

SFP has been conducted using a wide range of datasets. As evidenced by our observations, the NASA dataset is the most commonly used in SFP, following the PROMISE repository and open-source datasets. The datasets have been classified into six types. Most of these datasets are freely accessible, while a few are private. These datasets are described in the following.

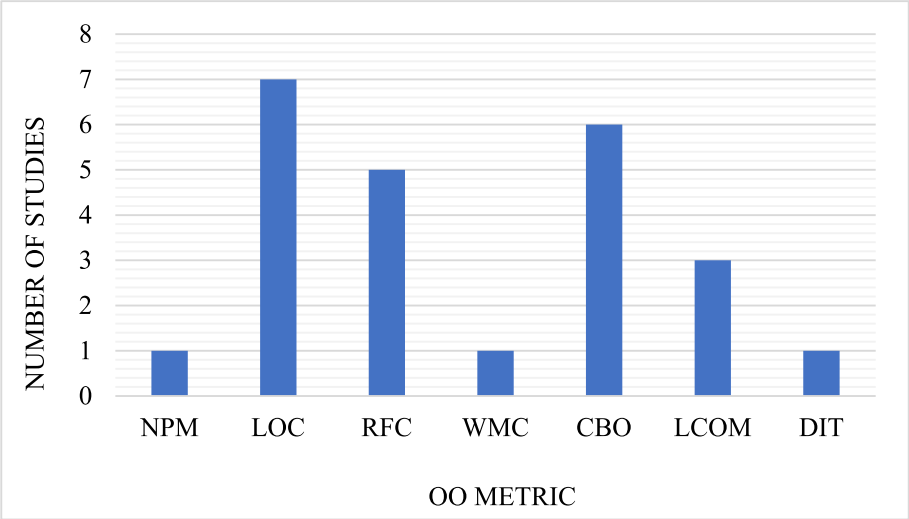

Fig. 3 OO metrics for SFP

- NASA dataset: It is one of the widely available datasets. Several research papers have employed NASA datasets in their research analyses. These datasets are available in the NASA metrics DATA program repository.
- PROMISE dataset: It is also widely used in the SFP field. PROMISE repository provides free access to the datasets.
- Eclipse dataset: The majority of its versions are free to download. During our investigation, we identified four studies that utilized eclipse datasets.
- Student dataset: It mainly pertains to academic studies produced by students. Out of 154 studies, four were based on student datasets.
- Open-source dataset: There are also other open-source software efforts, including Kspread, Kpdf, Klac, OpenOffice, Gnome, Apache, Lucene, and Xylan.
- Other: This is a private or enterprise dataset, such as a data set from a commercial banking dataset or a commercial Java application.

Data validation strategies are critical to verify SFP dataset accuracy and reliability. Cross-validation refers to dividing the data set into multiple subsets to iteratively test and validate the model, reducing overfitting and improving generalization. A data cleansing process comprises removing or correcting errors, duplicates, and inconsistencies. This is critical to eliminating noise that distorts predictions. Data quality is increasingly monitored in real-time as new information is generated, ensuring models remain relevant and accurate over time. As a result, SFP models become more robust and reliable, providing clean, precise, and timely predictions, thereby improving prediction results in real-world applications.

Future trends and recommendations in SFP

In real-world SFP scenarios, large datasets and imbalanced data are common challenges, especially in industrial applications. The efficiency of MFO and PSO algorithms makes them suitable for handling large datasets. Combining SVM with

ensemble methods like boosting, which improves accuracy while mitigating bias, is the best approach for dealing with imbalanced data. An example is the combination of MFO for feature selection and SVM for classification in a large-scale financial software application that significantly increased fault detection rates and reduced computational costs. These strategies ensure robust and scalable SFP implementations in industrial environments.

As shown in Fig. 4, useful information related to software faults can be predicted from a more complex model that distinguishes between defective and non-defective code. Besides predicting fault severity in terms of high, medium, and low, an estimation can be indicated by the Fibonacci series regarding story points or low, medium, and high values. Classifying faults according to a particular module, package, and historical fault dataset makes it possible to predict possible code changes for the specific faults. According to historical records, the resource allocation for the fault is also affected by a collection of variables, such as the team's resources, which have worked on similar faults in history and have a thorough understanding of the codebase.

Class imbalance, a common problem in SFP models, results from a large gap between fault-prone and non-faulty instances, leading to biased predictions. In the models discussed in this study, resampling and class weighting are typically used to address this problem. Resampling methods such as the synthetic minority oversampling technique (SMOTE) or undersampling balance the data set by increasing the samples of the minority classes or reducing the samples of the majority classes. Class weighting, on the other hand, gives greater importance to minority classes during model training to ensure that the model provides equal attention to underrepresented data. These techniques have been effectively implemented in various studies to enhance SFP accuracy and reliability, especially in datasets with significant class imbalances.

Forecasting or classifying the fault's functional or nonfunctional category is also possible. Does this represent a security vulnerability? Are there any problems resulting from a regression? These are the major data points for software development teams to consider when planning their projects. This results in an improved allocation of resources. In the production environment, prediction-based testing can reveal more defects than manual analysis, restricted by cost concerns, thus reducing the number of unidentified faults. The study does not suggest replacing traditional fault-fixing techniques with prediction-driven approaches. When combined with hybrid implementation, prediction-based methods can offer significant potential, and obtaining reliable prediction insights is crucial in making informed planning decisions. The following future trends are recommended to be studied in upcoming works.

- The training data distribution significantly impacts the efficiency of SFP models. As the name implies, class distribution refers to the number of class instances in a training dataset. The class imbalance problem arises when instances belonging to one class exceed those belonging to another. The majority classes are those with the most instances, while minority classes are those with fewer instances. Whenever there are fewer instances of the faulty class under consideration, the problem becomes more serious.

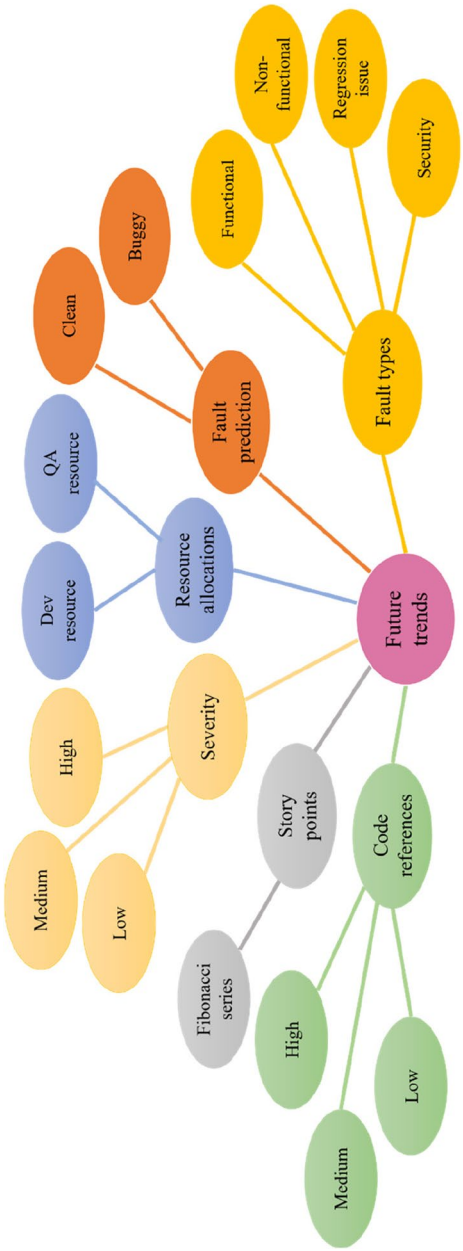

Fig. 4 Future trends for SFP

- Ironically, most of the work in SFP has been focused on its ease of use, while very few have considered its economic value. It can be extremely expensive to misclassify a component, particularly when faulty components are predicted to be non-faulty.
- The other problem with SFP is that researchers and scholars use different techniques on different datasets. There is, however, no standard framework or procedure for applying SFPs to local or cross-company projects.
- The collected dataset should include a wide range of features to predict more information about the software faults. A multi-label classification approach requires the application of appropriate Artificial Intelligence techniques.
- Effective data validation strategies must be implemented to produce credible predictions for each category of SFP. This is necessary to verify that all data used in making predictions is accurate, up-to-date, and reliable. Having reliable data is especially important for making predictions, as prediction accuracy is determined by the data quality used.
- As a future research direction, it might be useful to examine more sophisticated methods of assessing misclassified errors in SFP models. A cost-sensitive learning framework that dynamically adapts to different project contexts or the integration of more comprehensive economics-based metrics may provide deeper insight into the true cost–benefit ratio. Furthermore, expanding the Return on Investment (ROI) analysis to include long-term impacts such as maintenance savings and operational efficiency could provide a more holistic view of the economic value of SFP implementations.
- Future research directions could include expanding the study of comprehensive feature sets and advanced classification techniques, such as multi-label classification, which can significantly improve SFP models' effectiveness. Researchers could explore integrating more diverse and domain-specific metrics tailored to different software projects. Additionally, further development of multi-label classification techniques to better manage overlapping defect categories and interdependencies between software components could lead to more precise and actionable predictions.
- Investigating how SFP models can better adapt to software changes over time by using incremental learning and transfer learning techniques could be of substantial benefit. Incremental learning allows models to be updated as relevant data becomes available. This ensures that predictions remain accurate even as the software evolves. Transfer learning, on the other hand, allows models to leverage knowledge from previous projects or related areas, reducing the need for extensive retraining when applied to upcoming or evolving software systems. By focusing on these adaptive techniques, future research could improve the flexibility and longevity of SFP models, making them more resilient to software development dynamics and more applicable to a wider range of projects.
- Research into integrating SFP models into DevOps workflows could focus on using automated triggers for retraining based on software changes. By examining how these triggers can initiate real-time model updates in response to evolving software conditions, future studies can improve SFP models' adaptability and relevance. This integration would support continuous defect detection and improvement within DevOps pipelines, ensuring more reliable and resilient software systems.

- The labels in the collected dataset must be accurate and well-defined to ensure the model is well-equipped to predict the classes accurately. If the labels are inaccurate, the model may not be able to discern the features associated with each class properly. This can lead to misclassification errors, and the model's predictions may not accurately reflect the true classes of the data.
- Due to the extensive setup required to perform the prediction, SFP is relatively underutilized in the industry. A dedicated SFP tool can aggregate predictions and provide valuable knowledge of software faults. The tool should enable the user to perform the necessary steps for fault prediction through an interactive user interface.
- Investigate methods for adapting software fault prediction models dynamically as the software evolves. Software systems are subject to changes over time, and prediction models should be able to adjust their predictions based on these changes. This could involve techniques for incremental learning or transfer learning to update models with new data.
- Develop techniques to estimate uncertainty and confidence levels in the predictions provided by SFP models. This is particularly important when making critical decisions based on predictions. Users need to understand the reliability of the model's output to avoid making incorrect decisions.
- Consider incorporating temporal aspects into SFP models. Software faults may exhibit patterns over time, and analyzing these patterns could lead to more accurate predictions. Time series analysis techniques could capture and leverage this temporal information.
- Explore ensemble methods and model combination techniques to improve the robustness and accuracy of SFP models. Assembling predictions from diverse models can mitigate the limitations of individual models and provide more reliable predictions.
- Focus on developing interpretable and explainable AI techniques for SFP. Model predictions can enhance trust and understanding among stakeholders, such as developers and managers, and enable better decision-making.
- Investigate the generalization of fault prediction models across different projects and domains. Creating models that can be transferred from one context to another without significant loss of accuracy could save time and effort in developing new models for every project.
- Design prediction frameworks that incorporate human expertise and feedback. Collaboration between machine learning models and domain experts can enhance the quality of predictions and assist in fine-tuning models based on real-world insights.
- Extend the research to include cost-sensitive learning techniques. Misclassifying faults can have different costs based on severity or impact. Models considering these costs during learning could lead to more effective predictions.
- Integrate SFP models into DevOps practices to enable continuous monitoring and improvement. This could involve automated triggers for retraining models based on changing software and operational conditions.
- Establish benchmarks and standardized evaluation metrics for comparing different SFP models. This would enable researchers and practitioners to objectively assess the performance of different techniques and models.

- Extend fault prediction beyond identification and classification to include proactive measures such as predictive maintenance and proactive fault remediation. This could help in preventing faults from occurring in the first place.
- Develop tools and plugins that facilitate the integration of software fault prediction into the software development process. These tools could streamline the steps involved in prediction and provide actionable insights to developers.
- Investigate real-time fault prediction and monitoring techniques that can quickly identify emerging faults and anomalies in the software system. This is especially relevant for applications that require high availability and reliability.
- Involve end-users and stakeholders in the evaluation of fault prediction models. Understand how predictions impact their decision-making and gather feedback on the usefulness and effectiveness of the predictions.

Conclusions

This paper reviewed previous publications on the SFP problem to assess current research and propose suggestions for further research. Our evaluation focused on papers that used meta-heuristic algorithms with particular attention to parameters, strategies, and datasets. Review results indicate that public datasets have significantly increased, and the number of meta-heuristic algorithms used has increased slightly since 2015. Researchers working in the SFP field are encouraged to develop better fault predictors using public datasets and meta-heuristic algorithms. This trend underscores the growing recognition of the importance of SFP in ensuring software quality and reliability. Our findings suggest that while there has been progress in applying meta-heuristic algorithms to SFP, there is still ample room for improvement and innovation.

Researchers are encouraged to develop more sophisticated fault predictors by leveraging the wealth of available public datasets and refining existing meta-heuristic algorithms. The integration of these algorithms with advanced machine learning techniques and hybrid approaches could yield significant improvements in prediction accuracy and computational efficiency. Moreover, there is a noticeable gap in the literature concerning the practical applications of SFP. Most studies focus on theoretical and experimental validations, with limited emphasis on real-world implementations. Future research should aim to bridge this gapby exploring the practical challenges and solutions associated with deploying SFP models in industry settings. This includes addressing issues such as scalability, adaptability to evolving software environments, and the integration of SFP tools into existing software development workflows.

Abbreviations

ACO	Ant colony optimization
ARNN	Adaptive recurrent neural network
ACOFS	Ant colony optimization feature selection
ANN	Artificial neural network
ABC	Artificial bee colony
ALO	Ant-lion optimizer
BP	Backpropagation algorithm
BA	Bat-based search algorithm
BBBC	Big-bang big-crunch
BCE	Binary cross-entropy
BBO	Biogeography-based optimizer
BH	Black hole
BWO	Black widow optimization

CSS	Charged system search
CI	Choquet integral
CFS	Co-relation-based feature selection
CBO	Coupling between objects
CS	Cuckoo search
DT	Decision tree
DNN	Deep neural network
DE	Differential evolution
DEFS	Differential evolution feature selection
EP	Evolutionary programming
ES	Evolutionary strategy
EMA	Exchange market algorithm
FCNN	Faster convolutional neural networks
FA	Firefly algorithm
GbSA	Galaxy-based search algorithm
GA	Genetic algorithm
GPGSA	Genetic programming gravitational search algorithm
HS	Harmony search
HHO	Harris hawks optimization
IoT	Internet of Things
JA	Jaya algorithm
KNN	K-nearest neighbors
LCA	League championship algorithm
LICS	Levy-flight integrated cuckoo search optimization
LOC	Lines of code
LOA	Lion optimization algorithm
LR	Logistic regression
MI	Maintainability index
MFA	Memetic firefly algorithm
MASP	Model for assisted software process
MFO	Moth-flame optimization
MLP	Multilayer perceptron
MVO	Multi-verse optimization
NRMSE	Normalized root mean square error
NB	Naive bayes
OO	Object-oriented
PCA	Principal component analysis
PSO	Particle swarm optimization
RF	Random forest algorithm
QPSO	Quantum theory-particle swarm optimization
ROI	Return on investment
SA	Simulated annealing
SFLA	Shuffled frog leaping algorithm
SVM	Support vector machine
SFP	Software fault prediction
SSA	Sparrow search algorithm
SLR	Systematic literature review
TLBO	Teaching-learning-based optimization
WOA	Whale optimization algorithm

Acknowledgements
Not applicable.

Authors' contributions
ZD contributed to writing the draft, editing the manuscript, and conceptualizing the research. HW contributed to supervising the research, formatting the manuscript, and read and approved the final manuscript.

Funding
No funding.

Availability of data and materials
Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 23 July 2024 Accepted: 16 September 2024
Published online: 27 September 2024

References

1. Hayyolalam V et al (2022) Single-objective service composition methods in cloud manufacturing systems: recent techniques, classification, and future trends. Concurrency and Computation: Practice and Experience 34(5):e6698
2. Pourghebleh, B., et al., *A roadmap towards energy-efficient data fusion methods in the Internet of Things.* Concurrency and Computation: Practice and Experience, 2022: p. e6959.
3. Sakhnini J et al (2021) Security aspects of Internet of Things aided smart grids: a bibliometric survey. Internet of things 14:100111
4. Manchala P, Bisi M (2022) Diversity based imbalance learning approach for software fault prediction using machine learning models. Appl Soft Comput 124:109069
5. Rathi, S.C., et al., *Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction.* Expert Systems with Applications, 2023: p. 119806.
6. Thirukonda Krishnamoorthy Sivakumar Babu, R.B., S. Sivasubramanian, and S. Natarajan, *MLPNN-RF: software fault prediction based on robust weight based optimization and Jacobian adaptive neural network*. Concurrency and Computation: Practice and Experience, 2022. 34(21): p. e7122.
7. Shafiq, M., et al., *Scientific programming using optimized machine learning techniques for software fault prediction to improve software quality.* IET Software, 2023.
8. Chatterjee S, Roy A (2014) Web software fault prediction under fuzzy environment using MODULO-M multivariate overlapping fuzzy clustering algorithm and newly proposed revised prediction algorithm. Appl Soft Comput 22:372–396
9. Chen G et al (2015) A lightweight software fault-tolerance system in the cloud environment. Concurrency and Computation: Practice and Experience 27(12):2982–2998
10. Jin C, Jin S-W (2015) Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization. Appl Soft Comput 35:717–725
11. García Nieto PJ et al (2015) Hybrid PSO–SVM-based method for forecasting of the remaining useful life for aircraft engines and evaluation of its reliability. Reliab Eng Syst Saf 138:219–231
12. Arar ÖF, Ayan K (2016) Deriving thresholds of software metrics to predict faults on open source software: replicated case studies. Expert Syst Appl 61:106–121
13. Hryszko J, Madeyski L (2017) Assessment of the software defect prediction cost effectiveness in an industrial project. Software Engineering: Challenges and Solutions. Springer, pp 77–90
14. Singh, P., et al., *Fuzzy rule-based approach for software fault prediction.* IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2016.
15. Erturk E, Sezer EA (2015) A comparison of some soft computing methods for software fault prediction. Expert Syst Appl 42(4):1872–1879
16. Kumar, L., S. Misra, and S.K. Rath, *An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes.* Computer Standards & Interfaces, 2017.
17. Anju A, Judith J (2024) Hybrid feature selection method for predicting software defect. J Eng Appl Sci 71(1):124
18. Deng P et al (2015) An integrated framework of formal methods for interaction behaviors among industrial equipments. Microprocess Microsyst 39(8):1296–1304
19. Chen Y et al (2017) Application of fault tree analysis and fuzzy neural networks to fault diagnosis in the Internet of Things (IoT) for Aquaculture. Sensors 17(1):153
20. Arora I, Tetarwal V, Saha A (2015) Open issues in software defect prediction. Procedia Computer Science 46:906–912
21. Liu F, Zhou Z (2014) An improved QPSO algorithm and its application in the high-dimensional complex problems. Chemom Intell Lab Syst 132:82–90
22. Czibula G, Marian Z, Czibula IG (2014) Software defect prediction using relational association rule mining. Inf Sci 264:260–278
23. Wu, Y. and R. Yang. *Software reliability modeling based on SVM and virtual sample*. in *Reliability and Maintainability Symposium (RAMS), 2013 Proceedings - Annual*. 2013.
24. Catal C, Diri B (2009) A systematic review of software fault prediction studies. Expert Syst Appl 36(4):7346–7354
25. Mauša, G. and T.G. Grbac, *Co-evolutionary multi-population genetic programming for classification in software defect prediction: an empirical case study.* Applied Soft Computing, 2017.
26. Kaur, I., G.S. Narula, and V. Jain, *Differential analysis of token metric and object oriented metrics for fault prediction.* International Journal of Information Technology, 2017: p. 1–8.
27. Wu, X. and H. Zhu, *Formalization and analysis of the REST architecture from the process algebra perspective.* Future Gener. Comput. Syst., 2016. 56(C): p. 153–168.
28. Denaro, G., et al., *Deriving models of software fault-proneness*, in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. 2002, ACM: Ischia, Italy. p. 361–368.
29. Baier, C. and J.-P. Katoen, *Principles of model checking (representation and mind series)*. 2008: The MIT Press. 975.
30. Bowes, D., T. Hall, and J. Petrić, *Software defect prediction: do different classifiers find the same defects?* Software Quality Journal, 2017: p. 1–28.
31. Ali A, Gravino C (2021) An empirical comparison of validation methods for software prediction models. Journal of Software: Evolution and Process 33(8):e2367
32. Pourghebleh B, Navimipour NJ (2017) Data aggregation mechanisms in the Internet of things: a systematic review of the literature and recommendations for future research. J Netw Comput Appl 97:23–34
33. Catal C, Sevim U, Diri B (2011) Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. Expert Syst Appl 38(3):2347–2353
34. Hajimirzaei B, Navimipour NJ (2019) Intrusion detection for cloud computing using neural networks and artificial bee colony optimization algorithm. ICT Express 5(1):56–59
35. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179(13):2232–2248
36. Hassanzadeh, H.R. and M. Rouhani. *A multi-objective gravitational search algorithm*. in *2010 2nd international conference on computational intelligence, communication systems and networks*. 2010. IEEE.

37. Shah-Hosseini H (2011) Principal components analysis by the galaxy-based search algorithm: a novel metaheuristic for continuous optimisation. Int J Comput Sci Eng 6(1–2):132–140
38. Novák P et al (2013) RepeatExplorer: a galaxy-based web server for genome-wide characterization of eukaryotic repetitive elements from next-generation sequence reads. Bioinformatics 29(6):792–793
39. Mirjalili S, Mirjalili SM, Hatamlou A (2016) Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl 27(2):495–513
40. Tandu, C., et al. *A two-fold multi-objective multi-verse optimization-based time series forecasting*. in *Proceedings of the Seventh International Conference on Mathematics and Computing*. 2022. Springer.
41. Kirkpatrick, S., C.D. Gelatt Jr, and M.P. Vecchi, *Optimization by simulated annealing.* science, 1983. 220(4598): p. 671–680.
42. Friesz TL et al (1993) The multiobjective equilibrium network design problem revisited: a simulated annealing approach. Eur J Oper Res 65(1):44–57
43. Erol OK, Eksin I (2006) A new optimization method: big bang–big crunch. Adv Eng Softw 37(2):106–111
44. Singh R, Verma H (2012) Multi-objective big bang–big crunch optimization algorithm for recursive digital filter design. Int J Eng Innov Res (IJEIR) 1(2):194–200
45. Kaveh A, Talatahari S (2010) A novel heuristic optimization method: charged system search. Acta Mech 213(3):267–289
46. Kaveh A, Laknejadi K (2011) A novel hybrid charge system search and particle swarm optimization method for multi-objective optimization. Expert Syst Appl 38(12):15475–15488
47. Kumar S, Datta D, Singh SK (2015) Black hole algorithm and its applications. Computational intelligence applications in modeling and control. Springer, pp 147–170
48. Wu, C., et al., *AMOBH: adaptive multiobjective black hole algorithm.* Computational intelligence and neuroscience, 2017. 2017.
49. Geem, Z.W., J.H. Kim, and G.V. Loganathan, *A new heuristic optimization algorithm: harmony search.* simulation, 2001. 76(2): p. 60–68.
50. Sivasubramani S, Swarup K (2011) Multi-objective harmony search algorithm for optimal power flow problem. Int J Electr Power Energy Syst 33(3):745–752
51. Rao R (2016) Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int J Ind Eng Comput 7(1):19–34
52. Rao RV et al (2016) A new multi-objective Jaya algorithm for optimization of modern machining processes. Advances in Production Engineering & Management 11(4):271
53. Ghorbani N, Babaei E (2014) Exchange market algorithm. Appl Soft Comput 19:177–187
54. Ghorbani N, Babaei E, Sadikoglu F (2017) Exchange market algorithm for multi-objective economic emission dispatch and reliability. Procedia computer science 120:633–640
55. Rao RV, Savsani VJ, Vakharia D (2011) Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput Aided Des 43(3):303–315
56. Zou F et al (2013) Multi-objective optimization using teaching-learning-based optimization algorithm. Eng Appl Artif Intell 26(4):1291–1300
57. Kashan, A.H. *League championship algorithm: a new algorithm for numerical function optimization*. in *2009 international conference of soft computing and pattern recognition*. 2009. IEEE.
58. Subbaraj S, Thiagarajan R, Rengaraj M (2020) Multi-objective league championship algorithm for real-time task scheduling. Neural Comput Appl 32(9):5093–5104
59. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39
60. Gao Y et al (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci 79(8):1230–1242
61. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67
62. Aziz MAE, Ewees AA, Hassanien AE (2018) Multi-objective whale optimization algorithm for content-based image retrieval. Multimedia tools and applications 77(19):26135–26172
63. Mirjalili S (2015) The ant lion optimizer. Adv Eng Softw 83:80–98
64. Mirjalili S, Jangir P, Saremi S (2017) Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems. Appl Intell 46(1):79–95
65. Yang X-S (2010) Firefly algorithm, stochastic test functions and design optimisation. International journal of bio-inspired computation 2(2):78–84
66. Yang X-S (2013) Multiobjective firefly algorithm for continuous optimization. Engineering with Computers 29(2):175–184
67. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471
68. Akbari R et al (2012) A multi-objective artificial bee colony algorithm. Swarm Evol Comput 2:39–52
69. Kennedy, J. and R. Eberhart. *Particle swarm optimization*. in *Proceedings of ICNN'95-international conference on neural networks*. 1995. IEEE.
70. Coello, C.C. and M.S. Lechuga. *MOPSO: a proposal for multiple objective particle swarm optimization*. in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*. 2002. IEEE.
71. Yang, X.-S. and S. Deb. *Cuckoo search via Lévy flights*. in *2009 World congress on nature & biologically inspired computing (NaBIC)*. 2009. Ieee.
72. Yang X-S, Deb S (2013) Multiobjective cuckoo search for design optimization. Comput Oper Res 40(6):1616–1624
73. Heidari AA et al (2019) Harris hawks optimization: algorithm and applications. Futur Gener Comput Syst 97:849–872
74. Du, P., et al., *A novel hybrid model based on multi-objective Harris hawks optimization algorithm for daily PM2. 5 and PM10 forecasting.* Applied Soft Computing, 2020. 96: p. 106620.

75. Hayyolalam V, Kazem AAP (2020) Black widow optimization algorithm: a novel meta-heuristic approach for solving engineering optimization problems. Eng Appl Artif Intell 87:103249
76. Rahbari, M., et al., *A risk-based green location-inventory-routing problem for hazardous materials: NSGA II, MOSA, and multi-objective black widow optimization.* Environment, Development and Sustainability, 2021: p. 1–37.
77. Koza JR, Poli R (2005) Genetic programming. Search methodologies. Springer, pp 127–164
78. Zhao H (2007) A multi-objective genetic programming approach to developing pareto optimal decision trees. Decis Support Syst 43(3):809–826
79. Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4(2):65–85
80. Murata T, Ishibuchi H, MOGA: multi-objective genetic algorithms. in IEEE international conference on evolutionary computation. (1995) IEEE Piscataway. NJ, USA
81. Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12(6):702–713
82. Jamuna K, Swarup K (2012) Multi-objective biogeography based optimization for optimal PMU placement. Appl Soft Comput 12(5):1503–1510
83. Fleetwood, K. *An introduction to differential evolution*. in *Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia*. 2004.
84. Xue, F., A.C. Sanderson, and R.J. Graves. *Pareto-based multi-objective differential evolution*. in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* 2003. IEEE.
85. Alavi M, Henderson JC (1981) An evolutionary strategy for implementing a decision support system. Manage Sci 27(11):1309–1323
86. Binh, T.T. and U. Korn. *MOBES: a multiobjective evolution strategy for constrained optimization problems*. in *The third international conference on genetic algorithms (Mendel 97)*. 1997.
87. Zhang J-H, Xu X-H (1999) An efficient evolutionary programming algorithm. Comput Oper Res 26(7):645–663
88. Meza JLC, Yildirim MB, Masud AS (2009) A multiobjective evolutionary programming algorithm and its applications to power generation expansion planning. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 39(5):1086–1096
89. Eusuff M, Lansey K, Pasha F (2006) Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. Eng Optim 38(2):129–154
90. Arshi SS, Zolfaghari A, Mirvakili S (2014) A multi-objective shuffled frog leaping algorithm for in-core fuel management optimization. Comput Phys Commun 185(10):2622–2628
91. Khatri, Y. and S.K. Singh, *An effective software cross-project fault prediction model for quality improvement*. Science of Computer Programming, 2023: p. 102918.
92. Gill GK, Kemerer CF (1991) Cyclomatic complexity density and software maintenance productivity. IEEE Trans Softw Eng 17(12):1284–1288
93. Khoshgoftaar TM, Allen EB (1998) Classification of fault-prone software modules: prior probabilities, costs, and model evaluation. Empir Softw Eng 3(3):275–298
94. Gondra I (2008) Applying machine learning to software fault-proneness prediction. J Syst Softw 81(2):186–195
95. Khoshgoftaar TM, Allen EB (1999) A comparative study of ordering and classification of fault-prone software modules. Empir Softw Eng 4(2):159–186
96. Khoshgoftaar TM, Munson JC (1990) Predicting software development errors using software complexity metrics. IEEE J Sel Areas Commun 8(2):253–261
97. Khoshgoftaar TM, Lanning DL, Pandya AS (2006) A comparative study of pattern recognition techniques for quality evaluation of telecommunications software. IEEE J Sel A Commun 12(2):279–291
98. Lehman, M.M., D.E. Perry, and J.F. Ramil. *Implications of evolution metrics on software maintenance*. in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. 1998.
99. Li HF, Cheung WK (1987) An empirical study of software metrics. IEEE Trans Softw Eng 13(6):697–708
100. Jolliffe, I., *Principal component analysis*. Wiley StatsRef: Statistics Reference Online, 2002.
101. Neumann DE (2002) An enhanced neural network technique for software risk analysis. IEEE Trans Software Eng 28(9):904–912
102. Xing, F., P. Guo, and M.R. Lyu, *A novel method for early software quality prediction based on support vector machine*, in *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*. 2005, IEEE Computer Society. p. 213–222.
103. Catal, C. and B. Diri, *Software defect prediction using artificial immune recognition system*, in *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*. 2007, ACTA Press: Innsbruck, Austria. p. 285–290.
104. Vandecruys O et al (2008) Mining software repositories for comprehensible software fault prediction models. J Syst Softw 81(5):823–839
105. Dejaeger K, Verbraken T, Baesens B (2013) Toward comprehensible software fault prediction models using bayesian network classifiers. IEEE Trans Software Eng 39(2):237–257
106. Rajaganapathy C, Subramani A (2015) A comparative study of different software fault prediction and classification techniques. Res J Appl Sci Eng Technol 10(7):831–840
107. De Carvalho AB, Pozo A, Vergilio SR (2010) A symbolic fault-prediction model based on multiobjective particle swarm optimization. J Syst Softw 83(5):868–882
108. Kwok, J., B.-L. Lu, and L. Zhang, *Advances in neural networks--ISNN 2010: 7th International Symposium on Neural Networks, ISNN 2010, Shanghai, China, June 6–9, 2010, Proceedings*. Vol. 6063. 2010: Springer.
109. Malhotra R, Kaur A, Singh Y (2010) Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. International Journal of System Assurance Engineering and Management 1(3):269–281
110. Monden, A., et al. *A heuristic rule reduction approach to software fault-proneness prediction*. in *2012 19th Asia-Pacific Software Engineering Conference*. 2012.
111. Abdi Y, Parsa S, Seyfari Y (2015) A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction. Innov Syst Softw Eng 11(4):289–301

112. Pourghebleh, B., et al., *The importance of nature-inspired meta-heuristic algorithms for solving virtual machine consolidation problem in cloud environments.* Cluster Computing, 2021: p. 1–24.
113. Pourghebleh B, Hayyolalam V, Anvigh AA (2020) Service discovery in the Internet of Things: review of current trends and research challenges. Wireless Netw 26(7):5371–5391
114. Kitchenham B et al (2009) Systematic literature reviews in software engineering–a systematic literature review. Inf Softw Technol 51(1):7–15
115. Comert SE, Yazgan HR (2023) A new approach based on hybrid ant colony optimization-artificial bee colony algorithm for multi-objective electric vehicle routing problems. Eng Appl Artif Intell 123:106375
116. Farshidpour S, Keynia F (2012) Using artificial bee colony algorithm for MLP training on software defect prediction. Oriental Journal of Computer Science & Technology 5(2):231–239
117. Huang L, Ai J (2015) Automatic software fault localization based on artificial bee colony. J Syst Eng Electron 26(6):1325–1332
118. Hussein MK, Mousa MH (2020) Efficient task offloading for iot-based applications in fog computing using ant colony optimization. IEEE Access 8:37191–37201
119. Kumar K, Gyani DJ, Narsimha G (2018) Software defect prediction using ant colony optimization. Int J Appl Eng Res 13(19):14291–14297
120. Manivasagam, G. and R. Gunasundari, *An optimized feature selection using fuzzy mutual information based ant colony optimization for software defect prediction.* International Journal of Engineering & Technology, 2018. **7**(1.1): p. 456–460.
121. Singh P, Verma S (2020) ACO based comprehensive model for software fault prediction. International Journal of Knowledge-based and Intelligent Engineering Systems 24(1):63–71
122. Azar D, Vybihal J (2011) An ant colony optimization algorithm to improve software quality prediction models: case of class stability. Inf Softw Technol 53(4):388–393
123. Mohanthy, R., V. Naik, and A. Mubeen. *Software reliability prediction by using ant colony optimization technique*. in *2014 Fourth International Conference on Communication Systems and Network Technologies*. 2014. IEEE.
124. Mondal, S., et al. *Software fault prediction using wrapper based ant colony optimization algorithm for feature selection*. in *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*. 2023. IEEE.
125. Wahdan, H.G., S.S. Kassem, and H.M. Abdelsalam. *Product modularization using cuckoo search algorithm*. in *International Conference on Operations Research and Enterprise Systems*. 2016. Springer.
126. Song, K., et al., *Software defect prediction based on elman neural network and cuckoo search algorithm.* Mathematical Problems in Engineering, 2021. 2021.
127. Cai X et al (2020) An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. Concurrency and Computation: Practice and Experience 32(5):e5478
128. Badvath D et al (2022) Prediction of software defects using deep learning with improved cuckoo search algorithm. Concurrency and Computation: Practice and Experience 34(26):e7305
129. Rath, P.K., et al. *CSOFS: feature selection using cuckoo search optimization algorithm for software fault detection*. in *2024 International Conference on Emerging Systems and Intelligent Computing (ESIC)*. 2024. IEEE.
130. Yang, L., et al., *An analytical model of page dissemination for efficient big data transmission of C-ITS.* IEEE Transactions on Intelligent Transportation Systems, 2021.
131. Anbu, M. and G. Anandha Mala, *Feature selection using firefly algorithm in software defect prediction.* Cluster Computing, 2019. 22(5): p. 10925–10934.
132. Yenduri G, Gadekallu TR (2021) Firefly-based maintainability prediction for enhancing quality of software. Internat J Uncertain Fuzziness Knowledge-Based Systems 29(Suppl 2):211–235
133. Pemmada SK, Nayak J, Naik B (2023) A deep intelligent framework for software risk prediction using improved firefly optimization. Neural Comput Appl 35(26):19523–19539
134. Karimi, A., M. Irajimoghaddam, and E. Bastami, *Feature selection using combination of genetic-whale-ant colony algorithms for software fault prediction by machine learning.* Journal of Electronical & Cyber Defence, 2022. 10(1).
135. Jin, C. and E. Dong. *Software defect prediction using fuzzy integral and genetic algorithm*. in *Software Engineering and Information Technology: Proceedings of the 2015 International Conference on Software Engineering and Information Technology (SEIT2015)*. 2016. World Scientific.
136. Fazel FS (2016) A new method to predict the software fault using improved genetic algorithm. Bull Soc Roy Sci Liège 85:187–202
137. Nalini, C. and T.M. Krishna. *An efficient software defect prediction model using neuro evalution algorithm based on genetic algorithm*. in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. 2020. IEEE.
138. Kaliraj, S. and Y. Reddy, *Software fault prediction using an optimised feature selection process based on a genetic algorithm.* International Journal on Engineering Applications, 2023. 11(5).
139. Gupta M, Rajnish K, Bhattacharjee V (2024) Software fault prediction with imbalanced datasets using SMOTE-Tomek sampling technique and Genetic Algorithm models. Multimedia Tools and Applications 83(16):47627–47648
140. Moussa R, Azar D (2017) A PSO-GA approach targeting fault-prone software modules. J Syst Softw 132:41–49
141. Ibrahim, D.R., R. Ghnemat, and A. Hudaib. *Software defect prediction using feature selection and random forest algorithm*. in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*. 2017. IEEE.
142. Yang L et al (2021) Software defects prediction based on hybrid particle swarm optimization and sparrow search algorithm. IEEE Access 9:60865–60879
143. Alsghaier, H. and M. Akour, *Software fault prediction using whale algorithm with genetics algorithm.* Software: Practice and Experience, 2021. 51(5): p. 1121–1146.
144. Anju A, Judith J (2023) Adaptive recurrent neural network for software defect prediction with the aid of quantum theory-particle swarm optimization. Multimedia Tools and Applications 82(11):16257–16278
145. Alsghaier, H. and M. Akour, *Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier.* Software: Practice and Experience, 2020. 50(4): p. 407–427.

146. Goyal S, Bhatia PK (2021) Software fault prediction using lion optimization algorithm. Int J Inf Technol 13(6):2185–2190
147. Tumar I et al (2020) Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction. IEEE Access 8:8041–8055
148. Anjali, C., J.P.M. Dhas, and J. Singh, *Moth flame optimization based FCNN for prediction of bugs in software.* Intelligent Automation & Soft Computing, 2023. 36(2).
149. Wahono RS, Suryana N (2013) Combining particle swarm optimization based feature selection and bagging technique for software defect prediction. International Journal of Software Engineering and Its Applications 7(5):153–166
150. Malhotra, R., et al. *Software defect prediction using binary particle swarm optimization with binary cross entropy as the fitness function*. in *Journal of Physics: Conference Series*. 2021. IOP Publishing.
151. Hassouneh Y et al (2021) Boosted whale optimization algorithm with natural selection operators for software fault prediction. IEEE Access 9:14239–14258
152. Halstead, M.H., *Elements of software science (operating and programming systems series)*. 1977: Elsevier Science Inc.
153. McCabe TJ (1976) A complexity measure. IEEE Trans Software Eng 4:308–320

Publisher's Note