Soft Computing (2024) 28:1503–1544
https://doi.org/10.1007/s00500-023-08382-8

APPLICATION OF SOFT COMPUTING

Nature-inspired metaheuristic methods in software testing

Niloofar Khoshniat[1] · Amirhossein Jamarani[2] · Ahmad Ahmadzadeh[1] · Mostafa Haghi Kashani[2] · Ebrahim Mahdipour[1]

Accepted: 28 April 2023 / Published online: 8 June 2023

Abstract

Software quality is becoming a momentous challenge in software engineering processes, and *software testing* has a pivotal role in its measurements. Nature-inspired metaheuristic methods play an essential role in software testing, in which various studies have been conducted in this field; however, due to a lack of wide-ranging papers reviewing these methods, conducting a comprehensive systematic review to examine an array of crucial mechanisms in this field has become a necessity. This study aims to present a detailed analysis and taxonomically classifies the metaheuristic methods inspired by nature. This paper compromises a systematic literature review of 65 chosen studies published between 2015 and 2022. Genetic algorithm-based, hybrid, ant colony optimization-based, cuckoo search-based, firefly algorithm-based, artificial bee colony-based, and other metaheuristic methods constitute this systematic study's stratification. Evaluation methods, applied tools, merits, and demerits of each reviewed article are investigated. Additionally, future directions and open issues are addressed. This conducted paper not only expounds on software testing strengths, open issues, and future works, but also recognizes the quest for optimizing the insufficient metrics in software testing, such as mutation score, complexity, and scalability, which would be the propulsion of the testing process if consummated.

Keywords Software · Testing · Metaheuristic · Bio-inspired

1 Introduction

Software history goes back to *Alan Turing*, known as the father of software (Carpenter and Doran 1986). He proposed a theory in 1935 about Software in a paper which in turn led to the introduction of two novel academic fields called "computer science" and "software engineering" (Turing 1937). The theoretical concepts and software engineering, whose main focus is on using software development in practice, are both important building blocks of computer science (Malhotra 2016). In general, the collection of data or computer instructions that define the computer's working path and process calls computer software. Methods (Zuse 2019), processes (Fuggetta 2000), tools (Kernighan and Plauger 1976), and quality (Galin 2004) are the core components of software engineering. As software engineering is multiplying in this decade and there is a wide variety of novel methods, means, models, and concepts available, the potential for this amazing industry is steadily growing (Basili 1989).

Every year in the United States, software projects lead companies to invest billions of dollars in software development (Charette 2005; Collins et al. 2012). Also, 50 percent of the total cost in a typical programming project is devoted to testing the program or developing the system (Myers 2006). However, software companies require quality assurance engineers to control and reduce software implementation and maintenance costs. The most frequent

✉ Mostafa Haghi Kashani
mh.kashani@iau.ac.ir

Niloofar Khoshniat
niloofar.khoshniatt@gmail.com

Amirhossein Jamarani
ah.jamarani@qodsiau.ac.ir

Ahmad Ahmadzadeh
ahmet.ahmadzade@gmail.com

Ebrahim Mahdipour
mahdipour@srbiau.ac.ir

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

[2] Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran

method used by far to ensure and control quality in the process of software development is software testing (Burnstein et al. 1996). There are many reasons why test algorithms have become a software engineering priority. The development of technology, lack of time, limitation of resources, funding, and expenses imposed upon hiring professional staff are some reasons that can explain such prioritization (Nawaz and Malik 2008).

Also, to fulfill customers' expectations, there is a need for secure environments for the customized codes, which is another reason that companies prioritize testing practices. Software testing faces an issue of combinatorial optimization having NP-hard complexity, which is a severe challenge in software testing. Owing to the innate feature of being an NP-hard problem, the metaheuristic, in recent years, nature-inspired metaheuristic algorithms have presented viable solutions in software testing optimization, test data generation, test coverage, and test data efficiency, and quality. Nature-inspired metaheuristic methods in software have been employed to reach adequate suboptimal solutions for nature-inspired-based problems. In particular, search-based software testing is defined as the use of metaheuristic optimization techniques to partially or fully automate software testing tasks, such as test selection, generation, and prioritization (Harman et al. 2015). Recently, various nature-inspired metaheuristic algorithms have been presented in the papers and implemented in various applications of real-life nature.

Metaheuristic methods are global optimization devices typically aiming to model a natural phenomenon. They generate reasonable approximate solutions in sensibly bounded computation times (Bianchi et al. 2009; Kashani and Sarvizadeh 2011; Nikravan et al. 2011; Sarvizadeh et al. 2012). Specifically, in case there are incomplete problems or noisy and non-continuous ones, metaheuristics are preferred over analytical methods that make assumptions numerously about the problem surface. Compared with traditional methods used previously in research studies, these methods introduce many advantages, one of which is reasonable solutions are found and the computational effort needed is less than it used to be; also, they can even move quickly to achieve even better solutions. Hence, in dealing with complicated problems that are also large, employing these methods can be pretty efficient. Another advantage is that they are also worthwhile because these methods do not get trapped in local minima in contrast with traditional methods. The common application for these techniques is in combinatorial optimization problems (Bianchi et al. 2009; Yang 2010a).

Testing software with optimized and analytical methods can be a part of the software development lifecycle to lower costs. The primary purpose of this review is to research papers focused on nature-inspired metaheuristic methods for software testing. The algorithms that can be used in automated testing enable increased productivity and lower costs for different software environments. By looking at the most commonly used metaheuristic algorithms and introducing their strengths and weaknesses, this paper also explains the most potent metaheuristic methods. It compares the differences among all of them, examining the numerous widespread software testing with these methods. And, finally an overview of the types of challenges that could be faced due to the significant role that software testing plays; Therefore, in this study, a Systematic Literature Review (SLR) about software testing methods is presented, and the focus is on the presented classification solutions. In this regard, we have made a considerable effort to answer the following study questions:

- What classification of research techniques can be possibly utilized for software test methods?
- What evaluation metrics can be used for software test methods?
- What are the evaluation techniques for software test methods?
- What tools are used to evaluate the selected papers in software test methods?
- What are the future challenges and open perspectives in using algorithms for software test methods?

To prepare this state-of-the-art review, we imitated the general SLRs instructions represented in (Brereton et al. 2007; Kitchenham 2004). Our obvious target is to have a reasonably exhaustive and detailed comparison to critically analyze major threats, limitations, and unfulfilled potentials in software testing by using nature-inspired metaheuristic methods. For compiling this SLR, we have predominantly concentrated on reasonably suggested advanced techniques, optimal solutions, and nature-inspired algorithms in software testing. In this regard, we have reviewed 65 recently published articles in-depth. In addition, the reviewed papers were segregated, categorized, and analogized by directly applying the provided idiosyncratic taxonomy. The taxonomy is formed by seven main categories, namely genetic algorithm-based methods, hybrid methods, ant colony optimization-based methods, cuckoo search-based methods, firefly algorithm-based methods, artificial bee colony-based methods, and other metaheuristic methods. Although there are currently review papers that have worked on different nature-inspired algorithms, the overwhelming majority of them did not carefully consider the function of bio-inspired methods in software testing, principal issues, future works, merits, and demerits. To successfully curb the inadequacies in previous articles, this SLR is presented and its main contributions are set as follows:

- Presenting a systematic study of nature-inspired metaheuristic algorithms in software testing methods.
- Determining a technical taxonomy to categorize the nature-inspired metaheuristic algorithms in the software test methods.
- Preparing a side-by-side comparison between algorithm types, tools, evaluation methods, advantages, and disadvantages approaches.
- Defining the challenges and critical areas in which future studies could enhance the function of software testing methods.

The rest of this paper is arranged as Fig. 1 illustrates. A survey of the relevant work is presented in Sect. 2. In Sect. 3, the research methodology is described. Section 4 discusses the classification and comparison of software testing methods. Also, an analysis of the results is presented in Sect. 5. Section 6 aims to present the open issues and future directions. Section 7 describes the limitations of this systematic review. And lastly, in Sect. 8, we draw conclusions and limitations. In addition, the frequently used abbreviation in the paper is given in Table 1.

2 Related work and motivation

The relevant review papers on nature-inspired metaheuristic methods in software testing are outlined in this section. However, the selected literature reviews on the subject have different drawbacks, which we have noted in Sect. 2.1.3.

2.1 Review studies on nature-inspired metaheuristic methods in software testing

This section provides an apt summary of background information on software test methods to prepare a platform to compare the various recent related works. The reviewed studies are divided into two categories: survey and SLR. In addition, Table 2 displays an outline of the analyzed surveys and SLRs. The parameters like type of review, covered years, main topic, issuing year, the process of paper selection, taxonomy, and future work of each study are illustrated in this table.

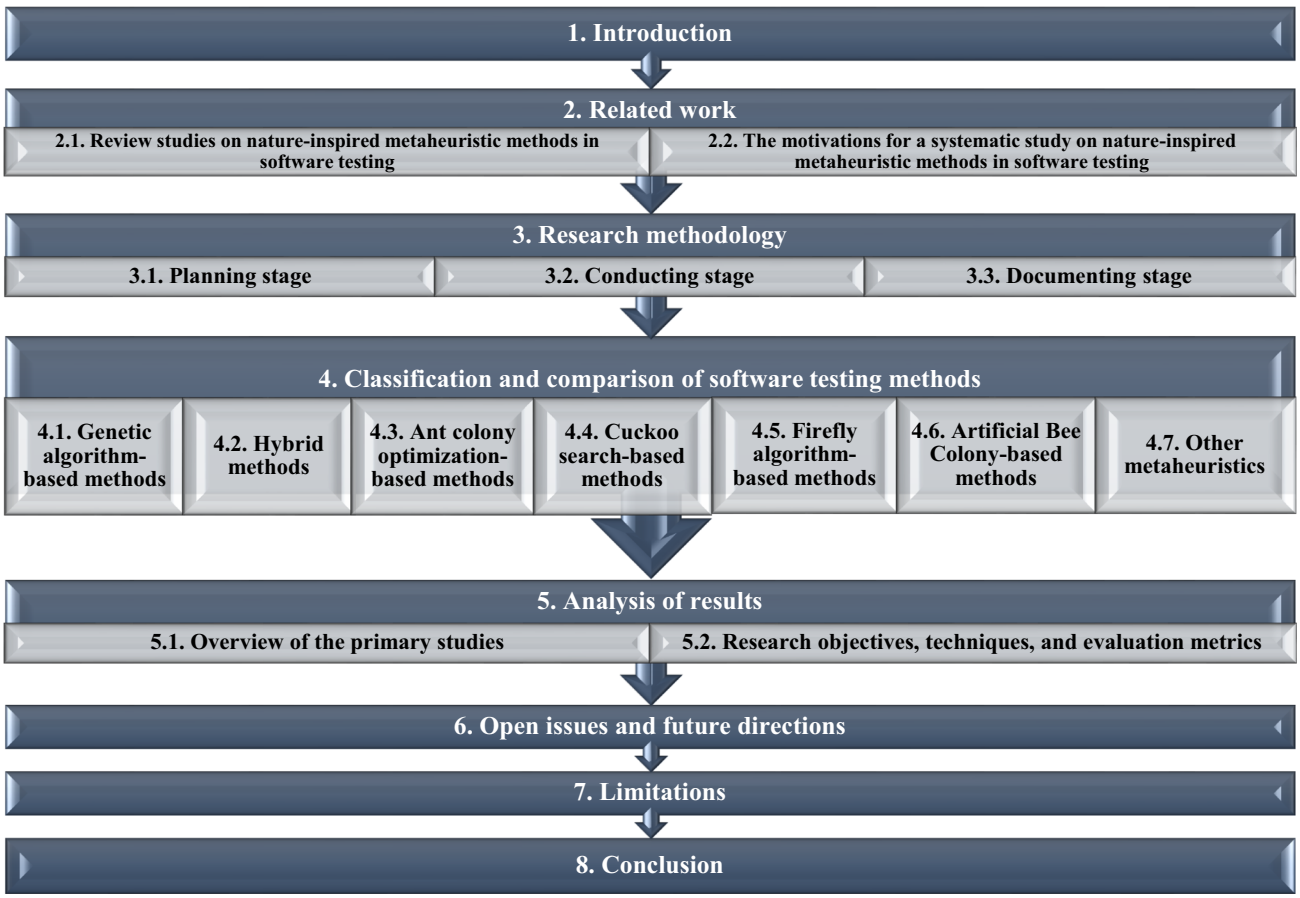


Fig. 1 Structure of this review

Table 1 Abbreviation table

Abbreviation	Definition
GA	Genetic Algorithm
ACO	Ant Colony Optimization
CS	Cuckoo Search
RG	Regenerate Genetic
NSGA	Nondominated Sorting Genetic Algorithm
GLS	Genetic Local Search
EA	Evolutionary Algorithm
ES	Evolutionary Strategy
MA	Mutation Analysis
PSO	Particle Swarm Optimization
IBEA	Indicator-Based Evolutionary Algorithm
SPEA	Strength Pareto Evolutionary Algorithm
SRGM	Software Reliability Growth Model
TEF	Testing Effort Function
CSC	Convergence Speed Controller
K-A	Kapur and Aggarwal
SA	Simulated Annealing
WRAC	Weighted Rank Ant Colony
DD	Diversity Dragonfly
FA	Firefly Algorithm
33FP	Flower Pollination
ABC	Artificial Bee Colony
SFL	Shuffled Frog Leaping
WRACMO	Weighted Rank Ant Colony Metaheuristic Optimization
MG	Micro-Genetic Algorithm
BA	Bat-inspired Algorithm
NEAT	NeuroEvolution of Augmenting Topologies
CMSA	Construct, Merge, Solve and Adopt
BFA	Bacterial Foraging Algorithm
EMVO	Enhanced Multiverse Optimizer
AB-CNS	African Buffalo-based Convolution Neural Slicing
SO-SA	Simultaneous Operation Simulated Annealing
EA-CSC	Evolutionary Algorithm with Convergence Speed Controller
MGSO	Modified Genetic Swarm Optimization
ICA	Imperialist Competitive Algorithms
WOA	Whale Optimization Algorithm
SSA	Salp Swarm Algorithm
MCDM	Multi-Criteria Decision Making
BPNN	Back Propagation Neural Network
ESAMP-SMO	Elitist Self-Adaptive Multi-Population Social Mimic Optimization
BTLBOd	Binary Teaching–Learning-Based Optimization
CGWO	Grey Wolf Optimization Algorithm

2.1.1 Surveys

Suri and Singhal (2012) gave a review of ACO applied in software testing. ACO has been employed at different steps in the process of software testing, as the survey suggests. Also, Anand et al. (2013) reviewed an SLR that presents a well-organized study of the unique techniques of automatically generating software test cases which are surveyed in separate sections. The reviewed approaches are as follows: (1) structural testing, which utilizes figurative execution, (2) testing based on models, (3) combined testing, (4) random testing along with the slightly varying

Table 2 Related surveys in the field of software testing approaches

Review type	Paper	Main topic	Publication year	Paper selection process	Taxonomy	Future work	Covered years
Survey	Suri and Singhal (2012)	Software test process	2012	Yes	No	Presented	2003–2010
	Anand et al. (2013)	Software test automation	2013	No	Yes	Presented	Not mentioned
	Nanthaamornphong and Carver (2017)	Test-driven development	2015	Yes	No	Not presented	2002–2014
	Harman et al. (2015)	Software test process	2015	No	No	Presented	Not mentioned
	Rodrigues and Dias-Neto (2016)	Software test automation	2016	No	No	Presented	1999–2015
	Jamil et al. (2016)	Software testing techniques	2016	No	Yes	Presented	Not mentioned
	Khari and Kumar (2019)	Software test process	2017	Yes	Yes	Presented	1996–2016
	Prabhakar et al. (2019)	Software test process	2018	Yes	Yes	Not presented	Not mentioned
	Alkawaz and Silvarajoo (2019)	TCPAO in RT	2019	No	Yes	Not presented	Not mentioned
SLR	Yusop and Ibrahim (2011)	Software test process	2011	No	Yes	Not presented	2002–2009
	Mäntylä et al. (2015)	The omnipresence of rapid releases in software testing	2015	Yes	No	Presented	2006–2012
	de Souza et al. (2015)	KM in software testing	2015	Yes	No	Presented	2003–2013
	Afzal et al. (2016)	Software test process	2016	Yes	Yes	Presented	Not mentioned
	Garousi and Mäntylä (2016)	Software test process	2016	Yes	No	Presented	1994–2015
	Garousi et al. (2017)	Software test process	2017	Yes	No	Presented	1995–2015
	Wiklund et al. (2017)	Software test automation	2017	Yes	Yes	Presented	1999–2014
	Gillenson et al. (2018)	Software test process	2018	Yes	No	Not presented	1999–2017
	Garousi and Küçük (2018)	Smells in software testing	2018	Yes	No	Presented	Not mentioned
	Dadkhah et al. (2020)	Semantic web in software testing	2020	Yes	No	Presented	2000–2019
	Tebes et al. (2020)	Nature of software testing	2020	No	No	Presented	2003–2018
	Alyahya (2020)	CST	2020	Yes	No	Not presented	Up to 2018
	Bluemke and Malanowska (2021)	Software testing effort approximation	2021	Yes	No	Presented	2016–2019
	Barbosa et al. (2022)	Software test process	2022	Yes	No	Presented	Up to 2021
	Our paper	Software test algorithms	2023	Yes	Yes	Presented	2015–2022

technique called adaptive random testing, and (5) search-based testing. Every section in this survey is a contribution made by famous active researchers to the corresponding method. These sections, in a nutshell, cover the fundamental ideas relating to the technique, the current methods, and finally, both a discussion of the research challenges and a view of the future advancements regarding the given approach. By and large, the survey gives an introductory section followed by a brief review related to contemporary research in automatic test case generation in a way that does not impede the comprehensiveness and authoritativeness of the total paper. However, the paper does not mention future work.

Nanthaamornphong and Carver (2017) surveyed test-driven development (TDD) in the life cycle of software development. The results of this survey provide experimental evidence on the capability of TDD for scientific software development. The primary impact of TDD is to

improve functionality in terms of software quality features. The authors also discussed the effects of using TDD, the difficulties of using TDD, the developers' methods, and the developers' refactoring techniques. However, the drawbacks of the TDD were mentioned as difficulties, and they suggested some solutions, the paper has not considered future works, and the paper did not represent classification. Harman et al. (2015) reviewed the main challenges and accomplishments for search-based software testing. They preliminarily analyzed defects, fixed them, and authenticated their reformations. But, taxonomy, paper selection, and covered years were undefined due to the lack of systematic structure.

Moreover, Rodrigues and Dias-Neto (2016) reviewed an SLR in critical software automation life cycle factors. This study also analyzed the influence that critical success factors had on the plain lifecycle of software test automation. As the participants acknowledged, factors can impact phases differently and can be given priorities to attain better results and get fewer losses during the whole process. Practitioners who had better experiences tended not to give priority to direct technology-related factors. For future work, this study made some recommendations; but, the paper had no classifications, and the process for paper selection was vague.

Jamil et al. (2016) surveyed software testing methods to pay attention to the current approaches and observe testing techniques for optimum quality and affirmation. Despite their presented taxonomy and future work, their study did not note the paper selection process and covered years which stems from the lack of systematic formation. Khari and Kumar (2019) conducted a review paper to compare the major topics and methods in search-based software testing. They divided software testing techniques into eight distinct categories: unit, integration, system, validation, acceptance, regression, stress, and load testing. The authors presented metaheuristic methods: functional, structural, grey-box, and non-functional testing as taxonomy. Their survey, however, was not systematic and the procedure for the collection of the reviewed papers was not apparent.

Prabhakar et al. (2019) provided a survey of the results and applications of ACO and GAs in software testing. And, Alkawaz and Silvarajoo (2019) surveyed regression testing (RT) as a principal method for software testing. They discovered that RT needs to use test case prioritization and optimization (TCPAO) to reduce the execution time in software testing. Although they provided taxonomy, the selection process of the reviewed papers was not transparent; no future works were discussed, the covered years of their survey was not noted, and it was not an SLR.

2.1.2 Systematic literature reviews

Yusop and Ibrahim (2011) reviewed numerous maintenance testing approaches such as the RT suite method, keyword-based approach, heuristic techniques, GUI-based approach, and model-based approach, which were assessed with the help of the software development framework. In approaches studied in the paper, modifications of test cases are supported. After conducting the SLR, a handful of outcomes were assumed and emphasized the restriction of the current methods; but, the paper selection process was not systematically transparent. Mäntylä et al. (2015) investigated a semi-systematic literature review alongside a case study regarding software testing with fast-paced releases. Their case study was chosen to able them to review the influences of transmission from conventional methods to rapid releases on a system. Additionally, their literature focused on the constraints of software testing and how it can be ameliorated in future. de Souza et al. (2015) reviewed knowledge management (KM) fundamentals on software testing. They presented a study and discussed vital features of automation in software testing. Their study also indicated that KM plays an integral part in software testing not just because it would make testing effective, but it paves the way for selecting and utilizing test cases' techniques. To review the software test process improvement techniques were surveyed by Afzal et al. (2016). This study has included an SLR to recognize possible software test process improvement (STPI) techniques and a case study. It has categorized and discussed the methods in four categories: STPI and related approaches, TMM, and related approaches, distinct approaches standard, and related methods in the SLR. The case study was investigated to review a couple of approaches in the case organization, and some collateral evaluations of the testing steps were conducted by employing such approaches. In this case study, the approaches used tested maturity model integration and TPI NEXT. However, future works were not checked.

Garousi and Mäntylä (2016) overviewed the right moment for automating software testing and what needs to be done to test the software. They presented a type of SLR named multivocal literature review (MLR) to survey the software testing systematically. The authors reviewed decision support to compare the metrics in software testing to note whether decision support is beneficial for the industries. Their study, however, lacked a comprehensive taxonomy and covered years of their studied papers were unspecified. Also, Garousi et al. (2017) carried out an MLR to evaluate and enhance the test processes and make them more mature. They used and test process improvement (TPI) and test maturity assessment (TMA). This review identified 58 distinct test maturity models and many sources with different degrees of experimental confirmations on

such subjects. To synthesize the challenges, drivers and points of TMA/TPI from the principal sources, they conducted a qualitative analysis (coding). The paper shows that methods and maturity models in TMA/TPI which exist today provide sensible guidance for this industry and the community of researchers. However, the paper selection process ignored papers after the year 2015 and the review has no taxonomy.

Wiklund et al. (2017) reviewed an SLR, searching for publications explaining obstacles encountered in automated software testing. The chosen publications were examined to categorize the obstacles, collect the evidence, and make a high-quality synthesis of them all. The methods are explained and classified into six categories: environment configuration, inadequate development practices, quality issues, technical limitations, usability, IT environment, and system under test. This study aims to propose an illustrative model for the impediments and how they interact to specify how they relate to one another and, in large, to the software development and testing business. They suggested that creating a model for evaluations and enhancements primarily targeting software test automation is necessary. However, the disadvantages of the software test automation were not mentioned, and the paper selection process has ignored the papers after 2014.

Also, Gillenson et al. (2018) explored the significance and worth that software test cases suggest on an introductory level and then allowed researchers to conduct further research about techniques used in software tests. The function evaluation goes through a two-level evaluation. The first evaluation is an assessment on the theoretical level. In this assessment, the final value function is compared with some currently available estimation functions usually employed in software engineering by considering the facility, precision, and flexibility the function owns. The subsequent evaluation is the assessment on the practical level. In this assessment, the function is applied in a pragmatic environment, and the effect is analyzed. Software testers use this method to judge the test cases and relevant resource distribution used in software testing. But, the method of paper selection is ambiguous, and they did not mention future work either.

Garousi and Küçük (2018) employed an MLR to investigate smells in software testing codes. Their initiated scope was separated into two different areas: academia and industry. They summarized guidelines, methods, and tools to cope with the challenges of smells. Dadkhah et al. (2020) employed an SLR to review the connotation of web empowerment in software testing. The authors illustrated the plus points of software testing in both academia and industry. They also surveyed major challenges and limitless potentials in software testing. Nonetheless, their SLR did not provide a taxonomy. Tebes et al. (2020) reviewed software testing ontologies. They presented three main questions regarding reviewing software testing nature by providing a non-functional requirement tree. The authors determined the concepts of ontologies for software testing zone, classification, and correlation between periodic that included concepts and axioms.

Alyahya (2020) systematically reviewed the usage of crowdsourced software testing (CST). The author investigated the improvements in assessments and value of CST and considered the detected challenges in software testing. Bluemke and Malanowska (2021) prepared an SLR to investigate the computation of related testing effort estimation. Furthermore, they focused on finding solutions for test evaluation. Albeit the authors classified the included papers based on different research questions, they did not provide a taxonomy. Barbosa et al. (2022) provided an SLR concerning software test case prioritization (TCP). They searched through the published works up to 2021 and found out that TCPs have the potential to decrease the cost and time for software testing. In this sense, the authors took advantage of Markov chains in TCPs, which allowed them to introduce different techniques.

2.1.3 Concluding remark

We presented this work to include the weak points with all due respect to the aforementioned review papers. The deficiencies, which are resolved in our paper, include the following:

- Most reviewed papers are about software test case generation processes or automation.
- Most papers neither proposed any taxonomy nor classification.
- Some papers have not presented possibilities, trends, challenges, threats, and future works.
- In related papers, the lack of methodical construction has caused the selection process to be vague.
- In some studies, there is no mention of assessment tools; in contrast, an assessment tool for each investigated research is proposed in this paper.
- Most of the review papers did not review the recently published papers.
- A notable number of review studies did not particularly consider the usage of nature-inspired metaheuristics methods in software testing.

By reviewing the discussed SLRs above (Afzal et al. 2016; Alyahya 2020; Barbosa et al. 2022; Bluemke and Malanowska 2021; Dadkhah et al. 2020; de Souza et al. 2015; Garousi et al. 2017; Garousi and Küçük, 2018; Garousi and Mäntylä, 2016; Gillenson et al. 2018; Mäntylä et al. 2015; Tebes et al. 2020; Wiklund et al. 2017; Yusop and Ibrahim 2011), we quickly discovered that nearly all of

them reviewed software testing in minor detail. Yusop and Ibrahim (2011) did not review the recently published papers. Challenges and future works were not investigated, and their paper selection process was ambiguous. In addition, none of the SLRs composed by de Souza et al. (2015), Mäntylä et al. (2015), Garousi and Mäntylä (2016), Garousi et al. (2017), Gillenson et al. (2018), Garousi and Küçük (2018), Dadkhah et al. (2020), Tebes et al. (2020), Alyahya (2020), Bluemke and Malanowska (2021), and Barbosa et al. (2022) provided a taxonomy. Furthermore, no future work nor taxonomy was represented in Gillenson et al. (2018) study. And, Afzal et al. (2016) did not note the covered years of their SLR. However, the sole paper that is fairly close to our SLR is (Wiklund et al. 2017); nevertheless, the authors only reviewed papers until 2014. Despite these limitations, we assert that the SLR we have presented is the most comprehensive to date that systematically reviews nature-inspired metaheuristic methods in software testing up to 2022.

2.2 The motivations for a systematic study on nature-inspired metaheuristic methods in software testing

The quest for a systematic review steers to the *identification, classification, and comparison* of the current studies related to using bio-inspired metaheuristic methods in software testing. This work predominantly concentrates on the detailed comparison and broad classification of nature-inspired metaheuristic methods. To claim that similar literature studies to ours have not yet been conducted, we browsed Google Scholar with the below research string:

software **AND**
(test **OR** testing) **AND**
(review **OR** survey **OR** overview **OR** challenges **OR** trends **OR** study **OR** state-of-the-art)

None of the observed reviews mainly answered our proposed research queries in Sect. 3.1.2. Due to the extreme criticality of nature-inspired metaheuristic methods in software testing, reinforcing the current evidence on metaheuristic methods in software testing is needful.

3 Research methodology

As opposed to a formless review operation, a systematic literature review lessens any discrimination, which follows a meticulous and invaluable succession of numerous methodological stages for researching literature (Brereton et al. 2007). It aims to recognize, assess, and interpret most existing research regarding specific research questions, topic areas, and phenomena (Brereton et al. 2007; Kitchenham 2004). We used SLR to conduct broad and methodic research of the software test techniques in the current section. While using a three-stage approach, namely *planning, conducting, and documenting*, we followed the regulations in (Brereton et al. 2007; Kitchenham 2004) as well. In addition, a transparent classification of the surveyed papers is issued—which is fundamental as a consequence of the analytical comparison of the selected papers. In addition, the data summary is depicted in Section 4, while research outcomes are analyzed in Section 5.

3.1 Planning stage

The planning stage is initiated and finalized by a concise protocol explanation for fulfilling the insatiable demands and strong motivations for an SLR.

3.1.1 Realizing the necessities and incentives for the systematic review

The main contributions, needs, and motivations are depicted in Sect. 2.2.

3.1.2 Establishing the research questions

We have stated five research questions (RQs) concerning related works' constraints and our motivations. The responses prepare a knowledge-based method for reviewing software testing using nature-inspired metaheuristic methods. The RQs are given in the following lines:

RQ 1: What classification of research techniques can be possibly utilized for software test methods?

We will answer this question in Sect. 4.

RQ 2: What evaluation metrics can be used for software test methods?

We will answer this question in Sect. 5.2.

RQ 3: What are the evaluation techniques for software test methods?

We will answer this question in Sect. 5.2.

RQ 4: What tools are used to evaluate the selected papers in software test methods?

We will answer this question in Sect. 5.2.

RQ 5: What are the future challenges and open perspectives in using algorithms for software test methods?

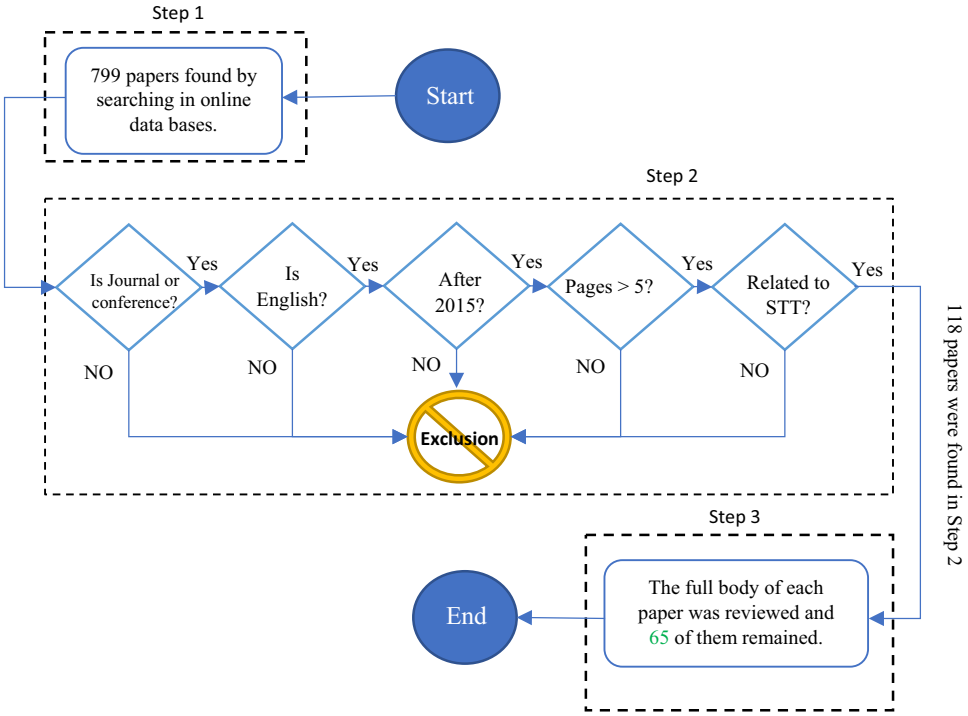


Fig. 2 Filtering method for the paper selection process

Any open issues and challenges like requirements (time, cost, resources, and language) that can be clearly classified for any algorithms are discussed in Sect. 6.

3.1.3 Defining review protocols

Concerning our prime objective, the noted research questions and the review scope were recognized to set the appropriate string for literature extraction (Brereton et al. 2007). Moreover, for attaining a systematic study, a protocol is developed based on our former relevant experiences with SLRs (Abkenar et al. 2020; Ahmadi et al. 2021; Bazzaz Abkenar et al. 2021; Etemadi et al. 2023; Fathi et al. 2022; Haghi Kashani et al. 2021, 2020; Karimi et al. 2021; Kashani and Mahdipour 2022; Nemati et al. 2023; Nikravan and Haghi Kashani 2022; Rahimi et al. 2020; Sheikh Sofla et al. 2022; Songhorabadi et al. 2023). An SLR's pilot study, comprising approximately 20 percent of our reviewed papers, is performed to assess this technique to eradicate any prejudice—which is the chief objective of this pilot study. As a consequence, inclusion, exclusion, search methods, and review scope are vastly enhanced.

3.2 Conducting stage

Conducting the review is the next stage of the research methodology. The main goal of this section is to illustrate paper selection, the results in data extraction, and information synthesis.

3.2.1 Selection of articles

The selection process comprises three-step regulations, which are depicted in Fig. 2. The search operation is carried out by investigating online scientific databases: Google Scholar,[1] ScienceDirect,[2] IEEE Explorer,[3] ACM,[4] Springer,[5] Taylor & Francis,[6] Hindawi,[7] Sage,[8] Emerald,[9] World Scientific,[10] Wiley,[11] and Inderscience[12] with the following search string:

[1] https://scholar.google.com.
[2] https://www.sciencedirect.com.
[3] http://ieeexplore.ieee.org.
[4] http://www.acm.org.
[5] http://link.springer.com.
[6] http://www.tandfonline.com.
[7] https://www.hindawi.com.
[8] http://online.sagepub.com.
[9] https://www.emerald.com.
[10] https://www.worldscientific.com.
[11] https://www.wiley.com.
[12] https://www.inderscience.com.

software **AND** (test **OR** tests **OR** testing) **AND** (metaheuristic **OR** meta-heuristic **OR** algorithm **OR** method **OR** mechanism **OR** technique **OR** approach **OR** nature-inspired **OR** "nature inspired" **OR** search-based **OR** optimal **OR** optimize **OR** optimise **OR** optimization **OR** optimisation **OR** optimizing **OR** optimising **OR** optimizer **OR** optimiser)

- *Step one:* We found 779 different types of papers in this initial step. We followed two other steps given below to ensure that only excellent publications and papers with high quality are nominated in the review.
- *Step two:* According to Table 3, we searched for any paper whose method directly relates to nature-inspired metaheuristic methods in software testing. In addition, only journal and conference papers that are composed in English are selected. Moreover, papers with more than five pages that were published between 2015 and 2022 are selected. Consequently, 118 papers were found in this step. Obviously, any other types of studies are excluded in this SLR, such as book chapters, working papers, and revolution editorial note commentaries.
- *Step three:* In this last step, by comprehensively reviewing the full body of selected papers, 65 papers whose methods directly relate to the metaheuristic-based methods for software testing were extracted. As a result, not only did the selected papers in this step help us propose an exhaustive taxonomy, but they were also perfectly able to provide precise answers to our research questions.

3.2.2 Data extraction and information synthesizing

By acquiring data from records of the noted online search databases, we organized a systematic structure by following the original format proposed in (Brereton et al. 2007; Kitchenham 2004). The classifications and comparisons of the introduced approaches for nature-inspired metaheuristic methods in software testing are shown in Sect. 4. Scrutinizing the studies' negative and positive points is depicted in Sect. 5.2. And, future directions are noted in Sect. 6.

3.3 Documenting stage

After documenting the reviewed papers, the outcomes are thoroughly inspected and conceptualized, which are outlined in Sect. 5. Afterward, threats and analysis limitations are investigated in Sect. 7 (Table 3).

4 Classification and comparison of the selected papers

In this section, we have defined a structured classification of the reviewed pieces of the literature. Since the studies on software testing algorithms, methods, and approaches are quite diverse, structuring the related works systematically is not simple. The framework of the suggested taxonomy scheme is indicated in Fig. 3. Seven main categories are detected: GA-based methods, hybrid methods, ACO-based methods, CS-based methods, FA-based methods, ABC-based methods, and other metaheuristic methods. Since the majority of research in this field deals with issues connected to one of these seven views, reviewing the literature from each of these perspectives enables the evaluated papers to be grouped into a general heading. In this case, we have sorted the 65 selected papers according to the above-noted criteria and the basic properties of the reviewed article's techniques and their differences, main ideas, evaluation methods, tools, disadvantages, and advantages are discussed. Nevertheless, other classified taxonomies on nature-inspired algorithms might be possible (Table 4).

4.1 Genetic algorithm-based methods

Initially inspired by the notion of evolution in biological studies, a GA is a global optimization technique algorithm that is playing a more important role day after day in the studies of complex adaptive systems, varying from adaptive agents in economic theory to the design of complex software and devices (Holland 1992). In the GA, a new set of the population is created for each generation, in which an optimal solution is reached. Every single of the population gives a possible solution to the problem. A chromosome contains a sequence of genes. Consequently, a

Table 3 Inclusion/exclusion criteria

Type	Criteria
Inclusion	Papers that explicitly offers methods, algorithms, solutions, and evaluations to facilitate software testing technique with nature-inspired metaheuristic algorithms
	Papers that are composed in English
	Studies published online from 2015 to 2022
	Papers that include more than five pages
Exclusion	The papers that are not related to software test techniques (STT)
	Revolution editorial note commentaries, reviews, and book chapter
	All papers that use uncommon algorithms or are out of selected classification

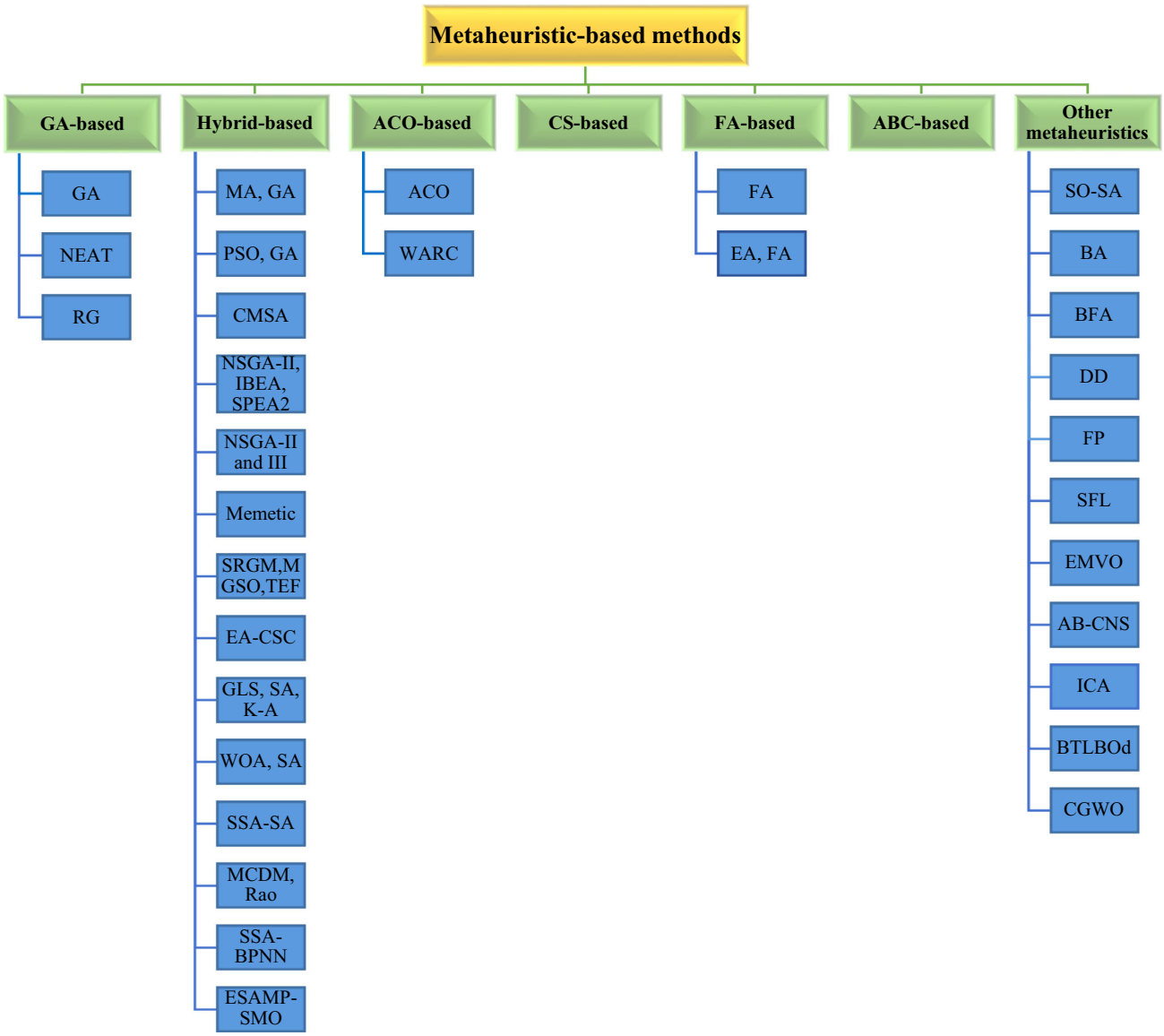


Fig. 3 The taxonomy of software testing methods

stochastic collection of chromosomes creates a primary population (Boopathi et al. 2014). GA-based algorithms are divided into three sub-sections, namely GA (Asoudeh and Labiche 2018; Bahaweres et al. 2017; Betts and Petty 2016; Boopathi et al. 2019; Dasoriya and Dashoriya 2018; Esparcia-Alcázar et al. 2018; Hema Shankari et al. 2021; Huo et al. 2017; Koteswara Rao and Raju 2019; Kumar and Sahni 2020; Li et al. 2018; Mahajan et al. 2015; Serdyukov and Avdeenko 2018; Wu et al. 2016; Yao et al. 2020; Zhan 2022; Zhu et al. 2017; Ziming et al. 2017), NEAT (Raj and Chandrasekaran 2018), and RG (Yang et al. 2016). In this section, we attempted to provide a summary of every single of the noted papers.

4.1.1 Overview of the selected genetic algorithm-based methods

Mahajan et al. (2015) introduced the component-based test suite sequence method accompanied by a GA implementation by employing the technique of Java decoding. This prioritization technique was presented for test cases utilized at the component-based software module level. This technique enabled us to predict the rate of extreme faults or bugs in much more enhanced ways. In this paper, the hardships attached to RT case prioritization are observed and measured. Hence, the evaluation of a component-based software test prioritization approach could also go hand-in-hand with RT to minimize the time necessary to carry out test suits. The introduced method employs the most

Table 4 Definition of evaluation metrics

Parameter Name	Description
Time	How long does it take for a user to get their test request responses
Cost	Required payments for testing a software
Performance	Requirements that determine the sufficiency of a software product to supply suitable proficiency relative to the number of resources essential to carry out full functionality under introduced situations
Efficiency	Average software product performance measurement considering available resources
Availability	The system is accessible at any time and situation when the user running a function
Code coverage	The measure applies to describe the proportion to which the source code of a product is executed when a specific test suite run
Scalability	Capacity to change the software product in various situations
Branch coverage	The requirement that each branch is tested at least once
Statement coverage	The measurement that calculates the numeral of executed statements
Safety	Requirements that the system is safe against and also designed to handle abrupt software treatment
Complexity	The quantitative measurement of the number of linearly absolute paths via a program's source code
Mutation score	The total proportion of killed mutants with the total number of mutants
Quality	The degree to which a software product has met specific needs or expectations
Reliability	Amount of error messages to total messages
Success rate	How many times does an algorithm prosper in ratio to the overall frequencies that it is executed
Resource utilization	The proportion of total time that a constraint exploits the resource
Accuracy	The total state of precise outcomes in software testing

beneficial tokens, such as the code's scope, the size of the project, information stream, bug inclination, and impact of fault or bug on the system. One beneficial aspect is the use of a modular testing technique for the project as a whole. On the other hand, the major disadvantage of the proposed system lies in RT; as reconciliation testing result returns, for each alteration inside the development module, the amount of RT becomes wasteful as there is a need for re-execution of each test for each component of the project.

Due to the critical information security condition ICS faces, Wu et al. (2016) explored the methods for detecting vulnerability. These methods are used for the configuration software. Relying on what characteristics the configuration software has; they offer a GA-based fuzzing test method. The authors explained the principal design parameters of the framework's fuzzing test. They also employed GA to enhance the process of test data generation on the foundation of investigating the weak points of contemporary fuzzy test techniques. In addition, the study supported the vulnerability discovery of the configuration and security test in software; nonetheless, the test work required a lot of manual operations, and limited sections of the test functions could only be mechanized.

Bahaweres et al. (2017) presented a study to implement a GA in testing the software and compare the automated and manual testing results with a GA. The test parameters are coverage measurements, such as branch, statement, loop coverage, and testing time. The study concluded that automated testing with a GA demands less time and fewer test cases to accomplish 100% coverage. Furthermore, Dasoriya and Dashoriya (2018) demonstrated an algorithm that can be applied to either black-box or white-box testing. Inspired by evolutionary biology and iterating through the five phases, these ES enhance the heuristic search technique. We can use these five phases to get some of the best test cases in software testing rather than testing the software as a whole. Not only is the time taken reduced, it can also be efficiently employed in the white box and black box testing to generate the optimal results.

Serdyukov and Avdeenko (2018) proposed a system for generating test data and investigated an approach to analyze the possibility of integrating a GA. Testing the algorithm and analyzing its performance are two goals that the generated tests could serve. The raw data already allow them to decide on the patterns in the program code. By conducting a further study, it is possible to allow researchers to come to conclusions on likely subsequent enhancement. Also, Huo et al. (2017), using genetic programming (GP), proposed a method to test the main problems of multi-objective data generation. They also implemented two GA-based methods only for comparison purposes. In addition to that, several multi-objective

optimization frameworks are employed, and a comparison is drawn to inspect how well the GP-based methods perform. The results have shown that the GP techniques have superior performance than the other two GA methods, or a baseline algorithm for random search.

By using a fitness function, Zhu et al. (2017) proposed an efficient GA. This algorithm generates a couple of test cases to cover multiple target paths, and therefore a reduced number of iterations is required; hence as opposed to the existing algorithms, it exhibits higher efficiency. The major distinction between their method and the other selected methods was calculating an individual's fitness. Every individual method runs independently 100 times. The number of iterations and the time necessary to create the test case are recorded, and averages of these computed values are calculated. The simulation results have also shown that the presented algorithm was high path coverage and was highly efficient, although it was not scalable, and to achieve the optimal solution, closer and more examination is needed. In addition, Esparcia-Alcázar et al. (2018) suggested using the GP to improve upon the action selection strategy, which is specified as a series of IF–THEN rules. Provided that sufficient primitives like functions and terminals are specified, GP has been demonstrated as being fitting for the enhancement of all kinds of programs, or even particular rules. There are three applications utilized as Software under test in the tests. And examinations show that GP has adequate potential to improve upon action selection strategies.

Li et al. (2018) established using a GA-based framework for test case generation of a software product line (SPL) while also integrating a technique called fault localization. Euclidean distance and K-Means clustering (KM) were presented along with the strategy to reuse the available test cases from one product to another. The authors prove that the framework, which could be fully automated, could be hugely time-saving in generating good-quality test cases for SPLs. The offered framework was employed as case studies in four SPLs. With appropriate conversion, as their results suggest, test cases generated in this method can be reused between varying products of the same family without tremendous difficulties. This, in turn, from a fault localization point of view, will aid in reducing the overall cost of testing as well as offering more information, but the proposed framework should be applied on more SPLs to continue assessing the effectiveness it offers and also to answer the requirements of companies in numerous sectors. Also, the procedure should be ameliorated.

Koteswara Rao and Raju (2019) proposed profitable subjective testing to test in light of the question straight dependence illustrates. GA uses the degree estimations of the investigations to reduce the prohibited and equitant inputs accuse slant. The technique prunes the data space by mixing the past commitment with new-made ones, increasing sufficiency augment and flexibility when programming testing occurs. Boopathi et al. (2019) employed the Markov chain method to measure the overall scope of the dd-graph by utilizing GA. Their main purpose was to utilize GAs for optimizing error-free software codes to achieve efficiency in software testing. Ziming et al. (2017) introduced a method to enhance the generation of data testing. To authenticate the applied approach, the authors tested it in different case studies. In addition, a symbolic execution tool is used in the study for gathering the elements of testing to decrease the search volume of tested data. Nevertheless, their proposed method did not pay a close attention to key features of the target path for testing.

Yao et al. (2020) created a mathematical model to test data generation. In the first step, the authors proposed an equal feature for testing software which was chosen haphazardly. Then, they offered an optimized model to mitigate the key pitfalls of testing the data generation. For the last stage, they introduced a solution using GA, which enhanced the quality and efficiency of the presented model. Betts and Petty (2016) utilized two search-based testing algorithms, GA and surrogate-based optimization (SBO), to test the automation of vehicle software. The authors compared two noted algorithms and discovered that, though GA was relatively slower than SBO in improving the system's execution time, its productivity fulfilled the theoretical maximums when the number of procedures was above 1%.

Kumar and Sahni (2020) presented a model to estimate the finding efforts in a dynamic surrounding with debugging values similar to every mastering curve phenomenon. The authors used a theoretical management approach to comprehend optimal policies and algorithmic genetic rules to predict testing. Asoudeh and Labiche (2018) investigated the authentication of various mechanisms by calculating the multiplicity of the whole test suit in a case study. The authors computed the test suit with nine practicable methods embedded in GA to reduce costs and reached the final measurement for testing the test suit's variations.

To solve fundamental problems derived from automated software testing, Hema Shankari et al. (2021) investigated software's performance by proposing an artificial neural network with GA-based case suite prioritization (ANNGCSP). Their introduced method reduced the interferences of humans in the regression of every software testing succession. However, they did not use various algorithms, such as SA, and ACO for prioritizing and comparing different test cases. Zhan (2022) investigated path generation and methods of selection. In specific terms, they tried to elevate test generation and path tests to make test requirements meet the usual requirements. Raj and

Chandrasekaran (2018) used a triangle classifier program to evaluate and compare their proposed method with other viable approaches. Their initial outcomes from testing the introduced method, which was under a NEAT algorithm, depicted that not only did the code coverage improve dramatically, but the number of test cases also decreased notably.

Also, Yang et al. (2016) proposed a new regeneration strategy called RG. RG is based on a novel, plain, stable, and efficiently implemented regeneration strategy which engages in making judgments about the process for population aging. Factors for population aging and its process for deciding on the degree of population aging are defined in the proposed strategy. The regenerate genetic algorithm (RGA) was evaluated by first conducting an experimental study and then comparing it with several other methods like basic random testing, GA, and numerous other approaches concerning effectiveness on the test programs' siemens suite. As the results have proven, the algorithm offered can successfully increase search productivity, restrict population aging, boost test coverage, and decrease the total number of test cases. Finally, the RGA presented better optimization capability than the traditional algorithms, particularly when the programs were large-scale and highly complex. However, there were various potential opportunities for enhancing RGA further by putting it together with other strategies or methods.

4.1.2 Summary of genetic algorithm-based methods

In Table 5, the categorization of nominated papers and influential elements for analyzing the GA-based methods in software testing are explained in further detail. In Table 6, the evaluation for the assigned studies by employing evaluation factors in GA-based methods is explained. Factors such as time, cost, performance, efficiency, code coverage, scalability, branch coverage, statement coverage, safety, and complexity are included. In GA-based methods, the majority of reviewed papers assessed their presented method using the efficiency factor.

4.2 Hybrid methods

Because of the drawbacks of each metaheuristic method, one needs to put them together to accomplish an efficient software testing technique. A combination of two or more methods ought to be made in various mechanisms to create new hybrid algorithms. A hybrid algorithm relies on the idea that combining two or more algorithms is quite common among researchers. Nevertheless, the algorithm is general and might have application in most optimization problems. We have divided hybrid methods into nine subsections; MA, GA (Khan and Amjad 2015), PSO, GA (Damia et al. 2021; Jin and Jin 2016; Koleejan et al. 2015; Mahmoud and Ahmed 2015), CMSA (Ferrer et al. 2021), NSGA-II, IBEA, SPEA2 (Gupta et al. 2020; Matnei Filho and Vergilio 2015), NSGA-II and NSGA-III (Jamil et al. 2019), Memetic (Bahrampour and Rafe 2021; Esnaashari and Damia 2021; Nejad et al. 2016), SRGM, MGSO, TEF (Rao and Anuradha 2016), EA-CSC (Liu et al. 2017), GLS, SA, K-A (Gao and Xiong 2015), WOA, SA (Zhu et al. 2021), SSA-SA (Kassaymeh et al. 2022b), MCDM, Rao (Thirumoorthy and J, 2022), SAA-BPNN (Kassaymeh et al. 2022a), and ESAMP-SMO (Thirumoorthy and Britto 2022). The remainder of this section discusses the major characteristics of the nominated hybrid methods.

4.2.1 Overview of the selected hybrid methods

Khan and Amjad (2015) presented a technique that uses MA with GA to generate automatic test cases that are better and mainly developed using GA operations mutation and crossover. The authors covered 20 suits of different sizes of the applied mutants. The result of the applied suits proposed that the method covered 100 %. The mutation score was deficient and was quantified using randomly generated test cases. An enhanced mutation score is gained when GA is employed for irregularly generated test cases; however, it is not suitable for black-box testing; also, their result was limited to 20 test suits, and they did not check the method for too many inputs and test suits.

Koleejan et al. (2015) proposed a multi-vector representation in GA and particle PSO that can create multiple sets of data cases to produce test data for sophisticated testing programs in a single run. To investigate and create a comparison of the performance of GAs and PSO, they have conducted some experiments on six frequently employed benchmark test programs along with three novel programs that have an approximately large amount of complex conditions. The offered multi-vector representation is, in fact, capable of enhancing the performance of PSO and GAs on each of the programs that were tested, as the experimental results confirmed, and the optimal 100% code coverage on accessible programs was achieved. Regarding the code coverage and the computational efficiency, PSO outperforms Gas, especially when the programs are complex; however, determining the optimal number of vectors prior was rather tricky. It also could not handle various types of input variables.

Mahmoud and Ahmed (2015) developed a PSO fuzzy-based methodology to tackle key downsides in traditional PSO algorithms. The authors controlled the introduced approach in detail through fuzzy logic, and after which the method was monitored, they implemented it in colony algorithm generation. Despite the effectiveness and coverage of the presented method, the performance was not

Table 5 Classification of recent studies and other information details on GA-based methods

Paper	Algorithm	Main idea	Evaluation method	Tool (s)	Advantage (s)	Disadvantage (s)
Mahajan et al. (2015)	GA	Component-Based test case prioritization with GA	Prototype	Java	Low cost Low time Low efficiency High performance High code coverage	High complexity
Wu et al. (2016)	GA	Using the fuzzing test method to increase the security of software based on GA	Simulation	IDA Pro, OllyDbg	High safety High availability High scalability	High complexity
Bahaweres et al. (2017)	GA	Statement branch's examination and loop with the use of GA	Case study	Not mentioned	High code coverage High branch High statement Low time	Low scalability
Dasoriya and Dashoriya (2018)	GA	Optimizing software testing by GA	Design	Not mentioned	High efficiency Low time	Low scalability Low accuracy
Serdyukov and Avdeenko (2018)	GA	Data generation for software testing with GA	Prototype	C#	High scalability Low complexity Low cost Low time	Low code coverage High complexity
Huo et al. (2017)	GA	Utilizing multi-objective optimization to improve the total performance and contribute to search-based test data generation	Real testbed	Java	High performance High efficiency High code coverage High branch Low time	Low scalability Low reliability
Zhu et al. (2017)	GA	Multiple paths in software test case generation with improved GA	Simulation	Not mentioned	Low time High coverage High efficiency Low cost	Low scalability
Esparcia-Alcázar et al. (2018)	GA	To discover the exciting primitives to utilize when developing an action selection rules with GP	Real testbed	Odoo, Testona	High performance High efficiency	Low scalability High complexity
Li et al. (2018)	GA	Integrating defects using GA	Case study	χSuds	Low cost Low time High efficiency High performance	Low scalability

Table 5 (continued)

Paper	Algorithm	Main idea	Evaluation method	Tool (s)	Advantage (s)	Disadvantage (s)
Koteswara Rao and Raju (2019)	GA	Interactive fault proneness reduction in software by using GA-based optimal	Real testbed	Java	High performance Low cost High efficiency	Low scalability
Boopathi et al. (2019)	GA	Using Markov chain approach and GA for obtaining an optimized path in software code	Simulation	Gcov coverage analysis	High efficiency High coverage Low cost	Not testing the proposed algorithm's efficiency in real world
Ziming et al. (2017)	GA	Taking an approach to grow the efficiency of datasets	Case study	Symbolic execution	High performance	Low scalability Low coverage
Yao et al. (2020)	GA	Testing software with random numbers	Formal	Spss	High efficiency	Low quality Low reliability Low scalability
Betts and Petty (2016)	GA	Using GA in the generation of test cases	Simulation	Not mentioned	High performance High robustness	Lack of actual testbed experiment Not testing the software in complex autonomous scenarios
Kumar and Sahni (2020)	GA	Optimizing resource allocation to software	Simulation	Spss	High reliability Low cost	Not analyzing the outcome in multi-diverse software Software's Release time was not measured
Asoudeh and Labiche (2018)	GA	Using GA to assess various potential fares in error recognition	Case study	ANOVA	Cost-efficient High efficiency	Low reliability
Hema Shankari et al. (2021)	GA	Utilizing ANNGCSP to optimize performance	Simulation	Selenium	High performance	Low scalability
Zhan (2022)	GA	Set a hierarchy by the usage of genetic algorithms and NSGA-II	Simulation	Not mentioned	Easy generation	Low test efficiency High complexity
Raj and Chandrasekaran (2018)	NEAT	Leveling up the currently produced test suit	Simulation	Python	High coverage Low cost	High time Low reliability
Yang et al. (2016)	RG	Judging the population aging process with RGA	Formal	Siemens	High scalability Low complexity High efficiency	High cost

maximum, and it can be enhanced if the design of the data structure becomes optimized in future. Jin and Jin (2016) proposed the employment of PSO for elevating the contributions of the SRGM. The division of SRGM with the s-shape no longer needed to be separated in their approach. It also does not necessarily depend on the anticipation for software defects. Having a wholly mechanized system, the SRGM model did not entail user collaboration because it was a wholly mechanized system.

Table 6 Comparison of existing GA-based methods evaluation metrics

Paper	Time	Cost	Performance	Efficiency	Code coverage	Scalability	Branch coverage	Statement coverage	Safety	Complexity
(Mahajan et al. 2015)	✔	✔	✔	✔	✔					
(Wu et al. 2016)						✔			✔	✔
(Bahaweres et al. 2017)					✔		✔	✔		
(Dasoriya and Dashoriya 2018)	✔			✔						
(Serdyukov and Avdeenko 2018)	✔	✔				✔				✔
(Huo et al. 2017)	✔		✔	✔	✔		✔			
(Zhu et al. 2017)	✔	✔		✔	✔					
(Esparcia-Alcázar et al. 2018)			✔	✔						
(Li et al. 2018)	✔	✔	✔	✔						
(Koteswara Rao and Raju 2019)		✔	✔	✔						
(Boopathi et al. 2019)	✔	✔	✔	✔	✔					
(Ziming et al. 2017)	✔		✔	✔	✔					
(Yao et al. 2020)				✔		✔				✔
(Betts and Petty 2016)			✔	✔						
(Kumar and Sahni 2020)	✔	✔		✔						
(Asoudeh and Labiche 2018)		✔		✔			✔			
(Hema Shankari et al. 2021)	✔	✔	✔	✔	✔	✔	✔	✔		✔
(Zhan 2022)	✔			✔	✔	✔	✔	✔		✔
(Raj and Chandrasekaran 2018)	✔	✔	✔	✔	✔		✔			
(Yang et al. 2016)				✔		✔				✔

So as to improve the efficiency of PSO, Damia et al. (2021) applied a method called an improved weight PSO. Due to the fragility of PSO in growing the quality of solutions obtained from every iteration, the authors imperceptibly altered the PSO structure to achieve higher performance. Ferrer et al. (2021) compared a hybrid metaheuristic algorithm, CMSA, with four different algorithms to achieve a viable solution to tackle data generation serious problems. The comparison results illustrated that CMSA is the most proper algorithm to solve the pitfalls of prioritized pairwise test data generation, though it requires more execution time than other algorithms.

Matnei Filho and Vergilio (2015) presented multiple objective techniques that include the drawbacks' representation, two other aims to calculate the number of test cases, dead mutants, and search operators. This approach was executed using three multi-objective and ESs: NSGA-II, SPEA2, and IBEA. An analysis of the solutions obtained and a comparison of the algorithms are given in the performed evaluation. As an advantage, this approach offers the tester a couple of reasonable solutions along with a decreased number of products and high mutation score values, having a high probability of recognizing faults specified by the mutation testing. However, other algorithms and also other kinds of targets, such as pair-wise coverage, can also be implemented and evaluated.

Gupta et al. (2020) investigated the effectiveness of NSGA-II in defect detection and fault positioning. They tried to minimize test cases, in which the minuscule proportion of test suits was 78% alongside 95.16% defect recognition. However, their proposed method was not optimum because the partitioning was not used to reach its maximum capacity. Jamil et al. (2019) investigated the usage of multi-object evolutionary algorithms in SPL

testing. Their proposed algorithm reduced the experimenting efforts by sorting the product line. In addition, the experimental outcomes illustrated that the tests' results were less susceptible to errors. They also compared NSGA-II with NSGA-III and concluded that the latter one is more efficient.

Nejad et al. (2016) proposed a model-based testing method that could prioritize tests by employing activity diagrams, control flow graph, GA, and memetic algorithms. Varying versions of memetic algorithms had been developed by stochastic local search, randomized iterative improvement, hill climbing, and SA algorithms. Employing local search methods in GA created efficiency and generated competitive results as opposed to GA, as the results have shown. However, model-based testing requires a formal specification or model to carry out testing. Using different case studies, Bahrampour and Rafe (2021) tested memetic algorithms' effectiveness, robustness, coverage, and error detection ability in a graph transformation system (GTS). They introduced an array of data-conflict characteristics in GTS to guide the generation process. Via mutation analysis, test data generation by memetic algorithms was evaluated, which radiated positive outcomes, such as outperformance and cost-efficiency.

To obtain lower costs, higher efficiency, and lower execution time in search processes, Esnaashari and Damia (2021) applied reinforcement learning served as a memetic method. They automated all operations to enclose finite paths. With the purpose of evaluating the efficiency and practicality of the proposed algorithm, the authors compared it with five different nature-inspired algorithms under four separate experiments. The test results indicated that not only did the memetic algorithm outperform in comparison with other algorithms, but it also had a higher success rate and required lower time. Furthermore, Rao and Anuradha (2016), using MGSO combined with logistic-exponential TEF, proposed a hybrid method for anticipating the parameters of the SRGM. Estimation of parameters of SRGM is of great help in reliability prediction, which, in turn, is helpful in the determination of the quality of Software. These parameters are optimized using MGSO after the parameters are estimated. The proposed method has been implemented in Java, and its practicality has been evaluated by comparing it with existing methods.

Also, Liu et al. (2017) designed an EA with CSC to handle the problem of automated generation of test data. The decision on how the population was propagated was founded on the heuristic data provided by the populace. The experiment results revolved around that, in most tested problems, EA-CSC can use the least number of test case overhead. The successful use of adaptive searching length CSC showed that EA's performance improved for automated test data generation problems. Nevertheless, EA-CSC was not suitable for test case generation in the tested issues that comprise numerous input values. Moreover, Gao and Xiong (2015) introduced an algorithm using GA and GLS to solve the optimal testing resource allocation problems. The experimental results show that the proposed algorithm performs better than some existing approaches to solving critical issues of software testing resources.

To create an improved metaheuristic search-based feature selection algorithm, named EMWS, Zhu et al. (2021) used a combination of WOA and SA. This hybridized method limited the selection coverage, but meticulously related the representatives. The authors specified their experimental setup, evaluation selectors, and Scott–Knott effect size difference to have their work comparable. Internal, external, and construct validities were assigned as threats to their work, in which the authors stated that in their future works, they will evaluate their model in more open-sourced details and use automated parameter methods. Kassaymeh et al. (2022b) hybridized the SA with SSA to adjust the ratio of extraction and development in SSA. To increase estimation accuracy and lower the total error rates, the authors used a BPNN estimator. However, the authors plan to use different metaheuristic methods, such as the monarch butterfly optimization and earthworm optimization algorithm, to address the same software engineering issue and optimize it further.

MCDM and Rao methods were combined by Thirumoorthy and J (2022) to enhance the software defect prediction rates. The authors used three NASA benchmark datasets to be able to compare their performance. In this sense, through the incorporation of crossover and mutation into the Rao optimization techniques, the suggested method enhanced the search capability toward the global solution. Kassaymeh et al. (2022a) improved the accuracy by combining the SSA optimizer and BPNN. Software fault prediction problems were resolved as the hybridization of the proposed algorithms in the author's work rose the exactness of the model. Thirumoorthy and Britto (2022) proposed ESAMP-SMO for agglomerating software defect modules. The authors used three NASA benchmark datasets for piloting works. As a result of implementing the hybrid methods, the authors could improve searchability.

4.2.2 Summary of hybrid methods

In Table 7, a categorization of the nominated studies and influential factors for analyzing the hybrid methods in software testing are explained in more detail. In Table 8, an evaluation of nominated studies by employing evaluation factors in hybrid methods is given. The factors included time, cost, performance, efficiency, mutation score, code coverage, scalability, quality, reliability, and success rate. Most research studies evaluated their investigated

technique in hybrid methods' efficiency, performance, code coverage, quality factors, and cost.

4.3 Ant colony optimization-based methods

ACO is employed in difficult combinatorial problems and is a population search method. Inspired by the foraging behavior of ant colonies in nature, Dorigo first proposed it in 1992 (Dorigo and Di Caro 1999). During hunting, ants leave a chemical level called pheromone along the paths they visit to better coordinate with other ants. By sensing such pheromone levels, the ants choose paths when they come across branches on the road. Initially, however, the paths are chosen randomly by the ants, and they are in the direction of the food. Nevertheless, after some time traversing, the pheromone level left on the shortest path becomes higher as opposed to others since, in lengthy paths, the pheromone has evaporated more than others (Biswas et al. 2015). We have set two sub-sections for ACO-based methods: ACO (Biswas et al. 2015; Mao et al. 2015; Sayyari and Emadi 2015), and WRAC (Bharathi and Sangeetha 2018).

4.3.1 Overview of the selected ant colony optimization-based methods

Biswas et al. (2015) offered an ACO-based algorithm that generates a set of routes and prioritizes them based on certain criteria. Moreover, to create inputs of the generated paths, the approach offered a sequence of test data within the domain to be used. The provided approach ensures complete software coverage having the least redundancy. They also try to demonstrate the offered process and apply it in a program module. Sayyari and Emadi (2015) proposed a solution based on the ACO algorithm and model-based testing to create faster test paths with maximized coverage and minimized time and cost. The Markov chain is the basis for the model in this study. The results achieved by the Markov chain are good nominees to study the feasibility of the testing process when in development. As the evaluation suggests, the proposed algorithm has higher performance than existing methods regarding cost, coverage, time, and user parameters.

Mao et al. (2015) proposed the basic ACO algorithm to reform into a discrete version for test data generation in structural testing. Strategies such as local transfer, global transfer, and pheromone updates are defined and applied. A particular optimization objective is coverage for program elements. The construction of customized fitness function, in their approach, is done by considering the nesting level and predicate type of branch comprehensively. Their approach outperforms the existing SA and GA in terms of test data quality and how stable it can be, as the results confirm, and it is also comparable to the PSO-based method.

Moreover, Bharathi and Sangeetha (2018) proposed the WRAC metaheuristic test suite optimization method that designed a WRACMO algorithm for reducing the test suite in combinatorial testing. Initially, the noted algorithm, with the help of test cases in the test suite, creates an initialization of ACO parameters, and then, the process happens repeatedly unless the ant locates the food source. This technique improves the rate of the test suite declination and lowers the execution time needed for test cases to perform effective combinatorial testing.

4.3.2 Summary of ant colony optimization-based methods

In Table 9, the categorization of the nominated papers and the decisive factors for analyzing the ACO-based methods in software testing are given in further detail. Table 10 shows an evaluation for the nominated studies by employing evaluation factors in ACO-based methods. In the ACO-based methods, the majority of research studies examined their presented approach in some factors such as time, cost, performance, efficiency, code coverage, scalability, and complexity.

4.4 Cuckoo search-based methods

CS, developed by (Yang and Deb 2009), is a metaheuristic search algorithm inspired by the manners of an interesting bird, named cuckoo, which lays its eggs in public nests. Nevertheless, these birds might remove the eggs belonging to another bird to ensure that their eggs are more likely to hatch. If the bird that is the host of the nest distinguishes the eggs belonging to the cuckoo from its eggs, it may either leave the nest completely or throw the eggs away. The rest of this section addresses the selected CS methods and major characteristics. The studied articles for this section are (Ahmed et al. 2015; Khari and Kumar 2016; Sharma et al. 2019).

4.4.1 Overview of the selected cuckoo search-based methods

Khari and Kumar (2016) proposed the application of the CS algorithm for test data generation. The authors validate the presented algorithm against test functions and then create a comparison between its performance and the performance of hill-climbing algorithms. How effective the presented algorithm is in test data generation is proved by the initial results. In contrast with the traditional hill-climbing algorithm, the high amount of test data for any given program was decreased with no redundancy. It has had a positive impact on test data minimization, and thus,

Table 7 Classification of recent studies and other information details on hybrid methods

Paper	Algorithm(s)	Main idea	Evaluation method	Tool(s)	Advantage(s)	Disadvantage(s)
(Khan and Amjad 2015)	MA, GA	Automatic test case generation with GA and MA	Simulation	Not mentioned	Low mutation score Low time High efficiency	Low Scalability High complexity
(Koleejan et al. 2015)	PSO, GA	Using PSO and GA to generate test data	Case study	Pseudo-code	High efficiency High code coverage High performance	Low scalability High cost
(Mahmoud and Ahmed 2015)	PSO	Adopting PSO to cover an array of fuzzy-based logics	Case study	Not mentioned	High efficiency High code coverage	High complexity High time Low performance
(Jin and Jin 2016)	PSO	PSO algorithms in optimizing the segments of SRGM	Simulation	Not mentioned	High flexibility High reliability High performance High efficiency	Low coverage
(Damia et al. 2021)	PSO	Automating test data generation	Simulation	Pseudo-code	High coverage	Low performance
(Ferrer et al. 2021)	CMSA	Testing immense SPLs	Simulation	Pseudo-code	High quality High coverage	High execution time Low performance
(Matnei Filho and Vergilio 2015)	NSGA-II, IBEA, SPEA2	Feature Testing by mutation and multi-objective test data generation	Real testbed	FMTS	High efficiency Low mutation score High scalability Low cost Low time	High complexity
(Gupta et al. 2020)	NSGA-II	Detecting defects based on coverage and mutant	Simulation	Cobertura	High coverage Low time High precision	High complexity Mutants separation was not optimized
(Jamil et al. 2019)	NSGA-II and NSGA-III	Overall performance of NSGA-II and III to enhance the SPL testing	Simulation	Pseudo-code	High performance High quality High coverage	Not testing the model on large-scale products

the efforts are reduced in the testing process; however, the algorithm takes some time to empty the buff list, and there was no check for a real-time or working application. Sharma et al. (2019) suggested a framework that produced ideal test sets by using CS. Their under-controlled software was transmitted to an appropriate delineation, such as a

Table 7 (continued)

Paper	Algorithm(s)	Main idea	Evaluation method	Tool(s)	Advantage(s)	Disadvantage(s)
(Nejad et al. 2016)	Memetic	Optimization automation test cases by model-based algorithms	Design	Not mentioned	High efficiency High Performance High scalability	High time High cost
(Bahrampour and Rafe 2021)	Memetic	Optimizing code coverage using memetic algorithms	Case study	GROOVE	Low cost High robustness High performance	Low accuracy Coverage needs to be optimized
(Esnaashari and Damia 2021)	Memetic	A systematic approach to mechanize the procedure of data generation	Simulation	Python, Pseudo-code, Pycfg	High success rate High efficiency High coverage	High complexity
(Rao and Anuradha 2016)	SRGM, MGSO, TEF	Reliability growth of software using MGSO	Design	Java	High reliability High quality High efficiency	Low scalability High complexity
(Liu et al. 2017)	EA-CSC	Test coverage and automation test case speed control by EA-CSC	Design	Not mentioned	High performance Low cost	Low scalability
(Gao and Xiong 2015)	GLS, SA, K-A	Minimizing the optimal testing resource allocation problems	Simulation	MATLAB, Pseudo-code	High efficiency High performance High reliability Low cost	Low scalability
(Zhu et al. 2021)	WOA, SA	An improved metaheuristic feature selection method	Simulation	Pseudo-code	High efficiency High performance High search capability	Limited data source Not using multi-source cross-projects to validate their model
(Kassaymeh et al. 2022b)	SSA-SA	Software defect estimation	Simulation	Pseudo-code	High accuracy High performance	Limited test coverage
(Thirumoorthy and J, 2022)	MCDM, Rao	Using binary Rao optimization algorithm for feature selection model	Simulation	Pseudo-code	High accuracy High performance High search capability	Not using the built-in parallelism methods
(Kassaymeh et al. 2022a)	SSA-BPNN	SSA for software fault prediction	Simulation	Pseudo-code	High accuracy	High computational costs
(Thirumoorthy and Britto 2022)	ESAMP-SMO	Clustering approach to predict software failures using a hybrid social simulation optimization algorithm	Simulation	Not mentioned	High performance High search capability	Dependency on clustering accuracy

Table 8 Comparison of existing hybrid methods evaluation metrics

Paper	Time	Cost	Performance	Efficiency	Mutation score	Code coverage	Scalability	Quality	Reliability	Success rate
Khan and Amjad (2015)	✔			✔	✔					
Koleejan et al. (2015)			✔	✔		✔				
Mahmoud and Ahmed (2015)	✔		✔	✔				✔		
Jin and Jin (2016)			✔	✔		✔		✔	✔	
Damia et al. (2021)			✔	✔		✔		✔		
Ferrer et al. (2021)	✔	✔	✔			✔		✔		
Matnei Filho and Vergilio (2015)	✔	✔		✔	✔		✔			
Gupta et al. (2020)	✔				✔	✔				
Jamil et al. (2019)	✔✔		✔			✔		✔		
Nejad et al. (2016)			✔	✔			✔			
Bahrampour and Rafe (2021)		✔	✔	✔	✔	✔		✔		
Esnaashari and Damia (2021)	✔	✔	✔	✔		✔		✔		✔
Rao and Anuradha (2016)				✔				✔	✔	
Liu et al. (2017)		✔	✔							
Gao and Xiong (2015)		✔	✔	✔					✔	
Zhu et al. (2021)			✔	✔			✔		✔	
Kassaymeh et al. (2022b)			✔	✔		✔	✔		✔	
Thirumoorthy (2022)			✔	✔		✔	✔	✔	✔	
Kassaymeh et al. (2022a)		✔	✔	✔		✔	✔	✔	✔	✔
(Thirumoorthy and Britto 2022)		✔	✔		✔	✔	✔	✔		

flow diagram that provided conditions where the input could be mapped and tracked. The authors introduced an object function to calculate the efficiency and maintain the quality of the proposed framework.

Ahmed et al. (2015) proposed a strategy for combinatorial test suite generation by employing CS. This strategy could be applied to functional testing activities that did not consider the inner structure of the code. It has been recently proven that CS is an effective optimization algorithm for solving NP-complete problems. Utilizing the CS to optimize the combinatorial test suites could mostly lead to better results than its counterpart strategies, as the evaluation results have shown. To have a real-world case study, they assessed the efficiency of the test suits. The strategy confirmed its efficacy in fault detection in programs by employing the functional testing approach. However, given the absence of results for the metaheuristic algorithms, only one program was employed for the case study, and other kinds of faults within the same case study are possible to be recognized by a higher combination degree which was the most critical weakness.

4.4.2 Summary of cuckoo search-based methods

In Table 11, the categorization of the above studies and the influential factors to analyze the CS-based methods in software testing are explained in further detail. In Table 12, an evaluation of the above studies using evaluation factors in CS-based methods is given. The following factors include cost, performance, efficiency, and code coverage. Most research studies used major metrics, cost, and efficiency, in CS-based methods to evaluate their presented approach.

4.5 Firefly algorithm-based methods

FA was initially originated by Yang (2009) in 2008. Since fireflies are genderless, they will be absorbed by other fireflies. In addition, their radiance of light is also governed

Table 9 Classification of recent studies and other information details on ACO-based methods

Paper	Algorithm	Main idea	Evaluation method	Tool	Advantage (s)	Disadvantage (s)
Biswas et al. (2015)	ACO	ACO to generate optimal paths in the control flow graph	Prototype	C++	High code coverage Low time Low cost	Low Scalability
Sayyari and Emadi (2015)	ACO	Automatic test case generation with ACO	Prototype	Not mentioned	High code coverage Low time Low cost	Low Scalability
Mao et al. (2015)	ACO	Software structural testing optimization by adapting ACO	Prototype	C++	High efficiency High coverage High performance	Low scalability High cost
Bharathi and Sangeetha (2018)	WRAC	Test suit reduction and software testing quality improvement by ACO	Simulation	Java	Low cost Low time High code coverage High efficiency High performance	High complexity Low scalability

Table 10 Comparison of existing ACO-based methods evaluation metrics

Paper	Time	Cost	Performance	Efficiency	Code coverage	Scalability	Complexity
Biswas et al. (2015)	✔	✔			✔	✔	
Sayyari and Emadi (2015)	✔	✔			✔	✔	
Mao et al. (2015)			✔	✔	✔	✔	
Bharathi and Sangeetha (2018)	✔	✔	✔	✔	✔	✔	✔

by the object functions in environments. To provide a summary of their primary functions in software testing, we have divided them into two distinct categories, which are as follows: FA (Anbu and Anandha Mala 2019; Khatibsyarbini et al. 2019; Sharma and Saha 2020; Shomali and Arasteh 2020), and EA, FA (Sahin and Akay 2016).

4.5.1 Overview of the selected firefly algorithm-based methods

Khatibsyarbini et al. (2019) presented an approach to prioritize the test cases by employing FA optimally. The authors employed the FA while setting the fitness function to use a similarity distance model to decide on the optimal prioritization of the test cases. By utilizing such a prioritization technique for the test cases, the achieved performance, concerning the average percentage of detected faults and execution time, was better, if not equal, than the current works. The result percentage of detected defects indicates that FA is a promising competitor in test case prioritization applications.

To confirm highly crucial test paths, Sharma and Saha (2020) used an FA-based approach. They tested the proposed algorithm with five varied case studies, such as trouble tickets, booking systems of hotels, workflow tasks, vending machines, and microwaves. They tested their presented algorithm on the noted case studies because all of them contributed to a wide range of states. Their empirical outcomes signaled that the path FA provided for testing was relatively error-free. Shomali and Arasteh (2020) used FA to recognize the most prone error paths to decrease the costs and time of the mutation test. Their proposed method

Table 11 Classification of recent studies and other information details on CS-based methods

Paper	Algorithm	Main idea	Evaluation method	Tool	Advantage(s)	Disadvantage(s)
Khari and Kumar (2016)	CS	Software test data generation using CS	Prototype	Java	High efficiency Low cost	High response time Low scalability
Sharma et al. (2019)	CS	Initiation of an optimized test using CS	Simulation	Not mentioned	Cost-efficient High performance	Low coverage
Ahmed et al. (2015)	CS	CS for minimizing combinational test suit configuration aware	Case study	Not mentioned	High efficiency High performance Low cost	Low scalability Low reliability Low stability

Table 12 Comparison of existing CS-based methods evaluation metrics

Paper	Cost	Performance	Efficiency	Code coverage
Khari and Kumar (2016)	✔		✔	
Sharma et al. (2019)	✔	✔	✔	✔
Ahmed et al. (2015)	✔	✔	✔	

steers clear of any faults, and the total sum of mutants plunged exponentially. They compared their presented approach with different GAs stationed by four criteria—success rate, convergence peace, stability, and quantity of mutants. The output of the comparison portrayed the outperformance of the introduced method.

To have a reduction of the size and improve the incorporation of accuracy, Anbu and Anandha Mala (2019) proposed a feature selection method with the usage of the Firefly algorithm. However, it is in the plan of the future works of the authors to implement the updated feature selection on FA to enhance accuracy and improve performance. Sahin and Akay (2016) employed some metaheuristics methods, such as ABC, PSO, FA, random search, and differential evolution, to solve optimization problems in software testing. The second section of the experimentations investigated the coverage performance of several metaheuristics on software test data generation, utilizing varying fitness functions, such as branch + level interval fitness function, path-based fitness function, and dissimilarity-based fitness function. For the third section of the experimentations, a comparison of the path-based, dissimilarity-based, and approximation branch + level interval fitness functions was made. As the results demonstrated, a random search was very effective for straightforward pitfalls in a short search space, whereas it was unsatisfactory for more complicated issues or in case there is a vast search space.

4.5.2 Summary of firefly algorithm-based methods

In Table 13, the categorization of the above studies and the influential factors to analyze the FA-based methods in software testing are explained in further detail. In Table 14, an evaluation of the above studies using evaluation factors in FA-based methods is given. The following factors include time, cost, performance, efficiency, code coverage, quality, reliability, and mutation score.

4.6 Artificial bee colony-based methods

ABC algorithm is stimulated by the aid of the predatory manner of honey bees to discover the most proper location for food sources. They determine food supplies in the area where they live with support from their long-term memories. They are also capable of remembering the most plentiful food sources (Karaboga and Basturk 2008). We have investigated the ABC-based methods in this section by reviewing two relevant papers (Aghdam and Arasteh 2017; Singhal et al. 2021).

4.6.1 Overview of the selected artificial bee colony-based methods

Aghdam and Arasteh (2017) proposed an ABC algorithm for dealing with the problem of branch coverage criterion and test data generation utilized as the fitness function for enhancing the proposed solutions. In this study, seven

Table 13 Classification of recent studies and other information details on FA-based methods

Paper	Algorithm (s)	Main idea	Evaluation method	Tool (s)	Advantage (s)	Disadvantage (s)
(Khatibsyarbini et al. 2019)	FA	Prioritization test case for software testing using FA	Real testbed	Java	Low cost High efficiency High performance	Low code coverage
(Sharma and Saha 2020)	FA	Determining the most crucial test paths	Case study	OPTCP	High efficiency High reliability	Low scalability
(Shomali and Arasteh 2020)	FA	Decreasing the time and cost of mutation test	Simulation	MATLAB, MuJava	Low cost High performance High coverage	High complexity
(Anbu and Anandha Mala, 2019)	FA	Feature selection using FA	Simulation	Not mentioned	High accuracy High reliability High performance	Not using the updated feature selection on FA
(Sahin and Akay 2016)	EA, FA	Software test data generation by metaheuristic methods and fitness function	Simulation	MATLAB, Python	High efficiency	Low scalability Low availability

prominent and conventional programs belonging to the literature were utilized as benchmarks, only for comparison purposes. The method, on average, outperforms GA, SA, ACO, and PSO according to four criteria, including 99.99% average branch coverage, 99.94% success rate, 3.59 average convergence generation, and 0.18ms median running time, as the experimental results confirm. However, one of the disadvantages of ABC algorithms is slow performance in sequential processing; hence, the method should be examined more closely in the sequential processing environment, and comparisons should be drawn to indicate whether improved performance is achieved in this situation or not.

Intending to improve the aspect-oriented of test cases, Singhal et al. (2021) used the ABC algorithm. To assess the total performance and efficiency of the utilized algorithm, the authors benchmarked ABC's applications on different models. Then, to calculate the computation time and complexity of the algorithm, they compared the usability of the ABC method with the results from GA-based methods. The initial outcomes depicted that the coverage of the algorithm was above 90% in comparison with GA.

4.6.2 Summary of artificial bee colony-based methods

In Table 15, the categorization of the above studies and the influential factors to analyze the ABC-based methods in software testing are explained in further detail. In Table 16, an evaluation of the above studies using evaluation factors in ABC-based methods is given. The following factors

Table 14 Comparison of existing FA-based methods evaluation metrics

Paper	Time	Cost	Performance	Efficiency	Code coverage	Quality	Reliability	Mutation score
Khatibsyarbini et al. (2019)		✔	✔	✔				
Sharma and Saha (2020)		✔	✔				✔	
Shomali and Arasteh (2020)	✔	✔	✔		✔	✔	✔	✔
Anbu and Anandha Mala (2019)			✔			✔	✔	
Sahin and Akay (2016)				✔				

include time, cost, performance, efficiency, code coverage, and reliability.

4.7 Other metaheuristic methods

The rest of the metaheuristic-based selected papers that are not included in the introduced taxonomies are categorized as other metaheuristic methods. A variety of well-known methods are reviewed and divided into nine sub-sections; SO-SA (Haraty et al. 2018), BA (Alsariera et al. 2015), BFA (Devika Rani Dhivya and Meenakshi, 2018), DD (Sugave et al. 2017), FP (Kabir et al. 2017), SFL (Ghaemi and Arasteh 2019), EMVO (Fakhouri et al. 2020), AB-CNS (Sivaji and Srinivasa Rao 2021), ICA (Arasteh and Hosseini 2022), BTLBOd (Khuat and Le 2019), and CGWO (Dhavakumar and Gopalan 2021). The remainder of this section discusses the other nominated metaheuristic methods and their major characteristics.

4.7.1 Overview of the selected other metaheuristic methods

Haraty et al. (2018) proposed a model to test web applications by using SO-SA. They compared their proposed algorithm with other algorithms, GA, greedy algorithm, and incremental simulated annealing. SO-SA's test results far outweigh other algorithms' performance. Nevertheless, there is still room for optimizing the proposed algorithm, and it needs to be compared with a broader range of metaheuristics. Alsariera et al. (2015) presented an approach for testing software product lines called T-way. Their implementation aimed at reducing test volumes and how correlating the interactions between segments. The authors used BA to eliminate the excessive constraints compound in testing. However, their proposed method ceased to function while the interactions peaked and failed to recognize defects in the software.

Aiming for optimizing the segments of adaptive random partition testing (ARPT), Devika Rani Dhivya and Meenakshi (2018) applied BFA in different software. They tested the proposed algorithm on three software, Univocity parser, marc4j, and Jsoup. The initial outcomes revealed that execution time was reduced in the software, identifying defect efficiency was precise, the software occupied less disk, and code coverage was at its maximum. Furthermore, a rather altered version of the standard diversity DD for nominating the test suite that matches the whole set of test necessities and reaches the constraint for minimum cost was proposed by Sugave et al. (2017). Removing the redundant test case while not impairing the software quality and ensuring that the more significant test cases are disregarded are considered the primary functions of the proposed DD. The experimentation was conducted by employing five subject programs compared with TBAT, Greedy Irreplaceability, and systolic genetic search proving the proposed DD was more capable of making reductions. Additionally, the percentage of improvement in the offered DD was larger and had a lower value of variance and attained a lower cost for execution.

Kabir et al. (2017) aimed to attain better convergence, inspected an FP algorithm optimization, and reformulated an algorithm. The suggested technique was a modified flower pollination algorithm that is developed by utilizing a strategy and iteratively renovating the parameter's step lengths. Such a strategy aimed to accelerate convergence. Consequently, the suggested algorithm has a high potential to be employed by researchers and engineers specializing in software tests to achieve an optimal test suite that requires the least time for software testing. Nevertheless, to find solutions for the significant problem with mammoth computational cost and to match the required limits for accuracy and convergence, an optimization algorithm that is much more efficient is needed.

Table 15 Classification of recent studies and other information details on ABC-based methods

Paper	Algorithm	Main idea	Evaluation method	Tool (s)	Advantage (s)	Disadvantage (s)
Aghdam and Arasteh (2017)	ABC	Test data generation optimization with ABC	Simulation	MATLAB	Low time Low cost High reliability	Low performance Low efficiency
Singhal et al. (2021)	ABC	Optimizing test data generation using ABC	Simulation	MATLAB, AspectJ	High coverage Low time Low complexity	Not being tested on more complex software to authenticate the provided outcomes

Table 16 Comparison of existing ABC-based methods evaluation metrics

Paper	Time	Cost	Performance	Efficiency	Code coverage	Reliability
Aghdam and Arasteh (2017)	✔	✔				✔
Singhal et al. (2021)	✔		✔	✔	✔	✔

Moreover, Ghaemi and Arasteh (2019) prepared an SFL algorithm that automatically generates data for software tests. The main focus was on four criteria which are as follows: average time, the average number of generations, success rate, and the branch coverage of the data generation. As the outcomes show, based upon SFL, the presented method has specific benefits over the previous EAs such as PSO, GA, ACO, and ABC. An average branch coverage of 99.99% was achieved via the proposed SFL algorithm, the rate for success was 99.97%, the average number of the generations produced for covering the whole number of branches was 2.03, and the average number of branches runtime was 0.15 milliseconds.

Fakhouri et al. (2020) introduced an EMVO algorithm to elevate the performance of software testing. They also compared the presented algorithm with PSO, sine cosine algorithm, grey wolf optimizer, and multiverse optimizer. Their experimented results depicted the optimized behavior of EMVO in comparison with other nature-inspired algorithms. Sivaji and Srinivasa Rao (2021) expressed the positive points of the AB-CNS algorithm in the regression of software testing. Initially, they gathered existing datasets; subsequently, they applied AB-CNS to organize and minimize test cases to elevate the performance of the used test cases. At the last stage, the simulation results illustrated that defect detection of the model outperformed, and resource consumption gradually declined. Arasteh and Hosseini (2022) used Traxtor to automize the test generation process. Simultaneously, they utilized (ICA) to enhance the effectiveness of code coverage, branch coverage, and success rate.

To resolve the challenge of imbalanced distribution of the classes in the datasets, Khuat and Le (2019) proposed BTLBOd initiate a balanced dataset. The authors took out experiments on nine imbalanced software fault datasets extracted from the Promise repository. However, the authors plan to implant BTLBOd to other optimization issues and have their methods evaluated based on accuracy and performance. Dhavakumar and Gopalan (2021) used the Pham–Zhang model to predict parameters in software defects. Also, to enhance the efficiency and reliability of the proposed method, the authors utilized Tandem datasets to have the capability to compare their work. Having compared the CGWO with PSO, the authors detected that the former performed better integration and had higher reliability.

4.7.2 Summary of other metaheuristic methods

In Table 17, the categorization of the aforementioned studies and the influential factors for analyzing other metaheuristic methods in software testing are explained in further detail. In Table 18, an evaluation of the studies mentioned is shown. The following factors include time, cost, performance, efficiency, code coverage, quality, reliability, resource utilization, and accuracy. Research studies have mainly evaluated their presented approach regarding cost, time, performance, and efficiency factors in other metaheuristic methods.

5 Analysis of results

This section presents a comparative analysis of the selected nature-inspired metaheuristic algorithms in software testing in Sect. 3. A recap of the nominated studies in Sect. 3.2 is represented in Sect. 5.1. Next, by answering the primary RQs in Sect. 3.1, we proposed a taxonomical categorization and comparison of the studies in Sect. 5.2.

5.1 Overview of the primary studies

In the analysis of the recently published literature in nature-inspired metaheuristic algorithms in software testing, there are a couple of supplementary questions (SQs) yet to be answered:
SQ_1: At which stage in the history of software testing did research on nature-inspired metaheuristic methods become active among the communities? Which year has more publications?
SQ_2: What are the fora in which publications on nature-inspired metaheuristic methods in software testing have been made? In which communities do they concentrate?
SQ_3: What are the research communities actively researching these days on nature-inspired metaheuristic methods in software testing research?

5.1.1 Temporal overview of studies

According to Fig. 4, 35% of the papers were published in IEEE, 25% in Springer, 20% in ScienceDirect, 6% in Taylor & Francis, 3% in Emerald, 3% in Inderscience, 3% in Hindawi, 2% Wiley, 2% ACM, and 1% World

Table 17 Classification of recent studies and other information details on other metaheuristic methods

Paper	Algorithm (s)	Main idea	Evaluation method	Tool (s)	Advantage (s)	Disadvantage (s)
Haraty et al. (2018)	SO-SA	Web applications being tested by using SA algorithms	Prototype	DOM, HTML, AJAX	Low energy High efficiency High coverage High performance	Energy function in large applications is not optimum
Alsariera et al. (2015)	BA	Using BA to decrease the SPL tests	Simulation	Pseudo-code	High availability High quality	Low scalability Low efficiency
Devika Rani Dhivya and Meenakshi (2018)	BFA	Improving ARPT 1 and ARPT 2 parameters	Case study	Java	High performance High coverage Low time High efficiency Low memory utilization	High complexity
Sugave et al. (2017)	DD	Reduction in cost and increasing quality of the test suite	Real testbed	Java	High quality Low time Low cost High Efficiency	High complexity
Kabir et al. (2017)	FP	Optimization by FP	Simulation	Java	Low time	Low scalability High cost Low accuracy
Ghaemi and Arasteh (2019)	SFL	Software structural test case generation by SFL	Simulation	MATLAB	High coverage High efficiency Low time Low cost	Low scalability Low reliability
Fakhouri et al. (2020)	EMVO	An approach for improving testing output by using EMVO	Simulation	MATLAB	High performance High efficiency High accuracy	Low reliability
Sivaji and Srinivasa Rao (2021)	AB-CNS	Using AB-CNS to improve the test model	Simulation	Python	High performance High efficiency Low time Low resource utilization High accuracy	Not testing the approach in real-world conditions
Arasteh and Hosseini (2022)	ICA	Traxtor in generating the test data by the usage of ICA	Simulation	Not mentioned	High success rate High coverage	Low generation rate

Table 17 (continued)

Paper	Algorithm (s)	Main idea	Evaluation method	Tool (s)	Advantage (s)	Disadvantage (s)
Khuat and Le (2019)	BTLBOd	Software defect prediction with BTLBOd	Simulation	Pseudo-code, Java	High accuracy High performance	Not applicable on multi-class imbalanced data
Dhavakumar and Gopalan (2021)	CGWO	Parameter optimization by CGWO	Simulation	Pseudo-code, Chebyshev graph	High reliability High accuracy High performance	Not covering a broad term of the algorithm to have their work compared with parameters

Table 18 Comparison of existing other metaheuristic methods evaluation metrics

Paper	Time	Cost	Performance	Efficiency	Code coverage	Quality	Reliability	Resource utilization	Accuracy
Haraty et al. (2018)	✔	✔	✔	✔	✔				
Alsariera et al. (2015)			✔	✔		✔			
Devika Rani Dhivya and Meenaksh (2018)	✔	✔	✔	✔	✔	✔			
Sugave et al. (2017)	✔	✔		✔		✔			
Kabir et al. (2017)	✔								
Ghaemi and Arasteh (2019)	✔	✔		✔	✔				
Fakhouri et al. (2020)			✔	✔		✔	✔		
Sivaji and Srinivasa Rao (2021)	✔	✔	✔	✔		✔		✔	✔
Arasteh and Hosseini (2022)	✔		✔	✔		✔		✔	
Khuat and Le (2019)			✔	✔		✔		✔	✔
Dhavakumar and Gopalan (2021)			✔	✔		✔		✔	✔

Scientific. In addition, Fig. 5 illustrates the number of published journal and conference papers from 2015 to 2022. Also, it depicts that the year 2015 experienced the highest number of published papers.

5.1.2 Publication fora

In Table 19, the papers are shown to have been published in two categories, including conferences and journals. Among the 65 included papers and the ones that were published in IEEE, three were published in Congress on Evolutionary Computation (CEC), and two were published in the International Conference on Intelligent Computing and Control Systems (ICICCS) conference. In addition, it is apparent that most of the papers were published in journals, as Table 19 indicates.

5.1.3 Active research communities

The subsequent distribution is related to active research communities. Author affiliations were examined after the study selection process ended, and the study synthesis for eradicating affiliation bias was finished. Initially, we recognized 105 active communities related to the subject; however, only active communities proposing at least two studies were selected, and the focus of their research studies is depicted in Table 20. A substantial number of studies are connected with optimizing test case generation in software testing, which is issued by researchers at Islamic Azad University, Iran. They predominantly focused on ACO, FA, ABC, and SFL algorithms to optimize the test case generation.

Researchers at the Victoria University of Wellington, New Zealand, concentrated on software testing based on GA methods, and their counterparts at Andhra University,

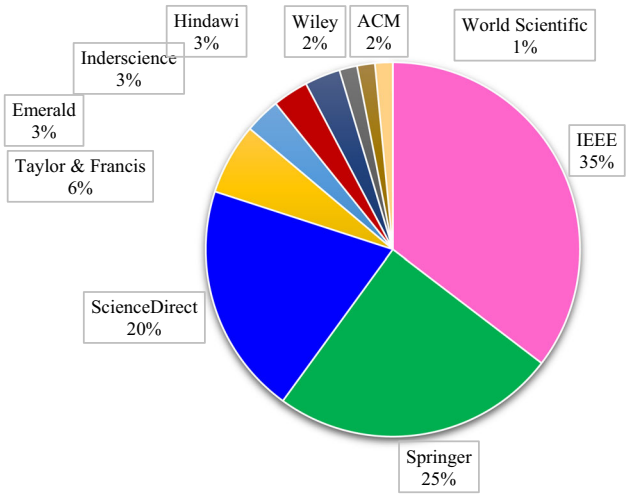


Fig. 4 Percentage of selected papers classification

India, researched software testing optimization utilizing GA and AB-CNS algorithms. Researchers of Bharati Vidyapeeth University College of Engineering, India, and Amity University, India, centrally focused on test case generation in software testing based on GA, ABC, and DD algorithms. In addition, researchers of Salahaddin University-Hawler (SUH), Kurdistan, worked on software testing utilizing PSO and CS algorithms. Furthermore, test data generation based on hybrid methods, PSO and Memetic, was researched by researchers at Toosi University of Technology, Iran. Researchers at Mepco Schlenk Engineering College and Ramco Institute of Technology mainly discussed Rao optimization along with hybrid social simulation optimization algorithms. Lastly, software fault prediction was the topic of research at the University of Kebangsaan, Ajman University, and Al-Balqa Applied University.

5.2 Research objectives, techniques, and evaluation metrics

In this section, we answer the research questions, which are presented in Sect. 3.1. The research questions give the necessary details on what one ought to exploit by reading the presented survey, and they also point to the path for further discussions. To respond to RQ1, the previous sections described the most popular software test methods in seven main categories. The classification consists of GA-based, hybrid-based, ACO-based, CS-based, FA-based, ABC-based, and other metaheuristic methods. It is evident in Fig. 6 that 62% of the selected papers in software testing worked on GA-based and hybrid methods, collectively, 17% on other metaheuristics, 6% on ACO-based methods, 7% on FA-based methods, 5% on CS-based methods, and 3% on ABC-based methods.

Table 21 provides details on the plus and negative points of selected classification for software testing techniques as follows: The GA-based methods are algorithms that try to reduce time, cost, and most of the time, increase efficiency, performance, and code coverage. The disadvantages of GA-based techniques are low scalability and high complexity. Our review in Sect. 5 shows that the GA-based methods have used different GA methods, including GA, NEAT, and RG. Moreover, the hybrid methods combine various methods that try to increase efficiency, performance and reduce time and cost. Same as GA-based methods, the disadvantages of hybrid methods are low scalability and high complexity. Based on our review, the hybrid methods use different combination methods, including MA, GA, PSO, CMSA, NSGA-II/III, IBEA, SPEA2, Memetic, SRGM, MGSO, TEF, EA-CSC, GLS, SA, K-A, WOA, SA, SSA-SA, MCDM, Rao, SAA-BPNN, and ESAMP-SMO.

In ACO-based methods, the technique always tries to mount the code coverage and increase the performance, efficiency, and time and cost. But, a disadvantage of ACO-based methods is low scalability. Also, the CS-based methods decrease costs and enhance efficiency. The disadvantages of CS-based methods are high complexity, low scalability, and reliability. The FA-based methods are practical in software testing, especially in terms of reducing costs and elevating performance. Nevertheless, the

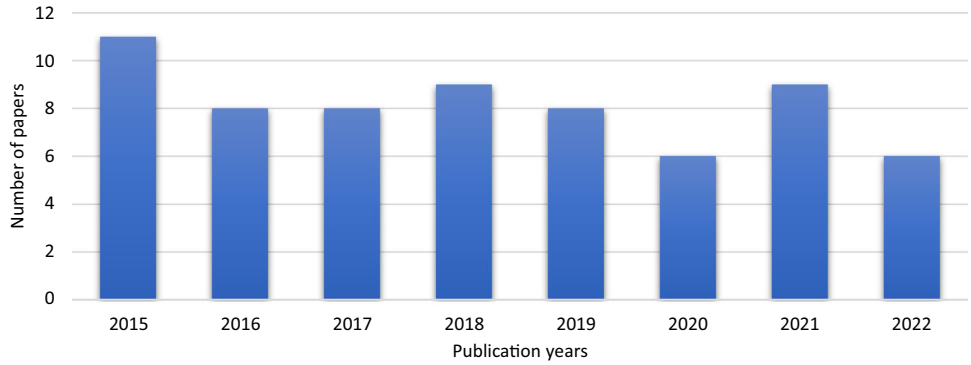


Fig. 5 Number of papers based on publication years

Table 19 Study distribution per publication channel

Category	Publication channel	Abbreviation	Count
Conferences	IEEE Congress on Evolutionary Computation	CEC	3
	International Conference on Intelligent Computing and Control Systems	ICICCS	2
	International Congress on Technology, Communication, and Knowledge	ICTCK	1
	International Conference on Electrical Engineering and Information Communication Technology	ICEEICT	1
	IEEE UP Section Conference on Electrical Computer and Electronics	UPCON	1
	International Conference on Computing Communication Control and Automation	ICCUBEA	1
	Brazilian Symposium on Software Engineering	SBES	1
	Conference on Swarm Intelligence and Evolutionary Computation	CSIEC	1
	International Conference on Research Advances in Integrated Navigation Systems	RAINS	1
	International Conference on Electrical Engineering, Computer Science and Informatics	EECSI	1
	International Conference on Electrical Information and Communication Technology	EICT	1
	IEEE International Students' Conference on Electrical, Electronics and Computer Science	SCEECS	1
	International Conference on Intelligent Computing	ICIC	1
	Australasian Joint Conference on Artificial Intelligence	AusAI	1
	International Conference on Swarm Intelligence	ICSI	1
	International Conference on Information and Communication Technology for Competitive Strategies	ICTCS	1
	Proceedings of First International Conference on Mathematical Modeling and Computational Science	ICMMCS	1
	International Conference on Engineering Technologies and Applied Sciences	ICETAS	1
	International Conference on Web Research	ICWR	1
	International Conference on Software Engineering and Computer Systems	ICSECS	1
	International Conference on Cloud Computing, Data Science & Engineering	Confluence	1
	International Scientific-Technical Conference on Actual Problems of Electronic Instrument Engineering	APEIE	1
	International Conference on Reliability, Maintainability, and Safety	ICRMS	1
	Symposium Series on Computational Intelligence	SSCI	1
Journals	Applied Soft Computing	ASOC	3
	Information and Software Technology	IST	2
	IEEE Access	IEEE Access	2
	Expert Systems with Applications	ESWA	2
	Journal of King Saud University—Computer and Information Sciences	JKSUCIS	2
	Memetic Computing	MEMET	1
	Empirical Software Engineering	EMSE	1
	Swarm and Evolutionary Computation	Swarm Evol. Comput	1
	International Journal of Software Engineering and Knowledge Engineering	IJSEKE	1
	International Journal of Computers and Applications	TJCA	1
	Journal of Software: Evolution and Process	JSME	1
	International Journal of Bio-Inspired Computation	IJBIC	1
	Modelling and Simulation in Engineering	MSE	1
	International Journal of Quality & Reliability Management	IJQRM	1
	Modelling Foundations and Applications	ECMFA	1
	Journal of Heuristics	HEUR	1
	Artificial Intelligence Review	AIRE	1
	Journal of Interdisciplinary Mathematics	JIM	1
	Data Technologies and Applications	DTA	1
	International Journal of Advanced Intelligence Paradigms	IJAIP	1

Table 19 (continued)

Category	Publication channel	Abbreviation	Count
	Applied Artificial Intelligence	UAAI	1
	Computational Vision and Bio Inspired Computing	ICCVBIC	1
	Journal of Experimental & Theoretical Artificial Intelligence	JETAI	1
	Wireless Communications and Mobile Computing	WCMC	1
	Journal of Electronic Testing	JETTA	1
	Materials Today: Proceedings	MATPR	1
	Journal of Ambient Intelligence and Humanized Computing	JAIHC	1
	Soft Computing	SOCO	1
	Cluster Computing	Clust. Comput	1
	Computing	Comput	1
	Knowledge-Based Systems	KBS	1
	Journal of Systems and Software	J. Syst. Softw	1

Table 20 Active communities and their research focus

Affiliation	Study ID	Research focus
Islamic Azad University, Iran	(Sayyari and Emadi 2015), (Shomali and Arasteh 2020), (Aghdam and Arasteh 2017), (Ghaemi and Arasteh 2019)	Optimization of test case generation in software testing using ACO, FA, ABC, and SFL algorithms
Victoria University of Wellington, New Zealand	(Koleejan et al. 2015), (Huo et al. 2017)	Software testing with GA algorithms
Andhra University, India	(Koteswara Rao and Raju 2019), (Sivaji and Srinivasa Rao 2021)	Software testing optimization using GA and AB-CNS algorithms
Bharati Vidyapeeth University College of Engineering, India	(Mahajan et al. 2015), (Sugave et al. 2017)	Test case generation in Software testing using GA and DD algorithms
Amity University, India	(Kumar and Sahni 2020), (Singhal et al. 2021)	Test case generation and resource allocation using GA and ABC algorithms
Salahaddin University-Hawler (SUH), Kurdistan	(Mahmoud and Ahmed 2015), (Ahmed et al. 2015)	Software testing using PSO and CS algorithms
Toosi University of Technology, Iran	(Damia et al. 2021), (Esnaashari and Damia 2021)	Test data generation based on hybrid methods using PSO and Memetic algorithms
Mepco Schlenk Engineering College, India	(Thirumoorthy and J, 2022), (Thirumoorthy and Britto 2022)	Using binary Rao optimization algorithm for feature selection model and clustering approach to predict software failures using a hybrid social simulation optimization algorithm
Ramco Institute of Technology, India	(Thirumoorthy and J, 2022), (Thirumoorthy and Britto 2022)	Using binary Rao optimization algorithm for feature selection model and clustering approach to predict software failures using a hybrid social simulation optimization algorithm
University of Kebangsaan, Malaysia	(Kassaymeh et al. 2022b), (Kassaymeh et al. 2022a)	Software fault prediction
Ajman University, United Arab Emirates	(Kassaymeh et al. 2022b), (Kassaymeh et al. 2022a)	Software fault prediction
Al-Balqa Applied University, Jordan	(Kassaymeh et al. 2022b), (Kassaymeh et al. 2022a)	Software fault prediction

algorithm is not capable of increasing mutation scores and quality.

ABC-based methods are reliable, as well as applicable, in reducing the needed time for software testing. However, low code coverage, efficiency, and high costs are the

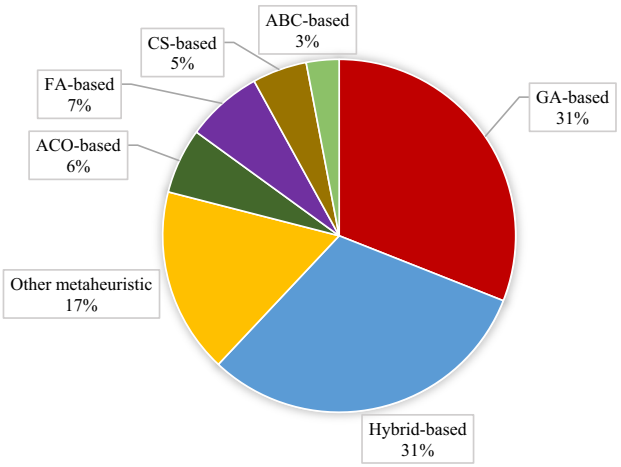


Fig. 6 Percentage of selected papers classification

negative sides of the ABC-based methods. Other metaheuristic methods did not involve the rest of the introduced taxonomies. Most of the time, they led to a decrease in time and cost while boosting efficiency. Also, the disadvantages of other methods are low scalability and reliability. Based on our review, the other metaheuristic methods are included by SO-SA, BA, BFA, DD, FP, SFL, EMVO, AB-CNS, ICA, BTLBOd, and CGWO.

To obtain the most significant and influential parameters of RQ2 and their percentage, we use the formula that is shown below. The number of occurrences of parameter $i\left(Occure_{no(i)}\right)$ is counted separately and divided by the sum of all parameters (Eq. (1)). The percentage of parameter $\mathrm{i}\left(\mathrm{Imp}_{\mathrm{percentage(i)}}\right)$ is determined by the multiplication of its calculated value and the number 100 [25].

$$\mathrm{Imp}_{\mathrm{percentage}(i)} = \frac{Occure_{no(i)}}{\sum_{j=1}^{param_{no}} Occure_{no(j)}} \quad (1)$$

As specified in Fig. 7, it is shown that due to the essential parameters of the software test methods, 18% of the selected papers have improved efficiency, 16% of them have increased performance, and 24% of the selected papers have lowered time and cost, collectively. 11% increased code coverage, and 31% tried to optimize software testing in other metrics. Also, Fig. 8 represents the taken evaluation metrics in each category. By considering (1), we have tried to provide an accurate proportion of each used metric in every group.

To answer the RQ3, as specified in Fig. 9, 56% of the evaluation method in selected papers are simulations, 15% are case studies, 11% are prototypes, 9% are real testbeds, 6% are designed with models, and 3% are formals. To specify RQ4, according to Fig. 10, nearly a third of the

Table 21 Main advantages and disadvantages of selected classification for software test techniques

Classification	Advantage	Disadvantage
GA-based	Appreciable efficiency	Unsatisfactory scalability
	Appreciable time	
	Appreciable cost	
	Appreciable performance	
	Appreciable code coverage	
Hybrid	Appreciable efficiency	Unsatisfactory scalability
	Appreciable cost	
	Appreciable performance	
	Appreciable time	
ACO-based	Appreciable code coverage	Unsatisfactory scalability
	Appreciable efficiency	
	Appreciable time	
	Appreciable cost	
CS-based	Appreciable cost	Unsatisfactory scalability
	Appreciable efficiency	Unsatisfactory reliability
FA-based	Appreciable cost	Unsatisfactory mutation score
	Appreciable performance	Unsatisfactory quality
		Unsatisfactory time
ABC-based	Appreciable time	Unsatisfactory cost
	Appreciable reliability	Unsatisfactory efficiency
		Unsatisfactory code coverage
Other metaheuristics	Appreciable time	Unsatisfactory scalability
	Appreciable cost	Unsatisfactory reliability

articles used other tools, and a fifth of the studies did not mention their utilized tool. However, Java was used in 16% of selected papers, MATLAB in 9%, Pseudo-code in 17%, Python in 5%, and C++ in 3%.

6 Open issues and future directions

Some vital issues about software testing techniques exist that have not yet been thoroughly and extensively investigated. To specify RQ5, according to Fig. 11, major challenges and future trends of software testing approaches are presented in this section. In this research, the SLR method has been used to gather information; therefore, the challenges and issues currently considered to be open issues are mentioned in the following sections. Such challenges and trends for the future have not been accurately and holistically presented in the reviewed papers.

- *Multi-objective optimization*: While reviewing and analyzing research papers, we understand that multi-objective goals in nature-inspired metaheuristic in software testing were not considered simultaneously in most papers. The focus of each paper is on a distinct challenge with varying features. For illustration, consider the following scenario: Scalability or response time is provided by a few techniques; however, some of the other techniques completely ignore such metrics. Hence, multi-objective goals are ignored mainly by the available techniques, and consequently, offering a method that is both effective and multi-objective for the parameters of software testing approaches while also aiming to create a trade-off between them can be considered a tempting open issue.
- *Accuracy*: Test accuracy is a critical challenge for software test methods. Due to the reviewed papers, the test accuracy did not check as a critical test result. This concept is the quality of proper scientific software. Evaluating this quality is one way to ensure confidence in the outcomes of simulations. The measure of quality is an accurate result. The concept of accuracy uses as a characteristic of validation and verification in scientific software tests. Accuracy and the related concepts of precision are pretty ambiguous in every language, but they should be clear when measured as a quality metric (Boisvert et al. 2005).
- *Heuristic and metaheuristic-based methods*: In the nominated studies, different metaheuristic-based methods in software testing are considered a subset of NP-complete or NP-hard class in terms of complexity. However, most of the researchers ignored heuristic-based-methods categories. Thus, employing heuristic or metaheuristic method is the optimum solution to solve them more rapidly when conventional methods are much slower or to find an approximate solution when classic methods fail when trying to find any precise solution. List of some of these methods are containing: grey wolf optimizer (Mirjalili et al. 2014), bacterial colony optimization (Niu and Wang 2012), artificial immune system (Dasgupta et al. 2003), bat algorithm (Yang 2010b), glowworm swarm optimization (Krishnanand and Ghose 2009), Hill-climbing (Hinson and Staddon 1983), Best First Search (Baby and Banu 2019), A* search algorithm, OA* search (Antelman et al. 2005), local beam search (Ow and Morton 1988), Constraint Satisfaction (Nadel 1989) and pruning algorithm (Margineantu and Dietterich 1997) and other algorithm. Hence, examination of the previously mentioned optimization techniques and algorithms could be a promising future work.
- *Formal methods*: The experimenters do not consider the challenge of using formal methods in software testing in the vast majority of the investigated methods. Formal methods are techniques employed when an attempt to create a mathematical model for complex systems is made. The possibility of thoroughly validating the properties of the system is increased when a precise model of a complex system is built mathematically compared with when empirical testing is employed. In comparison with other design systems, formal methods utilize mathematical proof hand-in-hand with system testing to complement one another to ensure correct behavior. With systems getting increasingly complicated and, consequently, safety becoming increasingly important, another level of insurance is offered by the formal approach to system design (Isaksen et al. 1996; Stocks 1993). So, it would be interesting for researchers to evaluate test optimization by using formal methods in software testing and incorporating such factors into existing methods is possible to result in higher efficiency.
- *Scalability*: Most of the existing papers did not consider software testing scalability. The scalability in software testing is a significant factor both in performance evaluation and in the analysis of parallel and distributed systems. Usually, the capability of a system to maintain its performance when the system ensemble size is scaled up is conceptualized as the scalability of the system. Even though studies for scalability have been primarily conducted in distributed computing, parallel computing, databases, and service-oriented architecture there are still several researchers nowadays that are using the ambiguous explanations that address scalability challenges faced when the relation between system load and performance is to be considered (Chen and Sun 2006; Hill 1990). Thus, adopting scalability by

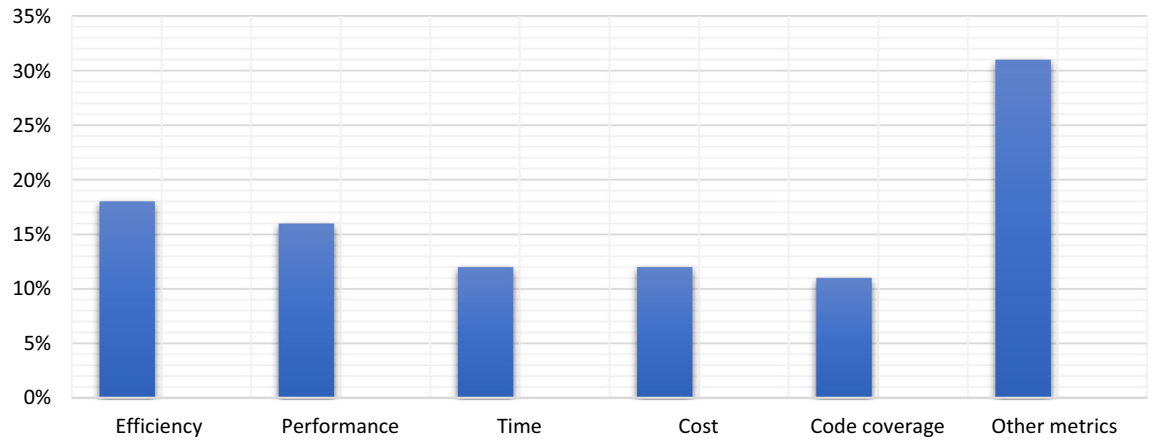
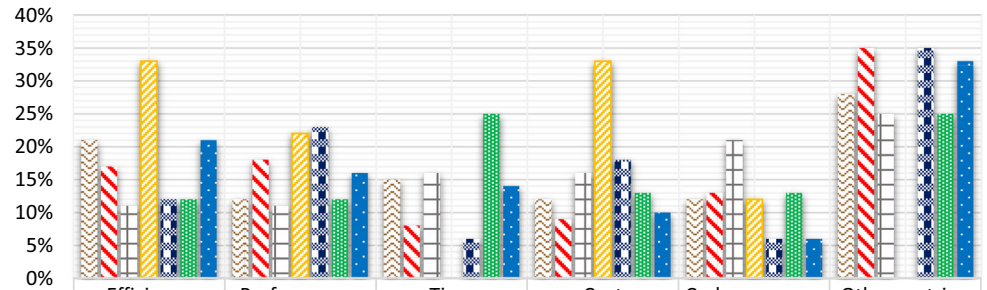


Fig. 7 Percentage of the improved evaluation metrics in the reviewed technique



	Efficiency	Performance	Time	Cost	Code coverage	Other metrics
GA	21%	12%	15%	12%	12%	28%
Hybrid	17%	18%	8%	9%	13%	35%
ACO	11%	11%	16%	16%	21%	25%
CS	33%	22%	0%	33%	12%	0%
FA	12%	23%	6%	18%	6%	35%
ABC	12%	12%	25%	13%	13%	25%
Other metaheuristics	21%	16%	14%	10%	6%	33%

Fig. 8 Percentage of selected papers evaluation metrics in each classification

current methods may be an exciting roadmap, and therefore, this issue is still challenging.

- *Failure management*: Our study symbolizes that there is no single reason for software projects to succeed. A number of methods that can level up project success rates are researched. Nonetheless, an overwhelming majority of these methods concentrate on recognizing and declination the influence of segments that can lead to project failure. The early forecasting of software system failures can assist in making decisions about possible solutions that could improve the success rate (Singh and Singh 2012). Failures are defects which are occurred while the test phase is happening and may cause software failure. Also, failures do not always occur as a response to undiscovered defects or unresolved bugs. The cause for faults in hardware or firmware may be due to environmental circumstances or a fault, including a wrong input value used during the test phase. Simply put, the conditions which cause failure may differ (Graham et al. 2008; Pan 1999). This is why incorporating failure management into existing methods could be of interest for future studies.
- *Test coverage*: The reviewed approaches showed that most of the researchers focused on software testing optimization and test case reduction, but in most of the reviewed papers, the authors ignored the test coverage concept. Test coverage in software testing can be an interesting subject as a metric that measures the amount of testing performed by a set of tests. Software test-coverage measures evaluate the degree of comprehensiveness of testing (Malaiya et al. 2002). These approaches techniques can be statements coverage, branch coverage, and path coverage which can be still challenging

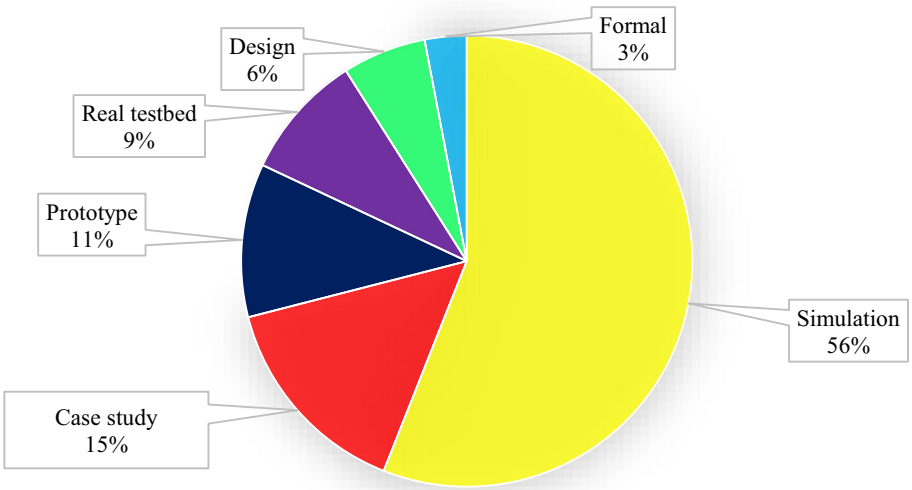


Fig. 9 Percentage of selected papers evaluation method for each classification

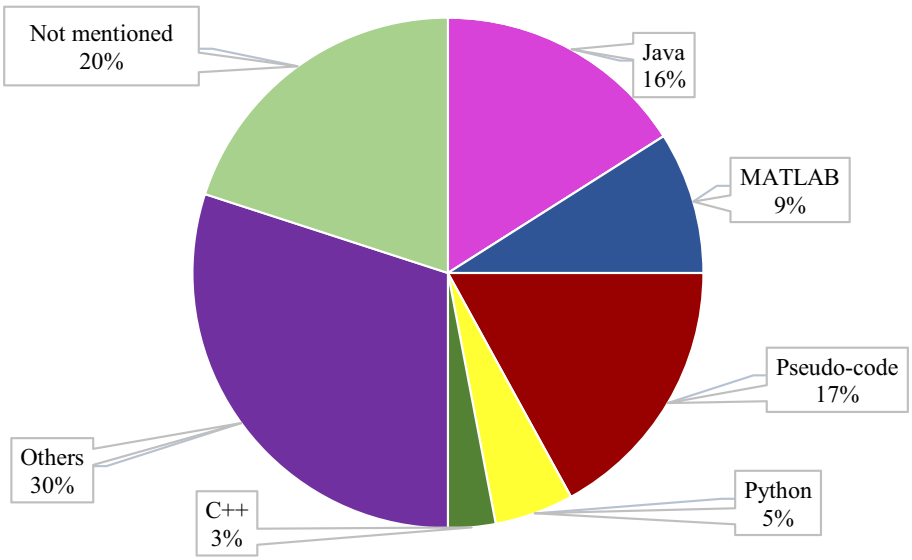


Fig. 10 Percentage of selected papers evaluation tools for each classification

- *Non-heuristic-based*: As non-heuristic-based approaches suggest, other promising lines for open issue research could be investigated by employing novel optimization methods, namely: graph theory (West 1996), linear programming (Murty 1983), and dynamic programming (Bellman 1966). Also, considering semantic information compared to these approaches, especially in a dynamic and distributed environment, might be the intriguing main route for future work (Zelenka et al. 2012). Moreover, analyzing software testing satisfaction and storing such info as a pattern for future software testing is indeed very enchanting.
- *Trust & Security*: A crucial challenge for software testing is trust management. The trust-based similarity methods make the interconnection between every potential system and request of user safe to provide support of permanency and political conclusions. The critical challenge of this is for the critical environment to be supported by the software with a safe condition. In addition, software security testing is testing that intends to uncover vulnerabilities and specifications that the system protects its data and resources against potential intruders (Potter and McGraw 2004; Thompson 2003). The focus area in software security testing includes system software security, network security, server-side application security, and client-side application security that can be used for future work. It is interesting for the researchers to evaluate software trust & security testing factors in any area.

Fig. 11 Software testing open issues and a future worklist

- *Interoperability*: Interoperability testing is a kind of software testing that checks for the software's capability to interoperate with another software component, software, or systems and in software different versions. A major benefit of standards-based products is, indeed, successful interoperation. Irrespective of the standards being proprietary or international/public, most users expect that the products backed by standards interact with similar products without interruptions (van der Veer and Wiles 2008). Testing in numerous forms is regularly hired to aid users in making decisions about what products to buy that will operate by other products and match with the same standard. The individual levels in interoperability testing include specification level interoperability, data-type interoperability, physical interoperability, and semantic interoperability, which can be an exciting topic for future work.
- *Oracle problem:* Much of the test process is practical and pleasing, but there is a need to make testing quicker, less costly, and more reliable. A test oracle should be formed to answer such necessity, which is simply a procedure differentiating between the correct and incorrect behaviors the system shows under test. However, there has been significantly less attention to the issue of test oracle automation, as opposed to many aspects of test automation, and it remains by far less well solved (Barr et al. 2014; Singhal et al. 2016). A concerted effort is needed in enhancing testing techniques to attain test automation that is greater and more wide-ranging, ways to address the test oracle problem that are much better, and test oracle solutions that are automated or partially automated.

7 Limitations

We attempted to provide a methodic and broad-ranging review of the software testing by using inspired-nature metaheuristic methods. However, it is possible that constraints still exist. Accordingly, future studies need to take the limitations of the current study into account, which are stated below:

- Research scope: The application of the nature-inspired metaheuristic methods for software testing has been included in various sources: thesis, editorial notes, academic publications, technical reports, and web pages, to name but a few. Nevertheless, our focus was

only on the main academic international journals and conferences to gain the best qualification. Additionally, we have omitted articles in national conferences and journals, along with those not in English.
- Classification: We categorized papers into seven main sections, including GA-based methods, hybrid-based methods, ACO-based methods, CS-based methods, FA-based methods, ABC-based methods, and other metaheuristic methods; however, a different categorization is also possible.
- Research query: Five RQs were considered in the proposed review; however, it is possible to consider more queries and different RQs.

Study and publication bias: To assure the reliability of electronic sources, several famous databases such as IEEE, Springer, ScienceDirect, ACM, and Google Scholar, were chosen for this paper. As statistics indicate, these famous databases offered the most relevant and authentic research; however, it is not guaranteed that all relevant studies have been chosen. Therefore, there is a possibility that some studies to have been overlooked. Furthermore, several related papers are likely to have been disregarded during the process of paper exclusion in Sect. 3.2.

8 Conclusion

This paper provided an SLR of nature-inspired metaheuristic mechanisms in software testing. The metaheuristic methods in software testing were investigated, and then the advantages and disadvantages of the systematically surveyed method were discussed and compared. This SLR selected 65 papers out of 779 journals and conferences between the years 2015 and 2022. Between the given period, 2015 accounted for the highest number of published articles; however, the years 2020 and 2022 saw the least number of published papers. As the results indicate, the vast majority of articles were published by IEEE and Springer with a proportion of 35% and 25%, respectively. Conversely, Wiley, ACM, and World Scientific comprised the least number of papers. These articles were divided into seven major metaheuristic-based methods: GA-based, hybrid-based, ACO-based, CS-based, FA-based, ABC-based, and other metaheuristics methods. The review result showed that researchers had published more papers using GA-based and hybrid methods with 62%, collectively. Based on statistics, the most commonly used tools and platforms were Java and MATLAB, of which 16% and 9% of them were used in the reviewed articles, respectively. As the comparison of evaluation methods showed, 56% of the studies applied a simulation environment. Having reviewed the nominated papers, some metrics were used for evaluation, including efficiency, performance, time, cost, code coverage, and other metrics. In many of the reviewed papers, other metrics, such as scalability, branch coverage, statement coverage, complexity, reliability, mutation score, and success rate, were considered the most. Nevertheless, 18% of the articles tried to improve testing efficiency by using metaheuristic methods in software testing. As reviewed papers illustrated, the authors emphasized some issues that can be examined more closely in future studies. They can help future researchers to employ appropriate techniques for the issues discussed. Eventually, we sincerely wish for the results of this research to be helpful in software testing techniques, mainly when nature-inspired metaheuristic algorithms are employed.

Author contributions The first and second authors have equal contributions to this work.

Funding No funding is provided for the preparation of the manuscript.

Data availability Enquiries about data availability should be directed to the authors

Declarations

Competing interests The authors have no relevant financial or non-financial interests to disclose.

Ethics approval This article does not contain any studies with human participants.

References

Abkenar SB, Kashani MH, Akbari M, Mahdipour E (2020) Twitter spam detection: a systematic review. arXiv:201114754

Afzal W, Alone S, Glocksien K, Torkar R (2016) Software test process improvement approaches: a systematic literature review and an industrial case study. J Syst Softw 111:1–33

Aghdam ZK, Arasteh B (2017) An efficient method to generate test data for software structural testing using artificial bee colony optimization algorithm. Int J Software Eng Knowl Eng 27:951–966

Ahmadi Z, Haghi Kashani M, Nikravan M, Mahdipour E (2021) Fog-based healthcare systems: a systematic review. Multim Tools Appl 80:36361–36400

Ahmed BS, Abdulsamad TS, Potrus MY (2015) Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm. Inf Softw Technol 66:13–29

Alkawaz MH, Silvarajoo A (2019) A survey on test case prioritization and optimization techniques in software regression testing. In: 2019 IEEE 7th conference on systems, process and control (ICSPC), pp 59–64

Alsariera YA, Majid MA, Zamli KZ (2015) SPLBA: an interaction strategy for testing software product lines using the Bat-inspired

algorithm. In: 2015 4th international conference on software engineering and computer systems (ICSECS), pp 148–153
Alyahya S (2020) Crowdsourced software testing: a systematic literature review. Inf Softw Technol 127:106363
Anand S, Burke EK, Chen TY, Clark J, Cohen MB, Grieskamp W, Harman M, Harrold MJ, Mcminn P, Bertolino A (2013) An orchestrated survey of methodologies for automated software test case generation. J Syst Softw 86:1978–2001
Anbu M, Anandha Mala GS (2019) Feature selection using firefly algorithm in software defect prediction. Clust Comput 22:10925–10934
Antelman K, Bakkalbasi N, Goodman D, Hajjem C, Harnad S (2005) Evaluation of algorithm performance on identifying OA
Arasteh B, Hosseini SMJ (2022) Traxtor: an automatic software test suit generation method inspired by imperialist competitive optimization algorithms. J Electron Test 38:205–215
Asoudeh N, Labiche Y (2018) Life sciences-inspired test case similarity measures for search-based, FSM-based software testing. In: Pierantonio A, Trujillo S (eds) Modelling foundations and applications. Springer International Publishing, Cham, pp 199–215
Baby B and Banu S (2019) Enhanced security for dynamic multi keyword ranked search using greedy best first search
Bahaweres RB, Zawawi K, Khairani D and Hakiem N (2017) Analysis of statement branch and loop coverage in software testing with genetic algorithm. In: 2017 4th international conference on electrical engineering, computer science and informatics (EECSI), pp 1–6, IEEE
Bahrampour A, Rafe V (2021) Using memetic algorithm for robustness testing of contract-based software models. Artif Intell Rev 54:877–915
Barbosa G, de Souza ÉF, dos Santos LBR, da Silva M, Balera JM, Vijaykumar NL (2022) A systematic literature review on prioritizing software test cases using Markov chains. Inf Softw Technol 147:106902
Barr ET, Harman M, McMinn P, Shahbaz M, Yoo S (2014) The oracle problem in software testing: a survey. IEEE Trans Software Eng 41:507–525
Basili VR (1989) Software development: a paradigm for the future. In: Proceedings of the thirteenth annual international computer software & applications conference. IEEE, pp 471–485
Bazzaz Abkenar S, Haghi Kashani M, Mahdipour E, Jameii SM (2021) Big data analytics meets social media: A systematic review of techniques, open issues, and future directions. Telematics Inform 57:101517
Bellman R (1966) Dynamic programming. Science 153:34–37
Betts KM, Petty MD (2016) Automated search-based robustness testing for autonomous vehicle software. Model Simul Eng 2016:5309348
Bharathi M, Sangeetha V (2018) Weighted rank ant colony metaheuristics optimization based test suite reduction in combinatorial testing for improving software quality. In: 2018 second international conference on intelligent computing and control systems (ICICCS). IEEE, pp 525–534
Bianchi L, Dorigo M, Gambardella LM, Gutjahr WJ (2009) A survey on metaheuristics for stochastic combinatorial optimization. Nat Comput 8:239–287
Biswas S, Kaiser MS, Mamun S (2015) Applying ant colony optimization in software testing to generate prioritized optimal path and test data. In: 2015 international conference on electrical engineering and information communication technology (ICEEICT). IEEE, pp 1–6
Bluemke I, Malanowska A (2021) Software testing effort estimation and related problems: a systematic literature review. ACM Comput Surv 54:Article 53
Boisvert RF, Cools R and Einarsson B (2005) Assessment of accuracy and reliability, in *Accuracy and reliability in scientific computing* pp 13–32, SIAM.
Boopathi M, Sujatha R, Kumar CS and Narasimman S (2014) The mathematics of software testing using genetic algorithm, in *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization* pp 1–6, IEEE.
Boopathi M, Sujatha R, Kumar CS, Narasimman S, Rajan A (2019) Markov approach for quantifying the software code coverage using genetic algorithm in software testing. International Journal of Bio-Inspired Computation 14:27–45
Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M (2007) Lessons from applying the systematic literature review process within the software engineering domain. J Syst Softw 80:571–583
Burnstein I, Suwanassart T and Carlson R (1996) Developing a testing maturity model for software test process evaluation and improvement, in *Proceedings International Test Conference 1996 Test and Design Validity* pp 581–589, IEEE.
Carpenter BE and Doran R (1986) *AM Turing's ACE report of 1946 and other papers*, Massachusetts Institute of Technology.
Charette RN (2005) Why software fails [software failure]. IEEE Spectr 42:42–49
Chen Y and Sun X-H (2006) Stas: A scalability testing and analysis system, in *2006 IEEE International Conference on Cluster Computing* pp 1–10, IEEE.
Collins E, Dias-Neto A and de Lucena Jr VF (2012) Strategies for agile software testing automation: An industrial experience, in *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops* pp 440–445, IEEE.
Dadkhah M, Araban S, Paydar S (2020) A systematic literature review on semantic web enabled software testing. J Syst Softw 162:110485
Damia A, Esnaashari M and Parvizimosaed M (2021) Automatic Web-Based Software Structural Testing Using an Adaptive Particle Swarm Optimization Algorithm for Test Data Generation, in *2021 7th International Conference on Web Research (ICWR)* pp 282–286.
Dasgupta D, Ji Z and Gonzalez F (2003) Artificial immune system (AIS) research in the last five years, in *The 2003 Congress on Evolutionary Computation, 2003 CEC'03* pp 123–130, IEEE.
Dasoriya R and Dashoriya R (2018) Use of Optimized Genetic Algorithm for Software Testing, in *2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)* pp 1–5, IEEE.
DevikaRani Dhivya andMee-nakshi (2018)"in the text andTables 17 and 18 isinactive. DhivyaDR, K andMeenakshi VS,(2018) AnOptimized AdaptiveRandom PartitionSoftware Testing byUsing BacterialForagingAlgorithm. In:Hemanth DJ, SmysS (eds)ComputationalVision and BioInspiredComputing.SpringerInternationalPublishing, Cham,pp 542–555
de Souza ÉF, Falbo RdA, Vijaykumar NL (2015) Knowledge management initiatives in software testing: A mapping study. Inf Softw Technol 57:378–391
Devika Rani Dhivya K, Meenakshi VS (2018) An Optimized Adaptive Random Partition Software Testing by Using Bacterial Foraging Algorithm. In: Hemanth DJ, Smys S (eds) Computational Vision and Bio Inspired Computing. Springer International Publishing, Cham, pp 542–555
Dhavakumar P, Gopalan NP (2021) An efficient parameter optimization of software reliability growth model by using chaotic grey wolf optimization algorithm. J Ambient Intell Humaniz Comput 12:3177–3188
Dorigo M and Di Caro G (1999) Ant colony optimization: a new meta-heuristic, in *Proceedings of the 1999 congress on*

evolutionary computation-CEC99 (Cat No 99TH8406) pp 1470–1477, IEEE.

Esnaashari M, Damia AH (2021) Automation of software test data generation using genetic algorithm and reinforcement learning. Expert Syst Appl 183:115446

Esparcia-Alcázar AI, Almenar F, Vos TE, Rueda U (2018) Using genetic programming to evolve action selection rules in traversal-based automated software testing: results obtained with the TESTAR tool. Memetic Computing 10:257–265

Etemadi M, Bazzaz Abkenar S, Ahmadzadeh A, Haghi Kashani M, Asghari P, Akbari M, Mahdipour E (2023) A systematic review of healthcare recommender systems: Open issues, challenges, and techniques. Expert Syst Appl 213:118823

Fakhouri SN, Hudaib A, Fakhouri HN (2020) Enhanced Optimizer Algorithm and its Application to Software Testing. J Exp Theor Artif Intell 32:885–907

Fathi M, Haghi Kashani M, Jameii SM, Mahdipour E (2022) Big Data Analytics in Weather Forecasting: A Systematic Review. Archives of Computational Methods in Engineering 29:1247–1275

Ferrer J, Chicano F, Ortega-Toro JA (2021) CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. Journal of Heuristics 27:229–249

Fuggetta A (2000) Software process: a roadmap, in *Proceedings of the Conference on the Future of Software Engineering* pp 25–34, ACM.

Galin D (2004) *Software quality assurance: from theory to implementation*, Pearson Education India.

Gao R and Xiong S (2015) A genetic local search algorithm for optimal testing resource allocation in module software systems, in *International Conference on Intelligent Computing* pp 13–23, Springer.

Garousi V, Felderer M, Hacaloğlu T (2017) Software test maturity assessment and test process improvement: A multivocal literature review. Inf Softw Technol 85:16–42

Garousi V, Küçük B (2018) Smells in software test code: A survey of knowledge in industry and academia. J Syst Softw 138:52–81

Garousi V, Mäntylä MV (2016) A systematic literature review of literature reviews in software testing. Inf Softw Technol 80:195–216

Ghaemi A and Arasteh B (2019) SFLA-based heuristic method to generate software structural test data. *Journal of Software: Evolution and Process*:e2228.

Gillenson ML, Zhang X, Stafford TF and Shi Y (2018) A Literature Review of Software Test Cases and Future Research, in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* pp 252–256, IEEE.

Graham D, Van Veenendaal E and Evans I (2008) *Foundations of software testing: ISTQB certification*, Cengage Learning EMEA.

Gupta N, Sharma A and Pachariya MK (2020) Multi-objective test suite optimization for detection and localization of software faults. *Journal of King Saud University - Computer and Information Sciences*.

Haghi Kashani M, Mahdipour E (2023) Load Balancing Algorithms in Fog Computing. IEEE Trans Serv Comput 16:1505–1521

Haghi Kashani M, Madanipour M, Nikravan M, Asghari P, Mahdipour E (2021) A systematic review of IoT in healthcare: Applications, techniques, and trends. J Netw Comput Appl 192:103164

Haghi Kashani M, Rahmani AM, Jafari Navimipour N (2020) Quality of service-aware approaches in fog computing. Int J Commun Syst 33:e4340

Haraty RA, Mansour N, Zeitunlian H (2018) Metaheuristic Algorithm for State-Based Software Testing. Appl Artif Intell 32:197–213

Harman M, Jia Y and Zhang Y (2015) Achievements, Open Problems and Challenges for Search Based Software Testing, in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)* pp 1–12.

Hema Shankari K, Mathivilasini S, Arasu D and Suseendran G (2021) Genetic Algorithm Based on Test Suite Prioritization for Software Testing in Neural Network, in *Proceedings of First International Conference on Mathematical Modeling and Computational Science* (Peng S-L, Hao R-X and Pal S eds) pp 409–416, Springer Singapore, Singapore.

Hill MD (1990) What is scalability? ACM SIGARCH Computer Architecture News 18:18–21

Hinson JM, Staddon J (1983) Hill-climbing by pigeons. J Exp Anal Behav 39:25–47

Holland JH (1992) *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT press.

Huo J, Xue B, Shang L and Zhang M (2017) Genetic Programming for Multi-objective Test Data Generation in Search Based Software Testing, in *Australasian Joint Conference on Artificial Intelligence* pp 169–181, Springer.

Isaksen U, Bowen JP, Nissanke N (1996) System and software safety in critical systems. The University of Reading, Whiteknights, United Kingdom

Jamil MA, Alhindi A, Arif M, Nour MK, Abubakar NSA and Aljabri TF (2019) Multiobjective Evolutionary Algorithms NSGA-II and NSGA-III for Software Product Lines Testing Optimization, in *2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)* pp 1–5.

Jamil MA, Arif M, Abubakar NSA and Ahmad A (2016) Software Testing Techniques: A Literature Review, in *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)* pp 177–182.

Jin C, Jin S-W (2016) Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization. Appl Soft Comput 40:283–291

Kabir MN, Ali J, Alsewari AA and Zamli KZ (2017) An adaptive flower pollination algorithm for software test suite minimization, in *2017 3rd International Conference on Electrical Information and Communication Technology (EICT)* pp 1–5, IEEE.

Karaboga D, Basturk B (2008) On the performance of artificial bee colony (ABC) algorithm. Appl Soft Comput 8:687–697

Karimi Y, Haghi Kashani M, Akbari M, Mahdipour E (2021) Leveraging big data in smart cities: A systematic review. Concurrency and Computation: Practice and Experience 33:e6379

Kashani MH and Sarvizadeh R (2011) A novel method for task scheduling in distributed systems using Max-Min Ant Colony Optimization, in *2011 3rd International Conference on Advanced Computer Control* pp 422–426.

Kassaymeh S, Abdullah S, Al-Betar MA, Alweshah M (2022a) Salp swarm optimizer for modeling the software fault prediction problem. Journal of King Saud University - Computer and Information Sciences 34:3365–3378

Kassaymeh S, Al-Laham M, Al-Betar MA, Alweshah M, Abdullah S, Makhadmeh SN (2022b) Backpropagation Neural Network optimization and software defect estimation modelling using a hybrid Salp Swarm optimizer-based Simulated Annealing Algorithm. Knowl-Based Syst 244:108511

Kernighan BW, Plauger PJ (1976) Software tools. ACM SIGSOFT Software Engineering Notes 1:15–20

Khan R and Amjad M (2015) Automatic test case generation for unit software testing using genetic algorithm and mutation analysis, in *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)* pp 1–5, IEEE.

Khari M and Kumar P (2016) A novel approach for software test data generation using cuckoo algorithm, in *Proceedings of the second*

international conference on information and communication technology for competitive strategies p 98, ACM.
Khari M, Kumar P (2019) An extensive evaluation of search-based software testing: a review. Soft Comput 23:1933–1946
Khatibsyarbini M, Isa MA, Jawawi DN, Hamed HNA, Suffian MDM (2019) Test Case Prioritization Using Firefly Algorithm for Software Testing. IEEE Access 7:132360–132373
Khuat TT, Le MH (2019) Binary teaching–learning-based optimization algorithm with a new update mechanism for sample subset optimization in software defect prediction. Soft Comput 23:9919–9935
Kitchenham B (2004) Procedures for performing systematic reviews. Keele, UK, Keele University 33:1–26
Koleejan C, Xue B and Zhang M (2015) Code coverage optimisation in genetic algorithms and particle swarm optimisation for automatic software test data generation, in *2015 IEEE Congress on Evolutionary Computation (CEC)* pp 1204–1211, IEEE.
Koteswara Rao K, Raju G (2019) Reducing interactive fault proneness in software application using genetic algorithm based optimal directed random testing. Int J Comput Appl 41:296–305
Krishnanand K, Ghose D (2009) Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intell 3:87–124
Kumar V, Sahni R (2020) Dynamic testing resource allocation modeling for multi-release software using optimal control theory and genetic algorithm. International Journal of Quality & Reliability Management 37:1049–1069
Li X, Wong WE, Gao R, Hu L, Hosono S (2018) Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. Empir Softw Eng 23:1–51
Liu F, Huang H and Hao Z (2017) Evolutionary algorithm with convergence speed controller for automated software test data generation problem, in *2017 IEEE Congress on Evolutionary Computation (CEC)* pp 869–875, IEEE.
Mahajan S, Joshi SD and Khanaa V (2015) Component-based software system test case prioritization with genetic algorithm decoding technique using java platform, in *2015 International Conference on Computing Communication Control and Automation* pp 847–851, IEEE.
Mahmoud T, Ahmed BS (2015) An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use. Expert Syst Appl 42:8753–8765
Malaiya YK, Li MN, Bieman JM, Karcich R (2002) Software reliability growth with test coverage. IEEE Trans Reliab 51:420–426
Malhotra R (2016) *Empirical research in software engineering: concepts, analysis, and applications*, Chapman and Hall/CRC.
Mäntylä MV, Adams B, Khomh F, Engström E, Petersen K (2015) On rapid releases and software testing: a case study and a semi-systematic literature review. Empir Softw Eng 20:1384–1425
Mao C, Xiao L, Yu X, Chen J (2015) Adapting ant colony optimization to generate test data for software structural testing. Swarm Evol Comput 20:23–36
Margineantu DD and Dietterich TG (1997) Pruning adaptive boosting, in *ICML* pp 211–218, Citeseer.
Matnei Filho RA and Vergilio SR (2015) A mutation and multi-objective test data generation approach for feature testing of software product lines, in *2015 29th Brazilian Symposium on Software Engineering* pp 21–30, IEEE.
Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
Murty KG (1983) *Linear programming*, Chichester.
Myers GJ (2006) The art of software testing. John Wiley & Sons
Nadel BA (1989) Constraint satisfaction algorithms 1. Comput Intell 5:188–224
Nanthaamornphong A, Carver JC (2017) Test-Driven Development in scientific software: a survey. Software Qual J 25:343–372
Nawaz A and Malik KM (2008) Software testing process in agile development. *Department of Computer Science School of Engineering Blekinge Institute of Technology Box 520 SE-372 25 Ronneby Sweden*.
Nejad FM, Akbari R and Dejam MM (2016) Using memetic algorithms for test case prioritization in model based software testing, in *2016 1st Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)* pp 142–147, IEEE.
Nemati S, Haghi Kashani M and Faghih Mirzaee R (2023) Comprehensive Survey of Ternary Full Adders: Statistics, Corrections, and Assessments. *IET Circuits, Devices & Systems* **n/a**.
Nikravan M, Haghi Kashani M (2022) A review on trust management in fog/edge computing: Techniques, trends, and challenges. J Netw Comput Appl 204:103402
Nikravan M, Jameii SM, Kashani MH (2011) An intelligent energy efficient QoS-routing scheme for WSN. International Journal of Advanced Engineering Sciences and Technologies 8:121–124
Niu B and Wang H (2012) Bacterial colony optimization. *Discrete Dynamics in Nature and Society* **2012**.
Ow PS, Morton TE (1988) Filtered beam search in scheduling. The International Journal of Production Research 26:35–62
Pan J (1999) Software reliability. Dependable Embedded Systems, Carnegie Mellon University 18:1–14
Potter B, McGraw G (2004) Software security testing. IEEE Secur Priv 2:81–85
Prabhakar N, Singhal A, Bansal A and Bhatia V (2019) A literature survey of applications of meta-heuristic techniques in software testing, in *Software Engineering* pp 497–505, Springer.
Rahimi M, Songhorabadi M, Kashani MH (2020) Fog-based smart homes: A systematic review. J Netw Comput Appl 153:102531
Raj HLP and Chandrasekaran K (2018) NEAT Algorithm for Testsuite generation in Automated Software Testing, in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)* pp 2361–2368.
Rao KM and Anuradha K (2016) A hybrid method for parameter estimation of software reliability growth model using Modified Genetic Swarm Optimization with the aid of logistic exponential testing effort function, in *2016 International Conference on Research Advances in Integrated Navigation Systems (RAINS)* pp 1–8, IEEE.
Rodrigues A and Dias-Neto A (2016) Relevance and impact of critical factors of success in software test automation lifecycle: A survey, in *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing* p 6, ACM.
Sahin O, Akay B (2016) Comparisons of metaheuristic algorithms and fitness functions on software test data generation. Appl Soft Comput 49:1202–1214
Sarvizadeh R, Kashani MH, Zakeri FS, Jameii SM (2012) A novel bee colony approach to distributed systems scheduling. Int J Comput Appl 42:1–6
Sayyari F and Emadi S (2015) Automated generation of software testing path based on ant colony, in *2015 International Congress on Technology, Communication and Knowledge (ICTCK)* pp 435–440, IEEE.
Serdyukov KS and Avdeenko TV (2018) Automatic Data Generation for Software Testing Based on the Genetic Algorithm, in *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)* pp 535–540, IEEE.

Sharma R, Saha A (2020) Identification of critical test paths using firefly algorithm for object oriented software. Journal of Interdisciplinary Mathematics 23:191–203

Sharma S, Rizvi SAM and Sharma V (2019) A Framework for Optimization of Software Test Cases Generation using Cuckoo Search Algorithm, in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)* pp 282–286.

Sheikh Sofla M, Haghi Kashani M, Mahdipour E, Faghih Mirzaee R (2022) Towards effective offloading mechanisms in fog computing. Multimedia Tools and Applications 81:1997–2042

Shomali N and Arasteh B (2020) Mutation reduction in software mutation testing using firefly optimization algorithm. *Data Technologies and Applications*.

Singh SK and Singh A (2012) *Software testing*, Vandana Publications.

Singhal A, Bansal A, Kumar A (2016) An approach to design test oracle for aspect oriented software systems using soft computing approach. International Journal of System Assurance Engineering and Management 7:1–5

Singhal A, Bansal A, Kumar A (2021) Meta-heuristic algorithm to generate optimised test cases for aspect-oriented software systems. International Journal of Advanced Intelligence Paradigms 18:134–153

Sivaji U and Srinivasa Rao P (2021) Test case minimization for regression testing by analyzing software performance using the novel method. *Materials Today: Proceedings*.

Songhorabadi M, Rahimi M, MoghadamFarid A, Haghi Kashani M (2023) Fog computing approaches in IoT-enabled smart cities. J Netw Comput Appl 211:103557

Stocks PA (1993) Applying formal methods to software testing, University of Queensland.

Sugave SR, Patil SH, Reddy BE (2017) DDF: Diversity dragonfly algorithm for cost-aware test suite minimization approach for software testing, in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)* pp 701–707, IEEE.

Suri B and Singhal S (2012) Literature survey of ant colony optimization in software testing. In: 2012 CSI Sixth International Conference on Software Engineering (CONSEG). IEEE, pp 1–7

Tebes G, Peppino D, Becker P, Matturro G, Solari M, Olsina L (2020) Analyzing and documenting the systematic review results of software testing ontologies. Inf Softw Technol 123:106298

Thirumoorthy K, Britto JJJ (2022) A clustering approach for software defect prediction using hybrid social mimic optimization algorithm. Computing 104:2605–2633

Thirumoorthy K, J JJB, (2022) A feature selection model for software defect prediction using binary Rao optimization algorithm. Appl Soft Comput 131:109737

Thompson HH (2003) Why security testing is hard. IEEE Secur Priv 1:83–86

Turing AM (1937) On Computable numbers, with an application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society s2–42:230–265

van der Veer H, Wiles A (2008) Achieving technical interoperability. European Telecommunications Standards Institute

West DB (1996) Introduction to graph theory. Prentice Hall, Upper Saddle River

Wiklund K, Eldh S, Sundmark D, Lundqvist K (2017) Impediments for software test automation: a systematic literature review. Software Test Verificat Reliab 27:e1639

Wu B, Yun L, Jin X, Liu B, Wei G (2016) Study on the fuzzing test method for industrial supervisory control configuration software based on genetic algorithm. In: 2016 11th international conference on reliability, maintainability and safety (ICRMS) pp 1–6, IEEE

Yang S, Man T, Xu J, Zeng F, Li K (2016) RGA: A lightweight and effective regeneration genetic algorithm for coverage-oriented software test data generation. Inf Softw Technol 76:19–30

Yang X-S (2009) Firefly algorithms for multimodal optimization. In: Watanabe O, Zeugmann T (eds) Stochastic algorithms: foundations and applications. Springer, Berlin, Heidelberg, pp 169–178

Yang X-S (2010a) Nature-inspired metaheuristic algorithms. Luniver Press

Yang X-S (2010b) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010b). Springer, pp 65–74

Yang X-S, Deb S (2009) Cuckoo search via Lévy flights. In: 2009 World congress on nature & biologically inspired computing (NaBIC). IEEE, pp 210–214

Yao X, Gong D, Li B, Dang X, Zhang G (2020) Testing method for software with randomness using genetic algorithm. IEEE Access 8:61999–62010

Yusop OM, Ibrahim S (2011) Evaluating software maintenance testing approaches to support test case evolution. Int J New Comput Architect Appl (IJNCAA) 1:74–83

Zelenka J, Budinská I, Dideková Z (2012) A combination of heuristic and non-heuristic approaches for modified vehicle routing problem. In: 2012 4th IEEE international symposium on logistics and industrial informatics. IEEE, pp 107–112

Zhan L (2022) Optimal model of software testing path selection based on genetic algorithm and its evolutionary solution. Wirel Commun Mob Comput 2022:7601096

Zhu E, Yao C, Ma Z, Liu F (2017) Study of an improved genetic algorithm for multiple paths automatic software test case generation. In: International conference on swarm intelligence. Springer, pp 402–408

Zhu K, Ying S, Zhang N, Zhu D (2021) Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. J Syst Softw 180:111026

Ziming Z, Xiong X, Li J (2017) Improved evolutionary generation of test data for multiple paths in search-based software testing. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp 612–620

Zuse H (2019) Software complexity: measures and methods. Walter de Gruyter GmbH & Co KG