Journal of Theoretical and Applied Information Technology
30th June 2022. Vol.100. No 12
© 2022 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A CRITICAL ANALYSIS OF SWARM INTELLIGENCE FOR REGRESSION TEST CASE PRIORITIZATION

BAKR BA-QUTTAYYAN[1], HASLINA MOHD[2], YUHANIS YUSOF[3]

School of Computing, College of Arts and Sciences, Universiti Utara Malaysia (UUM), 06010, Sintok, Kedah, Malaysia

E-mail: 1baksam1@gmail.com, 2haslina@uum.edu.my, 3yuhanis@uum.edu.my

ABSTRACT

In software development, applications always undergo frequent modifications to cope with the changing needs of the users. Hence, this will affect the reliability of the applications. Regression testing is used to tackle this problem in which test case prioritization (TCP) is considered as one of the effective approaches. Optimization methods are employed to enhance the proficiency of prioritizing test cases in terms of efficiency and effectiveness. Swarm intelligence (SI) algorithms are mostly applied to TCP as one of the effective optimization methods. This paper aims at providing state-of-the-art for TCP using SI algorithms. The methodological process of performing this review includes conducting a search in the Scopus database using predetermined keywords to obtain relevant primary studies where suitable papers have been selected according to a set of predefined criteria. Predefined research questions also provide a basis for assessing nominated studies. Hence, the most relevant papers (57 out of 420) have been identified, analyzed, and classified. Based on the findings, it is learned that the deployment of swarm intelligence algorithms in prioritizing test cases yields promising results. Moreover, this study also includes suggestions that can be deployed to improve existing studies.

Keywords: *Swarm Intelligence, Software Testing, Regression Testing, Test Case Prioritization, Optimization Algorithms*

1 INTRODUCTION

Digital technology plays an essential role in contemporary daily life, where computerized systems represent the backbone of everyday transactions in aspects such as education, health, government, and finance. Thus, accreditation is required to ensure the production of worthy and reliable software systems as failures in such systems may lead to financial risk, disaster, or death. Krasner [1] reported that the United States has lost nearly $2.08 trillion due to the poor quality of software in 2020. Likewise, Heathrow airport disruption in London, British Airways glitch, social media (Facebook, Instagram, and WhatsApp) outage, TSP bank outage, and others are clear examples of the negative impact of software errors in public life [2].

In the software industry, producing high-quality systems undergo a sequential, tedious, and costly process that affects the digital industry's reputation. As an arduous and critical part of the software development life cycle (SDLC), software testing is responsible for building confidence towards the developed software systems, costing around 50% of the overall production cost [3]. Therefore, the software under development is frequently modified to cope with the frequently changing requirements of customers and the market, leading to the occurrence of new programming errors requiring quality testers to deal with them seriously. Deeper in the testing artifacts, a test case is the smallest unit that is carefully designed with the intent of detecting errors, and sometimes test cases need to be optimized for effective implementation. In software testing, the branch that deals with modification errors and their adverse effects is called regression testing.

Regression testing is defined by the IEEE standard glossary of software engineering terminology as: *"Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements."* [4]. Furthermore, Rothermel and Harrold [5] define the regression testing problem as: *Given a program **P**, its modified version **P'**, and test set **T** used previously to test **P**, find a way, making use of **T**, to gain sufficient confidence in the correctness of **P'**.*

Despite its high cost and time-consuming, regression testing remains an unavoidable and vital process in software development. Regression testing can be performed by executing all test cases, selecting test cases, prioritizing test cases, or combining between selection and prioritization of test cases. Retest all is the first method in which all available test cases are re-executed to attain uncovering faults. Such a conventional method is costly, especially when the amount of test cases is huge due to increasing the size and complexity of the software system. The second method is called test selection which can select a significant subset of the test cases to check the amendments on the current program. However, discarding any related test case may leave some defects undiscovered. Test case prioritization (TCP) is a regression testing method that effectively reschedules test cases to attain a performance goal. Accordingly, this method avoids the drawbacks of omitting test cases as occurred in test selection and hybrid approaches. Figure 1 highlighted the regression testing approaches below:

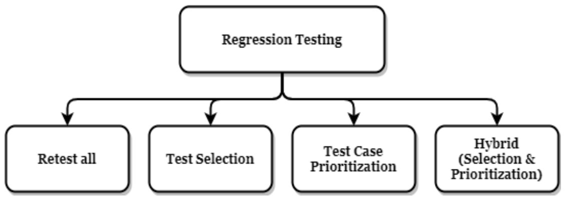

Figure 1: Regression Testing Approaches

Test case prioritization is one of the most effective approaches of regression testing as it reorders the execution of test cases in an effective way for detecting errors as early as possible during the testing process. Unlike test selection and hybrid approaches, TCP does not omit any test case, in order not to allow faults to slip away, and it provides an effective ranking of test cases in which the rate of fault detection can be accelerated [6]–[8]. Due to the complexity and complicated nature of regression testing, researchers have denoted the TCP problem as an NP-hard problem, implying the need for optimization algorithms [9], [10]. Recent research efforts indicated that artificial intelligence (AI) methods had gained much popularity and promising results for optimizing test cases [11], [12]. Among effective AI methods, evolutionary algorithms and swarm intelligence are mostly applied in TCP.

This paper investigates the application of swarm intelligence (SI) in the software testing domain, specifically the regression TCP. A total of 57 papers published between 2008 and 2021 have been identified. Hence, the study aims to extensively analyze and discuss the work that has been done in the software testing domain and provide future directions to help prospective studies in this field. Next section specifies the related work, followed by an overview of SI, the study method has been described in section 4, while section 5 discusses the results, then the study is concluded in section 6.

2 RELATED WORK

The increasing need for solving real-world problems led to the utilization of computation intelligence for solving such problems through optimization. Meanwhile, SI algorithms have become quite popular in which these algorithms are applied in different domains of software engineering and testing. In the context of regression testing, SI algorithms have been widely implemented for optimizing test cases, especially in TCP. Literature survey revealed several studies that involve the application of nature-inspired algorithms in the TCP domain.

Catal [13] studied the effectiveness of using genetic algorithms (GAs) for prioritizing test cases by selecting seven primary studies. He reported that among these seven studies, five had a positive impact of GA on TCP than other algorithms, such as hill-climbing, greedy algorithm, random, and other TCP techniques. This research concludes with specific recommendations for better implementation of TCP techniques, such as: assessing TCP methods using the most common measurements in the field and reporting the TCP factors for the algorithm.

Sharma and Singh [14] had briefly explored the implementation of ACO in the context of model-based TCP. Seven studies were collected and reviewed to conclude that ACO is a robust algorithm and can provide positive feedback in the model-based TCP.

Bajaj and Sangwan [15] investigated nature-inspired methods, including swarm algorithms, that have been implemented for optimizing test cases of regression testing. In this study, 15 works have been explored and summarized. In conclusion, they stated that GA is the most implemented and well-known among all nature-inspired algorithms for researchers in different approaches of the regression testing domain. The work concludes with the importance of nature-inspired algorithms for efficient optimization and cost-effectiveness, as well as the need for hybridization among these algorithms for more effectiveness.

Alkawaz and Silvarajoo [16] surveyed the optimization techniques applied in TCP. As a result, eight studies were collected and examined based on the applied algorithm, purpose, coverage, implementation, and metrics used for evaluation. However, their work does not cover SI algorithms in TCP.

Bajaj and Sangwan [17] systematically investigated and categorized the application of GAs in TCP, in which 20 relevant works have been selected from four different databases. The selected studies were investigated based on several criteria such as prioritization method, benchmarks, size of the test suite, GAs type, and metrics of scaling performance, as well as proving a taxonomy for GAs applied in TCP.

Paygude and Joshi [18] assessed the implementation of evolutionary algorithms in the context of TCP with a total of 33 papers reported and tabularly summarized. The study concludes that GA is the most popular and utilized algorithm, and the APFD metric is used by most of the studied papers. Besides evolutionary algorithms, however, this study also includes papers that addressed non-evolutionary algorithms such as ant colony, clustering, greedy, particle swarm, and other algorithms.

In summary, the work presented in this paper builds on previous research to explore how swarm intelligence algorithms affect the optimization of test cases in the regression testing prioritization process of the test cases. While earlier work focused either on the single algorithm (i.e., GA, ACO) or generally on a wide domain like evolutionary or nature-inspired algorithms, we narrowly focused on swarm algorithms implemented to improve TCP performance. In addition, we can study such algorithms at a much deeper scale than previously possible by considering only the SI discipline, which was applied in the TCP domain. In other words, this study differs from previous works in scope (i.e., a very narrow topic), presentation (i.e., classification based on TCP factors), and more suggestions for common issues were added.

3 OVERVIEW OF SWARM INTELLIGENCE

This section provides SI background, including classification, benefits, general structure, and a brief description of SI algorithms implemented in the regression TCP.

3.1 What is Swarm Intelligence?

Swarm Intelligence (SI) is an integral part of artificial intelligence (AI), which is recently gaining increasing usage among researchers for optimization due to the powerful performance and flexibility it provides [19], [20]. The term swarm intelligence was first coined in 1988 by Gerardo Beni, in the context of cellular robotic systems as simple agents interacting with each other by utilizing the self-organization principle [21]. Basically, SI computation mimics the intellectual cooperative behavior of the societies of the natural creatures such as ant colonies, bee colonies, cat swarms, etc. [20]. In the real world, there is a kind of problem called NP-hard problems where, in this situation, finding a global optimum (unique solution) becomes almost impossible because solutions always turn out to be infinite. In such cases, it becomes necessary to find a workable solution (near optimum) within the time constraints. Thus, SI algorithms found their usefulness in solving such real-world problems [19], [22].

Different types of SI algorithms exist, such as but not limited to ant colony optimization (ACO), particle swarm optimization (PSO), artificial bee colony (ABC), glowworm swarm optimization (GSO), and cat Swarm Optimization (CSO). Chakraborty and Kar [19] classify different types of SI algorithms as highlighted in Figure 2 below.

SI algorithms have several benefits that made researchers prefer applying them in solving optimization problems. Also, SI algorithms have shortcomings reported by researchers such as Slowik [23] and Yang et al. [24]. Both the benefits and drawbacks are emphasized briefly in Table 1.

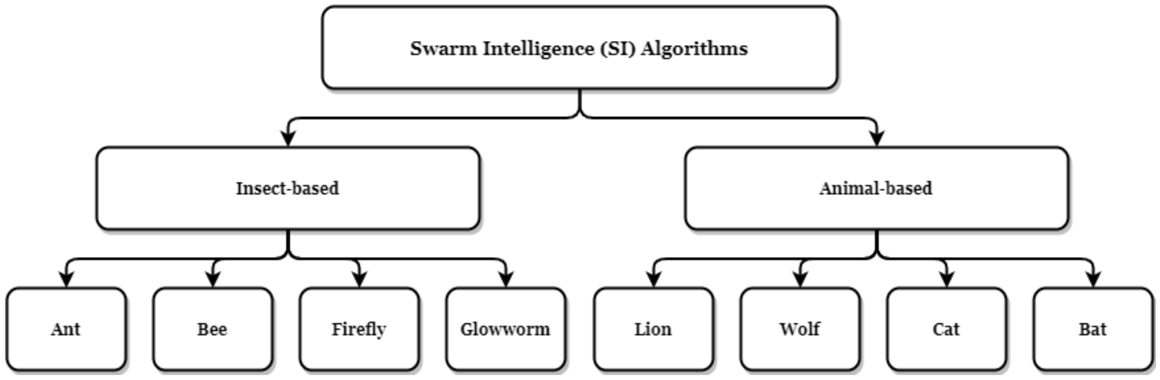

Figure 2: Swarm Intelligence Algorithms Classification (adapted from [19]*)*

Table 1: Swarm Intelligence Algorithms Benefits and Drawbacks

Benefits	Drawbacks
• Not start from a single point, but among potential solutions. • Provide optimal global search. • Flexible and simple to apply. • Suitable for hard and complex problems. • A diverse range of problems can be handled. • The simplicity of parallel implementation.	• Expensive computation. • Unique solutions cannot be guaranteed. • Not suitable for simple problems.

SI algorithms have been used to solve real-world problems, and their applications have pervaded many aspects, including routing, scheduling, tracking, sorting, and classification. The most famous problems in which SI has been successfully solved consist of the traveling-salesman problem (TSP), aircraft landing, vehicle routing problem (VRP), flow shop scheduling problem, knapsack problem, timetable problems, etc. [23]–[25].

3.2 General Structure of Swarm Intelligence

To understand how the SI algorithms work, the system consists of several phases that work in an iterative process. Tan [26] and Yang et al. [24] described the staged process briefly as beneath:

- *Swarm Population* – a collection of agents that interact locally with each other within an environment, i.e., the search space, have been initialized.

- *Fitness Function* – sometimes is called evaluation or objective function in which each individual agent in the search space has been assessed against an identified fitness formula formulated carefully to represent the chosen near-optimal solution for the desired problem.

- *Communication* – according to swarming behavior such as ants and bees, fitness values are shared among agents through the entire atmosphere.

- *Swarm Updating* – for every iteration, each agent updates once necessary its individual and community best information as well as its movement (i.e., velocity and position vector).

- *End* – the loop remains until termination conditions have been met as the optimal solution has been reached. Due to their stochastic nature, repeated solutions are challenging to obtain through SI.

Figure 3 clearly illustrates the aforementioned stages below.

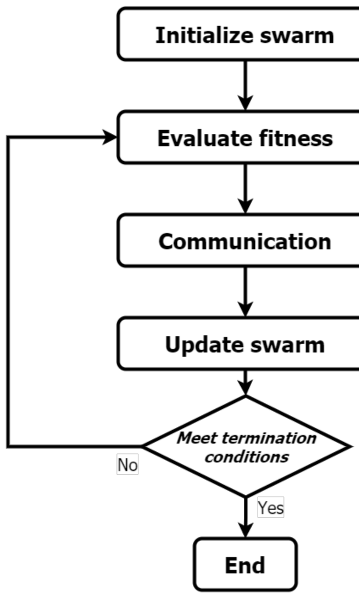

Figure 3: General Structure of SI Algorithms (as in [26]*)*

3.3 Swarm Intelligence Applications in TCP

Several SI algorithms have been implemented to optimize the test cases in regression TCP since 2008. Due to their nature, researchers reported that SI algorithms could increase the overall regression testing methods proficiency, cost-effectiveness, accuracy, and convergence incorporated with providing the ability to cover multiple criteria and decreasing complexity [14], [15].

In the context of testing, the optimization process through SI algorithms has several considerations to be taken into account. First, test cases usually are initialized as a population of the swarm algorithm, where the size of the population is the total number of the test cases to be optimized. Furthermore, each agent during communication can represent any single test case among the population. Thus, each test case will be evaluated against a fitness function. On the other hand, the fitness function should be carefully designed for representing the optimization basis, i.e., the prioritization factors test cases are optimized accordingly. Thus, there are three things to be known about the fitness functions: (1) They are responsible for the communication among agents, where the agent movement is controlled by its fitness value [27], [28]. (2) Fitness functions should be specified mathematically to guide search algorithms in evaluating the quality of solutions when using search algorithms [29]. (3) In the test optimization process, test cases are being prioritized in the test suites based on the obtained value of fitness, i.e., fitness scores, which allowed the optimization algorithm to rank test cases in a manner that the most likely important test cases have the priority to be executed first before others [30], [31]. It is well known that the fitness function can be any evaluation metric like APFD or any other mathematical expression that every test case will be assessed accordingly and should be defined based on the goal to be achieved in a way that makes the experimental results positive [32]–[34]. For multi-objective optimization, the fitness function should be designed as multi-objective to reflect these objectives and constraints precisely. For designing a suitable fitness function for optimizing test cases, Arrieta et al. [32] provided a piece of advice in terms of Lessons Learned at the end of their article.

Aside from this, a brief description is provided below for those algorithms that have been applied for prioritizing test cases in the regression testing domain:

1. *Ant Colony Optimization (ACO):*

   Marco Dorigo first introduced this algorithm in 1992, inspired based on the observations of ants' foraging behavior to solve computational problems [35], [36]. In reality, ants' strength lies in the highly organized colony structure of the whole ants, i.e., the teamwork, where the shortest path always has been chosen between their nest and food source using pheromone trails. The ACO algorithm emphasizes a probabilistic model that simulates the ants' behavior to choose the optimal solution in the shortest way [36]. Based on the optimization, researchers found the ACO algorithm promising in producing significant solutions when implemented in numerous fields, including the software testing domain.

2. *Particle Swarm Optimization (PSO):*

   PSO algorithm was invented by Eberhart and Kennedy in 1995, employing the idea of simulating the social behavior of swarms of the birds flocking and fish schooling, which emphasizes some organized structure [23], [37]. Meanwhile, the algorithm utilizes what was called particles as agents that move towards the optimal solution in the search space, i.e., swarm, under the control of velocity (acceleration) and position (direction) of each particle [38]. Slowik [38] reported that the appearance of ACO, then PSO later, led to the formation of a new research discipline called Swarm Intelligence (SI).

3. *Artificial Bee Colony (ABC):*

   In 2005, an ABC algorithm was introduced by Karaboga, which imitates the foraging behavior of honeybees out of the colony [39], [40]. For this process, bees are divided into three categories, each of which has a distinct task: the employed, the onlooker, and the scouts. Employed bees exploit and take nectar from flowers found to the hive. Onlooker bees are entrusted with selecting good food sources based on employed bee waggle dance (positive feedback). Scout bees search for new food sources once the existing food source is abandoned (negative feedback). Computationally, the ABC algorithm has three iterative searches: (1) scout – detects exhausted solutions in the search space, i.e., exploring new solutions (2) employed – exploits the existing information of discovered solutions, and (3) onlooker – evaluates then selects high-quality solutions randomly and performs local search around them [19], [23]. ABC algorithm is being touted by researchers as a powerful and efficient method for providing solid solutions to hard problems such as TCP.

4. *Glowworm Swarm Optimization (GSO):*

   GSO algorithm was introduced based on ACO in 2005 by Krishnanand and Ghose for mobile robotics in a way that mimics the flashing behavior of the glowworms or lighting bugs [23], [41]. In reality, glowworms naturally release a chemical substance, i.e., luciferin, as a signal for attracting mates or prey to feed. The algorithm begins by initializing glowworms randomly across the workspace; then, each loop consists of three phases: luciferin update – for fitness assessment, movement update, and neighborhood range update [19][41][38].

5. *Cat Swarm Optimization (CSO):*

   CSO was primarily invented in 2006 by Chu et al. based on observing the hunting behavior of cats. Since then, it has been used for solving difficult engineering problems. CSO algorithm placed a swarm of cats randomly in the search space. Each cat represents a solution and can be identified using velocity, position, and flag mode information, i.e., seeking or tracing mode. After certain iterations, the fittest cat among all will be chosen by the fitness function as the best solution for the intended problem [23], [38].

6. *Cuckoo Search Algorithm (CSA):*

   In 2009, Yang and Deb developed an algorithm that simulates the aging behavior of cuckoos in which the principles of the search of lévy flights and cuckoos brooding parasitism were adapted. In the case of CSA, three main steps employed are initialization, iteration loop, and solutions modifications. In the initialization step, each nest egg symbolizes a solution at the start of the algorithm, while the cuckoo egg represents a new solution. Not all cuckoo eggs will be survived until the end; around 20% of them will be abandoned due to the detection of parasitism mechanism in which they are replaced by new eggs. This is followed by the generation of new solutions, i.e., cuckoos, as well as performing lévy flights, and finally, the best solutions, i.e., cuckoos, were selected according to the fitness function [23], [38], [42].

7. *Firefly Algorithm (FFA):*

   In late 2007 and early 2008, Yang developed a firefly algorithm based on tropical fireflies flashing behavior [23][43]. FFA algorithm first initializes a swarm population of fireflies randomly, then updates the fitness function against each firefly in the search space in an iterative manner to attain the best sequence of individuals among the solution pool. FFA has been employed with different applications in a variety of domains, including software testing [19], [23], [43].

8. *Bat Algorithm (BA):*

   Bat algorithm was first designed by Xin-She Yang in 2010, which imitates the primary attributes of bat echolocation in terms of navigating the neighborhood surroundings. In this algorithm, bat swarms are firstly initialized as the population for search; then, these bats randomly fly by assigning their positions and velocity as their movement is controlled by frequency, loudness, and pulse emission rate (PER), which are updated at each iteration. Secondly, execute a local random walk (LRW) in which frequency increases, while loudness decreases once the bat is potentially getting closer to its prey or food source [19], [23], [44]. Unlike exploitation, the algorithm exploration is unsatisfactory due to local trapping as stated by Kanwar et al. [44].

9. *Lion Search Algorithm (LSA):*

   Rajakumar developed an algorithm in 2012 that is inspired by the lion's social life within generations as frequent fighting happens between bride leader (territorial lion) against nomadic lions [45]. Apart from nature, LSA undergoes several processes; the first is randomly releasing a set of lions as pride generation. However, some lions were selected to be nomadic lions in this step. Secondly, the mating process is performed in which new lions are generated and evaluated. The surviving lion in the territorial defense is the best solution in the iteration, while the defeated lions and matured cubs were considered as nomadic and weak solutions to be eliminated from the solution pool. This process continues until a satisfactory goal is reached [19], [45].

10. *Flower Pollination Algorithm (FPA):*

    In 2012, Yang et al. developed an algorithm in which the pollination attributes of the flowering plants were inspired. Accordingly, FPA has two steps of pollination, i.e., global and local. Globally, pollinators such as insects or animals carried flower pollens to long distances in which the fittest pollens survived as lévy flight simulated. Locally, pollens spread among different flowers of the same species in a limited neighborhood. FPA has been applied in different domains for solving complex problems, including software testing [46], [47].

11. *Grey Wolf Optimization (GWO):*

    In 2014, the grey wolf optimization algorithm was developed by Mirjalili et al. based on grey wolves' social behavior of hunting where wolves live in packs led by an alpha. Mathematically, the algorithm initializes a population of wolves ranked as alpha, beta, and delta. The best solution is represented by alpha, whereas the second and third best solutions were represented by beta and delta, respectively. In each cycle, the position of each wolve is updated according to the alpha, beta, and delta positions, i.e., the exploration process. Then, exploitation ends when the prey attack at the end of the iteration. The algorithm ends with attaining the best solution [19], [23], [38].

12. *Shuffled Frog-Leaping Algorithm (SFLA):*

    In 2006, Eusuff, Lansey, and Pasha developed an algorithm that simulates the social behavior of frogs during food search [21], [48]. The algorithm starts by placing virtual frogs randomly among the whole swamp to invent independently groups (memeplexes) for search within different directions. During memetic evolution, frogs with stronger memes (ideas) within memeplex are more likely to create new ideas than others, leading to a change in their positions, i.e., exploitation. After certain iterations, the shuffling process begins by forcing memeplexes to mix and creating new memeplexes as frogs start migration with sharing information, i.e., global exploration. The process ends with selecting the fittest frog (solution) among all [49].

13. *Whale Optimization Algorithm (WOA):*

    Mirjalili and Lewis established an algorithm in 2016 in which the hunting social behavior of humpback whales is modeled mathematically. Generally, WOA utilizes three major steps: (1) encircling prey – where all agents moved towards the best solution found in the search space, i.e., the leader; (2) bubble-net attacking method – in which the path of whales' movement imitated to get closer to the prey, i.e., the exploitation process; and (3) search for prey – for the exploration of the workspace wherever agents are chosen randomly to change the location of the ***i-th*** search agent. This algorithm has been applied to solve several engineering problems in the search community [23], [38].

3.4 Hybrid Swarm Intelligence

It is commonly known that not all algorithms have the same level of strength due to the diversity of search mechanisms used. So, researchers found that combining two or more algorithms can be more effective for improving performance in terms of accuracy and convergence in attaining optimal solutions. This process is known as hybridization [15], [50]. Accordingly, the powerful nature of hybrid algorithms in solving optimization problems has made them algorithms of choice in the research community [43], [47]. Concerted efforts are being made by researchers towards integrating several SI algorithms in the TCP for enhanced performance [15]. Example of hybridization among SI algorithms includes integrating SI algorithms with a genetic algorithm, and integration with fuzzy logic are obvious.

4 METHODOLOGY

TCP is classified as an NP-hard problem due to the nature and complexity of the testing environment in which the optimization of test cases can be affected by several factors. Consequently, SI is being proposed as a panacea to the TCP problems as it has not been investigated before. The aim of this research review is to investigate the work that has been done by researchers in relation to the application of SI in the field of regression test case prioritization. In short, the method consists of formulating research questions and defining the search strategy that includes database search identification, key terms determination, search syntax formulation, identifying selection criteria for the retrieved papers, and finally, the selection of the actual papers for the investigation. Moreover, this review enlightened by the guidance provided by Kitchenham et al. [51].

4.1 The Research Questions

The following research questions have been formulated to be answered throughout this review:

- To what extent SI algorithms have been implemented for optimizing test cases in TCP?
- On what basis have the test cases been optimized in the existing studies?
- How has the evaluation of these studies been carried out?
  - What metrics have been used to scale the performance of these proposed methods?
  - To which existing methods are these studies compared?

4.2 The Search Process

The search strategy conducted in this review is started by choosing the Scopus database as a starting point for obtaining the relevant studies up to the year 2021, then formulating the search string to be fulfilled in the next step for obtaining the desired related studies. For seeking the TCP studies where SI algorithms are applied, we used the following search string to compile the algorithms mentioned above:

```
[ ("test case prioritization" OR "test case prioritisation" OR "prioritizing test cases" OR "prioritising test cases") AND ("swarm" OR "intelligence" OR "algorithm" OR "colony" OR "artificial" OR "search" OR "optimization") ].
```

The above search string was designed according to the defined keywords regarding TCP and SI, considering the spelling differences of the word “prioritization” between British and American English as known in the academic environment.

4.3 The Study Selection

By conducting the search in the Scopus database, a total number of 420 papers were initially screened, and unrelated studies were discarded. Those studies which have been filtered based on specific features such as title, abstract, and conclusion are 67 in number. Finally, a full-text read round followed, resulting in the selection of 57 papers. These stages are illustrated in Figure 4 below:

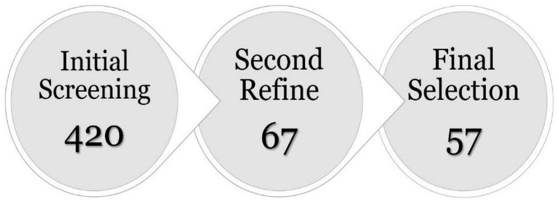

Figure 4: Paper Selection Stages

Towards the final selection process of the related studies among retrieved results, the suitable studies have been chosen, particularly according to predetermined selection criteria for inclusion and exclusion of related work as briefly emphasized in Table 2.

Table 2: Inclusion and Exclusion Criteria

Inclusion criteria	Exclusion criteria
• English published work. • Journals and proceedings papers. • Application of SI in TCP.	• Other languages published work. • Books, book chapters, posters, etc. • Other topics.

5 RESULTS AND DISCUSSION

At the end of the selection process, 57 papers were identified in the specified period, i.e., between 2008 and 2021, for the application of SI algorithms in the regression TCP, and those studies were to be analyzed according to the RQs. Across those years, there is no published papers found in the Scopus database before the year 2008 as only one paper corresponds to the year 2008, the later four years, unfortunately, have no publications found, to the year 2013, only one paper is published. After that, the publications grow slowly between 2014 and 2015. Three years later, i.e., 2016, 2017, and 2018, the amount of publication constantly stays at 6 papers per year. In the year 2019, the number of publications increased to 14 papers. Then, the year 2020 scored 8 published papers. Lastly, the year 2021 got 9 papers published. Figure 5 below presents the publication of papers according to years, while other aspects will be detailed discussed in the later subsections.

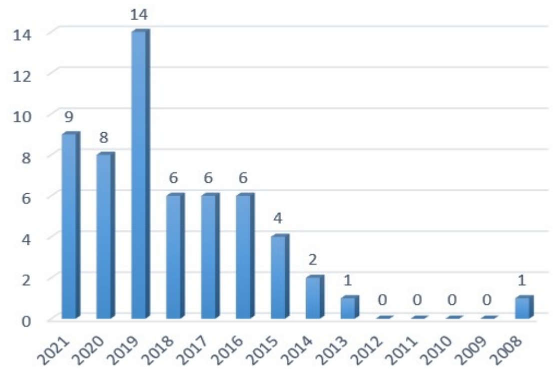

Figure 5: Distribution of The Studies with Respect to Publication Years

In summary, SI algorithms have been implemented in the context of TCP for regression testing since 2008. In terms of publication type, the obtained studies can be classified as either journal papers or conference proceedings, as denoted in Figure 6 underneath.

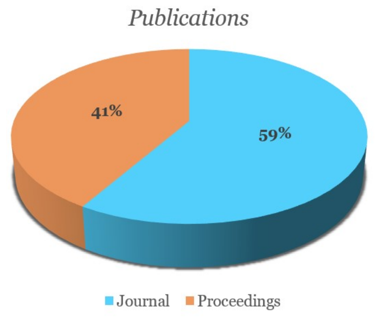

Figure 6: Publications Type

The application of SI algorithms varies according to the characteristics of the algorithms, nature of their implementation, and the prioritization basis, where ant colony is the most used algorithm, as reported in 19 of the papers reviewed. This is followed by PSO, ABC, CSO, cuckoo, bat, and FFA, respectively. On the other hand, hybrid algorithms showed increasing recently, i.e., in 2021, where 14 studies were reported. The publication distribution among SI algorithms has been presented in Figure 7 below.

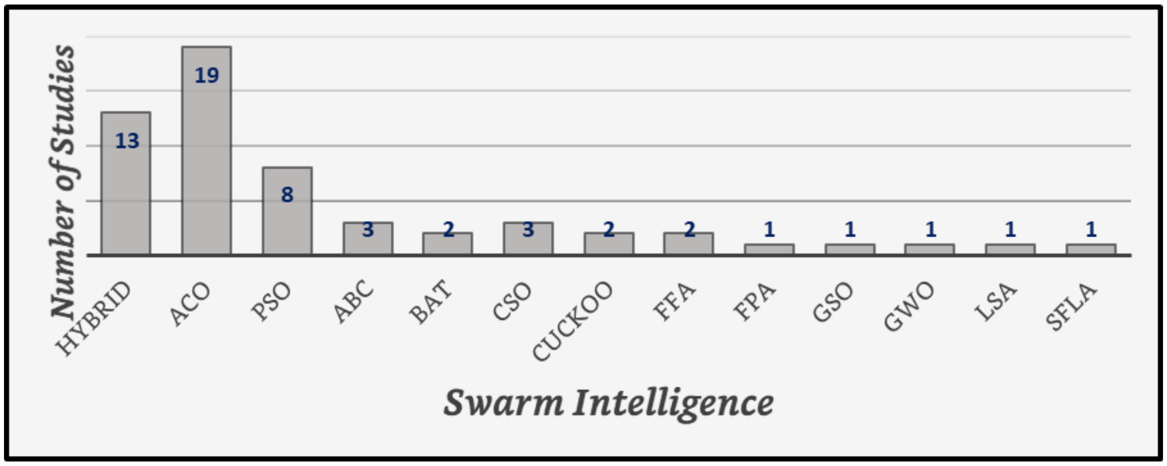

Figure 7: Swarm Intelligence Algorithms used in Regression Test Case Prioritization

5.1 Swarm Intelligence Algorithms

Throughout the conducted search process, the authors found that the application of SI algorithms in the context of regression TCP addressed the following algorithms: ant colony, particle swarm, artificial bee colony, glowworm swarm, cat swarm, cuckoo search, firefly, bat, lion search, flower pollination, grey wolf, shuffled frog-leap, dragonfly, and whale optimization. In this section, single SI algorithms have been discussed in detail.

1. **Ant Colony Optimization (ACO) Applications:**

   A total of nineteen proposed work has been implemented since 2013, based on the ACO implemented in TCP. Several factors were used for performing prioritization of test cases in these studies, such as code coverage, fault coverage, and others emphasized as follow:

   - ***Code coverage:*** Lu et al. [52], Lu and Zhong [53], and Panwar et al. [54] introduced ACO-based algorithms which prioritize test cases according to the additional statement, total statement, and optimal path coverage, respectively. However, only Lu and Zhang [53] framework achieved 96.04% of statement coverage, while Lu et al. [52] did not reach

90%. Panwar et al. [54] study lacks sufficient evaluation where there is no metric or benchmark specified besides a few numbers of test cases. On the other hand, Dhiman and Chopra [55] utilized ACO for TCP based on the importance of functions as the slicing technique was used to determine the importance of the function. Nevertheless, no evaluation was specified regarding this work, and it was compared with manual ordering.

- ***Incorporating execution time with code coverage (statement–coverage):*** Pachariya [56] and Bian et al. [57] implemented an ACO-based method for prioritizing test cases. While Bian et al. [57] work provide promising results in code-coverage (i.e., reached 99% of the statements coverage within 9.86 sec.). Code-coverage does not mean detecting all faults, especially when they are detected by test cases that are not unit test cases [58]. Besides, Pachariya [56] work, aside from not being evaluated against any popular metric, was not also compared to other techniques.

- ***Fault coverage:*** Chen and Zhang [59] proposed an ACO-based algorithm to cover all faults through prioritized test cases. However, this method is not compared with optimization-based methods for better performance evaluation. Another study by Kumar and Srinivas [60] proposed the ACO method for prioritizing test cases of object-oriented programs based on fault exposing potential. However, this algorithm lacks proper evaluation as it was only compared to random ordering and did not use any public dataset like SIR or GitHub. Lastly, Studies such as Solanki et al. [61], Solanki et al. [62], and Solanki et al. [63] performed TCP based on maximizing coverage of faults diversity using the m-ACO algorithm. Despite their promising results, none of Solanki et al. [61]–[63] studies reached 90% of faults covered, which means this method needs to be enhanced.

- ***Incorporating execution time with fault-based:*** Vescan et al. [64] came up with a multi-objective TCP method using ACO based on 3 factors of fault coverage, fault severity, and cost of execution (time execution of the test case). However, the evaluation is insufficient as only 5 test cases were used, the lower performance achieved, and the compared techniques are not AI-based. Likewise, no benchmark was specified. Next, Ansari et al. [65], and Suri and Singhal [66] came up with ACO-based methods for prioritizing test cases according to fault coverage and execution time. However, the first study requires more comparison, especially with optimization techniques under controlled experiments using public datasets like SIR to make it more reliable, while the other lacks adequate evaluation in terms of comparison and measurements. Last, Gao et al. [67] introduced an ant-based multi-objective TCP method for optimizing test cases according to three factors: numbers of faults detected, faults severity, and execution time of test cases. The performance of this method reaches 82.5% according to APFD and covers total faults within 16 units of time. This method has been compared to original, random, and optimal orderings. However, this work lacks adequate evaluation.

- ***Requirements:*** Zhang et al. [68] employed ACO for prioritizing test cases based on two approaches: distance-based and index-based. Still, this method lacks proper evaluation as only one program is used as a benchmark within 6 test cases.

- ***History of test execution:*** Noguchi et al. [69] suggested an ACO-based framework of black-box testing for TCP. However, this framework needs to be compared to optimization-based methods for better performance evaluation.

- ***Software component metrics:*** A fourth method proposed by Silva et al. [70] uses ACO for prioritizing test cases based on software component metrics. This study utilized three factors: test cases' criticality, execution time, and faults history. The proposed method used two metrics: APFD and cost-cognizant of the average percentage of faults detected per cost ($APFD_C$). This method yields an average of 92.71% and 92.97%, respectively. A comparison with original, random, optimal ordering and the greedy algorithm was carried out.

- ***Multiple factors:*** Ahmad et al. [11] suggested a multi-objective TCP based on six test factors which are (1) importance (of requirement), (2) volatility (change of requirement), (3) complexity (of requirement), (4) Fault Rate (of requirement), (5) Time (for test case), and (6) Coverage (of requirement). Based on the test factor's weight, test cases are generated and assigned values; then, they were prioritized using ACO. However, this technique is yet to be implemented.

2. **Particle Swarm Optimization (PSO) Applications:**

A total of 6 studies involved the applications of PSO in TCP in which several factors for prioritization were applied as follows.

- ***Fault coverage:*** Mann et al. [71] introduced a PSO method that first generates test cases using GA, PSO, and ABC, then prioritizes them using PSO based on maximum fault coverage. Nevertheless, this approach needs a more mature comparison with existing optimization techniques.

- ***Incorporating test-point coverage with execution time:*** Xing et al. [72] developed an artificial fish school algorithm (AFSA) for prioritizing test cases based on two factors, i.e., test points coverage and execution time. The AFSA implemented using python used 4 C programs from SIR for experiments and compared with GA and K-means. The results reported AFSA outperformed GA and K-means in terms of the average percentage of test-point coverage (APTC) and effective execution time (EET) metrics, i.e., gained an average of 98% of statement coverage within an average of 3.75 of time. However, this algorithm depends on source code in which it is absence make the algorithm stopped working. Also, if gained full coverage of code, it does not mean reached 100% of effectiveness.

- ***Incorporating execution time with code coverage:*** Sun et al. [73] proposed an epistatic PSO-based multi-objective method for prioritizing test cases according to branch coverage (code coverage) and effective execution time of test cases. However, the achievement of this method was not numerically specified. Likewise, the method has not been compared to other optimization algorithms such as GA, ACO, and ABC.

- ***Incorporating execution time with fault coverage:*** Tyagi and Malhotra [74] introduced a multi-objective PSO (MOPSO) method for reduction, selecting a minimal set of test cases, then prioritizing them for achieving maximum fault coverage within minimum execution time (time constraint). The shortcoming of this technique emanates from the fact that a set of test cases are selected first before being prioritized accordingly, which implies that some test cases will be skipped. Furthermore, the amount of test cases is insufficient, which means that this technique needs further empirical evaluation and comparison with optimization techniques.

- ***String distance of test cases:*** Khatibsyarbini et al. [75] proposed a PSO-based method that prioritizes test cases according to their weight-hybrid string distance. This study takes string inputs of test cases and utilized four variants of term frequency-inverse document frequency (TFIDF) for prioritizing test cases without source code consideration. The technique achieved an overall average of APFD is 94.35% and was compared to four string distance techniques. However, the precision of string distance techniques is based on the quality of test cases created and their documentation.

- ***Modifications of the software units:*** Hla et al. [76] introduced a PSO-based method for prioritizing test cases according to modifications of the software units. Meanwhile, the evaluation criteria are not clear due to the lack of sufficient information regarding achievement, comparison, and measurement. Therefore, this technique needs empirical evaluation and comparison against other optimization algorithms.

- ***Multiple factors:*** Samad et al. [77] built a multi-objective PSO that prioritizes test cases according to three factors, i.e., fault detection ability, code coverage, and execution time (cost of a test case). The algorithm achieved 17 – 86% inclusiveness, 33 – 85% precision, and 17

– 86% size reduction, compared with no ordering, reverse ordering, and random ordering techniques. However, the algorithm did not compare with AI techniques, as well as the comparison results with traditional techniques were not reported. Joseph et al. [78] proposed a modified PSO for prioritizing test cases according to seven factors: (1) customer-assigned priority (CP), (2) implementation complexity (IC), (3) changes in requirements (RC), (4) fault impact of requirements (FI), (5) completeness (CT), (6) traceability (TR), and (7) execution time (ET). To overcome the local optima trap, this algorithm introduced the GAs mutation operator. However, the study provides little information about implementation, experiment, comparison, and evaluation.

3. **Artificial Bee Colony (ABC) applications:**

ABC has been implemented for optimizing test cases for TCP in a few studies. As observed in this study, 3 studies were found in the Scopus database based on ABC.

- ***Fault exposing capability:*** Khanna et al. [79] presented an ABC-based algorithm for scheduling test cases of web applications. Hence, the test cases are generated from source code then ordered according to their fault exposing capability. However, this study did not clearly specify the numerical achievement of ABC in terms of APFD and APFDC.

- ***Change coverage:*** Manaswini and Rama Mohan Reddy [80] proposed an ABC-based algorithm for TCP using the bee's nature. However, this study did not specify any achievement or comparison with other techniques. Also, the metric and objective of the algorithm are to cover the change in different versions of a program under test, while the fitness function ensures the maximum number of the requirements, which makes it imprecise.

- ***Decision matrix:*** Vats and Kumar [81] suggested an ABC-based algorithm for prioritizing test cases, where the decision matrix is calculated as total faults divided by time of execution for each test case. However, it was not applied in a controlled experiment environment, and only 10 test cases were used.

4. **Glowworm Swarm Optimization (GSO) Applications:**

GSO has been implemented in several domains, including regression testing. In TCP, there is only one study that uses GSO for prioritizing test cases as below:

- ***Multiple factors:*** Raman and Subramani [82] presented a multi-objective GSO-based method for optimizing test cases according to statement and fault coverage. This approach reached 90% of APFD and was compared with PSO and ABC. Still, this approach lacks sufficient evaluation because it was not applied in a controlled experiment environment, and the number of test cases was very few.

5. **Cat Swarm Optimization (CSO) Applications:**

In terms of CSO applications in TCP, there are three studies found based on fault detection rate that was incorporated with other factors as below:

- ***Incorporating test case clustering with fault detection rate:*** Vats and Kumar [83] and Yadav and Dutta [84] presented CSO-based methods for prioritizing test cases. Nevertheless, there is insufficient information regarding evaluation and comparison as well as no controlled experiment was applied on these methods. Besides, the comparison in the second study was made using success rate rather than APFD metric.

- ***Multiple factors:*** Likewise, Manaswini and Rama Mohan Reddy [85] proposed a CSO-based algorithm for prioritizing test cases based on three parameters. These parameters are fault detection rate, time of execution for the test case, and coverage scope. However, the algorithm lacks adequate evaluation as there is no measurement or comparison nor any controlled experiments specified.

6. **Cuckoo Search Algorithm (CSA) Applications:**

There are only two applications of CSA proposed for TCP as follow:

- ***Code coverage:*** Dhareula and Ganpati [86] presented a CSA for TCP. Even though this method was based on code coverage, its proficiency is measured using APFD, which falls below the 80% mark.

- ***Incorporating execution time with faults coverage:*** Nagar et al. [87] proposed a CSA for optimizing test cases by prioritizing and selecting only effective test cases that capture faults. Nonetheless, this algorithm needs sufficient evaluation in terms of measurements, achievement, benchmarking, and comparison with other algorithms.

7. **Firefly Algorithm (FFA) Applications:**

Only two studies were found as regard applications of FFA in TCP as follows:

- ***Incorporating the importance of test cases with their edit distance:*** Su et al. [88] presented an FFA for prioritizing test cases based on a hybrid model, which is the correlation between edit distance of test cases, i.e., the similarity, and their importance, i.e., the test data. The algorithm was measured using APFD and time of execution as well as compared with PSO, Greedy, and original FFA. Besides that, the achievement gained was an average of 95.27 as in APFD within an average of 221 sec as execution time.

- ***String distance of test cases:*** Khatibsyarbini et al. [12] introduced an FFA method for TCP. The approach was evaluated using the APFD metric and execution time and was compared against PSO, local beam search (LBS), greedy, and GA. The reported achievements obtained was an average of 94.07% as APFD within 373 sec as a time of execution.

8. **Bat Algorithm (BA) Applications:**

In terms of bat algorithm application for TCP, there is only one study as detailed below:

- ***Fault coverage:*** Bajaj and Sangwan [89] suggested a bat algorithm for prioritizing test cases based on fault coverage. Compared with untreated, random, reverse order, and GA, this algorithm gained 84 – 88.07% APFD values. However, the algorithm lacks sufficient evaluation, only applied on one project with 10 test cases matched with 13 faults.

- ***Metrics of code maintainability:*** Öztürk [90] introduced a TCP method using the bat algorithm. In this study, 12 metrics of code maintainability were investigated, including LOC. However, the achievement of this method was not clearly specified as the comparison with other algorithms was only specified in figures instead of numbers.

9. **Lion Search Algorithm (LSA) Applications:**

LSA was employed for prioritizing test cases in only one study presented here:

- ***Historical failure patterns:*** Asthana et al. [91] introduced a lion-based algorithm for optimizing test cases. This algorithm first prioritizes test cases then selects only an effective set for execution where 80% of faults were detected as reported in the experiments. Even though the APFD metric has been used for measurement, no comparison and achievements are specified.

10. **Flower Pollination Algorithm (FPA) Applications:**

In TCP, FPA was applied in only one article found in this study as detailed:

- ***Code coverage:*** Dhareula and Ganpati [92] presented an FPA-based method for prioritizing test cases. However, this algorithm suffers from insufficient evaluation against optimization techniques.

11. **Gray Wolf Optimization (GWO) Applications:**

The applications of GWO in TCP were found in one study as presented in this work.

- ***Incorporating test case clustering with code coverage:*** Badanahatti and Murthy [93] introduced a cloud-based regression testing method for generating, clustering, and prioritizing test cases, respectively. This technique first generates test cases (in the cloud) based on code coverage (including loop, statement, line, and comment coverage), then clusters them according to closeness using Kernel Fuzzy C-Means (KFCM) clustering algorithm, and finally prioritizes test cases using a GWO algorithm. However, the comparison was conducted just between clustering algorithms, i.e., K-means against KFCM, and memory usage metric is merely related to regression testing.

12. **Shuffled Frog-Leaping Algorithm (SFLA) Applications:**

    For SFLA, a single study was found as detailed below:

    - ***Multiple factors:*** Manaswini and Rama Mohan Reddy [94] employed SFLA for prioritizing test cases according to the coverage of code, cost (time of execution), and fault. However, not enough information about implementation, comparison, and achievement is specified.

5.2 Hybrid Swarm Intelligence Algorithms

In terms of hybrid algorithms where two or more algorithms are integrated to improve proficiency, convergence, and accuracy, those hybrid algorithms were improved. As mentioned earlier, hybridization resulted in efficiency and effectiveness improvements for TCP methods. In this review, 13 studies were found that implemented hybrid algorithms in TCP as improvements in which they were popularly categorized as hybridization among SI algorithms, integrating SI algorithms with a genetic algorithm, integration with fuzzy logic, and integration with SVM.

1. **Hybrid within Swarm:**

   Five studies were found in terms of hybridization among SI.

   - ***Requirement-based:*** Dahiya and Solanki [95] provide a hybrid technique where ACO and PSO are combined for TCP. In this technique, four factors were used, i.e., customer-assigned priority (CP), implementation complexity (IC), requirement changes (RC), and fault impact of requirements (FI). This technique used an industrial case study written in java, matched with unprioritized, random, and reverse prioritized sequence, as well as ACO and PSO. Also, it was measured by APFD (91%) and PTR (54%) metrics.

   - ***Code-coverage:*** Bajaj and Sangwan [96] integrated CSA with PSO for optimizing test cases based on statement coverage. The algorithm first prioritizes test cases, selects modification-revealed among them, then eliminates redundant test cases. The algorithm used 3 java programs from SIR for experiments and achieved 98.27% as an average of APSC within 373.55 sec. Also, the algorithm outperformed GA, GSA and PSO.

   - ***Quality metrics:*** Kumar and Muthukumaravel [97] proposed a TCP approach using PSO and an improved CSA, prioritizing test cases based on quality metrics, i.e., coupling, cohesion, and intended fault IF value. The workings of this approach are by firstly generating test cases from the application, then extracting quality metrics from the test cases, and finally optimizing test cases using PSO. The PSO output is considered input for ICSA in producing final optimized test suites. However, no evaluation is specified for this approach to show hybridization improvements against other optimization methods.

   - ***Incorporating test case clustering with multiple factors:*** earlier, Kale and Murthy [98] presented a hybrid swarm algorithm by integrating FFA with ABC for regression testing. The hybrid algorithm utilized the scout bee phase of the ABC algorithm in the FFA to improve the search quality in exploitation and exploration. This algorithm has four phases. First, it generates test cases. Second, it identifies factors for identifying test cases for prioritization, i.e., time, trace events, behavioral dependency, and responsibility. Then, the identified test cases were clustered using the K-means algorithm to separate relevant and irrelevant test cases. The end, prioritizing relevant test cases using the previously identified factors. The algorithm was measured using APFD and compared with an existing method which is not identified. Authors claimed that the performance of their algorithms according to APFD metric reached 85% and 70%, respectively. However, it seems that this algorithm lacks sufficient evaluation in terms of the application in reliable projects as in controlled experiments and needs to be compared with other optimization algorithms to prove its superiority.

   - ***Fault coverage:*** Bajaj and Abraham [99] introduced a hybrid DAPSO (dragonfly with PSO) algorithm for optimizing test cases. The algorithm first prioritizes test cases based on fault coverage then minimizes them. DAPSO compared with random search, GA, PSO, Bat and dragonfly algorithms (discrete and combinatorial) and achieved an average of 93.64% of APFD for TCP and an average of

83.69% of test minimization percentage (TMP) for reduction.

2. **Swarm Intelligence with Genetic Algorithm (GA) Integration:**

GA has been combined with SI in which four works were found for optimizing test cases as detailed:

- ***Fault coverage:*** Bajaj and Sangwan [100] suggested a discrete CSA for prioritizing test cases based on fault coverage. The algorithm used 4 office applications in the experiment and was compared with random search (RS), ACO, GA, tree seed algorithm (TSA), CSA, where the APFD metric was used for measuring effectiveness.
- ***Code coverage:*** Dhareula and Ganpati [101] proposed FPA that is genetically modified for prioritizing test cases. In this work, the EclEmma tool is used for code coverage to optimize test cases accordingly. Then optimizing test cases starts using FPA while the output is supplied as input to the GA. GM-FPA algorithm outperforms GA, FPA, random, and reverse random ordering as it achieves an average of 73% of APFD within an average of 0.165 sec. of execution time. However, this method's performance falls below 80% of the APFD metric.
- ***Incorporating execution time with historical failure patterns:*** Padmnav et al. [102] presented a hybrid algorithm for TCP based on historical failure patterns of regression cycles and execution time of test cases. The lack of information about experiments and the comparison of this algorithm makes the evaluation of this algorithm unclear for researchers.
- ***Fault detection:*** Saraswat and Singhal [103] introduced a hybrid algorithm for optimizing test cases using GA and PSO. The algorithm first prioritizes test cases using GA, then takes the optimized test cases as an input to the PSO. Furthermore, in the PSO phase, the only global best value is shared with others, not the whole population. However, this hybrid algorithm was not compared with any optimization techniques.

3. **Swarm Integration with Fuzzy:**

Only a study was found that compounds with fuzzy logic with swarm as:

- ***Fault coverage:*** Nayak et al. [104] combined the honeybee algorithm with a fuzzy rule for prioritizing test cases according to the rate of fault detection. This method used a fault matrix matched with related test cases and execution time as inputs. The method experimented on two VB projects, and no prioritization, reverse, random, Kavitha and Sureshkumar [105], Tyagi and Malhotra [106], and Nayak et al. [107] techniques. They reported that their technique achieved 85% of APFD within 40% of tests executed to detect all faults. However, there is insufficient evidence on this technique's performance, as only 18 test cases were used within two VB projects, and no public dataset like SIR was used.

4. **Swarm integration with Artificial Neural Networks (ANN):**

One proposed work detected in this study is as below:

- ***Incorporating test case clustering with fault detection:*** Harikarthik et al. [108] presented a hybrid algorithm for prioritizing test cases. The algorithm first generates test cases from program source code and then clusters them to choose relevant test cases intended to be optimized using a modified kernel fuzzy c-means (MKFCM) algorithm. After picking appropriate test cases, optimization is then carried out using the ANN-Whale algorithm according to maximum faults detection. This algorithm outperforms the existing ANN algorithm in terms of the time of execution and the required amount of memory. Despite its success in reducing execution time and memory required during execution, this algorithm needs more evaluation in terms of controlled experiments as well as its specified performance is relatively low according to the APFD metric.

5. **Swarm Integration with Integer Linear Programming:**

Combing SI with linear programming was observed in one study as:

- ***Incorporating method-call with code coverage:*** Wong et al. [109] applied a TCP technique for a time-constraint environment using CSA and integer linear programming. The prioritization of test cases was carried out based on code coverage and method-call information using CSA, then selecting the best test cases was performed using integer linear programming model. Concerning the generalizability and reliability of the technique, the amount of optimized test cases is still very few, and more controlled experiments are needed to be involved in the evaluation.

6. **Swarm integration with SVM:**

There is only one study that SI integrated with Support Vector Machine (SVM) was observed as:

– ***Requirement-based:*** Dahiya and Solanki [110] recommended a CSA integrated with SVM for prioritizing test cases according to the distance matrix of requirements. This algorithm matched with firefly algorithm, where the achievement is 90.02% of APFD within 108.4456 MS, Max = 0.994366, Min = 0.187267, mean = 0.810716, median = 0.999526, and SD = 0.057313. However, the experimental evaluation has not been specified and compared to just one algorithm, which may be insufficient for evaluation.

5.3 Measurement and Evaluation

In regression testing, it is known that new TCP methods must be assessed for their performance by formal measurement and comparison against existing methods [9], [111]. Studies in this domain, such as Hao et al. [9] and Harman et al. [112], have identified two aspects of performance to be measured, which are efficiency and effectiveness; efficiency can be scaled using actual execution time of prioritization, i.e., sometimes noted as testing cost and effectiveness depends on the bases of the prioritization process. Accordingly, Figure 8 demonstrates the current evaluation metrics in TCP that are widely employed.

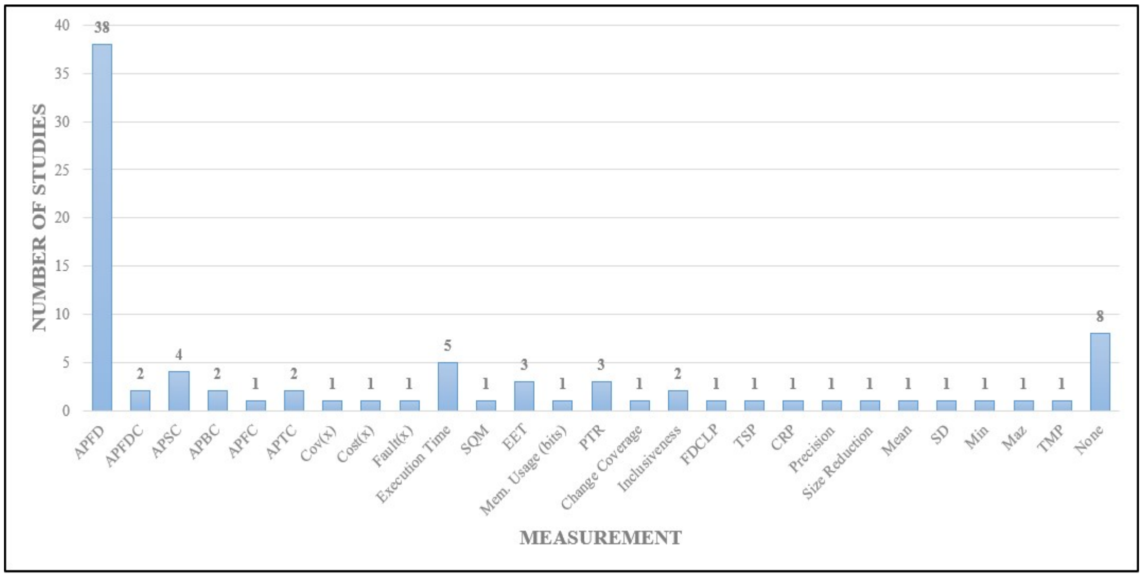

Figure 8: Measurements used in Current Test Case Prioritization

As evident in Figure 8, it can be observed that the weighted average percentage faults detected (APFD) metric is the most used, in which approximately 66.7% of the current studies have been evaluated and compared against others. This is followed by execution time at 8.8% of these studies. Next, the average percentage of statement coverage (APSC) for measuring statements coverage. Then cost-cognizant of the average percentage of faults detected per cost (APFDC), Average percentage of branch coverage (APBC), effective executing time (EET), and percentage of test suite required for complete fault coverage (PTR), which was used in just three studies for each of them. Finally, the Average percentage of function coverage (APFC), the average percentage of test-point coverage (APTC), Cov(x), Cost(x), Fault(x), SQM, Memory usage (bits), and change coverage metrics have only been used in one study for each. Notably, inclusiveness, precision, size reduction, test selection percentage (TSP), mean, min, max, and standard deviation are metrics used for scaling the reduction of test cases in studies that combined TCP with test selection or reduction. Remarkably, 14% of the previous studies did not disclose any evaluation metric used for assessing their performance.

On the experimental side, benchmarking is necessary for evaluating the performance of any TCP method. For reliability and scalability, assessing testing techniques via real-world or industrial projects is the best choice as used only in 5% of the reported studies. However, industrial projects cannot be available due to confidentiality concerns [17]. For this reason, public datasets such as Software-artifact Infrastructure Repository (SIR) and GitHub are an excellent choice for evaluating testing techniques in which researchers can replicate or empirically assess such methods [17]. As public datasets, SIR is the most popular dataset, which has been used in 22.8%, and GitHub also has been used in 7% of these current studies. However, around 59.7% of the studies were not implemented on public datasets, adversely affecting their results' generalizability and reproducibility, as stated by Bajaj and Sangwan [17]. For seeking more information, refer to Table 3 in the appendix.

5.4 Single vs Multi-objective TCP

With the increasing use of multi-objective optimization to solve NP-hard problems, TCP has been treated as a multi-objective problem in the research community [9], [17], [56], [104]. Several researchers reported that multi-objective TCP methods outperform single-objective TCP for several reasons, such as their capability to cope with the increasing complexity of test case optimization and to tackle two or more objectives for optimization [111], [113]–[115]. Recently, there has been a notable growth of multi-objective TCP among SI algorithms in which 24 (i.e., around 42%) of the current studies were multi-objective since 2014, as demonstrated by Figure 9 below.

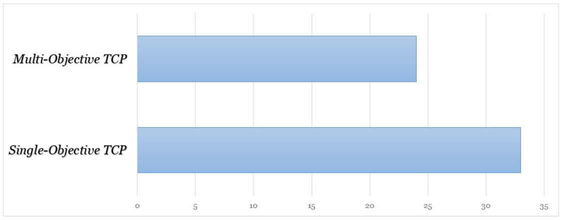

Figure 9: Single and Multi-objective TCP

6 PROBLEMS AND OPEN RESEARCH ISSUES

In conclusion, based on the observations in this study, there are a few issues and future directions to be drawn as follows:

1. ***Incorporating more swarm techniques:*** more SI algorithms have not been explored in TCP, such as African buffalo, bacterial foraging, monkey search, cockroach, wolf search, chicken swarm, grasshopper optimization, etc. Exploring more swarm algorithms will enrich the domain of TCP with more methods and empirical studies with different perspectives in which experiencing such algorithms in terms of optimization proficiency, scalability, and reliability.

2. ***Evaluation issues:*** at the end of this study, it was observed that most of the work reviewed suffers from a sufficient amount of evaluation at different levels of perspectives. This issue is evidently clear when looking at Table 3 in the appendix. It was also confirmed by previous studies such as Arrieta et al. [32], Ba-Quttayyan et al. [116], Bajaj and Sangwan [17] and Brunetto et al. [117].

   (a) *Empirical Studies:* the existing publications were mostly expressed in the initial form of ideas or methods in which their results still need to be confirmed empirically. So, more empirical studies were needed to confirm the reliability of such studies.

   (b) *Dataset Publicity:* using public datasets, such as SIR or GitHub, provides a stable environment for assessing and reproducing researchers' work for comparison purposes against other modern methods. However, most researchers used either private or industrial datasets. Due to confidentiality, other researchers cannot reproduce and evaluate their results against prior research, which affects the generalizability and reliability of such publications.

   (c) *Fitness Function:* Most reviewed studies do not express their fitness functions, while others use famous metrics as fitness functions. Hence, fitness function should be precisely determined cost-effectively due to its significant effects on producing accurate results, especially when optimizing multiple objectives.

   (d) *Measurements:* according to Figure 8, the most common metrics used are APFD and Execution Time. Moreover, most of the current publications used single performance metrics, whereas researchers like Bajaj and Sangwan [17] and Arrieta et al. [32] suggested the usage of two or more metrics for the robustness of the proposed work.

   (e) *Experiments:* it is noticed that most of the studies compared their results with traditional ordering. However, this will not make results solid unless matched against current optimization methods. Besides, when proposing multi-objective methods, it should also be compared against the multi-objective method. Likewise, the number of test cases and the size of subjects (programs) used during experiments should be sufficient to obtain satisfying results. Equally, it is good to take advantage of the support of commonly and publicly available tools such as *selenium*, *gcov*, and *EclEmma*, for improving the optimization process. Lastly, when applying optimization methods to the industry, Arrieta et al. [32] suggested using a user-friendly tool form because practitioners are always concerned about friendly tool support.

3. ***Hybridization:*** due to the increased performance and interest of hybrid algorithms, more hybridization is needed for SI and other optimization techniques for enhancing algorithms proficiency.

4. ***Multi-objective:*** In this review, it is noticed that there is an increasing growth in fault-based TCP methods of about 38.6% (22 papers), much more than code-based TCP methods with about 24.6% (14 paper). Also, 4 studies (around 7%) combined code with fault. Consequently, test cases were prioritized according to several bases in regression testing. TCP factors that have been used independently as reported by this study were: code coverage, statement coverage, fault coverage, fault detection, requirements (distance-based and index-based), history of test execution, string distance of test cases, modifications of the software units, fault exposing capability, change coverage, decision matrix (total faults/time of execution for each test case), historical failure patterns. It is also remarked that there is increasing interest in using multi-objective optimization of test cases. Due to this, the diversity of factors (objectives and criteria/constraints) can improve the proficiency of TCP methods. As observed in this study, TCP is carried out based on multiple factors as:

   (a) *Execution Time:* as observed in this review, test cases were not prioritized based on their execution time alone but incorporated with other factors such as code coverage, statement coverage, fault coverage, fault detection, coverage scope, historical failure patterns, etc. However, the execution time can also be incorporated with other factors such as fault history, fault severity, test case importance, and others.

   (b) *Test Case Clustering:* clustering test cases as observed has been incorporated with other factors such as fault detection rate, code coverage, test case clustering with multiple factors (time, trace events, behavioral dependency, and responsibility for), and fault detection. However, test case clustering can also be incorporated with fault severity, requirements, change coverage, and other factors. Furthermore, test cases can be clustered based on different bases such as similarity, dissimilarity, relevance, or other bases.

   (c) *Statement Coverage:* As mentioned before, statement coverage has been incorporated with execution time. It is also incorporated with fault coverage in one study.

   (d) *Metrics of Code Maintainability:* It is a group of metrics for code maintenance. The used metrics of code maintainability were 12 as the total number of fields in the class (NOF), the total number of methods in the class (NOM), the total number of properties in the class (NOP), the total number of public fields in the class (NOPF), the total number of public methods in the class (NOPM), the total number of source-code lines in the class (LOC), weighted methods per class (WMC), the total number of children (sub-classes) of the class (NC), depth of inheritance tree (DIT), lack of cohesion of methods (LCOM), the total number of classes that reference the class (FANIN), and the total number of classes referenced by the class (FANOUT). Yet, the above-mentioned were class metrics. Other metrics, called method metrics, can be used that consist of LOC, cyclomatic complexity (CC), and parameter count (PC). Likewise, there are other code maintainability metrics such as Lines of Executable code, Efferent Coupling (EC), number of Lines Changed in the Class (CHANGE), Comment Line of Code (CLOC), maintainability Index (MI), etc. as reported by Ardito et al. [118] in their review.

   (e) *Software Component Metrics:* in terms of component metrics, the used metrics were test cases' criticality, execution time, and faults history. These metrics were closely related to test execution history, and other data can be considered.

(f) *Importance of Test Case with their Edit Distance:* as observed in this study, which is related to test data, more features of test cases are needed, such as similarity and dissimilarity.

(g) *Quality Metrics:* in this study, quality metrics used were coupling, cohesion, and intended fault (IF) value. Other metrics also can be integrated.

(h) *Method-Call with Code Coverage:* as mentioned before, method-call is a kind of code maintainability metric.

(i) *Multiple Factors:* in one of the previous studies, a combination of factors was used as (1) importance (of requirement), (2) volatility (change of requirement), (3) complexity (of requirement), (4) Fault Rate (of requirement), (5) Time (for test case), and (6) Coverage (of requirement). Also, customer-assigned priority (CP), implementation complexity (IC), requirement changes (RC), and fault impact of requirements (FI) were used in one study. Such factors related more to requirements. Another study used multiple diverse factors, which were: (1) customer-assigned priority (CP), (2) implementation complexity (IC), (3) changes in requirements (RC), (4) fault impact of requirements (FI), (5) completeness (CT), (6) traceability (TR), and (7) execution time (ET). Lastly, fault detection ability, code coverage, and execution time have been used in one study here. As observed, a more diverse combination of related factors is needed for TCP enhancements.

Accordingly, more factors were needed to be involved in the optimization process of test cases and integrating them with others as a multi-objective process against single-objective optimization. Furthermore, Parejo et al. [119] asserted that better results and distribution could be gained for APFD values when using multi-objective combinations for prioritizing test cases than single objectives.

Lastly, this study has limitations on the scope, search databases, and studied domain which are briefly emphasized below:

- ***Scope:*** only the SI applications on the TCP were investigated in this study as a specific topic and narrowed research area. In general, AI, ML, and evolutionary computation are wider areas and have been studied broadly in the literature.
- ***Search databases:*** this study considered only the Scopus database for obtaining the primary studies. Other research databases, including google scholar, web of science, research gate, etc., can provide more primary studies for investigating SI in the TCP domain.
- ***Domain:*** regression testing domain has different categories as TCP is considered one of them. Test case selection, hybrid, test generation, and other testing categories are not included here. These are recent trends for potential studies in the research community.

7 CONCLUSION

Prioritizing test cases is one of the most effective ways of performing regression testing, especially when optimization methods are employed. SI has attained remarkable importance among optimization methods, with a growing number of publications based on SI in the context of TCP. In this study, 57 papers were identified, analyzed, and classified according to predefined research questions to determine the current state of SI applications in TCP. The study addresses internal structure, single, hybrid SI algorithms, multi-objective optimization, and evaluating such studies in the field. In summary, this study offered insights into the application of SI to prioritize test cases ideally and provide ideas for future research, in the previous sub-section, in terms of employing SI for TCP.

REFERENCES:

[1] H. Krasner, “The Cost of Poor Software Quality in the US: A 2020 Report,” 2021. [Online]. Available: https://www.it-cisq.org/the-cost-of-poor-software-quality-in-the-us-a-2020-report.htm.

[2] U. staff Computerworld, “Top software failures in recent history,” *Computerworld*, 2020. https://www.computerworld.com/article/3412197/top-software-failures-in-recent-history.html (accessed Jan. 10, 2021).

[3] R. Kazmi, D. N. A. A. Jawawi, R. Mohamad, and I. Ghani, “Effective Regression Test Case Selection: A Systematic Literature Review,” *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–32, Jun.

2017, doi: 10.1145/3057269.

[4] IEEE Standards Coordinating Committee, "IEEE Standard Glossary of Software Engineering Terminology (Std 610.12-1990(R2002))," *IEEE Computer Society*. IEEE, New York, USA, pp. 1–88, 1990, [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342.

[5] G. Rothermel and M. J. Harrold, "A framework for evaluating regression test selection techniques," in *Proceedings of 16th International Conference on Software Engineering*, 1994, pp. 201–210, doi: 10.1109/ICSE.1994.296779.

[6] H. Do, "Recent Advances in Regression Testing Techniques," in *Advances in Computers*, vol. 103, A. M. Memon, Ed. Elsevier, 2016, pp. 53–77.

[7] A. B. Sanchez, S. Segura, and A. Ruiz-Cortes, "A Comparison of Test Case Prioritization Criteria for Software Product Lines," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, Mar. 2014, pp. 41–50, doi: 10.1109/ICST.2014.15.

[8] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Softw. Testing, Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, Mar. 2012, doi: https://doi.org/10.1002/stvr.430.

[9] D. Hao, L. Zhang, and H. Mei, "Test-case prioritization: achievements and challenges," *Front. Comput. Sci.*, vol. 10, no. 5, pp. 769–777, Oct. 2016, doi: 10.1007/s11704-016-6112-3.

[10] K.-W. Shin and D.-J. Lim, "Model-Based Test Case Prioritization Using an Alternating Variable Method for Regression Testing of a UML-Based Model," *Appl. Sci.*, vol. 10, no. 21, p. 7537, Oct. 2020, doi: 10.3390/app10217537.

[11] S. F. Ahmad, D. K. Singh, and P. Suman, "Prioritization for Regression Testing Using Ant Colony Optimization Based on Test Factors," in *Intelligent Communication, Control and Devices*, 1st ed., vol. 624, R. Singh, S. Choudhury, and A. Gehlot, Eds. Singapore: Springer, Singapore, 2018, pp. 1353–1360.

[12] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, H. N. A. Hamed, and M. D. Mohamed Suffian, "Test Case Prioritization Using Firefly Algorithm for Software Testing," *IEEE Access*, vol. 7, pp. 132360–132373, 2019, doi: 10.1109/ACCESS.2019.2940620.

[13] C. Catal, "On the application of genetic algorithms for test case prioritization," in *Proceedings of the 2nd international workshop on Evidential assessment of software technologies - EAST '12*, Sep. 2012, pp. 9–14, doi: 10.1145/2372233.2372238.

[14] S. Sharma and A. Singh, "Model-based test case prioritization using ACO: A review," in *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2016, pp. 177–181, doi: 10.1109/PDGC.2016.7913140.

[15] A. Bajaj and O. P. Sangwan, "A Survey on Regression Testing Using Nature-Inspired Approaches," in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, Dec. 2018, pp. 1–5, doi: 10.1109/CCAA.2018.8777692.

[16] M. H. Alkawaz and A. Silvarajoo, "A Survey on Test Case Prioritization and Optimization Techniques in Software Regression Testing," in *2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, Dec. 2019, no. December, pp. 59–64, doi: 10.1109/ICSPC47137.2019.9068003.

[17] A. Bajaj and O. P. Sangwan, "A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms," *IEEE Access*, vol. 7, pp. 126355–126375, 2019, doi: 10.1109/ACCESS.2019.2938260.

[18] P. Paygude and S. D. Joshi, "Use of Evolutionary Algorithm in Regression Test Case Prioritization: A Review," in *Proceeding of the International Conference on Computer Networks, Big Data and IoT (ICCBI - 2018)*, vol. 31, no. August 2016, A. P. Pandian, T. Senjyu, S. M. S. Islam, and H. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 56–66.

[19] A. Chakraborty and A. K. Kar, "Swarm Intelligence: A Review of Algorithms," in *Nature-Inspired Computing and Optimization*, 1st ed., vol. 10, no. March, S. Patnaik, X.-S. Yang, and K. Nakamatsu, Eds. Springer, Cham, 2017, pp. 475–494.

[20] D. Karaboga, B. Akay, and N. Karaboga, "A survey on the studies employing machine learning (ML) for enhancing

artificial bee colony (ABC) optimization algorithm," *Cogent Eng.*, vol. 7, no. 1, p. 1855741, Jan. 2020, doi: 10.1080/23311916.2020.1855741.

[21] A. Nayyar and N. G. Nguyen, "Introduction to swarm intelligence," in *Advances in swarm intelligence for optimizing problems in computer science*, 1st ed., A. Nayyar, D.-N. Le, and N. G. Nguyen, Eds. CRC Press, 2019, pp. 53–77.

[22] N. Gunantara, "A review of multi-objective optimization: Methods and its applications," *Cogent Eng.*, vol. 5, no. 1, p. 1502242, Jan. 2018, doi: 10.1080/23311916.2018.1502242.

[23] A. Slowik, J. Yang, N. K. Meena, and P. Singh, *Swarm Intelligence Algorithms: A Tutorial*, 1st ed. First edition. | Boca Raton : Taylor and Francis, 2020.: CRC Press, 2020.

[24] X. Yang, S. Deb, S. Fong, X. He, and Y.-X. Zhao, "From Swarm Intelligence to Metaheuristics: Nature-Inspired Optimization Algorithms," *Computer (Long. Beach. Calif).*, vol. 49, no. 9, pp. 52–59, Sep. 2016, doi: 10.1109/MC.2016.292.

[25] D. M. Utama, S. K. Dewi, A. Wahid, and I. Santoso, "The vehicle routing problem for perishable goods: A systematic review," *Cogent Eng.*, vol. 7, no. 1, p. 1816148, Jan. 2020, doi: 10.1080/23311916.2020.1816148.

[26] Y. Tan, *Gpu-Based Parallel Implementation of Swarm Intelligence Algorithms*. Elsevier, 2016.

[27] A. P. Agrawal and A. Kaur, "A Comprehensive Comparison of Ant Colony and Hybrid Particle Swarm Optimization Algorithms Through Test Case Selection," in *Data Engineering and Intelligent Computing*, vol. 542, S. C. Satapathy, V. Bhateja, K. S. Raju, and B. Janakiramaiah, Eds. Singapore: Springer, Singapore, 2018, pp. 397–405.

[28] S. Kumar and P. Ranjan, "A Comprehensive Analysis for Software Fault Detection and Prediction using Computational Intelligence Techniques," *Int. J. Comput. Intell. Res.*, vol. 13, no. 1, pp. 65–78, 2017, [Online]. Available: http://www.ripublication.com.

[29] S. Wang, S. Ali, T. Yue, Ø. Bakkeli, and M. Liaaen, "Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search," in *Proceedings of the 38th International Conference on Software Engineering Companion*, May 2016, pp. 182–191, doi: 10.1145/2889160.2889240.

[30] A. A. Ahmed, M. Shaheen, and E. Kosba, "Software testing suite prioritization using multi-criteria fitness function," in *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*, Oct. 2012, no. October, pp. 160–166, doi: 10.1109/ICCTA.2012.6523563.

[31] D. B. Mishra, R. Mishra, A. A. Acharya, and K. N. Das, "Test Case Optimization and Prioritization Based on Multi-objective Genetic Algorithm," in *Harmony Search and Nature Inspired Optimization Algorithms*, 1st ed., vol. 741, N. Yadav, A. Yadav, J. C. Bansal, K. Deep, and J. H. Kim, Eds. Singapore: Springer Singapore, 2019, pp. 371–381.

[32] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems," *IEEE Trans. Ind. Informatics*, vol. 14, no. 3, pp. 1055–1066, Mar. 2018, doi: 10.1109/TII.2017.2788019.

[33] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "Hypervolume-Based Search for Test Case Prioritization," in *Search-Based Software Engineering. SSBSE 2015*, 1st ed., vol. 9275, M. Barros and Y. Labiche, Eds. Bergamo, Italy: Springer, Cham, 2015, pp. 157–172.

[34] X. Xie, M. Stumptner, and T. H. Tse, "Introduction to the special issue on program debugging," *J. Syst. Softw.*, vol. 140, pp. 109–110, Jun. 2018, doi: 10.1016/j.jss.2018.03.006.

[35] M. Dorigo, "Optimization, learning and natural algorithms," Politecnico di Milano, Italy, 1992.

[36] P. Singh, N. K. Meena, and J. Yang, "Ant colony optimization, modifications, and application," in *Swarm intelligence algorithms modifications and applications*, vol. 2, A. Slowik, Ed. CRC Press, 2020, p. 392.

[37] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on*

Neural Networks, 1995, vol. 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

[38] A. Slowik, Ed., *Swarm Intelligence Algorithms: Modifications and Applications*, 1st ed. First edition. | Boca Raton : Taylor and Francis, 2020.: CRC Press, 2020.

[39] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," 2005.

[40] A. Slowik, Ed., *Swarm Intelligence Algorithms: Modifications and Applications*, 1st Editio. CRC Press, 2020.

[41] K. N. Krishnanand and D. Ghose, "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics," in *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, 2005, pp. 84–91, doi: 10.1109/SIS.2005.1501606.

[42] T. T. Nguyen, D. Ngoc Vo, and B. H. Dinh, "A cuckoo bird-inspired meta-heuristic algorithm for optimal short-term hydrothermal generation cooperation," *Cogent Eng.*, vol. 3, no. 1, p. 1266863, Dec. 2016, doi: 10.1080/23311916.2016.1266863.

[43] X.-S. Yang, "Firefly Algorithm: Variants and Applications," in *Swarm Intelligence Algorithms: Modifications and Applications*, vol. 7, A. Slowik, Ed. CRC Press, 2020, pp. 175–186.

[44] N. Kanwar, N. K. Meena, and J. Yang, "Bat Algorithm - Modifications and Application," in *Swarm Intelligence Algorithms: Modifications and Applications*, A. Slowik, Ed. CRC Press, 2020, pp. 43–55.

[45] B. R. Rajakumar, "Lion Algorithm and Its Applications," in *Frontier Applications of Nature Inspired Computation*, M. Khosravy, N. Gupta, N. Patel, and T. Senjyu, Eds. Singapore: Springer, Singapore, 2020, pp. 100–118.

[46] M. A. Al-Betar, M. A. Awadallah, I. Abu Doush, A. I. Hammouri, M. Mafarja, and Z. A. A. Alyasseri, "Island flower pollination algorithm for global optimization," *J. Supercomput.*, vol. 75, no. 8, pp. 5280–5323, Aug. 2019, doi: 10.1007/s11227-019-02776-y.

[47] X.-S. Yang, S. Deb, Y.-X. Zhao, S. Fong, and X. He, "Swarm intelligence: past, present and future," *Soft Comput.*, vol. 22, no. 18, pp. 5923–5933, Sep. 2018, doi: 10.1007/s00500-017-2810-5.

[48] A. Shuaibu Hassan, Y. Sun, and Z. Wang, "Optimization techniques applied for optimal planning and integration of renewable energy sources based on distributed generation: Recent trends," *Cogent Eng.*, vol. 7, no. 1, p. 1766394, Jan. 2020, doi: 10.1080/23311916.2020.1766394.

[49] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Eng. Optim.*, vol. 38, no. 2, pp. 129–154, Mar. 2006, doi: 10.1080/03052150500384759.

[50] A. E. Hassanien and E. E. Emary, *Swarm Intelligence: Principles, Advances, and Applications*. CRC Press, 2016.

[51] S. Keele, others, and S. E. Group, "Guidelines for performing systematic literature reviews in software engineering," sn, UK, 2007.

[52] C. Lu, J. Zhong, Y. Xue, L. Feng, and J. Zhang, "Ant Colony System With Sorting-Based Local Search for Coverage-Based Test Case Prioritization," *IEEE Trans. Reliab.*, vol. 69, no. 3, pp. 1004–1020, Sep. 2020, doi: 10.1109/TR.2019.2930358.

[53] C. Lu and J. Zhong, "An efficient ant colony system for coverage based test case prioritization," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Jul. 2018, pp. 91–92, doi: 10.1145/3205651.3205680.

[54] D. Panwar, P. Tomar, H. Harsh, and M. H. Siddique, "Improved Meta-Heuristic Technique for Test Case Prioritization," in *Soft Computing: Theories and Applications*, 1st ed., vol. 583, M. Pant, K. Ray, T. K. Sharma, S. Rawat, and A. Bandyopadhyay, Eds. Singapore: Springer Singapore, 2018, pp. 647–664.

[55] R. Dhiman and V. Chopra, "Novel Approach for Test Case Prioritization Using ACO Algorithm," in *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, Mar. 2019, pp. 292–295, doi: 10.1109/INFOCT.2019.8711039.

[56] M. K. Pachariya, "Building Ant System for Multi-Faceted Test Case Prioritization," *Int. J. Softw. Innov.*, vol. 8, no. 2, pp. 23–37, Apr. 2020, doi: 10.4018/IJSI.2020040102.

[57] Y. Bian, Z. Li, R. Zhao, and D. Gong,

"Epistasis Based ACO for Regression Test Case Prioritization," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 1, no. 3, pp. 213–223, Jun. 2017, doi: 10.1109/TETCI.2017.2699228.

[58] O. Banias, "The drawbacks of statement code coverage test case prioritization related to domain testing," in *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, May 2016, pp. 221–224, doi: 10.1109/SACI.2016.7507373.

[59] L. L. Chen and L. Zhang, "Study of Test Cases Prioritization Based on Ant Colony Algorithm," *Appl. Mech. Mater.*, vol. 263–266, no. PART 1, pp. 2168–2172, Dec. 2013, doi: 10.4028/www.scientific.net/AMM.263-266.2168.

[60] M. Sudhir Kumar and P. Srinivas, "An Ant Colony Algorithm to Prioritize the Regression Test Cases of Object-Oriented Programs," *Indian J. Sci. Technol.*, vol. 9, no. 19, May 2016, doi: 10.17485/ijst/2016/v9i19/89458.

[61] K. Solanki, Y. Singh, and S. Dalal, "Test case prioritization: An approach based on modified ant colony optimization (m-ACO)," in *IEEE International Conference on Computer, Communication and Control, IC4 2015*, Sep. 2016, pp. 1–6, doi: 10.1109/IC4.2015.7375627.

[62] K. Solanki, Y. Singh, and S. Dalal, "Experimental Analysis of m-ACO Technique for Regression Testing," *Indian J. Sci. Technol.*, vol. 9, no. 30, Aug. 2016, doi: 10.17485/ijst/2016/v9i30/86588.

[63] K. Solanki, Y. Singh, and S. Dalal, "A Comparative Evaluation of 'm-ACO' Technique for Test Suite Prioritization," *Indian J. Sci. Technol.*, vol. 9, no. 30, pp. 1–10, Aug. 2016, doi: 10.17485/ijst/2016/v9i30/86423.

[64] A. Vescan, C.-M. Pintea, and P. C. Pop, "Solving the Test Case Prioritization Problem with Secure Features Using Ant Colony System," in *International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on EUropean Transnational Education (ICEUTE 2019). CISIS 2019, ICEUTE 2019*, 1st ed., vol. 951, F. M. Álvarez, A. T. Lora, J. A. S. Muñoz, H. Quintián, and E. Corchado, Eds. Seville, Spain: Springer International Publishing, 2020, pp. 67–76.

[65] A. Ansari, A. Khan, A. Khan, and K. Mukadam, "Optimized Regression Test Using Test Case Prioritization," *Procedia Comput. Sci.*, vol. 79, pp. 152–160, Jul. 2016, doi: 10.1016/j.procs.2016.03.020.

[66] B. SURI and S. SINGHAL, "DEVELOPMENT AND VALIDATION OF AN IMPROVED TEST SELECTION AND PRIORITIZATION ALGORITHM BASED ON ACO," *Int. J. Reliab. Qual. Saf. Eng.*, vol. 21, no. 06, p. 1450032, Dec. 2014, doi: 10.1142/S0218539314500326.

[67] D. Gao, X. Guo, and L. Zhao, "Test case prioritization for regression testing based on ant colony optimization," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Sep. 2015, vol. 2015-Novem, no. 91118007, pp. 275–279, doi: 10.1109/ICSESS.2015.7339054.

[68] W. Zhang, Y. Qi, X. Zhang, B. Wei, M. Zhang, and Z. Dou, "On Test Case Prioritization Using Ant Colony Optimization Algorithm," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 2767–2773, doi: 10.1109/HPCC/SmartCity/DSS.2019.00388.

[69] T. Noguchi, H. Washizaki, Y. Fukazawa, A. Sato, and K. Ota, "History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2015, pp. 1–2, doi: 10.1109/ICST.2015.7102622.

[70] D. S. Silva, R. Rabelo, P. S. Neto, R. Britto, and P. A. Oliveira, "A Test Case Prioritization Approach Based on Software Component Metrics," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Oct. 2019, vol. 2019-Octob, pp. 2939–2945, doi: 10.1109/SMC.2019.8914670.

[71] M. Mann, P. Tomar, and O. P. Sangwan, "Bio-inspired metaheuristics: evolving and

prioritizing software test data," *Appl. Intell.*, vol. 48, no. 3, pp. 687–702, Mar. 2018, doi: 10.1007/s10489-017-1003-3.

[72] Y. Xing, X. Wang, and Q. Shen, "Test case prioritization based on Artificial Fish School Algorithm," *Comput. Commun.*, vol. 180, no. September, pp. 295–302, Dec. 2021, doi: 10.1016/j.comcom.2021.09.014.

[73] J. Sun, J. Chen, and G. Wang, "Multi-Objective Test Case Prioritization based on Epistatic Particle Swarm Optimization," *Int. J. Performability Eng.*, vol. 14, no. 10, pp. 2441–2448, 2018, doi: 10.23940/ijpe.18.10.p20.24412448.

[74] M. Tyagi and S. Malhotra, "Test case prioritization using multi objective particle swarm optimizer," in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*, Jul. 2014, pp. 390–395, doi: 10.1109/ICSPCT.2014.6884931.

[75] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and D. N. Abang, "A hybrid weight-based and string distances using particle swarm optimization for prioritizing test cases," *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 12, pp. 2723–2732, 2017.

[76] K. H. S. Hla, YoungSik Choi, and Jong Sou Park, "Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software Retesting," in *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, Jul. 2008, pp. 527–532, doi: 10.1109/CIT.2008.Workshops.104.

[77] A. Samad, H. Bin Mahdin, R. Kazmi, R. Ibrahim, and Z. Baharum, "Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method," *Sci. Program.*, vol. 2021, pp. 1–13, Jun. 2021, doi: 10.1155/2021/9988987.

[78] A. K. Joseph, G. Radhamani, and V. Kallimani, "Improving test efficiency through multiple criteria coverage based test case prioritization using Modified heuristic algorithm," in *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, Aug. 2016, pp. 430–435, doi: 10.1109/ICCOINS.2016.7783254.

[79] M. Khanna, N. Chauhan, and D. K. Sharma, "Search for Prioritized Test Cases during Web Application Testing," *Int. J. Appl. Metaheuristic Comput.*, vol. 10, no. 2, pp. 1–26, Apr. 2019, doi: 10.4018/IJAMC.2019040101.

[80] B. Manaswini and A. Rama Mohan Reddy, "A Regression Test Based on the Test Case Prioritization Techniques by using the Nature of Bees," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 9S3, pp. 320–324, Aug. 2019, doi: 10.35940/ijitee.I3059.0789S319.

[81] R. Vats and A. Kumar, "Artificial Bee Colony Based Prioritization Algorithm for Test Case Prioritization Problem," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 5, pp. 8347–8354, Oct. 2020, doi: 10.30534/ijatcse/2020/207952020.

[82] B. Raman and S. Subramani, "An Efficient Specific Update Search Domain based Glowworm Swarm Optimization for Test Case Prioritization," *Int. Arab J. Inf. Technol.*, vol. 12, no. 6A, pp. 748–754, 2015, [Online]. Available: http://iajit.org/PDF/Vol 12, No. 7 (Special Issue)/7035.pdf.

[83] R. Vats and A. Kumar, "Test Case Prioritization Using Cat Swarm Optimization," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 5, pp. 8142–8148, Oct. 2020, doi: 10.30534/ijatcse/2020/175952020.

[84] D. K. Yadav and S. Dutta, "A New Cluster-Based Test Case Prioritization Using Cat Swarm Optimization Technique," in *Proceedings of the Third International Conference on Microelectronics, Computing and Communication Systems*, 1st ed., V. Nath and J. K. Mandal, Eds. Singapore: Springer, Singapore, 2019, pp. 441–450.

[85] B. Manaswini and A. Rama Mohan Reddy, "A Cat Swarm Optimization Based Test Case Prioritization Technique to Perform Regression Testing," *Int. J. Recent Technol. Eng.*, vol. 8, no. 1, pp. 2677–2682, 2019.

[86] P. Dhareula and A. Ganpati, "Cuckoo Search Algorithm for Test Case Prioritization in Regression Testing," *Int. J. Recent Technol. Eng.*, vol. 8, no. 3, pp. 6004–6009, Sep. 2019, doi: 10.35940/ijrte.C4488.098319.

[87] R. Nagar, A. Kumar, G. P. Singh, and S. Kumar, "Test case selection and prioritization using cuckoos search algorithm," in *2015 International Conference on Futuristic Trends on*

Computational Analysis and Knowledge Management (ABLAZE), Feb. 2015, pp. 283–288, doi: 10.1109/ABLAZE.2015.7155012.

[88] W. Su, Z. Li, Z. Wang, and D. Yang, "A Meta-heuristic Test Case Prioritization Method Based on Hybrid Model," in *2020 International Conference on Computer Engineering and Application (ICCEA)*, Mar. 2020, pp. 430–435, doi: 10.1109/ICCEA50009.2020.00099.

[89] A. Bajaj and O. P. Sangwan, "Test Case Prioritization Using Bat Algorithm," *Recent Adv. Comput. Sci. Commun.*, vol. 14, no. 2, pp. 593–598, May 2021, doi: 10.2174/2213275912666190226154344.

[90] M. M. Ozturk, "Adapting code maintainability to bat-inspired test case prioritization," in *2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, Jul. 2017, pp. 67–72, doi: 10.1109/INISTA.2017.8001134.

[91] M. Asthana, K. D. Gupta, and A. Kumar, "Test Suite Optimization Using Lion Search Algorithm," in *Ambient Communications and Computer Systems*, 1st ed., vol. 1097, Y.-C. Hu, S. Tiwari, M. C. Trivedi, and K. K. Mishra, Eds. Singapore: Springer, Singapore, 2020, pp. 77–90.

[92] P. Dhareula and A. Ganpati, "Flower Pollination Algorithm for Test Case Prioritization in Regression Testing," in *ICT Analysis and Applications*, 1st ed., vol. 93, S. Fong, N. Dey, and A. Joshi, Eds. Singapore: Springer, Singapore, 2020, pp. 155–167.

[93] S. Badanahatti and Y. Murthy, "Optimal Test Case Prioritization in Cloud based Regression Testing with Aid of KFCM," *Int. J. Intell. Eng. Syst.*, vol. 10, no. 3, pp. 96–105, Apr. 2017, doi: 10.22266/ijies2017.0430.11.

[94] B. Manaswini and A. Rama Mohan Reddy, "A shuffled frog leap algorithm based test case prioritization technique to perform regression testing," *Int. J. Eng. Adv. Technol.*, vol. 8, no. 5, pp. 671–674, 2019.

[95] O. Dahiya and K. Solanki, "An Efficient APHT Technique for Requirement-Based Test Case Prioritization," *Int. J. Eng. Trends Technol.*, vol. 69, no. 4, pp. 215–227, Apr. 2021, doi: 10.14445/22315381/IJETT-V69I4P230.

[96] A. Bajaj and O. P. Sangwan, "Tri-level regression testing using nature-inspired algorithms," *Innov. Syst. Softw. Eng.*, vol. 17, no. 1, pp. 1–16, Mar. 2021, doi: 10.1007/s11334-021-00384-9.

[97] K. Senthil Kumar and A. Muthukumaravel, "A Hybrid Approach for Test Case Prioritization using PSO Based on Software Quality Metrics," *Int. J. Eng. Technol.*, vol. 7, no. 3.12, p. 300, Jul. 2018, doi: 10.14419/ijet.v7i3.12.16046.

[98] S. Kale and Y. S. S. R. Murthy, "Hybrid firefly algorithm based regression testcase prioritisation," *Int. J. Bus. Intell. Data Min.*, vol. 12, no. 4, p. 340, 2017, doi: 10.1504/IJBIDM.2017.086983.

[99] A. Bajaj and A. Abraham, "Prioritizing and Minimizing the Test Cases using the Dragonfly Algorithms," *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.*, vol. 13, no. July, pp. 062–071, 2021.

[100] A. Bajaj and O. P. Sangwan, "Discrete cuckoo search algorithms for test case prioritization," *Appl. Soft Comput.*, vol. 110, p. 107584, Oct. 2021, doi: 10.1016/j.asoc.2021.107584.

[101] P. Dhareula and A. Ganpati, "Software test case prioritization using genetically modified flower pollination algorithm (Gm-fpa)," *Int. J. Sci. Technol. Res.*, vol. 8, no. 12, pp. 298–306, 2019.

[102] P. Padmnav, G. Pahwa, D. Singh, and S. Bansal, "Test Case Prioritization based on Historical Failure Patterns using ABC and GA," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Jan. 2019, pp. 293–298, doi: 10.1109/CONFLUENCE.2019.8776936.

[103] P. Saraswat and A. Singhal, "A hybrid approach for test case prioritization and optimization using meta-heuristics techniques," in *2016 1st India International Conference on Information Processing (IICIP)*, Aug. 2016, pp. 1–6, doi: 10.1109/IICIP.2016.7975319.

[104] S. Nayak, C. Kumar, S. Tripathi, N. Mohanty, and V. Baral, "Regression test optimization and prioritization using Honey Bee optimization algorithm with fuzzy rule base," *Soft Comput.*, vol. 25, no. 15, pp. 9925–9942, Aug. 2021, doi: 10.1007/s00500-020-05428-z.

[105] R. Kavitha and N. Sureshkumar, "Test Case Prioritization for Regression Testing based on Severity of Fault," *Int. J. Comput. Sci. Eng.*, vol. 02, no. 05, pp. 1462–1466, 2010.

[106] M. Tyagi and S. Malhotra, "An Approach for Test Case Prioritization Based on Three Factors," *Int. J. Inf. Technol. Comput. Sci.*, vol. 7, no. 4, pp. 79–86, Mar. 2015, doi: 10.5815/ijitcs.2015.04.09.

[107] S. Nayak, C. Kumar, and S. Tripathi, "Effectiveness of prioritization of test cases based on Faults," in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, Mar. 2016, pp. 657–662, doi: 10.1109/RAIT.2016.7507977.

[108] S. K. Harikarthik, V. Palanisamy, and P. Ramanathan, "Optimal test suite selection in regression testing with testcase prioritization using modified Ann and Whale optimization algorithm," *Cluster Comput.*, vol. 22, no. S5, pp. 11425–11434, Sep. 2019, doi: 10.1007/s10586-017-1401-7.

[109] Y. Wong, H. Zeng, H. Miao, H. Gao, and X. Yang, "The Cuckoo Search and Integer Linear Programming Based Approach to Time-Aware Test Case Prioritization Considering Execution Environment," in *Collaborative Computing: Networking, Applications and Worksharing. CollaborateCom 2018.*, 1st ed., vol. 268, H. Gao, X. Wang, Y. Yin, and M. Iqbal, Eds. Shanghai, China: Springer, Cham, 2019, pp. 734–754.

[110] O. Dahiya and K. Solanki, "An Efficient Requirement-based Test Case Prioritization Technique using Optimized TFC-SVM Approach," *Int. J. Eng. Trends Technol.*, vol. 69, no. 1, pp. 5–16, Jan. 2021, doi: 10.14445/22315381/IJETT-V69I1P202.

[111] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, Jan. 2018, doi: 10.1016/j.infsof.2017.08.014.

[112] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Search Based Software Engineering: Techniques, Taxonomy, Tutorial," in *Empirical Software Engineering and Verification. LASER 2010, LASER 2009, LASER 2008*, vol. 7007, B. Meyer and M. Nordio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–59.

[113] M. D. C. De Castro-Cabrera, A. García-Dominguez, and I. Medina-Bulo, "Trends in prioritization of test cases: 2017-2019," in *SAC '20: Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 2005–2011, doi: 10.1145/3341105.3374036.

[114] M. Harman, "Making the Case for MORTO: Multi Objective Regression Test Optimization," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, Mar. 2011, pp. 111–114, doi: 10.1109/ICSTW.2011.60.

[115] M. Harman, Y. Jia, and Y. Zhang, "Achievements, Open Problems and Challenges for Search Based Software Testing," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2015, no. Icst, pp. 1–12, doi: 10.1109/ICST.2015.7102580.

[116] B. Ba-Quttayyan, H. Mohd, and Y. Yusof, "Regression Test Case Prioritization Frameworks: Challenges and Future Directions," *Int. J. Recent Technol. Eng.*, vol. 8, no. 4, pp. 8457–8462, Nov. 2019, doi: 10.35940/ijrte.D9735.118419.

[117] M. Brunetto, G. Denaro, L. Mariani, and M. Pezzè, "On introducing automatic test case generation in practice: A success story and lessons learned," *J. Syst. Softw.*, vol. 176, p. 110933, Jun. 2021, doi: 10.1016/j.jss.2021.110933.

[118] L. Ardito, R. Coppola, L. Barbato, and D. Verga, "A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review," *Sci. Program.*, vol. 2020, pp. 1–26, Aug. 2020, doi: 10.1155/2020/8840389.

[119] J. A. Parejo, A. B. Sánchez, S. Segura, A. Ruiz-Cortés, R. E. Lopez-Herrejon, and A. Egyed, "Multi-objective test case prioritization in highly configurable systems: A case study," *J. Syst. Softw.*, vol. 122, pp. 287–310, Dec. 2016, doi: 10.1016/j.jss.2016.09.045.

APPENDIX A: ALL PAPERS SUMMARY

Table 3: The Summary of All Involved Studies

No.	*Study*	*Year*	*Algorithm*	*Application*	*# of TC*	*Benchmarks*	*Tools*	*Metric*
S1	[72]	2021	AFSA	Python	8,156	SIR	–	APTC, EET
S2	[100]	2021	CSA + GA	MATLAB	2,400	Terp Office applications	Minitab software	APFD
S3	[104]	2021	ABC + Fuzzy Logic	C	18	Private	QTP	APFD
S4	[95]	2021	ACO + PSO	MATLAB	100	Industrial “Cosmosoft Technologies”	–	APFD, PTR
S5	[96]	2021	GSA + PSO	MATLAB	1,184	SIR	JUnit, Emma	APSC, Inclusiveness, FDCLP, TSP, CRP
S6	[77]	2021	PSO	MATLAB	313	Open source	EclEmma	Inclusiveness, precision, and size reduction
S7	[99]	2021	PSO + DA	MATLAB	2,400	Open source	–	APFD, TMP
S8	[110]	2021	CSA + SVM	MATLAB	–	–	–	APFD, Mean, SD, Min, Max
S9	[89]	2021	Bat	–	10	Private	–	APFD
S10	[52]	2020	ACO	–	1,198	Defects4J	–	–
S11	[81]	2020	ABC	–	10	Private	–	APFD
S12	[83]	2020	CSO	–	10	–	–	APFD, Exec. time
S13	[56]	2020	ACO	MATLAB	8,259	SIR	gcov	–
S14	[88]	2020	FFA	–	1,593	SIR	–	APFD, Exec. time
S15	[64]	2020	ACO	C++	5	–	–	APFD
S16	[92]	2020	FPA	Java	191	Open source	Eclipse IDE, TestNG, EclEmma	APFD
S17	[91]	2020	LSA	MATLAB	888	Private	Excel	APFD
S18	[101]	2019	FPA+GA	Java	191	SIR	Eclipse, EclEmma	APFD, Exec. time
S19	[70]	2019	ACO	Java	35392	SIR	–	APFD, APFDC
S20	[108]	2019	Whale + ANN	Java	–	–	–	APFD
S21	[86]	2019	CSA	Java	≈ 33	Open source	Eclipse, TestNG and EclEmma	APFD
S22	[68]	2019	ACO	–	6	–	–	APTC
S23	[80]	2019	ABC	–	14	SIR	–	Change coverage
S24	[94]	2019	SFLA	–	10	SIR	–	Cov(x),

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

No.	*Study*	*Year*	*Algorithm*	*Application*	*# of TC*	*Benchmarks*	*Tools*	*Metric*
								Cost(x), Fault(x)
S25	[55]	2019	ACO	MATLAB	-	-	–	–
S26	[85]	2019	CSO	–	15	SIR	–	–
S27	[79]	2019	ABC	Python	500	Private	Selenium IDE	APFD, APFDC
S28	[102]	2019	ABC+GA	–	2086	Private	–	APFD
S29	[84]	2019	CSO	MATLAB	50	–	–	APFD
S30	[12]	2019	FFA	Java	6,336	SIR (Siemens)	NetBeans	APFD, Exec. time
S31	[109]	2019	CSA + ILP	–	10	SIR	Emma, Source monitor, time.pl, Lingo	APFD
S32	[73]	2018	PSO	–	–	GitHub	–	APBC, EET
S33	[53]	2018	ACO	–	–	SIR (Siemens)	–	APSC
S34	[71]	2018	PSO	MATLAB	1,628	Industry (Visual Analytics Benchmark Repository)	–	APFD
S35	[54]	2018	ACO	–	7	–	–	–
S36	[97]	2018	PSO+ICSA	–	–	–	–	Software quality metrics
S37	[11]	2018	ACO	–	–	–	–	–
S38	[90]	2017	Bat	–	–	GitHub	–	APFD
S39	[103]	2017	PSO+GA	Java	30	Private	–	APFD
S40	[75]	2017	PSO	–	1608	Siemens	–	APFD
S41	[57]	2017	ACO	–	8,216	SIR, Google	gcov, GCC	APSC, EET
S42	[93]	2017	GWO	Java	–	Private	Cloud	Exec. Time, Memory Usage (bits)
S43	[98]	2017	FFA+ABC	Java	82	Private	–	APFD
S44	[78]	2016	PSO	–	8	–	–	APFD
S45	[60]	2016	ACO	MATLAB	794	Private	Emma, ant	APFD
S46	[61]	2016	ACO	Perl	24	Private	–	APFD
S47	[65]	2016	ACO	–	6	–	–	APFD
S48	[62]	2016	ACO	Perl	45	Defects4J	–	APFD, PTR
S49	[63]	2016	ACO	Perl	24	Private	–	APFD, PTR

Journal of Theoretical and Applied Information Technology
30th June 2022. Vol.100. No 12
© 2022 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

No.	***Study***	***Year***	***Algorithm***	***Application***	***# of TC***	***Benchmarks***	***Tools***	***Metric***
S50	[82]	2015	GSO	–	8	–	–	APFD
S51	[67]	2015	ACO	–	8	–	–	APFD
S52	[69]	2015	ACO	–	20,000	Industry	–	APFD
S53	[87]	2015	CSA	Java	6	–	MATLAB	–
S54	[74]	2014	PSO	MATLAB	18	–	–	APFD
S55	[66]	2014	ACO	Turbo C++	61	Private	–	–
S56	[59]	2013	ACO	–	8	–	–	APFD
S57	[76]	2008	PSO	–	20	JUnit	–	–