ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

A Comprehensive Review of Metaheuristic and Hybrid Models for Automated Test Case Generation: Techniques, Challenges, and Optimization

Nurudeen Muhammad Bala[1]; Sadiq Umar Abubakar[2]; Buhari Mohammed Abubakar[3]; Musabi Mohammed Shitta[4]; Umar Ahmadu Ardo[5]; Okoro Edith[6]; Abbas Salamatu[7]; Abdulrasheed Badamasuiy[8]; Hauwa Dauda Balami[9]; Bashar Muhammad Arzika[10]

[1] Center for Satellite Technology Development, National Space Research and Development Agency, Abuja, Nigeria

Publication Date: 2026/02/28

Abstract: Software testing ensures quality and reliability of software necessitates effective testing; however, traditional manual testing methods can be time-consuming and inefficient. Automated test case generation (ATCG) has emerged as a promising solution, utilizing metaheuristic algorithms to enhance efficiency, minimize human effort, and improve fault detection. This paper presents an in-depth review of metaheuristic techniques employed in ATCG, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Simulated Annealing, and other evolutionary and swarm-based methods. It also explores hybrid models that integrate multiple metaheuristic strategies or combine them with machine learning and symbolic execution to boost performance. The study addresses key challenges such as scalability, computational complexity, and constraint handling, while discussing optimization strategies to improve the effectiveness, diversity, and efficiency of generated test cases. Additionally, it examines hybrid approaches like the Hybrid Harmony Search and Particle Swarm Optimization (HSPSO) framework, analyzing their advantages, limitations, and potential enhancements. By outlining current trends, open research challenges, and future directions, this paper provides valuable insights for researchers and practitioners in automated software testing.

Keywords: *Automated Test Case Generation (ATCG), Metaheuristic Algorithms, Hybrid Models, Software Testing, Optimization Techniques.*

How to Cite: Nurudeen Muhammad Bala; Sadiq Umar Abubakar; Buhari Mohammed Abubakar; Musabi Mohammed Shitta; Umar Ahmadu Ardo; Okoro Edith; Abbas Salamatu; Abdulrasheed Badamasuiy; Hauwa Dauda Balami; Bashar Muhammad Arzika (2026) A Comprehensive Review of Metaheuristic and Hybrid Models for Automated Test Case Generation: Techniques, Challenges, and Optimization. *International Journal of Innovative Science and Research Technology*, 11(2), 2049-2068. https://doi.org/10.38124/ijisrt/26feb433

I. INTRODUCTION

Software testing is essential to software development because it makes sure that programs fulfil both functional and non-functional requirements while maintaining reliability and security [1]. However, traditional manual test case generation is often time-consuming, error-prone, and lacks scalability [2]. To overcome these challenges, researchers have developed automated test case generation (ATCG) techniques that leverage search-based software testing (SBST) and metaheuristic optimization [3].

Metaheuristic algorithms, inspired by natural and evolutionary processes, offer efficient solutions for complex optimization problems [4]. Among these, Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) are widely adopted due to their ability to explore the extensive search spaces effectively [5]. However, these methods often suffer from slow convergence and suboptimal performance in large-scale software systems [3].

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

As software systems grow in complexity, traditional manual testing methods become increasingly inefficient, requiring significant time and effort while failing to provide comprehensive coverage [7]. In response, automated test case generation (ATCG) has emerged as a more efficient alternative, utilizing optimization techniques to enhance accuracy and scalability [8]. SBST is a widely used ATCG approach that applies metaheuristic optimization algorithms to generate high-quality test cases. GA and PSO, in particular, have demonstrated effectiveness in navigating large search spaces [9]. However, both techniques encounter limitations such as slow convergence and inefficiencies in handling complex software systems [10].

To mitigate these issues, researchers have developed hybrid optimization approaches that combine multiple algorithms to enhance performance [11]. One such approach is the Hybrid Harmony Search and Particle Swarm Optimization (HSPSO) framework, which integrates Harmony Search (HS) and PSO to improve test case generation. HS, inspired by musical improvisation, efficiently explores large solution spaces, while PSO, modeled after swarm intelligence, optimizes test cases through collaboration and information sharing. By combining these strengths, HSPSO enhances test case quality and reduces execution time, making it a promising solution for large-scale software testing.

This paper explores the role of metaheuristic algorithms in automated test case generation, analyzing their strengths and limitations while emphasizing the advantages of hybrid models like HSPSO in addressing modern software testing challenges..

II. SOFTWARE TESTING AND TEST CASES

Software testing is vital across various fields, vulnerable to human errors that could lead to operational failures or even fatalities [1]. Detecting errors early is crucial, as it helps reduce escalating development costs, pinpoint flaws, improve overall quality, and shorten execution time[2]. Prior studies identified two methodologies in software testing literature, such as static and dynamic testing, possess unique approaches [4], [5], [6]argued that static testing verifies without execution, concentrating on early defect identification, whereas dynamic testing assesses application behavior, ensuring expected functionality and performance. While, dynamic testing, particularly in the context of white-box testing, enhances code quality by validating expected behavior and exploring various execution paths, as observed by [7]. However, test case is a critical concept that aligns closely with above testing methodologies.

In the context of test case, the process is integral to both these testing methodologies. For instance, [3], the static testing, the test cases are generated to systematically check code for potential issues like syntax errors, coding standard violations, and possible vulnerabilities. This ensures that many defects are caught early in the development process, aligning with [8] emphasis on early defect identification. Similarly, [9] found that fault localization, and efficient debugging processes contributes to code coverage analysis, reducing defects. In addition, [10] argued that automation, especially in white-box testing, minimizes manual effort, as highlighted by Improving system execution time assures enhanced performance and efficiency, while adaptable solutions drive innovation and competitiveness. Exploring dynamic testing and execution duration enhances software system quality, efficiency, and dependability, as discussed by [11].

In real-time system settings, often operate in critical environments, that must be highly reliable and available. They need to function correctly and be accessible when require [12]. Therefore, testing such a system involves rigorous and comprehensive methodologies to ensure they meet their strict performance and reliability requirements. Priors' studies have identified both functional and non-functional testing activities essential for real-time systems [13], [14]. For example, [15] emphasized the importance of unit testing in real-time systems to validate individual components before integration. They found that early detection of errors through unit testing significantly reduces the cost of fixing defects later in the development cycle. Similarly,[16] highlighted integration testing, where components are combined and tested as a group. While functional testing is essential to ensure that a system behaves as expected in terms of specific operations and functionalities, non-functional testing is more crucial for ensuring that the system performs well under various conditions and meets the broader quality attributes that affect user satisfaction [17].

In the context of non-functional testing which concentrates ensuring reliability, efficiency, and performance within stringent time limits as described by [18]. Similarly, [19] reported that verifying deadline adherence, maintaining stable performance, optimizing resource usage, and managing tasks promptly. Confirming adherence to standards ensures legal compliance[20], while evaluating responsiveness and user-friendliness guarantees a satisfactory user experience [21]. Despite the benefits of non-functional testing in real time environment, the testing process can be labour-intensive and expensive, often surpassing 60% of project expenses as highlighted by [14]. However, automation of testing through a test case generation can ultimately offers cost reduction, efficiency and improvement through streamlined processes for faster test execution [22] . In essence, such a process of creating specific test cases based on various criteria such as requirements, specifications, or user scenarios that are essential for meeting reliability, efficiency, performance, safety, compliance, and user experience standards in critical applications. Therefore, a review of test case generation will be found in the next section.

III. CHALLENGES IN TEST CASE GENERATION

Metaheuristic algorithms have improved test case generation, but challenges remain, including scalability issues, parameter tuning difficulties, and the high computational cost of hybrid approaches [23]. In the early stages of software testing automation, many test case generators relied on meta-heuristic gradient descent algorithms [24], [25]. This approach outlined test case generation as a computationally challenging optimization problem [26]. According to [27], [28], [29],

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

highlighted the critical and time-consuming nature of producing test cases.

Similarly, [1] suggested generating test cases from various sources such as requirements, specifications, source code, or design documentation, particularly for mass-level testing. However, the above method can introduce complexities, such as the need to manage and integrate information from diverse sources, which can be time-consuming and require significant effort to ensure consistency and completeness. Also, [30] improved test planning by using design-based testing to evaluate the application's performance based on requirement specifications and design documentation. However, design-based testing faces challenges such as dependency on accurate and complete design documentation, as errors or omissions can lead to inadequate test cases. Establishing a robust framework also requires significant initial effort and resources, involving time-consuming tasks like creating detailed design documents and aligning them with test cases.

[31] aargued the challenges with test cases in software testing that make generation essential due to insufficient code coverage, which can lead to undetected bugs. [32] report poor design or thoroughness may result in unnoticed defects, and redundant tests could provide no additional value. [33] argued that the under-testing of complex code paths and ignoring performance and scalability issues are also common challenges. Similarly, [34], Software testing metrics encounter challenges in execution time accuracy (system load, asynchronous operations), uncertain cost estimation (fluctuating budgets), requirement coverage complexities (ambiguous needs), dependency management (integration testing issues), and limitations of performance metrics (BCET, the shortest execution time, and WCET, the longest execution time, to explore test scenarios and determine the minimum and maximum execution times of tasks in real-world prediction).

[35] emphasize the importance of coverage and execution time in software testing, particularly for code testing and quality control. Coverage measures how extensively a software product is tested, assessing the proportion of code, requirements, or functionalities exercised by a test suite [36]. Higher coverage generally indicates a lower probability of undetected software defects [12].

Coverage criteria include Statement coverage which calculates the percentage of executable statements that have been tested. Branch coverage measures the percentage of decision points that have been executed. Path coverage determines the percentage of unique code paths that have been traversed. Function coverage assesses the percentage of functions or methods that have been called during testing[37].

Achieving high coverage ensures thorough testing of various parts of the code, thereby reducing the risk of undetected bugs [38]. However, high coverage does not guarantee the absence of defects, as some test cases or scenarios might still be missed [39].

High coverage is important because it indicates that more of the codebase has been tested, potentially uncovering more defects. It provides confidence that the code has been thoroughly tested, contributing to higher software quality. High coverage helps maintain quality over time by ensuring new changes are tested [40]. However, achieving 100% coverage is often impractical due to complex code paths and conditions. High coverage does not necessarily mean tests are effective; test quality is crucial. Instrumenting code for coverage analysis can introduce performance overhead, affecting execution time [41].

Execution time, referring to the duration needed to run code or a test suite, is crucial in software testing for various reasons[42]. It is utilized in performance testing to evaluate software responsiveness, throughput, and efficiency under diverse loads and conditions [14]. Additionally, it serves as a benchmarking tool to compare the performance of different algorithms, implementations, or software versions. Analyzing execution time aids in identifying performance bottlenecks and optimization opportunities within the codebase, which is essential for efficient resource allocation, including determining hardware requirements[42].

However, execution time issues can adversely affect user experience and efficiency. Longer execution times can result in slower convergence, increased time consumption, and higher costs[43]. Optimizing execution time is complex due to dependencies and the need to balance factors like memory usage and computational power. Moreover, both coverage and execution time are indispensable for high-quality software while comprehensive coverage ensures robustness and correctness, efficient execution time ensures performance and scalability, often involving automated testing and optimization techniques [44].

IV. MANUAL TEST CASE GENERATION

Prior studies [45], [46] argue that manual test case generation have extensively explored various methodologies and approaches within software testing, highlighting its foundational role in the field. However, while manual test case generation has historically been crucial, it is increasingly seen as labour-intensive and potentially error-prone compared to automated methods. For example, [47] highlighted the challenges and complexities involved in manual test case generation, advocating structured approaches to maintain effectiveness, while [48] emphasized the importance of thoroughness and precision in manual test case generation, particularly in early software development practices.

Critically, scalability issues in large-scale projects have been identified as a significant limitation of manual test case generation, as noted by [49], who argued for automated testing frameworks to enhance efficiency and coverage, highlighting those manual processes may introduce biases and inconsistencies due to human error, which automated testing can mitigate.

Despite these challenges, several studies [24], [25], [31] also have highlighted some drawbacks of manual testing, such as time-consuming, proneness to errors, which complicates defect detection. For instance, [50] emphasized the significant

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

time and effort required for manual test case generation, leading to a growing interest in semi-automated or fully automated methods based on specified requirements and design documents [51].

Moreover, [52]highlighted that automated test case generation not only reduces defects and errors but also saves time and effort compared to manual testing. While manual testing provides a perspective from an end-user standpoint, assessing software performance and uncovering bugs, it requires substantial resources. However, full automation remains impractical in some contexts, underscoring the continued necessity of manual testing as a crucial preliminary step before automation implementation [53].

In real-time systems, manual testing often falls short because it cannot consistently replicate precise timing and performance requirements, making automated testing essential for simulating and measuring exact timings and responses. The Instrument PUT, referring to the program or system undergoing testing, is classified as a real-time system based on specific requirements such as strict timing constraints, deterministic behavior, immediate processing needs, and priority task scheduling, using predefined inputs or test cases as benchmarks for expected performance across various conditions. Hence, automated test case generation is discussed in the following section to address these specific requirements.

V. AUTOMATED TEST CASES GENERATION

Automatic test cases are essential in software development, aiding in fault assessment and resolution, as highlighted by [54]. While generating test cases can be time-consuming, various tools streamline the process, enhancing its efficiency. Factors such as execution time and path coverage significantly influence testing efficiency and effectiveness, as noted by[55].

Further studies [56], [57] highlight the importance of optimizing execution time to improve fault detection rates and enhance overall thoroughness of the test suite, considering resource constraints and minimizing computational overhead. Additionally, [58] emphasizes that lengthy test cases can impede thorough testing, potentially leaving some software areas inadequately tested. Optimizing execution time not only improves path coverage but also enhances fault detection rates and overall test suite thoroughness, as observed by [24], [59].

[60] discuss how automated testing aims to generate test cases that meet specific coverage criteria, ensuring efficiency, effectiveness, and comprehensive fault detection, while also enhancing software quality and optimizing testing processes through considerations of execution time and path coverage.

This study focuses on automatic test case generation to simultaneously minimize BCET, ACET, and WCET. The research will employ a proposed HSPSO framework, incorporating a fitness function and its algorithms as part of the theoretical development and implementation. Recent studies will serve as references to validate and compare results, specifically addressing problems related to test case generation in execution time.

VI. TECHNIQUE FOR AUTOMATED TEST CASE GENERATION

Automated test case generation aims to maximize coverage of test goals in the SUT based on selected adequacy criteria such as branch coverage[61]To achieve comprehensive coverage, test cases need to be designed with the maximum potential for detecting errors or bugs [62], ensuring their effectiveness is measured by the number of reported defects or errors.

[63] identify test cases generation that meet specific criteria, aiding in the identification of potential issues. Automated test case generation streamlines the process, reducing the time and cost associated with developing test cases. [64] utilize automated heuristics for generating test cases or data. Among these methods, SBST stands out, offering a versatile approach for various test case creation objectives, testing levels, and other features. SBST entails generating new test cases from initially random ones to minimize and eliminate their distance from undiscovered targets. Meta-heuristics like GA play a crucial role in this process.

According to [65] , SBST is useful for identifying inputs causing long or short software execution times. It can generate test data, prioritize test cases, reduce test suites, optimize test oracles, lower the cost of using human oracles, test service-oriented architectures, build test suites for interaction testing, and validate real-time properties [66].

Due to their heuristic nature, SBST procedures require empirical examination to assess their effectiveness and scalability in achieving test goals. However, there's considerable heterogeneity in the literature regarding methods used to empirically study SBST strategies[2]. In terms of test case execution time, the literature offers a thorough description of several evolutionary and swarm intelligence approaches like GA, ABC, HS, ACO, PSO, FA, and CS[61], [67], [68], [69], [70].

A. Single Techniques

Metaheuristics have been found to be the most efficient way for tackling many difficult optimization problems [71]. According to [72] metaheuristics is a computational technique where the problem is iteratively optimized to improve the proposed solution. Because they may achieve good outcomes depending on execution times, many metaheuristic algorithms have been created to help logistics planners carry out their daily operations in a realistic manner [73]. When using single-based algorithms, it ends the search after just one starting point. In order to enhance their solution, it will investigate the areas around it [71]. Thus, Optimization algorithms are discussed in details.

- *Genetic Algorithm (GA*):

Fraser and Bremermann are credited with conceptualizing the Genetic Algorithm (GA), a meta-heuristic method devised by Holland in 1975 [74]. GA finds applications in search and

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

optimization problems. Initially, populations are randomly created, and then GA evolves them using genetic operators like mutation and crossover. Fitness assessment selects the fittest members of the evolving population, crucially relying on a well-defined fitness function to validate GA's efficacy and efficiency.

Population diversity, crossover probability, and mutation probabilities significantly impact the quality of the best answer. Several studies, including those by [75], [76], have employed this fitness function. The process continues until halting conditions are met. In software testing, GA is utilized for test data generation, mutant optimization, and test data minimization [77]. Despite its diverse applications in SBMT, research on GA has been conducted in various approaches

In the early 1990s, GA was utilized for test case generation,[78] applied mutation analysis to improve test cases for integration testing with GA. (Khari & Kumar, 2019) optimized C# parser test cases using GA. [76], [80] have proposed a GA model to reveal flaws by breaking the target program into small parts, effectively reducing the cost and execution time of test cases.

[81] employed GA to reduce mutants, improving WS-BPEL compositions' initial test cases' quality. Additionally, [82] optimized test cases using mutation analysis and assessed them across 10 open-source libraries. [83] introduced a better grouping technique to cover various paths efficiently using GA, reducing search space for test data.

[84] have proposed test case generation to attain maximum path coverage, it can be challenging to emphasise covering a crucial path in a test resource. [85], used GA for automated test data generation, satisfying decision and condition coverage criteria. [86] combined local and global test data optimization using GA for test data generation. In this study by [87] have proposed GA for data flow test case generation, reducing test cases but with no effect on coverage percentage.

According to [59], designed a global search method using GA for search-based optimization techniques, demonstrating its effectiveness since 1992 in structural software testing. [88] prioritized fault-based regression test cases using GA, achieving total fault coverage with minimal execution time. [29] employed GA to cover branches, proposing a branch distance fitness function based on similarity. [89] compared fitness functions with GA for path coverage, effectively creating test cases for the goal path while determining branch distance fitness.

According to [90] have developed test sequences covering transitions and classes using GA. [86] combined GA & PSO to generate fewer but distinct test scenarios. [91] proposed a GA-based Whole Suite (WS) generation approach, generating all test cases at once. [92] suggested combining scatter search with GA for path prediction in NLP programs.

Overall, genetic algorithms prove useful for automating software testing [93], outperforming local and thorough search methods, despite challenges such as excessive iterations and early convergence [94]. However, determining when to halt the search remains a random choice.

- *Ant Colony Optimization (ACO):*

Ant behavior was first studied in the early 1970s. The behavior was then refined into an optimization technique by [95]. After then, Dorigo published more studies on ant colony optimization, including [95]. ACO is a metaheuristic method that can be used for a variety of issues. The idea behind it is to find the quickest path between food sources and colonies.

ACO is a metaheuristic technique inspired by the stigmaria behavior of ants in finding the quickest path between food sources and colonies [96]. Since its introduction, ACO has become one of the most commonly used metaheuristics for solving combinatorial problems. Many researchers have explored basic versions of ACO and introduced updates to achieve better results.

Studies [97], [98] have applied ACO to maximize code coverage among test cases by mimicking ants' foraging behavior. The experimental findings have demonstrated that this approach can outperform other cutting-edge approaches in terms of solution quality and search effectiveness. [99] have developed ACO to prioritize test cases based on identified defects, execution time, and fault severity, efficiently optimizing test case orderings. According to [100] ACO has been proposed for various testing tasks, including test case selection to minimize execution time and maximize fault coverage and automatic test data generation in mutation testing.

[101] suggested ACO for test case prioritization and selection, yielding solutions close to optimum. Enhanced ACO algorithms have also been introduced to address limitations and improve search effectiveness [102].

[103] have suggested new methods built on ACO. The results demonstrate that this method's test vectors outperform other simulated-based ATPGs in terms of highest fault coverage in the shortest amount of time for a number of benchmark circuits. While ACO has shown promise, it has some drawbacks, such as slower convergence performance for local minimums due to the lack of operators like crossover found in GA [104]. Comparatively, ACO has demonstrated better effectiveness and capability in locating optimal solutions than GA [105]. However, further research is needed to explore ACO's performance in test case generation, filling a gap in the literature

- *Harmony Search Optimization (HS):*

HS algorithm was developed by Geem et al. (2001) to tackle practical optimization problems, drawing inspiration from the process of seeking better musical harmony. It operates on the principle of musician seeking pleasing harmony, the algorithm continually adjusts parameters to achieve optimal solutions. The HS algorithm is versatile, capable of solving both discrete and continuous problems, and comprises two optional and three mandatory control settings [106].

[107] have further explored the HS algorithm's potential, emphasizing its adaptability to various problem types. The

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

algorithm retains its original principles while incorporating dynamic parameter adjustments based on previous values, enhancing its effectiveness. Additionally, alternative iterations of the HS method, such as the global-best harmony search (GHS) proposed by [108], offer simplified pitch adjustment processes to improve efficiency.

[109] have proposed variations of the HS algorithm, such as the novel global harmony search (NGHS) and the dynamic control parameters harmony search (NDHS), introduce innovative features like genetic mutation operations and dynamic parameter adaptation systems. [110] propose an improve variations of the algorithm's performance and versatility in solving optimization problems.

The original HS algorithm's memory consideration, pitch adjustment, and random selection, while dynamically updating the PAR and BW parameter values is shown in eqn (1)

$$PAR_{(t)} = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{maxNI} . t \qquad (1)$$

Where is the pitch adjustment rate in generation t, are the maximum and minimum adjustment rate respectively. Following, the generation number dynamically modifies the first parameter PAR is shown in eqn (2)

$$PAR_{(t)=} PAR_{(min)} + \left(PAR_{(maX)} - PAR_{(min)} \right) . \mathrm{e} - kt_{(maxN)} \qquad (2)$$

Furthermore, the learning automata-based harmony search (LAHS) proposed by [111], [112] introduces a learning mechanism that adjusts parameters based on feedback signals, enhancing adaptability and efficiency.

Despite these advancements, theoretical analyses of HS variations remain limited [113] compared to other population-based algorithms like GA and ACO. Moreover, research on the effectiveness of HS in test case generation is scarce [114], indicating a gap in the literature that warrants further exploration. Future studies should focus on examining the probabilistic convergence qualities of HS and providing comprehensive population convergence proofs to enhance its theoretical foundation.

- *Firefly Algorithm (FA):*

[115] first proposed FA, a metaheuristic algorithm based on swarms inspired by fireflies. These insects possess the ability to produce light through a biological mechanism called bioluminescence [112], [116]. However, its precise intended application remains unknown [115] . The two primary functions of fireflies are the attraction of potential prey and the search for mates. Additionally, the flashing light serves to shield them from other predators [106], [117]

When developing the firefly algorithm, three fundamental assumptions or rules were idealized [118]. Firstly, all fireflies are assumed to be unisex. Secondly, the attractiveness of a firefly is inversely correlated with the brightness of its flash. Thirdly, brightness diminishes as distance increases [119].

According to[116]has propose FA for generating and optimizing random test cases. The results demonstrate that this algorithm is capable of efficiently and effectively generating suitable automated test cases and test data. [120] have propose a test case minimization method performed on UML state charts to produce optimal test cases while considering higher coverage criteria. The results show successful coverage of various types of test coverage, including all states, all transitions, all transition pairs, and all-one-loop paths, while minimizing the number of test cases.

[121] also propose FA to solve high-dimensional problems, with results showing accuracy in finding optimal solutions and fast convergence speeds. Similarly, [122] have propose noisy non-linear continuous mathematical models to find optimal solutions, with FA showing better performance for higher levels of noise. [123] have developed a new version of FA that uses a Gaussian distribution to speed up convergence, resulting in more accurate and promising experimental results compared to traditional FA.

Likewise, [124] have develop a framework for analysing the inefficiency and convergence studies of meta-heuristics, examining the effects of Levy flight and Gaussian random walk on meta-heuristic outcomes. Similarly, [125], uses Levy flight movement strategy as inspiration, develop a new meta-heuristic firefly algorithm that will outperform PSO and GA in terms of efficacy and success rates [115] analysed the convergence and effectiveness for various meta-heuristics, including cuckoo search, FA, swarm intelligence, Levy flight, and random walk.

[126] have proposed the combination of chaotic maps with the firefly algorithm to improve convergence, enabling effective escape from local optima. In [127], a parallelized firefly algorithm is tested using common benchmark functions, resulting in a quick and less time-consuming algorithm. However, it is only applicable when considering more than one population generation. FA has undergone numerous changes and advancements in the past.

FA is useful for synchronizing and successfully locating both global and local solutions, as well as for simultaneous implementation due to the independence of the fireflies. However, it is essential to adjust parameters such as randomization, attractiveness, and absorption coefficient [128]. Hence, no research done on the performance of FA in test case generation, this research will fill the gap.

- *Cuckoo Search (CS):*

[129] highlighted the swarm-based metaheuristic search algorithm CS for global optimization that mimics cuckoo behavior. The bird removes the host bird's eggs in addition to laying its own eggs in the nest of another species. The egg will be well-cared for if it resembles the host bird's egg and the latter is unable to tell the difference between the two [23]. The host bird, however, will either abandon the nest or push the eggs outside if it learns they are not its own.

According to [130], the chances of being found by the host bird are between 0 and 1. [131] investigated the spring and welded bean designs and contrasted the outcomes with CS and its derivatives have been effectively investigated and implemented by numerous researchers in a variety of test case

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

generation. [54] have develop the cuckoo search approach for selecting the best test cases for software testing. This approach has been compared to the hill climbing algorithm, the higher performance. The method's computational complexity must be improved.

[132] have suggested using the Cuckoo algorithm in software testing to make the process less complicated. The results outperformed the Firefly algorithm in terms of time and cost. The method's efficacy has to be increased. [133] have suggested using the cuckoo search algorithm to identify the best software testing test case. The performance of this approach is better than that of the hill climbing algorithm. The method's computational complexity must be decreased.

The Pairwise Cuckoo Search strategy (PairCS), a new pairwise approach based on Cuckoo Search, has been developed by [133]. The outcomes using PairCS have been encouraging as they were able to obtain good test sizes for the majority of the configurations taken into consideration and exceed the current nature-inspired-based as well as other computational-based tactics.

[134] has developed CS to reduce test case quantity in configuration-aware structural testing, showcasing its effectiveness in both combinatorial optimization and software testing. In [129] a heuristic method combining CS, Lévy flight, and TS have been proposed for test data generation. This method demonstrates superior performance compared to other meta-heuristic techniques in terms of node coverage, iterations, and parameter counts. CS stands out among search algorithms like GA, PSO, ACO, and ABC due to its efficiency and global convergence, making it a prominent choice for researchers in the testing industry. However, further research on Cuckoo Search's performance in test case generation is needed to address existing gaps in the field.

- *Particle Swarm Optimization (PSO):*

[135], develop PSO, a swarm intelligence-based optimization method inspired by bird flocking behavior. Several studies [136], [137], [138], highlight PSO operations with a population of potential solutions, each assigned a randomly generated velocity, these solutions referred to as particles, move in hyperspace to find the optimal answer through social and individual behavior. [139], presented the movement of particles, the best solution guides others towards better positions, facilitating quick convergence. [140], introduce a PSO simpler initialization compared to genetic algorithms GAs and demonstrates efficiency in finding the ideal solution when termination requirements are met.

Further studies [141], [142] have proposed various modifications to PSO, addressing parameter adjustment, neighbourhood topology, and sub-swarm topology. [143], [144] have propose a PSO method paired with metamorphic relations to reduce the cost of test data generation. [145] introduces PSAPF, combining PSO and APF to handle high dimensionality, albeit with a high evaluation cost. ([146] demonstrate PSO's superiority in covering critical paths compared to branch distance fitness functions. [147] have propsed prioritization of test cases efficiently using PSO for regression testing.

Similarly, [83], [148]have applied PSO's effectiveness in generating test data compared to GA. [136], [137], [138] have introduce APSO and GPSCA, respectively to improve test data production efficiency. [149] have proposed RAPSO, which outperforms APSO and GPSCA in convergence speed. [150], [151] has developed PSO to create test cases while fully covering program codes. Despite improvements, PSO's performance in high-dimensional search spaces remains limited. Hence, no research done on the performance of PSO in test case generation, this research will fill the gap.

- *Artificial Bee Colony (ABC):*

A swarm-based optimization method called Artificial Bee Colony (ABC), proposed by [152], mimics the behaviour of bees to solve optimization problems. The colony includes worker, employed, and scout bees, simulating the foraging process. ABC has been applied in software testing, offering efficiency in terms of time and cost [153].

(Lakshminarayana & Sureshkumar, 2020), introduced ABC to minimize execution time and optimize clusters for datasets, showing improved efficiency compared to PSO. [154] proposed ABC for software test suite management and automatic pathway development, accelerating test suite optimization.

[155] utilized an ABC-based strategy with symbolic execution, yielding satisfactory results, despite drawbacks such as slow convergence due to the lack of crossover operators like GA. Additionally, ABC has not been applied to the test case execution time problem, highlighting a research gap that needs to be addressed.

- *Hill Climbing Algorithm (HC):*

One meta-heuristic search method that attempts to enhance the current solution by investigating all of its neighbors in the search space is hill climbing [156]. The HC algorithm is applied to functional and non-functional empirical values; sometimes, these empirical values can be missing. It produces quick results and is a straightforward procedure that starts with the selection of the first candidate solution at random in the initialization stage. After that, all of its neighbors are searched and assessed until no better answer can be located.

[81] highlighted the major drawback of this technique as local convergence, the search includes only the neighborhood space. (Gutjahr & Montemanni, 2019) has developed requirements for three properties of convergence: convergence to a global optimum, convergence in the distribution of the generated solutions, and asymptotic independence of the initial conditions; they also describe the pace of convergence.

[157] have proposed the use of HC with GA to handle the influx of numerous higher-order mutants produced from lower-order mutants. In another study, (R. A. Silva et al., 2017) have employed the Alternating Variable Method, a HC version, to provide test data for mutation testing with a better fitness function. (Jana et al., 2018; Pizzoleto et al., 2019) have

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

proposed a local convergence as the main flaw of this technique because the search only covers the neighbourhood space. Hence, further studies are needed as no research has been done on the performance of HC in test case generation; this research will fill the gap.

- *Simulated Annealing (SA):*

Simulated annealing (SA) is a simple and reliable method that, with a significant reduction in calculation time, offers effective solutions to single- and multiple-objective optimisation problems. It is a way to get a Pareto set of solutions for multi-objective optimisation issues as well as an optimal solution to a single-objective optimisation problem [158].

Previous studies [9], [159]have demonstrated that a simulating annealing model could be utilized to solve optimization problems where the energy of the metal's states is the goal function to be minimized. Nowadays, SA is one of many heuristic methods that aim to provide a good, but not always ideal, solution. It can easily handle mixed discrete and continuous problems and is easy to formulate.

Prior research [160], [161] highlighted the application of SA algorithm in solving optimization problems. For example, [162] have employed SA to address the issue of machine loading in the Flexible Manufacturing System (FMS) in an effort to reduce system imbalance. Another study by [163] has establish the routes of vehicles and delivery quantities for each retailer in a one-warehouse multi-retailer distribution system, the multi-period multi-stop transportation planning problem has been taken into consideration.

[164] have suggested lowering the overall product delivery transportation distance over the planning horizon in order to meet retailer requests. [151] defined a priority list as any permutation of a set of symbols in which the symbol of each job appears exactly as many times as its operations. They have also suggested a two-stage heuristic algorithm based on SA as an alternative for large problems that cannot be solved by the column generation algorithm in a reasonable amount of time. Every potential plan evolves in the same manner, and every workable schedule may be inherently connected to a list of priorities.

As a result, priority lists serve as an example of timetables that are realistic and prevent the issues that arise from unrealistic schedules. [165] has introduced a priority list-based Monte Carlo SA implementation that competes with the top schedule-based SA and tabu search strategies at the time. The modified insertion scheme (MIS), a proposed perturbation method, has been used to create new task sequences that have been used in the proposed SA algorithm to get a close to global optimal result.

The SA algorithm with the suggested MIS significantly reduced system imbalance. The FMS machine loading issue has presented its other applications [166], part classification [167], resource constrained project scheduling [168], etc. In another study [96] proposed a two stage SA algorithm to generate water distribution pump schedules quickly enough for everyday use. The model was created based on autonomous interaction with a hydraulic simulator, which addresses nonlinear impacts from changing reservoir levels.

Similar study by [169] have employed it to cluster database tuples. They have tried to apply SA in circuit board layout design, and their experience suggests that doing so could be helpful for clustering tuples in a database to improve query response time. It has recently been used for pattern recognition, object categorization, and optimization in addition to optimization [170].

[171] have suggested a hybrid approach to pattern recognition that uses evolutionary algorithms and quick SA to identify patterns that have been transformed by rotation, scaling, or translation alone or in combination. The object recognition issue has been defined as the matching of a global model graph with an input scene graph that either represents a single object or numerous overlapping objects.

[172] Analyse simulated annealing at a fixed temperature. [173] demonstrates how employing random restarts can increase convergence rates and talks about broad acceptance probabilities. For thorough analyses of the convergence findings of simulated annealing. [174]. Another study by [175] has proposed a fresh technique for altering the parameters of multi-objective optimization algorithms online.

The SA-based method has been used to optimize the parameters. Measure C and generalization distance have been taken into consideration as objective functions. They have undergone optimization to achieve the best algorithmic parameter values. Any multi-objective optimization algorithm can be used to apply the described technique. When SA is compared against other methods, the computational results in the aforementioned literature have mixed findings.

[40] proposed SA to resolve the path testing issue. In order to cover the goal path and shorten execution time, their work has made use of the GA, SA, and PSO algorithms. Because SA can swiftly produce test data that covers the target path, experimental results demonstrate that SA is the ideal evolutionary algorithm for path testing.

Similarly, [176] presents GA and SA, compare the effectiveness of each algorithm based on how many generations are needed to obtain the desired result and how long it takes to create the test cases. The findings show that GA is preferable to SA since more work needs to be done on SA's test case development than on other coverage criteria like branch coverage and data flow testing.

Therefore, it is too soon to evaluate SA's performance. By utilising an optimal annealing schedule, other algorithmic parameters, and combining SA with another algorithm, multi-objective multi-objective algorithms can perform better. Consequently, utilising SA is limited in the domain of optimising testing of non-functional software system properties. Hence, there is need for further studies.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

Table 1 shows the summary of the strength and weaknesses based on evolutionary computations and Swarm intelligence-based algorithms include genetic algorithm (GA), ant colony optimization (ACO), harmony search (HS), firefly algorithm (FA), cuckoo search (CS), particle swarm optimization (PSO), artificial bee colony (ABC), Hill Climbing Algorithm (HC), Simulating Annealing (SA).

The majority of algorithms have benefits and drawbacks. One of the earliest Evolutionary Algorithms (EAs) that has been effectively used in the development of test cases, for instance, is GA. It can provide numerous solutions to a problem and resolve any optimisation issue that can be specified through chromosomal encoding.

Nonetheless, it requires algorithm-specific factors such as mutation, crossover, and reproduction for optimal tuning. Moreover, there is no assurance that a genetic algorithm will identify a global optimum, and comprehending the intricacies of simulation-optimization linkages requires considerable work. ACO has a high convergence speed and is robust enough to deal with complex, non-linear, and non-uniform situations. Nevertheless, the computation of a problem is affected when it is of the explicit or implicit stochastic type. Additionally, it requires tweaking factors like evaporation, metaheuristic data, and relative pheromone trails [177]

HS has fewer mathematical requirements and does not require the initial value to set the decision variables, although having more factors like memory size and pitch adjustment. In addition, the speed at which the memory and nearby values are selected matters [178].

FA is helpful in synchronizing and successfully locating both global and local solutions. Because separate fireflies can operate independently, FA is helpful for parallel implementation. According to [121], the randomization parameter, attraction, and absorption coefficient all need to be tuned.

Levy flights, a technique used by CS, improve the exploration of the search space. An effective and universal convergence solution is offered by CS.

PSO is easy to code, offers quick convergence, has a memory-like quality, and has a low computational cost. However, it is necessary to fine-tune variables like inertia weight, social, and cognitive characteristics. The method can produce a global solution if the parameters are properly adjusted [179].

ABC is adaptable, straightforward, durable, simple to use, and capable of conducting a global search. However, it involves modifying parameters like scout, onlooker, and employed bees and is rather slow in sequential processing [180].

Table 1 Summary of Strength and Weaknesses of Single Algorith

Authors	Algorithms	Strengths	Weaknesses
[181]	Genetic algorithm (GA)	It is capable of solving any optimisation problem that can be represented using chromosomal encoding. It can provide numerous solutions for resolving a problem.	Algorithm-specific characteristics like mutation, crossover, and reproduction must be correctly tuned. The discovery of a global optimum using a genetic algorithm is not guaranteed.
[182]	Ant Colony Optimization (ACO)	It has the strength to handle challenging, non-uniform, non-linear problems. It can converge in a short time.	When a situation is stochastic either explicitly or implicitly, its computation is altered. Evaporation, heuristic data, and relative pheromone trails are a few tuning parameters that are required.
[180]	Artificial Bee Colony (ABC)	It is adaptable, straightforward, durable, modest to use, and capable of doing a global search. It can investigate local search.	Sequential processing is quite sluggish. It needs to be tuned using factors like scout, onlooker, and employed bees.
[183]	Particle Swarm Optimization (PSO)	It needs minimal computational effort and provides fast convergence. Its quality is depending on memory.	It is necessary to tune variables like inertia weight, social, and cognitive characteristics. The population is not spread out widely enough.
[58]	Harmony Search (HS)	HS has fewer mathematical requirements and does not require the starting value to set the decision variables.	Pitch adjustment and memory size are among the many variables required. Furthermore, the rate at which the memory and surrounding is important.
[115]	Firefly Algorithm (FA)	It is helpful in synchronising and successfully identifying both global and local solutions. Because separate fireflies can function independently, FA is helpful for parallel implementation.	It needs the randomization parameter to be tuned, as well as attraction and absorption coefficient.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

[129]	Cuckoo Search (CS)	It utilises Levy flights, a technique that facilitates exploration of the search space. CS provides an efficient global convergence solution.	The findings are not particularly sensitive to these settings, and it only needs a few tuning parameters, such the probability factor.
[184]	Hill Climbing (HC)	It requires much less conditions than other search algorithms, it is good for finding local minima/maxima efficiently and its good option in optimizing problems when limited to computational power.	Due to the possibility of becoming stuck at the local maximum, it is unable to locate the global optimal (best possible) solution and never moves in the direction of a lower value that is definitely incomplete.
[185]	Simulating Annealling (SA)	it accepts solutions that are poor compared to the best candidate, it is simple to implement and has the potential to find a global optimal even after a local minimum has been achieved.	It can be a time-consuming procedure, depending on which materials are being annealed. Since it is metaheuristic and always possible to get stuck at local minima, several parameters need to be adjusted.

B. Hybrid Metaheuristic Techniques

A crucial component of software testing, which aims to ensure the quality and dependability of software systems, is the creation of software test cases for testing a system's non-functional qualities [186]. In order to automate the process of creating test cases and, consequently, provide solutions for a more efficient testing procedure, metaheuristic search techniques have become widely used. A metaheuristic is an advanced approach or heuristic that looks for, creates, or selects a heuristic that might offer a good enough solution to an optimisation problem [128].

The main objective is to illustrate the most advanced search-based metaheuristic optimisation methods used to evaluate test case execution time[128]. Employing search-based approaches or meta-heuristics with a particular fitness function to direct the search towards a possibly viable solution inside a search space, test data production can be automated [64].

Combinations of two or more algorithms that cooperate and enhance one another to produce a beneficial synergy are known as hybrid algorithms[187]. These algorithms are often referred to as hybrid metaheuristics [188].

Prior literature examines the existing research on hybrid algorithms in the context of software test case generation. PSO is a common example of a hybrid algorithm [189] and combining it with other auxiliary search methods appears to significantly enhance its performance [190]. Genetic algorithm hybrids, which combine genetic operators with other techniques, are the most extensively researched. PSO has used genetic operators including crossover, mutation, and selection to generate better candidates [191].

One prominent hybrid algorithm that has been widely studied is the Genetic Algorithm-Particle Swarm Optimization (GA-PSO) approach [192] combining GA and PSO to generate test cases for object-oriented software. Their results demonstrated that the hybrid approach outperformed traditional methods in terms of achieving higher coverage and detecting more faults.

Although, [192] did not only requires the selection and fine-tuning of several parameters, such as population size, mutation rate, crossover rate, swarm size, inertia weight, and acceleration coefficients. But finding of this approach shows the optimal values for the above mention parameters can be a challenging task and might require significant experimentation and expertise.

In [193], the Hybrid Genetic Particle Swarm Technique Algorithm (HGPSTA) was presented, which combines the strengths of PSO and GA with a novel multi-objective fitness function to produce test cases that are effective in terms of the number of generations. Experimental results show HGPSTA compared with GA and PSO performs better and achieves 100% coverage in a smaller number of generations. However, the search functions are iterated for sufficient number of generations.

Hybrid Ant Colony Genetic Algorithm (HACGA) has been proposed in similar studies [194] to maximise test case effectiveness and prevent the issue of becoming trapped in local optima. The result reduces the total time needed to execute a test case and outperforms conventional methods.

In [195] have proposed Cuckoo Search and Bee Colony Algorithm (CSBCA) to provide path coverage and optimise test cases in the shortest amount of execution time. The results demonstrate that, in comparison to PSO, CS, FA, and BCA, test cases and test data are maximised with the least amount of time and iterations, and have a higher fitness function value and path coverage generation.

According to [196] have proposed a hybrid BCA to optimize the test cases and generation of path coverage within the minimal execution time. This gave better result in comparison with particle swarm and Bee colony algorithm. HBCA system took less time to choose the best test path and it is more capable, reliable for the development of software. The method was unable to enhance the test case or test data generation for large programs. The approach consumed large amount of time.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

Another hybrid algorithm, Genetic Algorithm-Ant Colony Optimization (GA-ACO) has also been investigated by [197] for generating test cases in the context of service-oriented systems. The hybrid algorithm utilized the global search capability of GA and the local search capability of ACO to enhance the diversity and effectiveness of generated test cases. The experimental results indicated that the GA-ACO approach yielded higher coverage and fault detection rates compared to traditional methods. Although, this hybrid approach can be computationally expensive, especially for large-scale software systems with a high number of test cases and complex search spaces. The execution time and resource requirements can increase significantly, impacting the overall efficiency of the testing process. In addition, Faust et al. did not show the complexity of search space. Moreover, they did not explore the execution time and resource requirements.

Similarly, [198] has propose the utilization of ant colony optimization (ACO) and genetic algorithm (GA) to improve the fault detection rate. Previous research has focused on single objective solutions, but this work combines ACO and GA to offer a multi-objective solution. Experimental evaluations considering test case quantity, iterations, and ant traversal path demonstrate that the proposed model outperforms existing methods, achieving high fault detection rates.

Overall, the study presents an effective approach for test case prioritization in regression testing, yielding superior results compared to current approaches. The adoption of a real-time TCP dataset, and evaluation was not considered in this approach. Hence the need for more research to investigated other hybrid such as HSPSO.

Additionally, a hybrid algorithm combining GA and Tabu Search (GA-TS) has been explored. For example, [199] have presented a GA-TS-based approach for test case generation in software product lines. The hybrid algorithm aimed to effectively explore the large search space and improve the efficiency of test case generation. The experimental analysis showed that the GA-TS strategy was superior in terms of coverage and fault detection rates. This approach has limit as difficulty in handling constraints such as resource constraints, and dependencies among test cases. This limitation can impact its ability to generate test cases that adequately cover the specific non-functional properties of the software under test. Moreover, in order to minimize this limitation another hybrid such as HSPSO should be utilized in test case generation of execution time of software system.

The aim of this study is to propose a hybrid algorithm that will be assisted in resolving the issue of time consumption for generating test cases in software testing especially when associated with poor requirement gathering. However, further study is required to determine the performance and structure of the presented algorithms to outperform the HGPSTA and PSABC especially when involving more complex programs. However, this study proposes the combination of HS and PSO named HSPSO which has strengths of both the algorithms. The proposed HSPSO algorithm will have advantages of both the algorithms which will enable the algorithm to optimized test case with a less execution time, demonstrate fast convergence and avoid local optima problem.

Table 2 shows the summary of strength and weaknesses based on evolutionary computations and Swarm intelligence-based algorithms include HGPSTA, PSABC, HACGA, CSBCA, GA-PSO, GA-ACO and GA-TS respectively.

Table 2 Summary of Strength and Weaknesses of Hybrid Algorithms

Authors	Algorithms	Strengths	Weaknesses
[200]	Hybrid Genetic Particle Swarm Technique Algorithm (HGPSTA)	It Coverage in less number of generations.	the search functions are iterated for sufficient number of generations
[201]	Particle Swarm Artificial Bee Colony (PSABC)	This approach workable, broad applicable, very flexible, robust against initiation and permits modification and knowledge introduction.	time to convergence is uncertain.
[202]	Hybrid Ant Colony Genetic Algorithm (HACGA)	It has the ability to shorten the total time needed to execute test cases.	It gets stuck in the local optima
[203]	Cuckoo Search and Bee Colony Algorithm (CSBCA)	It has a higher fitness function value and generation of path coverage	It took a lot of time to implement.
[204]	Genetic Algorithm-Particle Swarm Optimization (GA-PSO)	It can produce an effective, high-quality solution with more consistent convergence in a shorter amount of computational time.	It requires the selection and fine-tuning of several parameters as well as significant experimentation and expertise.
[205]	Genetic Algorithm-Ant Colony Optimization (GA-ACO)	It can search the subspace, locate feasible solutions before departing from local optima, and avoid premature convergence.	The approach is computationally expensive, especially for large-scale software systems with a high number of test cases and complex search spaces.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

[206]	GA and Tabu Search (GA-TS)	It has a superiority approach in terms of coverage and fault detection rates.	This approach has limit in handling resource constraints, and dependencies among test cases

FUTURE DIRECTIONS

Despite significant progress in both single and hybrid algorithms such as GA, ABC, HS, PSO, HGPSTA, PSABC, HACGA, CSBCA, GA-PSO, GA-ACO, and GA-TS software testing remains a challenging, time-consuming, and expensive process that often fails to achieve optimal results. Addressing issues related to test case execution time and path coverage is crucial. This study introduces a hybrid approach aimed at improving these aspects in software systems. Future research should focus on adaptive metaheuristic techniques that dynamically adjust algorithm parameters based on software complexity [207]. Moreover, integrating machine learning into metaheuristic optimization could further enhance both efficiency and accuracy [208].

VII. CONCLUSION

The literature emphasizes the importance of leveraging computing advancements to address practical challenges. Metaheuristic algorithms, known for their broad applications, play a key role in simplifying test case generation by utilizing search populations to explore solution spaces in parallel.

Hybrid algorithms such as HGPSTA, PSABC, HACGA, CSBCA, GA-PSO, GA-ACO, and GA-TS combine different optimization strategies to enhance software test case generation, particularly in reducing execution time. These methods outperform traditional approaches by achieving better test coverage, higher fault detection rates, and overall improved efficiency.

Despite substantial empirical support for hybrid algorithms, research on optimizing software test case execution time using these techniques remains limited. Addressing this gap requires further exploration of hybrid algorithms like HSPSO to refine and improve test case execution efficiency.

Automating test case generation with metaheuristic algorithms has significantly enhanced the speed and effectiveness of software testing. While GA and PSO are widely used, hybrid models like HSPSO demonstrate superior performance. Future advancements, incorporating adaptive techniques and machine learning, will further strengthen automated testing, making it more scalable, reliable, and efficient.

REFERENCES

[1]. Z. A. Hamza and M. Hammad, “Analyzing UML use cases to generate test sequences,” *International Journal of Computing and Digital Systems*, vol. 10, no. 1, pp. 125–134, 2021, doi: 10.12785/ijcds/100112.

[2]. N. Alshahwan *et al.*, “Some challenges for software testing research (invited talk paper),” in *ISSTA 2019 - Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 1–3. doi: 10.1145/3293882.3338991.

[3]. K. Srinivas, “Static and Dynamic Testing of Engineering Materials and Components,” 2020.

[4]. M. Fisher, V. Mascardi, K. Y. Rozier, B.-H. Schlingloff, M. Winikoff, and N. Yorke-Smith, *Towards a framework for certification of reliable autonomous systems*, vol. 35, no. 1. Springer US, 2021. doi: 10.1007/s10458-020-09487-2.

[5]. V. Lavrik, H. Alieksieieva, I. Bardus, and O. Shchetynina, “Object-oriented Representation of Mechanical Systems for the Automated Design,” *2021 International Conference on Intelligent Technologies, CONIT 2021*, no. September, 2021, doi: 10.1109/CONIT51480.2021.9498445.

[6]. M. A. Umar and C. Zhanfang, “A Comparative Study Of Dynamic Software Testing Techniques,” *International Journal of Advanced Networking and Applications*, vol. 12, no. 03, pp. 4575–4584, 2020, doi: 10.35444/ijana.2020.12301.

[7]. S. Sutiah and S. Supriyono, “Software testing on e-learning Madrasahs using Blackbox testing,” *IOP Conf Ser Mater Sci Eng*, vol. 1073, no. 1, 2021, doi: 10.1088/1757-899x/1073/1/012065.

[8]. V. Garousi, A. Rainer, P. Lauvås, and A. Arcuri, *Software-testing education: A systematic literature mapping*, vol. 165. 2020. doi: 10.1016/j.jss.2020.110570.

[9]. M. Khari and P. Kumar, “An extensive evaluation of search-based software testing: a review,” Mar. 29, 2019, *Springer Verlag*. doi: 10.1007/s00500-017-2906-y.

[10]. K. Srivisut, J. A. Clark, and R. F. Paige, *Search-Based Temporal Testing in an Embedded Multicore Platform*, vol. 10784 LNCS. Springer International Publishing, 2018. doi: 10.1007/978-3-319-77538-8_53.

[11]. B. Oliinyk and T. Volodymyr, “Automation in software testing, can we automate anything we want?,” 2019.

[12]. A. Bajaj and O. P. Sangwan, “Test Case Prioritization Using Bat Algorithm,” *Recent Advances in Computer Science and Communications*, vol. 14, no. 2, pp. 593–598, Mar. 2019, doi: 10.2174/2213275912666190226154344.

[13]. N. Honest, “Role of Testing in Software Development Life Cycle,” *International Journal of Computer Sciences and Engineering*, vol. 7, no. 5, pp. 886–889, May 2019, doi: 10.26438/ijcse/v7i5.886889.

[14]. C. T. M. Hue, D. H. Dang, N. N. Binh, and A. H. Truong, “USLTG: Test Case Automatic Generation

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

by Transforming Use Cases," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 9, pp. 1313–1345, 2019, doi: 10.1142/S0218194019500414.

[15]. O. Dahiya, K. Solanki, and A. Dhankhar, "RISK-BASED TESTING : IDENTIFYING , ASSESSING , MITIGATING & MANAGING," vol. 11, no. 3, pp. 192–203, 2020.

[16]. N. Prabhakar, A. Singhal, A. Bansal, and V. Bhatia, *A literature survey of applications of meta-heuristic techniques in software testing*, vol. 731. Springer Singapore, 2019. doi: 10.1007/978-981-10-8848-3_47.

[17]. M. H. Shirvani, "A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems ☆," *Eng Appl Artif Intell*, vol. 90, p. 103501, 2020, doi: 10.1016/j.engappai.

[18]. L. Chrisantyo, A. Wibowo, M. N. Anggiarini, and A. R. Chrismanto, "Blackbox Testing on the ReVAMP Results of The DutaTani Agricultural Information System," 2022. doi: 10.29007/1sx8.

[19]. N. Khoshniat, A. Jamarani, A. Ahmadzadeh, M. Haghi Kashani, and E. Mahdipour, "Nature-inspired metaheuristic methods in software testing," *Soft comput*, vol. 28, no. 2, 2024, doi: 10.1007/s00500-023-08382-8.

[20]. A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand, "Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, 2023, doi: 10.1109/TSE.2022.3184842.

[21]. Sutiah and Supriyono, "Software Testing on The Learning of Islamic Education Media Based on Information Communication Technology Using Blackbox Testing," *International Journal of Information System & Technology Akreditasi*, vol. 3, no. 36, 2020.

[22]. A. A. B. Baqais and M. Alshayeb, "Automatic software refactoring: a systematic literature review," 2020. doi: 10.1007/s11219-019-09477-y.

[23]. R. R. Sahoo and M. Ray, "Metaheuristic techniques for test case generation: A review," *Journal of Information Technology Research*, vol. 11, no. 1, pp. 158–171, Jan. 2018, doi: 10.4018/JITR.2018010110.

[24]. X. Dai, W. Gong, and Q. Gu, "Automated test case generation based on differential evolution with node branch archive," *Comput Ind Eng*, vol. 156, no. November 2020, p. 107290, 2021, doi: 10.1016/j.cie.2021.107290.

[25]. R. K. Medicherla, R. Komondoor, and A. Roychoudhury, "Fitness Guided Vulnerability Detection with Greybox Fuzzing," *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020*, no. Statement 2, pp. 513–520, 2020, doi: 10.1145/3387940.3391457.

[26]. R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empir Softw Eng*, vol. 27, no. 2, 2022, doi: 10.1007/s10664-021-10066-6.

[27]. S. Jana, Y. Tian, K. Pei, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," *Proceedings - International Conference on Software Engineering*, vol. 2018-May, pp. 303–314, 2018, doi: 10.1145/3180155.3180220.

[28]. M. Panda, S. Dash, A. Nayyar, M. Bilal, and R. M. Mehmood, "Test suit generation for object oriented programs: A hybrid firefly and differential evolution approach," *IEEE Access*, vol. 8, pp. 179167–179188, 2020, doi: 10.1109/ACCESS.2020.3026911.

[29]. M. O. Sullivan *et al.*, "A Review of Random Test Case Generation using Genetic Algorithm," *Inf Softw Technol*, vol. 114, no. 0, pp. 1–7, Jul. 2020, doi: 10.1016/j.infsof.2017.08.014.

[30]. C. Hutchison *et al.*, "Robustness testing of autonomy software," *Proceedings - International Conference on Software Engineering*, pp. 276–285, 2018, doi: 10.1145/3183519.3183534.

[31]. T. T. Chekam, M. Papadakis, M. Cordy, and Y. Le Traon, "Killing Stubborn Mutants with Symbolic Execution," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 2, 2021, doi: 10.1145/3425497.

[32]. T. Yadavalli, "An industrial case study to improve test case execution time," 2020, Accessed: Nov. 28, 2022. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1514587/FULLTEXT02

[33]. M. C. Júnior, D. Amalfitano, L. Garcés, A. R. Fasolino, S. A. Andrade, and M. Delamaro, "Dynamic Testing Techniques of Non-functional Requirements in Mobile Apps: A Systematic Mapping Study," *ACM Comput Surv*, vol. 54, no. 10s, pp. 1–38, Jan. 2022, doi: 10.1145/3507903.

[34]. A. Pandey and A. K. Malviya, "Enhancing test case reduction by k-means algorithm and elbow method," *International Journal of Computer Sciences and Engineering*, vol. 6, no. 6, pp. 299–303, 2018, doi: 10.26438/ijcse/v6i6.299303.

[35]. K. Juhnke, M. Tichy, and F. Houdek, "Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality," *Software Quality Journal*, vol. 29, no. 1, 2021, doi: 10.1007/s11219-020-09523-0.

[36]. H. Ben Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software*, vol. 164, p. 110542, 2020, doi: 10.1016/j.jss.2020.110542.

[37]. W. Khamprapai, C. F. Tsai, P. Wang, and C. E. Tsai, "Performance of enhanced multiple-searching genetic algorithm for test case generation in software testing," *Mathematics*, vol. 9, no. 15, pp. 1–17, 2021, doi: 10.3390/math9151779.

[38]. M. A. Albadr, S. Tiun, M. Ayob, and F. Al-Dhief, "Genetic algorithm based on natural selection theory for optimization problems," *Symmetry (Basel)*, vol. 12, no. 11, 2020, doi: 10.3390/sym12111758.

[39]. J. Sun, H. Zhang, H. Zhou, R. Yu, and Y. Tian, "Scenario-Based Test Automation for Highly

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

Automated Vehicles: A Review and Paving the Way for Systematic Safety Assurance," 2022. doi: 10.1109/TITS.2021.3136353.

[40]. [40] M. Khari, A. Sinha, E. Verdú, and R. G. Crespo, "Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization," *Soft comput*, vol. 24, no. 12, pp. 9143–9160, 2020, doi: 10.1007/s00500-019-04444-y.

[41]. Z. Saeed, C. M. Firrone, and T. M. Berruti, "Joint identification through hybrid models improved by correlations," *J Sound Vib*, vol. 494, p. 115889, 2021, doi: 10.1016/j.jsv.2020.115889.

[42]. M. Mahdieh, S. H. Mirian-Hosseinabadi, and M. Mahdieh, "Test case prioritization using test case diversification and fault-proneness estimations," *Automated Software Engineering*, vol. 29, no. 2, 2022, doi: 10.1007/s10515-022-00344-y.

[43]. N. Khanduja and B. Bhushan, "Recent advances and application of metaheuristic algorithms: A survey (2014–2020)," in *Studies in Computational Intelligence*, 2021. doi: 10.1007/978-981-15-7571-6_10.

[44]. Z. Yu, J. C. Carver, G. Rothermel, and T. Menzies, "Searching for better test case prioritization schemes: A case study of ai-assisted systematic literature review," 2019.

[45]. M. A. Umar, "Comprehensive study of software Testing," *International Journal of Advance Research, Ideas and Innovations in Technology*, vol. 5, no. November, 2020.

[46]. S. Lukasczyk, F. Kroiß, and G. Fraser, "An empirical study of automated unit test generation for Python," *Empir Softw Eng*, vol. 28, no. 2, 2023, doi: 10.1007/s10664-022-10248-w.

[47]. S. O. Barraood, H. Mohd, and F. Baharom, "An initial investigation of the effect of quality factors on Agile test case quality through experts' review," *Cogent Eng*, vol. 9, no. 1, 2022, doi: 10.1080/23311916.2022.2082121.

[48]. S. Khastgir, S. Brewerton, J. Thomas, and P. Jennings, "Systems Approach to Creating Test Scenarios for Automated Driving Systems," *Reliab Eng Syst Saf*, vol. 215, 2021, doi: 10.1016/j.ress.2021.107610.

[49]. H. Kirinuki and H. Tanno, "Automating End-to-End Web Testing via Manual Testing," *Journal of Information Processing*, vol. 30, 2022, doi: 10.2197/ipsjjip.30.294.

[50]. J. Godoy, J. P. Galeotti, D. Garbervetsky, and S. Uchitel, *Enabledness-based Testing of Object Protocols*, vol. 30, no. 2. 2021. doi: 10.1145/3415153.

[51]. M. Sánchez-Gordón, L. Rijal, and R. Colomo-Palacios, "Beyond Technical Skills in Software Testing: Automated versus Manual Testing," in *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020*, 2020. doi: 10.1145/3387940.3392238.

[52]. Y. Zhang, J. Wang, X. Li, S. Huang, and X. Wang, "Feature selection for high-dimensional datasets through a novel artificial bee colony framework," *Algorithms*, vol. 14, no. 11, Nov. 2021, doi: 10.3390/a14110324.

[53]. M. Viggiato, D. Paas, C. Buzon, and C.-P. Bezemer, "Using natural language processing techniques to improve manual test case descriptions," 2022. doi: 10.1145/3510457.3513045.

[54]. P. Lakshminarayana and T. V. Sureshkumar, "Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59–72, Jan. 2020, doi: 10.1515/jisys-2019-0051.

[55]. J. Carrasco, S. García, M. M. Rueda, S. Das, and F. Herrera, "Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review," *Swarm Evol Comput*, vol. 54, 2020, doi: 10.1016/j.swevo.2020.100665.

[56]. F. Ahsan, F. A.-A. S. Engineering, and undefined 2024, "A systematic literature review on software security testing using metaheuristics," *Springer*, vol. 31, no. 2, Nov. 2024, doi: 10.1007/S10515-024-00433-0.

[57]. M. A. Khan *et al.*, "Machine learning-based test case prioritization using hyperparameter optimization," *dl.acm.org*, pp. 125–135, Apr. 2024, doi: 10.1145/3644032.3644467.

[58]. S. Gupta, "Enhanced harmony search algorithm with non-linear control parameters for global optimization and engineering design problems," *Eng Comput*, vol. 38, 2022, doi: 10.1007/s00366-021-01467-8.

[59]. R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia, "A Survey on Adaptive Random Testing," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2052–2083, 2021, doi: 10.1109/TSE.2019.2942921.

[60]. J. Lee, S. Kang, and P. Jung, "Test coverage criteria for software product line testing: Systematic literature review," 2020. doi: 10.1016/j.infsof.2020.106272.

[61]. A. Aghamohammadi, S. H. Mirian-Hosseinabadi, and S. Jalali, "Statement frequency coverage: A code coverage criterion for assessing test suite effectiveness," *Inf Softw Technol*, vol. 129, no. September, p. 106426, 2021, doi: 10.1016/j.infsof.2020.106426.

[62]. Y. Lin, J. Sun, G. Fraser, Z. Xiu, T. Liu, and J. S. Dong, "Recovering fitness gradients for interprocedural Boolean flags in search-based testing," *ISSTA 2020 - Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 440–451, 2020, doi: 10.1145/3395363.3397358.

[63]. D. N. Gribkov, M. R. Dengina, V. V. Matveev, and T. V. Masharova, "The impact of the mobile applications usage on the quality of tourism specialists training," *RUDN Journal of Informatization in Education*, vol. 20, no. 4, 2023, doi: 10.22363/2312-8631-2023-20-4-396-409.

[64]. T. E. Colanzi, W. K. G. Assunção, S. R. Vergilio, P. R. Farah, and G. Guizzo, "The Symposium on Search-Based Software Eengineering: Past, Present and Future," *Inf Softw Technol*, vol. 127, p. 106372, 2020,

doi: 10.1016/j.infsof.2020.106372.
[65]. J. M. Balera and V. A. de Santiago Júnior, "A systematic mapping addressing Hyper-Heuristics within Search-based Software Testing," Oct. 01, 2019, *Elsevier B.V.* doi: 10.1016/j.infsof.2019.06.012.
[66]. J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic Improvement of Software: A Comprehensive Survey," *IEEE Transactions on Evolutionary Computation*, 2018, doi: 10.1109/TEVC.2017.2693219.
[67]. A. Arcuri, "RESTful API automated test case generation with Evomaster," *ACM Transactions on Software Engineering and Methodology*, vol. 28, no. 1, pp. 1–37, 2019, doi: 10.1145/3293455.
[68]. D. Gong, B. Sun, X. Yao, and T. Tian, "Test Data Generation for Path Coverage of MPI Programs Using SAEO," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 2, 2021, doi: 10.1145/3423132.
[69]. R. Khan, M. Amjad, and A. K. Srivastava, "Optimization of automatic test case generation with cuckoo search and genetic algorithm approaches," *Advances in Intelligent Systems and Computing*, vol. 554, pp. 413–423, 2018, doi: 10.1007/978-981-10-3773-3_40.
[70]. T. Tian, D. Gong, F. C. Kuo, and H. Liu, "Genetic algorithm based test data generation for MPI parallel programs with blocking communication," *Journal of Systems and Software*, vol. 155, pp. 130–144, 2019, doi: 10.1016/j.jss.2019.04.049.
[71]. E. H. Houssein, M. A. Mahdy, D. Shebl, and W. M. Mohamed, "A Survey of Metaheuristic Algorithms for Solving Optimization Problems," in *Studies in Computational Intelligence*, vol. 967, 2021. doi: 10.1007/978-3-030-70542-8_21.
[72]. B. Geetha and D. Jeya Mala, "A multi objective binary bat approach for testcase selection in object oriented testing," *J Ambient Intell Humaniz Comput*, vol. 12, no. 7, pp. 6997–7006, 2021, doi: 10.1007/s12652-020-02360-w.
[73]. J. E. Gómez-Lagos, M. C. González-Araya, W. E. Soto-Silva, and M. M. Rivera-Moraga, "Optimizing tactical harvest planning for multiple fruit orchards using a metaheuristic modeling approach," *Eur J Oper Res*, vol. 290, no. 1, 2021, doi: 10.1016/j.ejor.2020.08.015.
[74]. A. P. Agrawal and A. Kaur, "A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection," in *Advances in Intelligent Systems and Computing*, Springer Verlag, 2018, pp. 397–405. doi: 10.1007/978-981-10-3223-3_38.
[75]. V. Garousi, M. Felderer, M. Kuhrmann, K. Herkiloğlu, and S. Eldh, "Exploring the industry's challenges in software testing: An empirical study," *Journal of Software: Evolution and Process*, 2020, doi: 10.1002/smr.2251.
[76]. S. Rani, B. Suri, and R. Goyal, "On the effectiveness of using elitist genetic algorithm in mutation testing," *Symmetry (Basel)*, vol. 11, no. 9, 2019, doi: 10.3390/sym11091145.
[77]. D. Silva Rodrigues, "Using Genetic Algorithms in Test Data Generation: A Critical Systematic Mapping," *ACM Comput Surv*, vol. 51, no. 41, p. 41, 2018, doi: 10.1145/3182659.
[78]. A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, "A systematic literature review of techniques and metrics to reduce the cost of mutation testing," *Journal of Systems and Software*, vol. 157, p. 110388, 2019, doi: 10.1016/j.jss.2019.07.100.
[79]. M. Khari and P. Kumar, "An extensive evaluation of search-based software testing: a review," Mar. 29, 2019, *Springer Verlag*. doi: 10.1007/s00500-017-2906-y.
[80]. M. Dadkhah, S. Araban, and S. Paydar, "A systematic literature review on semantic web enabled software testing," *Journal of Systems and Software*, vol. 162, p. 110485, 2020, doi: 10.1016/j.jss.2019.110485.
[81]. L. Gutiérrez-Madroñal, I. Medina-Bulo, and J. J. Domínguez-Jiménez, "Evaluation of EPL mutation operators with the MuEPL mutation system," *Expert Syst Appl*, vol. 116, pp. 78–95, 2019, doi: 10.1016/j.eswa.2018.09.003.
[82]. G. Fraser and J. M. Rojas, "Software Testing," 2019.
[83]. N. Rodrigues, "Archive-Based Swarms," pp. 1460–1467, 2020.
[84]. R. K. Sahoo, S. Satpathy, S. Sahoo, and A. Sarkar, "Model driven test case generation and optimization using adaptive cuckoo search algorithm," *Innov Syst Softw Eng*, 2021, doi: 10.1007/s11334-020-00378-z.
[85]. M. Motwani, M. Soto, Y. Brun, R. Just, and C. Le Goues, "Quality of Automated Program Repair on Real-World Defects," *IEEE Transactions on Software Engineering*, vol. 5589, no. c, pp. 1–1, 2020, doi: 10.1109/tse.2020.2998785.
[86]. A. Damia, M. Esnaashari, and M. Parvizimosaed, "Software Testing using an Adaptive Genetic Algorithm," no. x, 2021, doi: 10.22044/JADM.2021.10018.2138.
[87]. S. Jiang, J. Chen, Y. Zhang, J. Qian, R. Wang, and M. Xue, "Evolutionary approach to generating test data for data flow test," *IET Software*, vol. 12, no. 4, pp. 318–323, Aug. 2018, doi: 10.1049/IET-SEN.2018.5197.
[88]. D. B. Mishra, R. Mishra, A. A. Acharya, and K. N. Das, "Test case optimization and prioritization based on multi-objective genetic algorithm," *Advances in Intelligent Systems and Computing*, vol. 741, pp. 371–381, 2019, doi: 10.1007/978-981-13-0761-4_36.
[89]. B. Swathi and H. Tiwari, "Genetic algorithm approach to optimize test cases," *International Journal of Engineering Trends and Technology*, vol. 68, no. 10, pp. 112–116, 2020, doi: 10.14445/22315381/IJETT-V68I10P219.
[90]. A. Bombarda and A. Gargantini, "An Automata-Based Generation Method for Combinatorial Sequence Testing of Finite State Machines," in *Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2020*, 2020. doi:

10.1109/ICSTW50294.2020.00036.
[91]. M. Kalra, S. Tyagi, V. Kumar, … M. K.-M., and undefined 2021, "A comprehensive review on scatter search: techniques, applications, and challenges," *hindawi.com*, Accessed: Dec. 09, 2022. [Online]. Available: https://www.hindawi.com/journals/mpe/2021/5588486/
[92]. S. Fan, N. Yao, L. Wan, B. Ma, and Y. Zhang, "An Evolutionary Generation Method of Test Data for Multiple Paths Based on Coverage Balance," *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3089196.
[93]. M. Manikandan and R. S. Pant, "Research and advancements in hybrid airships—A review," 2021. doi: 10.1016/j.paerosci.2021.100741.
[94]. W. Khamprapai, C. F. Tsai, and P. Wang, "Analyzing the performance of the multiple-searching genetic algorithm to generate test cases," *Applied Sciences (Switzerland)*, vol. 10, no. 20, pp. 1–16, 2020, doi: 10.3390/app10207264.
[95]. M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artif Life*, vol. 5, no. 2, pp. 137–172, 1999, doi: 10.1162/106454699568728.
[96]. D. Zhao *et al.*, "Ant colony optimization with horizontal and vertical crossover search: Fundamental visions for multi-threshold image segmentation," *Expert Syst Appl*, vol. 167, p. 114122, 2021, doi: 10.1016/j.eswa.2020.114122.
[97]. S. Mirjalili, "Ant colony optimisation," *Studies in Computational Intelligence*, vol. 780, no. November, pp. 33–42, 2019, doi: 10.1007/978-3-319-93025-1_3.
[98]. N. Nayar, S. Gautam, P. Singh, and G. Mehta, "Ant Colony Optimization: A Review of Literature and Application in Feature Selection," in *Lecture Notes in Networks and Systems*, Springer, Singapore, 2021, pp. 285–297. doi: 10.1007/978-981-33-4305-4_22.
[99]. B. Ba-Quttayyan, H. Mohd, and Y. Yusof, "A CRITICAL ANALYSIS OF SWARM INTELLIGENCE FOR REGRESSION TEST CASE PRIORITIZATION," *J Theor Appl Inf Technol*, vol. 30, no. 12, 2022, [Online]. Available: www.jatit.org
[100]. W. Zhang, Y. Qi, X. Zhang, B. Wei, M. Zhang, and Z. Dou, "On test case prioritization using ant colony optimization algorithm," in *Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019*, Institute of Electrical and Electronics Engineers Inc., Aug. 2019, pp. 2767–2773. doi: 10.1109/HPCC/SmartCity/DSS.2019.00388.
[101]. 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.
[102]. A. C. R. Paiva, A. Restivo, and S. Almeida, "Test case generation based on mutations over user execution traces," *Software Quality Journal*, vol. 28, no. 3, pp. 1173–1186, 2020, doi: 10.1007/s11219-020-09503-4.
[103]. D. Bruce, H. D. Menéndez, E. T. Barr, and D. Clark, "Ant Colony Optimization for Object-Oriented Unit Test Generation," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12421 LNCS, pp. 29–41, 2020, doi: 10.1007/978-3-030-60376-2_3.
[104]. M. Dorigo and T. Stützle, "Handbook of Metaheuristics; Ant colony optimization: Overview and recent advances," in *International Series in Operations Research and Management Science*, vol. 272, 2019, pp. 311–351.
[105]. A. Ma, X. Zhang, C. Zhang, and B. Zhang, "An adaptive hybrid ant colony optimization algorithm for the classification problem," *Information Technology and Control*, vol. 48, no. 4, pp. 590–601, 2019, doi: 10.5755/j01.itc.48.4.22330.
[106]. T. Zhang and Z. W. Geem, "Review of harmony search with respect to algorithm structure," *Swarm Evol Comput*, 2019, doi: 10.1016/j.swevo.2019.03.012.
[107]. M. T. Abdulkhaleq *et al.*, "Harmony search: Current studies and uses on healthcare systems," 2022. doi: 10.1016/j.artmed.2022.102348.
[108]. T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Comput Ind Eng*, vol. 137, no. May, p. 106040, 2019, doi: 10.1016/j.cie.2019.106040.
[109]. E. Feldman, A. Contzius, M. Lutch, and K. Bugaj, Instrumental Music Education: Teaching with the Musical and Practical in Harmony, Third Edition. 2020. doi: 10.4324/9780429028700.
[110]. A. E. Kayabekir, G. Bekdaş, M. Yücel, S. M. Nigdeli, and Z. W. Geem, "Harmony Search Algorithm for Structural Engineering Problems," 2021. doi: 10.1007/978-981-33-6773-9_2.
[111]. J. Yi, C. Lu, and G. Li, "A literature review on latest developments of Harmony Search and its applications to intelligent manufacturing," vol. 16, no. December 2018, pp. 2086–2117, 2019.
[112]. Q. Zhu, X. Tang, Y. Li, and M. O. Yeboah, "An improved differential-based harmony search algorithm with linear dynamic domain," *Knowl Based Syst*, vol. 187, no. xxxx, p. 104809, 2020, doi: 10.1016/j.knosys.2019.06.017.
[113]. L. Fu, H. Zhu, C. Zhang, H. Ouyang, and S. Li, "Hybrid Harmony Search Differential Evolution Algorithm," *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3055530.
[114]. L. Flores-Pulido, E. A. Portilla-Flores, E. Santiago-Valentin, E. Vega-Alvarado, M. B. C. Yanez, and P. A. Nino-Suarez, "A Comparative Study of Improved Harmony Search Algorithm in Four Bar Mechanisms," *IEEE Access*, vol. 8, 2020, doi: 10.1109/ACCESS.2020.3015942.
[115]. V. Kumar and D. Kumar, "A Systematic Review on Firefly Algorithm: Past, Present, and Future," *Archives of Computational Methods in Engineering*, vol. 28, no. 4, pp. 3269–3291, Jun. 2021, doi:

10.1007/s11831-020-09498-y.
[116]. L. Wang, H. Hu, R. Liu, and X. Zhou, "An improved differential harmony search algorithm for function optimization problems," *Soft comput*, vol. 23, no. 13, pp. 4827–4852, 2019, doi: 10.1007/s00500-018-3139-4.
[117]. H. Shayanfar and F. S. Gharehchopogh, "Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems," *Applied Soft Computing Journal*, vol. 71, pp. 728–746, 2018, doi: 10.1016/j.asoc.2018.07.033.
[118]. E. Maythaisong and W. Songpan, "Mutation-Based Harmony Search Algorithm for Hybrid Testing of Web Service Composition," *Comput Intell Neurosci*, vol. 2018, pp. 1–15, 2018, doi: 10.1155/2018/4759405.
[119]. M. Ghasemi, S. kadkhoda Mohammadi, M. Zare, S. Mirjalili, M. Gil, and R. Hemmati, "A new firefly algorithm with improved global exploration and convergence with application to engineering optimization," *Decision Analytics Journal*, vol. 5, 2022, doi: 10.1016/j.dajour.2022.100125.
[120]. N. L. Hashim and Y. S. Dawood, "Test case minimization applying firefly algorithm," *Int J Adv Sci Eng Inf Technol*, vol. 8, no. 4–2, pp. 1777–1783, 2018, doi: 10.18517/ijaseit.8.4-2.6820.
[121]. L. Zhou, L. Ding, M. Ma, and W. Tang, "An accurate partially attracted firefly algorithm," *Computing*, vol. 101, no. 5, pp. 477–493, May 2019, doi: 10.1007/s00607-018-0645-2.
[122]. S. Liaquat *et al.*, "Application of Dynamically Search Space Squeezed Modified Firefly Algorithm to a Novel Short Term Economic Dispatch of Multi-Generation Systems," *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2020.3046910.
[123]. K. Hussain, M. N. Mohd Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artif Intell Rev*, vol. 52, no. 4, pp. 2191–2233, Dec. 2019, doi: 10.1007/s10462-017-9605-z.
[124]. M. Oliveira, D. Pinheiro, M. Macedo, C. Bastos-Filho, and R. Menezes, "Uncovering the social interaction network in swarm intelligence algorithms," *Appl Netw Sci*, vol. 5, no. 1, Dec. 2020, doi: 10.1007/s41109-020-00260-8.
[125]. J. Li, Q. An, H. Lei, Q. Deng, and G. G. Wang, "Survey of Lévy Flight-Based Metaheuristics for Optimization," *Mathematics*, vol. 10, no. 15, 2022, doi: 10.3390/math10152785.
[126]. S. L. Tilahun, J. M. T. Ngnotchouye, and N. N. Hamadneh, "Continuous versions of firefly algorithm: a review," *Artif Intell Rev*, vol. 51, no. 3, pp. 445–492, Mar. 2019, doi: 10.1007/s10462-017-9568-0.
[127]. C. M. Li X., *Swarm Intelligence*, vol. 21, no. 1. 2019. doi: 10.1007/978-3-319-91086-4_11.
[128]. V. Kumar and D. Kumar, "A Systematic Review on Firefly Algorithm: Past, Present, and Future," *Archives of Computational Methods in Engineering*, vol. 28, no. 4, pp. 3269–3291, Jun. 2021, doi: 10.1007/s11831-020-09498-y.
[129]. A. Sharma, A. Sharma, V. Chowdary, A. Srivastava, and P. Joshi, "Cuckoo search algorithm: A review of recent variants and engineering applications," in *Studies in Computational Intelligence*, vol. 916, 2021. doi: 10.1007/978-981-15-7571-6_8.
[130]. A. M. Kamoona and J. C. Patra, "A novel enhanced cuckoo search algorithm for contrast enhancement of gray scale images," *Applied Soft Computing Journal*, vol. 85, Dec. 2019, doi: 10.1016/j.asoc.2019.105749.
[131]. M. Reda, A. Onsy, M. A. Elhosseini, A. Y. Haikal, and M. Badawy, "A discrete variant of cuckoo search algorithm to solve the Travelling Salesman Problem and path planning for autonomous trolley inside warehouse," *Knowl Based Syst*, vol. 252, Sep. 2022, doi: 10.1016/j.knosys.2022.109290.
[132]. A. Bajaj, A. Abraham, S. Ratnoo, and L. A. Gabralla, "Test Case Prioritization, Selection, and Reduction Using Improved Quantum-Behaved Particle Swarm Optimization," *Sensors*, vol. 22, no. 12, Jun. 2022, doi: 10.3390/s22124374.
[133]. R. R. Sahoo, M. Ray, and G. Nayak, "Test Case Generation Based on Search-Based Testing," in *Smart Innovation, Systems and Technologies*, 2021. doi: 10.1007/978-981-15-5971-6_33.
[134]. R. R. Sahoo and M. Ray, "PSO based test case generation for critical path using improved combined fitness function," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 4, pp. 479–490, May 2020, doi: 10.1016/j.jksuci.2019.09.010.
[135]. D. D. Ramírez-Ochoa, L. A. Pérez-Domínguez, E. A. Martínez-Gómez, and D. Luviano-Cruz, "PSO, a Swarm Intelligence-Based Evolutionary Algorithm as a Decision-Making Strategy: A Review," 2022. doi: 10.3390/sym14030455.
[136]. B. H. Nguyen, B. Xue, and M. Zhang, "A survey on swarm intelligence approaches to feature selection in data mining," *Swarm Evol Comput*, vol. 54, no. April 2019, p. 100663, 2020, doi: 10.1016/j.swevo.2020.100663.
[137]. A. P. Piotrowski, J. J. Napiorkowski, and A. E. Piotrowska, "Population size in Particle Swarm Optimization," *Swarm Evol Comput*, vol. 58, no. May, p. 100718, 2020, doi: 10.1016/j.swevo.2020.100718.
[138]. D. Yousri, D. Allam, M. B. Eteiba, and P. N. Suganthan, "Static and dynamic photovoltaic models' parameters identification using Chaotic Heterogeneous Comprehensive Learning Particle Swarm Optimizer variants," *Energy Convers Manag*, vol. 182, no. April 2018, pp. 546–563, 2019, doi: 10.1016/j.enconman.2018.12.022.
[139]. J. Shang, Y. Tian, Y. Liu, and R. Liu, "Production scheduling optimization method based on hybrid particle swarm optimization algorithm," in *Journal of Intelligent and Fuzzy Systems*, M. Sundhararajan, X.-Z. Gao, and H. Vahdat Nejad, Eds., IOS Press, Feb. 2018, pp. 955–964. doi: 10.3233/JIFS-169389.
[140]. X. W. Lv, S. Huang, Z. W. Hui, and H. J. Ji, "Test cases generation for multiple paths based on PSO algorithm with metamorphic relations," *IET Software*, vol. 12, no. 4, pp. 306–317, Aug. 2018, doi:

10.1049/IET-SEN.2017.0260.
[141]. E. El Rassy, A. Delaroque, P. Sambou, H. K. Chakravarty, and A. Matynia, "On the Potential of the Particle Swarm Algorithm for the Optimization of Detailed Kinetic Mechanisms. Comparison with the Genetic Algorithm," *Journal of Physical Chemistry A*, vol. 125, no. 23, pp. 5180–5189, 2021, doi: 10.1021/acs.jpca.1c02095.
[142]. V. Jindal and P. Bedi, "An improved hybrid ant particle optimization (IHAPO) algorithm for reducing travel time in VANETs," *Applied Soft Computing Journal*, vol. 64, pp. 526–535, 2018, doi: 10.1016/j.asoc.2017.12.038.
[143]. L. Abualigah, A. Diabat, and Z. W. Geem, "applied sciences A Comprehensive Survey of the Harmony Search Algorithm in Clustering Applications," no. 1, pp. 1–26, 2020, doi: 10.3390/app10113827.
[144]. I. U. Rahman, M. Zakarya, M. Raza, and R. Khan, "An n-state switching PSO algorithm for scalable optimization," *Soft comput*, vol. 24, no. 15, pp. 11297–11314, 2020, doi: 10.1007/s00500-020-05069-2.
[145]. K. Łapa, "Meta-optimization of multi-objective population-based algorithms using multi-objective performance metrics," *Inf Sci (N Y)*, vol. 489, pp. 193–204, 2019, doi: 10.1016/j.ins.2019.03.054.
[146]. R. R. Sahoo and M. Ray, "PSO based test case generation for critical path using improved combined fitness function," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 4, pp. 479–490, May 2020, doi: 10.1016/j.jksuci.2019.09.010.
[147]. N. M. Minhas, S. Masood, K. Petersen, and A. Nadeem, "A systematic mapping of test case generation techniques using UML interaction diagrams," *Journal of Software: Evolution and Process*, vol. 32, no. 6, pp. 1–21, 2020, doi: 10.1002/smr.2235.
[148]. R. Nshimirimana, *A multi-objective particle swarm for constraint and unconstrained problems*, vol. 6. Springer London, 2021. doi: 10.1007/s00521-020-05555-6.
[149]. Y. Shen, W. Cai, H. Kang, X. Sun, Q. Chen, and H. Zhang, "A particle swarm algorithm based on a multi-stage search strategy," *Entropy*, vol. 23, no. 9, 2021, doi: 10.3390/e23091200.
[150]. A. Ben Mabrouk and E. Zagrouba, "Abnormal behavior recognition for intelligent video surveillance systems: A review," *Expert Syst Appl*, vol. 91, pp. 480–491, 2018, doi: 10.1016/j.eswa.2017.09.029.
[151]. X. Han, Y. Dong, L. Yue, and Q. Xu, "State Transition Simulated Annealing Algorithm for Discrete-Continuous Optimization Problems," *IEEE Access*, vol. 7, pp. 44391–44403, 2019, doi: 10.1109/ACCESS.2019.2908961.
[152]. M. A. Awadallah, M. A. Al-Betar, A. L. Bolaji, E. M. Alsukhni, and H. Al-Zoubi, "Natural selection methods for artificial bee colony with new versions of onlooker bee," *Soft comput*, vol. 23, no. 15, pp. 6455–6494, 2019, doi: 10.1007/s00500-018-3299-2.
[153]. X. S. Yang, "Nature-inspired optimization algorithms: Challenges and open problems," *J Comput Sci*, vol. 46, no. xxxx, p. 101104, 2020, doi: 10.1016/j.jocs.2020.101104.
[154]. S. S. Ilango, S. Vimal, M. Kaliappan, and P. Subbulakshmi, "Optimization using Artificial Bee Colony based clustering approach for big data," *Cluster Comput*, vol. 22, pp. 12169–12177, Sep. 2019, doi: 10.1007/s10586-017-1571-3.
[155]. S. S. Jadon, R. Tiwari, H. Sharma, and J. C. Bansal, "Hybrid Artificial Bee Colony algorithm with Differential Evolution," *Applied Soft Computing Journal*, 2017, doi: 10.1016/j.asoc.2017.04.018.
[156]. B. Chatterjee, T. Bhattacharyya, K. K. Ghosh, P. K. Singh, Z. W. Geem, and R. Sarkar, "Late Acceptance Hill Climbing Based Social Ski Driver Algorithm for Feature Selection," *IEEE Access*, vol. 8, pp. 75393–75408, 2020, doi: 10.1109/ACCESS.2020.2988157.
[157]. M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, *Mutation Testing Advances: An Analysis and Survey*, 1st ed., vol. 112. Elsevier Inc., 2019. doi: 10.1016/bs.adcom.2018.03.015.
[158]. A. Martín and O. Schütze, "Pareto Tracer: a predictor–corrector method for multi-objective optimization problems," *Engineering Optimization*, vol. 50, no. 3, pp. 516–536, 2018, doi: 10.1080/0305215X.2017.1327579.
[159]. E. Mirsadeghi and S. Khodayifar, "Hybridizing particle swarm optimization with simulated annealing and differential evolution," *Cluster Comput*, vol. 24, no. 2, pp. 1135–1163, 2021, doi: 10.1007/s10586-020-03179-y.
[160]. K. Amine, "Multiobjective Simulated Annealing: Principles and Algorithm Variants," *Advances in Operations Research*, vol. 2019, 2019, doi: 10.1155/2019/8134674.
[161]. M. Cunha and J. Marques, "A New Multiobjective Simulated Annealing Algorithm—MOSA-GR: Application to the Optimal Design of Water Distribution Networks," *Water Resour Res*, vol. 56, no. 3, pp. 0–2, 2020, doi: 10.1029/2019WR025852.
[162]. N. Darjani and H. Omranpour, "Phase space elliptic density feature for epileptic EEG signals classification using metaheuristic optimization method," *Knowl Based Syst*, vol. 205, p. 106276, 2020, doi: 10.1016/j.knosys.2020.106276.
[163]. M. C. Vélez-Gallego, A. Teran-Somohano, and A. E. Smith, "Minimizing late deliveries in a truck loading problem," *Eur J Oper Res*, vol. 286, no. 3, pp. 919–928, 2020, doi: 10.1016/j.ejor.2020.03.083.
[164]. Y. Klochkov, E. Klochkova, E. Kiyatkina, D. Skripnuk, and D. Aydarov, "Development of methods for business modeling," *2017 International Conference on Infocom Technologies and Unmanned Systems: Trends and Future Directions, ICTUS 2017*, vol. 2018-Janua, pp. 366–369, 2018, doi: 10.1109/ICTUS.2017.8286034.
[165]. T. Østergård, R. L. Jensen, and S. E. Maagaard, "A comparison of six metamodeling techniques applied to building performance simulations," *Appl Energy*, vol. 211, no. October 2017, pp. 89–103, 2018, doi:

10.1016/j.apenergy.2017.10.102.
[166]. K. Dorgham, I. Nouaouri, H. Ben-Romdhane, and S. Krichen, "A hybrid simulated annealing approach for the patient bed assignment problem," *Procedia Comput Sci*, vol. 159, pp. 408–417, 2019, doi: 10.1016/j.procs.2019.09.195.
[167]. G. Julirose, "GENETIC ALGORITHM OPTIMIZATION OF PRODUCT ve rs ity of ay a rs," 2018.
[168]. E. B. Tirkolaee, A. Goli, M. Hematian, A. Kumar, and S. Tao, "Multi-objective multi-mode resource constrained project scheduling problem using Pareto-based algorithms," *Computing*, 2019, doi: 10.1007/s00607-018-00693-1.
[169]. N. A. Khan, A. Awang, and S. A. A. Karim, "Security in Internet of Things: A Review," 2022. doi: 10.1109/ACCESS.2022.3209355.
[170]. M. Micev, M. Ćalasan, Z. M. Ali, H. M. Hasanien, and S. H. E. Abdel Aleem, "Optimal design of automatic voltage regulation controller using hybrid simulated annealing – Manta ray foraging optimization algorithm," *Ain Shams Engineering Journal*, vol. 12, no. 1, pp. 641–657, 2021, doi: 10.1016/j.asej.2020.07.010.
[171]. Seema and A. R. Dixit, "Application of soft computing techniques for cell formation problems: A review," 2017 International Conference on Advances in Mechanical, Industrial, Automation and Management Systems, AMIAMS 2017 - Proceedings, pp. 245–251, 2017, doi: 10.1109/AMIAMS.2017.8069219.
[172]. A. Franzin and T. Stützle, "Revisiting simulated annealing: A component-based analysis," *Comput Oper Res*, vol. 104, pp. 191–206, 2019, doi: 10.1016/j.cor.2018.12.015.
[173]. S. Jianqi, H. Yanhong, L. Ang, C. F.-O. Physics, and undefined 2018, "An optimal solution for software testing case generation based on particle swarm optimization," *degruyter.com*, Accessed: Dec. 12, 2022. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/phys-2018-0048/html
[174]. C. Roques-Carmes *et al.*, "Heuristic recurrent algorithms for photonic Ising machines," *Nat Commun*, vol. 11, no. 1, 2020, doi: 10.1038/s41467-019-14096-z.
[175]. J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020, doi: 10.1109/ACCESS.2020.2990567.
[176]. S. Bernardo *et al.*, "Software Quality Assurance as a Service: Encompassing the quality assessment of software and services," *Future Generation Computer Systems*, vol. 156, 2024, doi: 10.1016/j.future.2024.03.024.
[177]. M. Shehab *et al.*, "A Comprehensive Review of Bat Inspired Algorithm: Variants, Applications, and Hybridization," 2023. doi: 10.1007/s11831-022-09817-5.
[178]. F. Misni, L. S. Lee, and H. V. Seow, "Hybrid harmony search-simulated annealing algorithm for location-inventory-routing problem in supply chain network design with defect and non-defect items," *Applied Sciences (Switzerland)*, vol. 10, no. 18, 2020, doi: 10.3390/APP10186625.
[179]. Y. Zhang, J. Wang, X. Li, S. Huang, and X. Wang, "Feature selection for high-dimensional datasets through a novel artificial bee colony framework," *Algorithms*, vol. 14, no. 11, Nov. 2021, doi: 10.3390/a14110324.
[180]. M. Thirunavukkarasu, Y. Sawle, and H. Lala, "A comprehensive review on optimization of hybrid renewable energy systems using various optimization techniques," 2023. doi: 10.1016/j.rser.2023.113192.
[181]. R. D. Goswami, S. Chakraborty, and B. Misra, "Variants of Genetic Algorithms and Their Applications," 2023. doi: 10.1007/978-981-99-3428-7_1.
[182]. B. Ba-Quttayyan, H. Mohd, and Y. Yusof, "A CRITICAL ANALYSIS OF SWARM INTELLIGENCE FOR REGRESSION TEST CASE PRIORITIZATION," *J Theor Appl Inf Technol*, vol. 30, no. 12, 2022, [Online]. Available: www.jatit.org
[183]. M. Jahandideh-Tehrani, O. Bozorg-Haddad, and H. A. Loáiciga, "Application of particle swarm optimization to water management: an introduction and overview," 2020. doi: 10.1007/s10661-020-8228-z.
[184]. C. Sathiyaraj, M. Ramachandran, M. Amudha, and R. Kurinjimalar, "A Review on Hill Climbing Optimization Methodology," *Recent trends in Management and Commerce*, vol. 3, no. 1, 2022, doi: 10.46632/rmc/3/1/1.
[185]. T. Guilmeau, E. Chouzenoux, and V. Elvira, "Simulated Annealing: A Review and a New Scheme," in *IEEE Workshop on Statistical Signal Processing Proceedings*, 2021. doi: 10.1109/SSP49050.2021.9513782.
[186]. M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *International Journal of Applied Science and Engineering*, vol. 17, no. 4, pp. 331–344, 2020, doi: https://doi.org/10.6703/IJASE.202012.
[187]. R. García-Ródenas, L. J. Linares, and J. A. López-Gómez, "A Memetic Chaotic Gravitational Search Algorithm for unconstrained global optimization problems," *Applied Soft Computing Journal*, vol. 79, pp. 14–29, 2019, doi: 10.1016/j.asoc.2019.03.011.
[188]. B. Farnad, A. Jafarian, and D. Baleanu, "A new hybrid algorithm for continuous optimization problem," *Appl Math Model*, vol. 55, pp. 652–673, 2018, doi: 10.1016/j.apm.2017.10.001.
[189]. M. Shariati *et al.*, "Application of a hybrid artificial neural network-particle swarm optimization (ANN-PSO) model in behavior prediction of channel shear connectors embedded in normal and high-strength concrete," *Applied Sciences (Switzerland)*, vol. 9, no. 24, 2019, doi: 10.3390/app9245534.
[190]. D. Tian, "Particle swarm optimization with chaos-based initialization for numerical optimization," *Intelligent Automation and Soft Computing*, vol. 24, no. 2, pp. 331–342, 2018, doi: 10.1080/10798587.2017.1293881.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26feb433

[191]. H. Pu *et al.*, "Mountain railway alignment optimization using stepwise & hybrid particle swarm optimization incorporating genetic operators," *Applied Soft Computing Journal*, vol. 78, pp. 41–57, 2019, doi: 10.1016/j.asoc.2019.01.051.

[192]. Y. Zhi, H. Wang, and L. Wang, "A state of health estimation method for electric vehicle Li-ion batteries using GA-PSO-SVR," *Complex and Intelligent Systems*, vol. 8, no. 3, pp. 2167–2182, 2022, doi: 10.1007/s40747-021-00639-9.

[193]. M. Žižović, T. Živković, N. Bačanin Džakula, and I. Štrumberger, "Nature-Inspired Approaches in Software Testing Optimization," 2021. doi: 10.15308/sinteza-2021-28-33.

[194]. Palak and P. Gulia, "Hybrid swarm and GA based approach for software test case selection," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 6, pp. 4898–4903, 2019, doi: 10.11591/ijece.v9i6.pp49898-4903.

[195]. P. Lakshminarayana and T. V. Sureshkumar, "Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59–72, Jan. 2020, doi: 10.1515/jisys-2019-0051.

[196]. R. Ku Sahoo, S. Kumar Nanda, D. Prasad Mohapatra, and M. Ranjan Patra, "Model driven test case optimization of UML combinational diagrams using hybrid bee colony algorithm.," *mecs-press.net*, vol. 6, pp. 43–54, 2017, doi: 10.5815/ijisa.2017.06.05.

[197]. O. S. Faust, C. G. Mehli, T. Hanne, and R. Dornberger, "A Genetic Algorithm for Optimizing Parameters for Ant Colony Optimization Solving Capacitated Vehicle Routing Problems," *ACM International Conference Proceeding Series*, pp. 52–58, 2020, doi: 10.1145/3396474.3396489.

[198]. T. K. Akila and M. Arunachalam, "Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing," *International Journal of Advanced Technology and Engineering Exploration*, vol. 9, no. 88, pp. 384–400, 2022, doi: 10.19101/IJATEE.2021.874727.

[199]. [199] M. D. Akbar and R. Aurachmana, "Hybrid genetic–tabu search algorithm to optimize the route for capacitated vehicle routing problem with time window," *International Journal of Industrial Optimization*, vol. 1, no. 1, p. 15, 2020, doi: 10.12928/ijio.v1i1.1421.

[200]. S. Sharma, S. A. M. Rizvi, and V. Sharma, "A framework for optimization of software test cases generation using cuckoo search algorithm," in *Proceedings of the 9th International Conference On Cloud Computing, Data Science and Engineering, Confluence 2019*, 2019. doi: 10.1109/CONFLUENCE.2019.8776898.

[201]. R. Beed, A. Roy, S. Sarkar, and D. Bhattacharya, "A hybrid multi-objective tour route optimization algorithm based on particle swarm optimization and artificial bee colony optimization," *Comput Intell*, vol. 36, no. 3, 2020, doi: 10.1111/coin.12276.

[202]. M. S. Ajmal, Z. Iqbal, F. Z. Khan, M. Ahmad, I. Ahmad, and B. B. Gupta, "Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers," *Computers and Electrical Engineering*, vol. 95, 2021, doi: 10.1016/j.compeleceng.2021.107419.

[203]. P. Lakshminarayana and T. V. Sureshkumar, "Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59–72, Jan. 2020, doi: 10.1515/jisys-2019-0051.

[204]. A. Raghavan, P. Maan, and A. K. B. Shenoy, "Optimization of day-ahead energy storage system scheduling in microgrid using genetic algorithm and particle swarm optimization," *IEEE Access*, vol. 8, 2020, doi: 10.1109/ACCESS.2020.3025673.

[205]. T. K. Akila and M. Arunachalam, "Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing," *International Journal of Advanced Technology and Engineering Exploration*, vol. 9, no. 88, 2022, doi: 10.19101/IJATEE.2021.874727.

[206]. R. Chandran, S. Rakesh Kumar, and N. Gayathri, "Genetic algorithm-based tabu search for optimal energy-aware allocation of data center resources," *Soft comput*, vol. 24, no. 21, 2020, doi: 10.1007/s00500-020-05240-9.

[207]. A. Panichella, F. M. Kifetew, and P. Tonella, "Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, Feb. 2018, doi: 10.1109/TSE.2017.2663435.

[208]. J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine Learning Testing: Survey, Landscapes and Horizons," *IEEE Transactions on Software Engineering*, vol. X, no. X, pp. 1–1, 2020, doi: 10.1109/tse.2019.2962027.