Enhanced Software Performance Testing for Big Data Platforms Using Clock-Controlled Computation Tree Logic with Particle Swarm and Genetic Optimization

Yuan Sun[1, 2], Md Gapar Md Johar[1, *], Jacquline Tham[1]
[1]Postgrduate Center, Management and Science University, Shah Alam 40100, Malaysia
[2]School of Information Engineering, Gongqing Institute of Science and Technology, Jiujiang 332020, China
E-mail: Yuan Sun: sunyuan12123@gmail.com, Md Gapar Md Johar: mdgapar@126.com, jacquline@msu.edu.my
[*]Corresponding author

Keywords: software testing, particle swarm algorithm, bell-controlled computational tree logic method, genetic algorithm, testing effectiveness

Received: November 22, 2024

This research aims to solve the problems of testing inefficiency and lack of accuracy in software testing, and proposes a software performance testing system for big data platforms based on the clock-controlled computational tree logic method. The particle swarm algorithm finds the optimal solution through the movement and mutual cooperation of particles in the search space. Genetic algorithm evolves the population through selection, crossover, and mutation operations, ultimately finding the optimal solution. Secondly, long short-term memory networks and linear autoregressive models also have advantages in software testing, which can improve the effectiveness and efficiency of software testing through reasonable selection and combined use. The new algorithm utilizes the ability of PSO and GA algorithms to search for optimal solutions through particle motion and group cooperation in the search space, in order to determine key moment parameters and other relevant information in software testing systems. The research uses the algorithmic logic of the particle swarm algorithm and the genetic algorithm to confirm the moment parameters and other information of the software testing system. At the same time, an algorithmic model research on the joint coverage and the use of the value of the system, and finally makes use of the big data platform to analyze the research system. The specific indicators used in the study include 100% test case coverage, as well as the functional coverage of genetic algorithms and particle swarm optimization algorithms. The innovative combination of CCTL method and optimization algorithm in the research has improved the accuracy and stability of software testing. CCTL is an extended computational tree logic that introduces the concept of time, allowing testers to explicitly specify time constraints in software testing, thereby more accurately simulating real-world scenarios. The research results show that using the system to test software can achieve a coverage rate of 100% for its component use cases, while the functional coverage rates of genetic algorithm and particle swarm algorithm reach 90.36% and 91.32%, respectively. The accuracy of software testing research methods is 5% and 6% higher than that of LSTM and LAR methods. When the moment range of the particle parameter position information of the model is [150 ms, 250 ms], the maximum value of the target parameter velocity is 80 m/s and the minimum value is 0 m/s. The maximum value of the target azimuth velocity is 20 rad/s, and the minimum value is 0 rad/s. The system is able to determine the various parameters of the software, and at the same time in the software test results on the test results are normal, fault analysis can be completed normally, the performance of the algorithm is also superior to other algorithm models such as LSTM and LAR, and the study of the use of algorithms with a higher degree of stability. It can be seen that the system and methodology used in this research is superior to traditional methods and the test results of software testing have improved. This study provides a new research direction for platform software afterwards.

Povzetek: Opisan je sistem za testiranje zmogljivosti programske opreme na platformah za velike podatke, ki uporablja metodologijo CCTL, optimizirano s pomočjo algoritmov PSO in GA.

1 Introduction

In the current digital age, big data platforms have become a core technology for processing complex data sets. With the increasing volume of data, it is critical to ensure stable platform software performance [1]. Software performance testing is a key component in ensuring efficient and accurate data processing [2]. The existing performance testing methods for big data platform software suffer from low efficiency and insufficient accuracy when dealing with large-scale data and complex scenarios. Therefore, how to build an efficient and accurate method for testing the performance of big data platform software has become a difficult problem that still needs to be solved [3]. The research objectives specifically include improving testing accuracy, expanding testing coverage, and clarifying the

performance of CCTL in handling specific types of constraints, as well as enhancing the adaptability of the model in handling large-scale datasets and real-time parameter changes. Simultaneously studying the hypothesis that combining CCTL methods and optimization algorithms can significantly improve the accuracy and coverage of big data platform software testing, optimization algorithms can effectively address the limitations of CCTL in processing large-scale datasets and real-time parameter changes, and the combination of PSO and GA can provide more stable and comprehensive testing results in different testing scenarios. These algorithms are used to optimize the search for critical moment parameters and other relevant information in software testing systems, thereby improving the accuracy and efficiency of the testing process. In the current digital age, big data platforms have become the core technology for processing complex datasets. As the amount of data increases, ensuring the stability of platform software performance becomes crucial. Software performance testing is a key component in ensuring efficient and accurate data processing. The existing performance testing methods for big data platform software face problems of low efficiency and insufficient accuracy when dealing with large-scale data and complex scenarios. Clock-Controlled Computation Tree Logic (CCTL) is a temporal logic that improves the traditional Computation Tree Logic (CTL) by introducing the concept of time [4]. Long Short Term Memory Network (LSTM) is a special type of recurrent neural network that can remember cells and gate mechanisms to solve the problems of gradient vanishing and exploding in traditional RNNs when processing long sequence data. Linear Autoregressive Model (LAR) is a statistical model used for time series analysis. This approach is particularly important in Big Data processing, where it allows the testing process to consider the temporal properties of the data flow, thus more accurately simulating real-world situations. CCTL can more effectively identify and analyze performance bottlenecks and potential problems in big-data platforms. Although the clock controlled computation tree logic method has achieved certain application results in other fields, there is still relatively little research on its application to software performance testing on big data platforms. CCTL is used to verify the temporal attributes of critical tasks and in fields such as aviation electronics, automotive control, and industrial automation. It ensures that data transmission and reception are completed within specified time intervals, which is crucial for network design and optimization. The novelty of the research lies in the innovative combination of clock controlled computation tree logic with PSO and GA, which not only achieves 100% test case coverage, but also enhances adaptability to different testing environments and conditions through optimized algorithm logic. In addition, this method has broad application prospects in potential fields such as big data processing, cloud computing platforms, Internet of Things (IoT) devices, as well as artificial intelligence and machine learning. Firstly, the new model utilizes the algorithm logic of particle swarm optimization and genetic algorithm to improve and analyze the process, in order to enhance the accuracy and stability of software testing parameter validation. Secondly, the research constructed testing scenarios and test cases suitable for big data platforms, and verified the performance and accuracy of the new method in different testing environments through experiments. And provided new ideas and methods for performance testing of big data platform software. By utilizing CCTL, the accuracy and efficiency of software performance testing have been significantly improved, especially for big data platforms that handle dynamic and large-scale datasets. The proposed method can be applied to other big data platforms, providing a scalable and effective performance testing solution applicable to various fields. This research is divided into four parts; the first part is an overview of domestic and international research; the second part is a study of the system and method of software testing; the third part is mainly to test and analyze the performance of the system; and the fourth part is a summary of the current research.

2 Literature review

Software is usually tested for different problems; therefore, different research methods are required to solve these problems. To address the complexities of data analysis encountered by strength and conditioning professionals who use strength platforms to conduct CMJ assessments during training, this study proposes a solution to create a data analysis program using MATLAB. The findings suggest that the program can help coaches simplify the process and improve the accuracy and reliability of the data analysis. In addition, the sample scripts provided allow further learning and mastery of basic scripting strategies to create separate analysis programs for the CMJ and other performance tests [5]. Kaur and Agrawal proposed a new approach based on the Bat Search Algorithm and the Cuckoo Search Algorithm to solve the problem of regression test case selection and improve the efficiency and accuracy of software maintenance. The results of this study show that both algorithms are effective in reducing the number of required test cases and improving the testing efficiency in regression testing. Among them, the cuckoo search algorithm is slightly better in terms of performance parameters. The algorithm proposed by Kaur and Agrawal performs well in reducing the number of test cases and improving testing efficiency, but still has limitations when dealing with large-scale datasets [6]. Chen et al., proposed an auxiliary method based on machine learning to study the benchmarking method in performance unit testing. The results of this study show that the method can effectively identify benchmarking methods, thus improving the accuracy and efficiency of performance unit testing. It was also found that the Random Forest algorithm performs the best in predicting performance and can retrieve 43% of the true BDMs by examining only 5% of the candidate methods detected by the model. Chen et al.'s method can effectively identify benchmark methods and improve the accuracy and efficiency of testing. However, this method has poor adaptability when dealing with dynamic and large-scale datasets [7]. In order to

address the problem of parameter estimation for software reliability growth models, this paper proposes a framework modelled on the Non-Homogeneous Poisson Process (NHPP). The framework integrates test coverage (TC), error propagation and troubleshooting efficiency while limiting the number of parameters. The results show that the model is more reliable than the existing models and can effectively assess and predict software reliability. Through sensitivity analyzes, we demonstrated that the model parameters have less impact on the mean function. The model proposed by Khurshid et al. performs well in handling test coverage, error propagation, and troubleshooting efficiency, but has shortcomings in handling real-time parameter constraints [8].

Qian et al. propose a method for prioritizing test scripts to address the memory bloat problem in web applications, as well as to improve the efficiency of performance testing. The new method uses a learning ranking technique to predict which test scripts are more likely to cause memory bloats, and thus prioritizes the execution of these scripts. Experimental results show that the method is effective in speeding up testing and improving the efficiency of detecting memory bloats. The method proposed by Qian et al. significantly improves the efficiency of detecting memory expansion. However, this method is mainly targeted at web applications and lacks adaptability to complex scenarios on big data platforms [9]. To fill a gap in the research on programming language security, this study proposes a methodology for benchmarking the security and performance of languages. The methodology compares six well-known programming languages and uses quantitative and qualitative methods to determine which language is best in terms of security and performance by testing the code and analyzing the available information. The results of the study show that the Rust performs best in terms of security and performance, achieving an excellent balance [10].

To investigate the effectiveness of Metamorphic Testing (MT) in different application contexts, this study revisited the use of MT in Sentiment Analysis (SA) systems and found that false satisfaction is an important factor affecting the validity of MT. An in-depth analysis of false gratification reveals how it can occur and how it can affect the effectiveness of MT. Our study also suggests that MT may overestimate the consistency of the system with the relevant MR if the occurrence of false satisfaction is not taken into account. These findings will help the MT community use MT test results more fairly and reliably [11]. To compare the predictive performance of an ensemble species distribution model with that of a single model, a study was conducted using a large eucalyptus species presence-absence dataset. Two spatial blocking strategies were used to partition the dataset, and all models within the calibration fold were calibrated and cross-validated using repeated random partitioning of data and spatial chunking. The results of the study showed that the ensemble models performed well in some tests, but did not always outperform their untuned individual models or the tuned BRT. Additionally, good external performance was obtained by selecting untuned individual models with the best cross-validation performance [12]. Hosseini et al. proposed a quantitative data error propagation rate and a mutation location recognition method based on genetic algorithm to reduce the cost of mutation detection. The research results showed that this method effectively reduced the number of mutants by about 24%, while increasing the mutation score by about 5.6%. Only 7.46% of the generated mutants were equivalent, significantly reducing testing time and cost [13]. Zeb et al. found that heuristic algorithms have been well studied in multiple fields, among which the use of heuristic algorithms such as particle swarm optimization in software testing can reduce the defects of software testing, improve the accuracy and reliability of software testing. It can be seen that using particle swarm optimization algorithm can improve the accuracy of software testing [14]. Pan et al. proposed a similarity search test case minimization technique based on genetic algorithm to improve the efficiency and fault detection capability of software testing. The research results indicate that the new method achieves a higher average fault detection rate compared to the existing technology FAST-R, with only 50% of test cases running [15].

In summary, existing research has made significant progress in the field of software testing, but there are still some shortcomings. Although MATLAB's data analysis program simplifies the process, it relies on specific environments; Although machine learning methods have improved the efficiency of regression testing, they lack applicability and have poor performance in evaluating data. Research has shown that although other framework models can accelerate the detection of webpage memory inflation, there is still a problem of poor network environment testing [16]. Therefore, this study proposes a new solution to address the issues of insufficient accuracy and performance in software testing. Firstly, the clock controlled computation tree logic method is used to generate software moment cases in the big data platform, which solves the problems of environment dependence, testing efficiency, and accuracy in existing research for software performance testing in big data platforms. Secondly, the model selects parameters such as moment examples and determines their values to ensure the accuracy and performance of software testing, solving the problems of limited applicability and unstable results of existing methods.

3 Method

3.1 Analysis of CCTL model

This chapter mainly focuses on the time platform for software testing to build a system, using the CCTL method to analyze the software testing system, build the moment component use case generation model and the software testing system model, and then analyze the system model to achieve system building for software testing performance. The main workflow of the current research is shown in Figure 1.

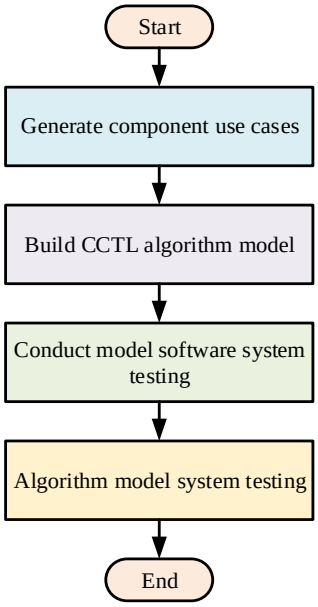

Figure 1: Main workflow

Description: Shows the main workflow.

From Figure 1, it can be seen that the first step is to analyze and generate test cases for the software. Secondly, by using the CCTL algorithm model, a software testing system based on the algorithm model is built. Then, based on the constructed model, implement software testing. Finally, the algorithm model constructed was tested for its actual effectiveness through experiments.

3.2 Research on software testing model

In the software for testing, software parameters of the moment input will have certain requirements, must be input in a specific time parameter to make the whole operation is effective, through this effective time input to be able to follow up on the input function operation, this time point and parameter point is the current software input parameters of the space moment. Generally, there are three types of input time space for software: interval input, cycle input, and discrete input. The interval input selects a fixed point in the cycle and selects a moment for input; the cycle input selects a time moment within the cycle and inputs different time points; the discrete input selects any time point from the discrete time collection for input. Interval input is suitable for systems that require regular and consistent data input, loop input is suitable for systems with periodic tasks, and discrete input is suitable for systems with irregular or user driven events. By classifying the input temporal space in this way, we can better understand and model the temporal behavior of software systems, which is crucial for accurate and effective software testing.

The number of parameters covered by the input will inevitably exist because of the moment processing constraints, and the expression of the constraints before the test generation is an important step in the analysis of its parameters. The general constraint moment is divided into two types: independent moment constraints and related moment constraints, which are mainly due to the non-existence of correlation between the parameters and parameters. Therefore, the two parameters do not affect each other, and at the same time the parameters simultaneously have their own moment constraint limitations. Independent moment constraints are temporal constraints on individual parameters that are not dependent on other parameters, thus simplifying the testing process and increasing testing efficiency. Correlation moment constraints, on the other hand, involve temporal dependencies between multiple parameters, ensuring that multiple parameters work together at a specific point in time or time range. Separating the independent moment constraints and correlated moment constraints into two distinct parts improves the clarity, efficiency and accuracy of testing. Correlated moment constraints, on the other hand, refer to the existence of identical moment constraints as well as different correlated moment constraints between two parameters. The CCTL method enables the description of constraints to reduce the moment constraints of the parameters. As shown in Eq. (1) is the independent time-constraint formula for the CCTL method [17].

$$\begin{cases} EX_{t_0,t_1}\phi \\ EX_{t_0,t_1}(\phi)^n \end{cases} \tag{1}$$

In Eq. (1), t_0, t_1 denotes the time interval, EX denotes the relationship between the constraints present, ϕ denotes the event expression of the input parameters, and n denotes the number of events satisfied by the test. The times of the different parameter constraints in the CCTL method can be expressed as shown in Eq. (2) [18]:

$$EX_{t_0,t_1}\phi \to EX_{t_2,t_3}\psi \tag{2}$$

In Eq. (2), ψ is expressed in terms of execution in the interval time, and the rest of the parameter expressions are the same as above. The constraint expression for the same moment means that at this time, the time will move to the next pointing interval after execution in that interval. The parameter expression for time can be substituted for the separate moments. When the time moments are replaced as separate moments the relative time constraints are also induced as n, the expression is shown in Eq. (3).

$$EX_{t_0,t_1}\phi \to EX_{t_2,t_3}\psi \to \cdots \to EX_{t_{2n-4},t_{2n-3}}\omega \to EX_{t_{2n-2},t_{2n-1}}\zeta \tag{3}$$

In Eq. (3), ω, ζ indicates the input conditions for different parameters. The remaining parameters were the same as those decribed. Because the existence of constraints will cause some combinations to not be in a time test case at the same time, for the current time constraints software test cases need to deal with constraint combinations; typically, there are four ways to deal with time constraints under the CCTL approach: abstract parameters, sub-models, substitution, and avoidance of

selection methods. Abstract parameters simplify the model and reduce complexity by abstracting specific temporal parameters, but can lead to oversimplification of the model and loss of important details. Submodels decompose a complex model into multiple submodels, but coordination between submodels can be complex, increasing the complexity of the overall model. Substitution methods simplify temporal constraints, but may lead to loss of temporal information. Selection methods choose specific temporal parameters or time points to avoid conflicts and ensure model consistency, but may require additional logic and computation, increasing complexity. Their principle is to transform the models and convert the models that appear to be in conflict with valid combination methods. However, the problem with this method is that when large parameters are encountered, more unnecessary and redundant information parameters appear [19]. At the same time, when using the CCTL method for software parameter moment determination and combinatorial testing, it is necessary to input a large number of consecutive parameters; therefore, it is necessary to study its parameter coverage during the analyzis. At this time it is necessary to use generative algorithms to study and analyse the parameter inputs of the method.

3.3 Time parameter combination and generation algorithm model

In the case of continuous input and transmission of the software moment parameters selected by the CCTL method, the parameter information at this time is analyzed. The algorithm model of data analysis selects the particle swarm algorithm and genetic algorithm for analysis, through the parameters of the population optimal solution and evolutionary optimal solution to find, to output the optimal value of the current parameters, to achieve the analysis of the parameters of the judgement. The analysis of parameter coverage helps ensure that all possible input conditions and scenarios are fully tested, thereby improving the comprehensiveness and accuracy of testing. PSO and GA algorithms can better generate test cases, ensuring the reliability and effectiveness of test results. At the same time, these algorithms perform well in parameter optimization problems and can quickly find the optimal solution, thereby improving the efficiency and accuracy of testing. The algorithm joint process includes PSO and GA initializing populations separately, with each individual representing a test case or parameter combination. Simultaneously, both models undergo iterative optimization and evaluate the quality of test cases through fitness functions. Finally, during the iteration process, excellent individuals are exchanged between PSO and GA to improve optimization efficiency. Figure 2 shows the flow of the moment-combination generation algorithm.

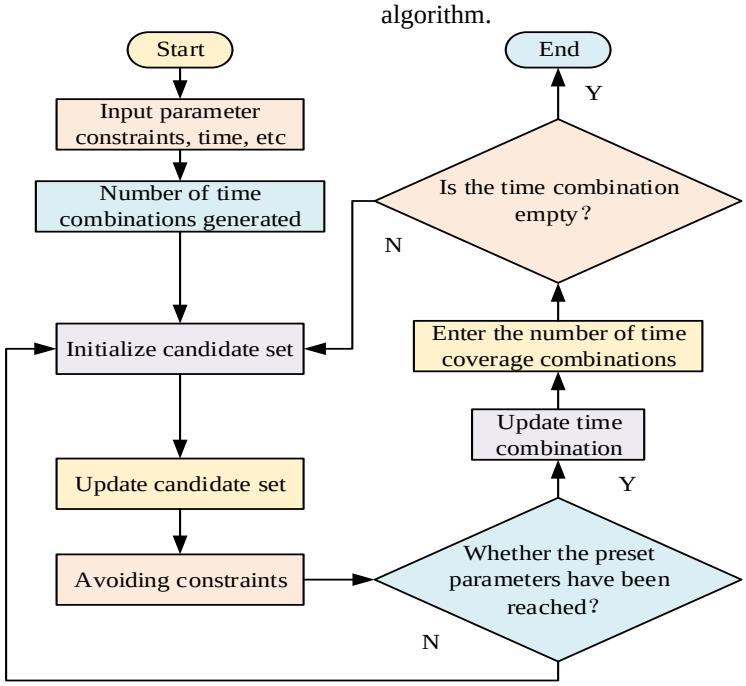

Figure 2: Time combination generation algorithm process

Description: Illustrates how the parameter information is analysed by particle swarm algorithm and genetic algorithm.

As shown in Figure 2, in the analysis phase of the algorithm, the number of parameters is first input to select the constraints and input moments, after which the combination of parameters at the current moment is generated, the parameter candidate set is initialized, and the set is updated so as to select a better individual for constraint evasion. Then, it is judged whether the current parameters under test reach the pre-set parameter data, and if they do, the combination of the target time is updated and then the combination is output. If it is reached, then

update the target time combination and output the combination; if it is not reached, then reinitialise the candidate collection. In software performance testing, selecting the optimal value of the current parameter is achieved through optimization algorithms. The algorithm gradually optimizes the combination of parameters to find the optimal test case. After outputting the number of combinations again, judge whether the combination set is empty; if it is empty, end the algorithm; if not, initialize the candidate set. Firstly, when generating examples, the model initializes the population of genetic algorithm and the particle swarm of particle swarm algorithm, with each individual representing a time combination. Based on the differences in algorithm structures, generate and optimize time combinations separately. Then regularly exchange individuals of genetic algorithm and particle swarm algorithm, add excellent particles from particle swarm algorithm to the population of genetic algorithm, and add excellent chromosomes from genetic algorithm to the particle swarm of particle swarm algorithm. Finally, after the iteration is completed, the results of the two algorithms are fused and the optimal time combination is selected as the final solution. When the algorithm is running, an initial population is formed by randomly generating a set of individuals, which ensures the diversity of the population. Secondly, setting a larger population can provide more solutions, but the computational complexity will also increase; Smaller populations require less computation, but may not be able to cover all possible solutions. The evolutionary process of the population requires four steps: population selection, population crossover, population mutation, and population replacement. Finally, suppose that when solving a problem, individuals can be represented using binary encoding. In software performance testing, genetic operations such as crossover and mutation can be easily performed by encoding parameter values as binary strings. First, the initial population randomly generates binary strings, and then the selection operation selects individuals with high fitness. Finally, a portion of new individuals will replace some individuals in the current population, maintaining the continuous evolution of the population. The data input process of the algorithm first selects the number of data parameters as k. Then generate a combination function through target coverage combination, which includes combinations between parameters, combinations between parameter values and input time, and direct combinations between input parameter data and time. After selecting the set of algorithm data parameters, the algorithm data is expressed by integrating the data parameters. In software performance testing, selecting and integrating data parameters is a key step in ensuring the effectiveness and accuracy of the testing. By selecting parameters directly related to the testing objectives and ensuring that these parameters comprehensively cover the testing scenario, the efficiency and accuracy of testing can be improved. By optimizing parameter combinations through PSO and GA algorithms, optimal test cases can be generated. This not only improves the efficiency and accuracy of testing, but also ensures that the test results can truly reflect the performance of the software under various conditions. At the same time, for the purpose of concluding the data space, all the parameter data mentioned above are collected. Due to the need to determine both time data parameters and initial candidate sets during algorithm execution, the study selects two parameter values p_1, p_2 through the algorithm process, and the set of parameter sets is shown in Eq. (4).

$$CT^{n}_{p_1,p_2} = \{(a,b) \mid a \in [0, |v_1| - 1], b \in [0, |v_2| - 1]\} \quad (4)$$

In Eq. (4), $CT^{n}_{p_1,p_2}$ denotes the set of parameters, a,b denote the constraint moment parameter expression at that moment, and $|v_1|$ denotes the numerical magnitude of the parameter expression. At this time, the particle swarm algorithm is used to obtain the particle velocity formula as shown in Eq. (5) [20]:

$$v_i^t = (v_{i1}^t, v_{i2}^t, \cdots, v_{ik}^t) \quad (5)$$

In Eq. (5), v_i^t represents the total position information of the particle, and $(v_{i1}^t, v_{i2}^t, \cdots, v_{ik}^t)$ represents the particle velocity information at different moments. The position information at this time is expressed as shown in Eq. (6).

$$x_i^t = (x_{i1}^t, x_{i2}^t, \cdots, x_{ik}^t) \quad (6)$$

In Eq. (6), x_i^t represents the total position information of the particle, and $(x_{i1}^t, x_{i2}^t, \cdots, x_{ik}^t)$ represents the position coordinate information of the particle at different moments. In software performance testing, the calculation formula for particle velocity is the core part of PSO algorithm, used to optimize parameter combinations. By adjusting the speed of particles reasonably, the movement speed of particles in the search space can be accelerated, thereby finding the optimal solution faster. This not only improves the efficiency of testing, but also enhances the accuracy of testing. Chromosome combination query refers to evaluating the fitness of chromosomes through fitness functions in genetic algorithms, and selecting chromosomes with high fitness for optimization. This process involves querying and selecting chromosome combinations to generate better combinations of test case parameters. Through these steps, the efficiency and accuracy of test cases can be ensured, and the efficiency and reliability of testing can be improved. Because the position information of the particle in the algorithm calculation is a vector coordinate of a dimension, the position of each coordinate needs to correspond to a specific moment in the dimension to be selected, and the value of the position coordinates at this time is shown in Eq. (7).

$$x_{ij}^{t} \in [0, |v_j| - 1] \tag{7}$$

In Eq. (7), the parameter expression is the same as that described above: The chromosome combination query by initialization adaptation comparing the current parameters completes the extraction of chromosomes; then, at this time, the adaptation is calculated as shown in Eq. (8) [21]:

$$p_i = \frac{s_i}{\sum_{i=1}^{k} s_i} \tag{8}$$

In Eq. (8), p_i denotes the corresponding adaptation value of the chromosome and s_i denotes the calculated probability of the population. The fitness function is used to evaluate the performance of individuals in the search space. By quantifying the proximity of individuals to the objective function, it provides a measurement standard for the algorithm. In software testing, the fitness function can help assess the quality of test cases, such as whether they can cover more functional modules and whether they can detect potential defects. Individuals with higher fitness values have a higher probability of being selected. The genetic algorithm probability calculation formula is given by Eq. (9).

$$s_i = \frac{\sum_{i=1}^{i} p_i}{\sum_{j=1}^{k} p_j} \tag{9}$$

Eq. (9) is shown, and the parameter expression is the same as that above. Through the crossover calculation after the expression of the operator of the above formula there will be a new chromosome pairing, and the crossover probability at this time is indicated by p_c, The crossover method in the method algorithm selects a point crossover through the crossover and then the exchange of the number of gene positions. The dot crossing method generates two offspring chromosomes by selecting a random crossing point and exchanging gene fragments of two parent chromosomes. Gene location plays a crucial role in this process, determining which genes will be exchanged to generate new parameter combinations. By selecting appropriate intersection points and exchanging gene fragments, the point crossing method can effectively explore the search space, generate better combinations of test case parameters, and improve the efficiency and accuracy of testing. Simultaneously, in the algorithm performed by particle updating, as shown in Eq. (10) [22, 23].

$$v_{ij}^{t+1} = v_{ij}^{t}\omega + c_1 r_1 (pBest_{ij}^{t} - x_{ij}^{t}) + c_2 r_2 (gBest_{j}^{t} - x_{ij}^{t}) \tag{10}$$

In Eq. (10), ω denotes the weight value of the inertia factor, c_1, c_2 denote the learning factor, r_1, r_2 denote random numbers in the interval 0-1, $pBest_{ij}^{t}$ denotes the limit of individual values, $gBest_{j}^{t}$ denotes the global limit value, and i, j denotes the iterative updating completed in the first particle. Iterative updating of the particles can achieve optimal selection of the current parameters. Because the data selected in the time selection of software parameters are only valid for a period of time, it is necessary to perform time selection and value constraints before executing the CCTL method. Therefore, the joint algorithm combining value and time is shown in Figure 3.

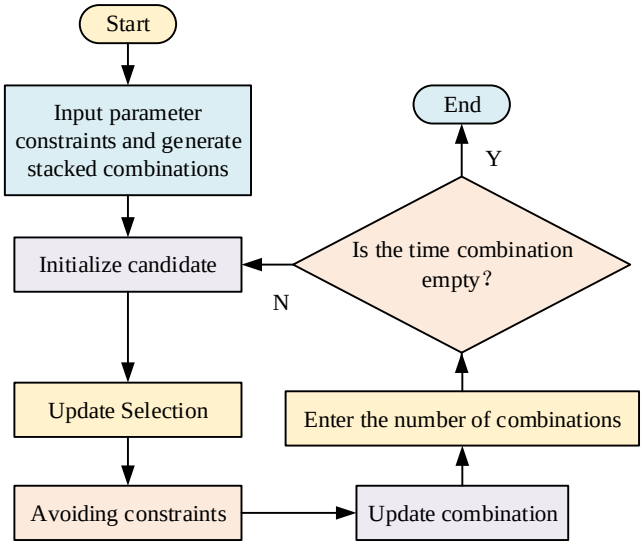

Figure 3: Joint algorithm flow combining value and time

Description: Explains the detailed steps to generate the final coverage combination during testing.

Figure 3 shows that, the algorithm input the parameters to be tested, time selection, and related constraints. Based on the input parameters and constraints, generate preliminary coverage combinations. Then initialize the candidate individual set and generate the final coverage combination through subsequent selection and optimization steps. Secondly, based on the current candidate individuals and coverage combinations, select new candidate individuals to optimize the coverage combination, ensuring that the selected individuals meet the constraint conditions. And when selecting new candidate individuals, it is necessary to ensure that they do not violate the previously set constraints. After selecting eligible candidate individuals, update the target coverage combination, generate the current joint coverage combination, and output the result. Finally, determine whether the currently generated joint coverage combination is an empty set. If it is an empty set, it indicates the end of the algorithm and the final test result can be output. If it is not an empty set, proceed to the next round of candidate initialization and selection. Selecting and optimizing new candidate individuals by evaluating the performance of each individual in the current population, selecting outstanding individuals, and generating new candidate individuals through crossover and mutation operations can ensure that the algorithm can effectively explore new solution spaces and find better solutions. In software performance testing, the joint algorithm combines the advantages of PSO and GA algorithms to optimize parameter combinations and generate high-quality test cases. The process of joint algorithm includes inputting parameters and constraints, generating initial parameter combinations, optimizing parameter combinations, evaluating and updating, and checking termination conditions. Through these steps, the joint algorithm not only improves the efficiency and accuracy of testing, but also adapts to complex testing scenarios, ensuring that the test results can truly reflect the performance of the software under various conditions. After the judgment is completed, the user inputs a new number of joint combinations to prepare for the next round of candidate individual initialization and optimization process. In the initialization of candidate individuals, to facilitate subsequent calculations, needs to be added to the set of values of the parameters through the construction of a simple index relationship to achieve its dimensionality reduction process. The formula is given by Eq. (11) [24, 25].

$$index = i\left|T_i\right| + t_i \tag{11}$$

In Eq. (11), $index$ denotes the index of the computation and $i\left|T_i\right| + t_i$ denotes the Cartesian set of the set composition. Dimensionality reduction is a data processing technique primarily aimed at simplifying data, improving efficiency, and enhancing interpretability. The parameters referred to in dimensionality reduction usually include test case parameters, system performance parameters, and constraint conditions. The idea of the particle swarm algorithm and genetic algorithm is also used in the selection of the combination of the union for the change in the same way as described above. PSO and GA are chosen for parameter information analysis because these two algorithms perform well in dealing with complex parameter optimisation problems, and are able to find the global optimal solution quickly and adapt to different test environments and conditions. The analysis process includes initialising the parameter candidate set, optimising the parameter combinations, evaluating the results and iterative updating. However, the difference lies in the update selection of their algorithms using the update as in Eq. (12) [26].

$$\begin{aligned} v_{ij} &= d_{ij} / \left|T_j\right| \\ t_{ij} &= d_{ij} \% \left|T_j\right| \end{aligned} \tag{12}$$

In Eq. (12), i denotes the first chromosome, j denotes the chromosome position, d_{ij} denotes the gene value size at the j position, and $\%$ denotes the residual value. The parameter analysis of the software test and the moment selection in the CCTL method are completed by generating and overriding the target parameters.

3.4 Analysis of software testing model system

The main purpose of the software testing tool system is to load and parse software models, generate test case parameters and constraint combinations, and import test cases into the testing database. The CCTL method serves as the core logical foundation and generates test cases by introducing time constraints; PSO and GA algorithms are used to optimize the generation process of test cases, improve testing efficiency and coverage. The testing tool system of software mainly exists in the form of components; the main function is to load the model, after completing the parsing to obtain the data information of the protocol, and then to match the strength constraints and other requirements using case analysis. A use case analysis of the software testing tool is shown in Figure 4.

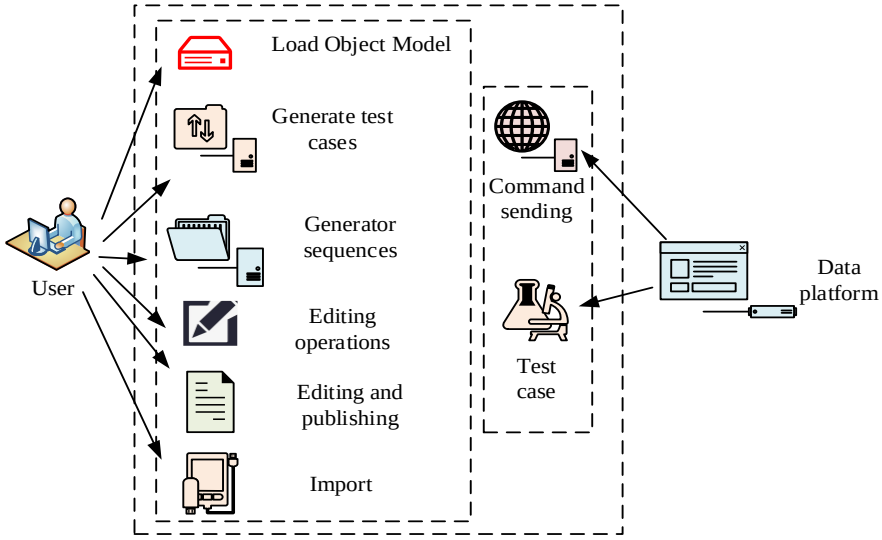

Figure 4: Use case analysis of software testing tools

Description: Demonstrates the use case analysis of user task operations in a software testing system

As shown in Figure 4, in the software testing system, the user needs to load the software model and generate test software case parameters and combinations of constraints and other settings, while inputting the sequence of generation events, editing the current need to release the task operation, edit the current need to release the software testing methods, as well as test cases and other information for the import and export operations. At the same time, the system needs to join the communication protocol to manipulate and receive instructions and send the current software test cases and test cases generated into the database to complete the import and transfer process of the test database components. Through a specific analysis of the system, it is necessary needs to add it to the dynamic analysis system of the model, as shown in Figure 5.

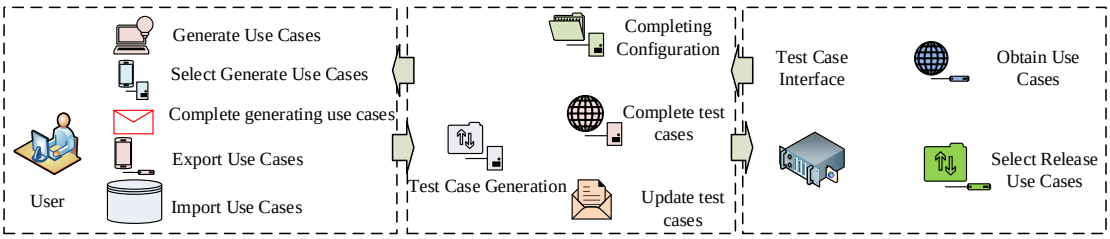

Figure 5: Example of model dynamic analysis

Description: Illustrates how the user analyses the system software or model according to the object model in the testing phase and completes the dynamic model construction

Figure 5 shows that in the testing phase, the user is required to analyze the system software or model in the component according to the object model, and input the information into the parameters of the test case after completing the analysis. Then, the operation is performed after receiving the analysis information of the parameters, including the combination of settings such as choosing the value of the parameter at the moment and setting the current moment and constraints, etc., and then completing the dynamic model. Construction of the dynamic model. Dynamic analysis is aimed at more accurately simulating and evaluating the behavior and performance of software in actual operating environments. The CCTL method generates test cases through time constraints, while dynamic analysis verifies the effectiveness and coverage of these test cases at runtime; PSO and GA algorithms optimize the generation of test cases, and dynamic analysis further verifies the actual effectiveness of these optimized test cases. Simultaneously, in the test, the user needs to deploy the front information in advance as shown in Figure 6 for the parameter sequence image.

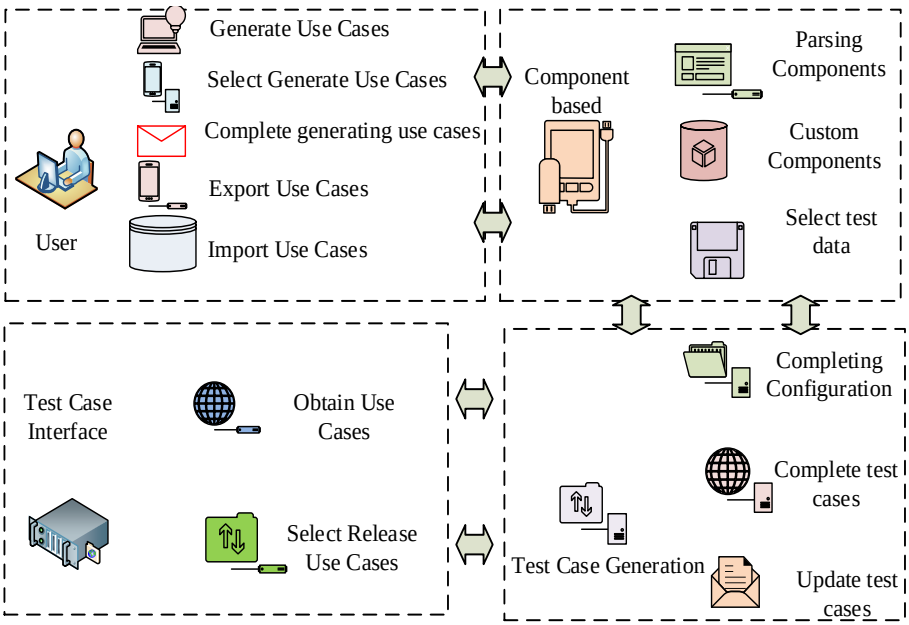

Figure 6: Example of deployment pre parameter sequence process

Description: Describe how to generate test case information before deployment and generate test cases through the CCTL method and algorithm process

As shown in Figure 6, after the deployment of the predecessor information, the test case information is generated, after receiving the generation signal, the predecessor information is read and analyzed to obtain the generated parameter value data and moments, constraints, etc., and the test cases are produced through the CCTL method and algorithmic process, and the user is reminded of the completion of the generation after the end of the generation, and the data information is then exported to be used for other operations, and the test data needs to be constantly updated when the test cases are generated. Parameter selection is based on the time constraints of test case generation requirements and CCTL methods, optimized through PSO and GA algorithms to ensure that test cases can cover all key scenarios. Pre information refers to the initial conditions and contextual information before the generation of test cases, providing necessary background and initial conditions for the generation of test cases. The pre deployment parameter sequence process is an important part of the testing process, which directly affects the generation of test cases and the accuracy of test results. Simultaneously, it is necessary to updat the use cases when generating test data. The entire software testing system platform is shown in Figure 7.

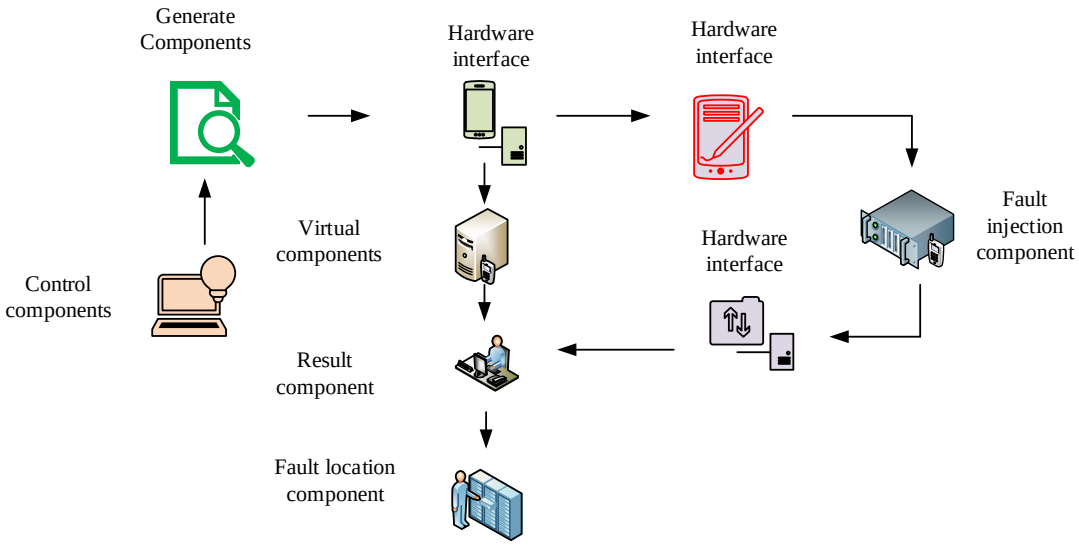

Figure 7: System platform for software testing

Description: Outline how the test system through the CCTL method and particle swarm algorithm genetic algorithm to generate and distribute software test case information

As shown in Figure 7, the main idea in the testing system is to generate and distribute the use case information for software testing through the main idea of the CCTL method and the particle swarm algorithm genetic algorithm. Therefore, in the general framework of the system, the test system includes a test case, hardware interface, virtual comparison model, result comparison and fault location components. The simulation to be tested included a hardware interface component and a fault injection component. The control component is responsible for controlling the testing process and execution. Generate components that can use different generated test cases. Virtual components can create tested system components through virtualization methods. The hardware interface can connect the generated test cases and virtual components with the actual hardware through the hardware interface. The result component is capable of collecting and processing test results for comparison. The fault location component can locate and analyze faults in the system. The hardware interface will conduct hardware level testing through the hardware interface. The fault injection component is mainly used to introduce faults into the system, verify the system's fault handling capability and robustness. The fault detection and analysis component is responsible for analyzing the test result data collected during the testing execution phase, comparing the actual test results with the expected results, and detecting whether there are deviations or abnormal behaviors. The fault location component uses parameters generated by the CCTL method, and utilizes these parameters and constraints to more accurately locate the time and location of the fault occurrence. The fault injection component is responsible for introducing predefined faults during the testing process, verifying the system's fault handling capability and stability, and injecting faults at specific time points and conditions based on the parameters generated by the CCTL method. During software testing, control components to generate test cases and distribute them to the generating components. Generate test cases using CCTL method and PSO/GA algorithms, and transmit them to virtual components. Virtual components interact with actual hardware through hardware interfaces and apply test cases to the system under test. During the testing process, the result component collects and processes the test results. The test results are transmitted to the fault location component for fault analysis and localization. Finally, the hardware interface is also used to interact with the fault injection component, introducing faults into the system to verify its stability and robustness. The higher the error detection rate, the more defects are discovered in the test case set, and the better the testing effect. The calculation formula is error detection rate = (number of discovered defects/total number of introduced defects) × 100%. The criteria for determining the coverage of a model are its functional coverage, with a functional coverage of ≥90%, a state coverage of ≥85%, a transition coverage of ≥80%, and a path coverage of 75%. These all indicate that the current model has good coverage. However, the study only uses the functional coverage of the model as the criterion for judgment, so a functional coverage rate of ≥90% is considered to be better for the current model. Functional coverage can directly reflect the extent to which test cases cover software functionality, and high coverage can help identify potential defects. A functional coverage rate of ≥ 90% can significantly improve the accuracy and reliability of testing, covering most key functions and ensuring the practicality and reference value of research results. The test case generation module generates test cases that meet time constraints and functional coverage requirements based on CCTL method and PSO/GA algorithm. The virtual comparison model simulates the actual operating environment through virtualization technology, executes test cases, and dynamically adjusts configurations. Compare the accuracy and completeness of the verification test results in the module, and generate a test report. Fault location module analysis report, accurately locate the fault. The hardware interface module and fault injection module respectively support the execution of test cases on actual hardware and fault injection, ensuring system stability and reliability.

4 Results and discussion

4.1 Results

Research the use of an online shopping system program for software testing. Firstly, the system needs to verify user login, product search, shopping cart functionality, and order payment. Next, configure the web server and database server, and set up development, testing, and production environments. Write test cases for user login, product search, addition, deletion, quantity modification, and order payment of shopping cart items. At the same time, the software uses PSO and GA to generate multiple input combinations during testing, ensuring coverage of all testing scenarios. And use automation tools such as Selenium to perform functional testing of the web interface, use JMeter for performance testing, and use OWASP ZAP for security testing. Finally, record the execution status of all test cases and analyze the failed cases. Use Selenium for automated testing of user interfaces, as it supports multiple browsers and operating systems and can simulate real user operations. Use JMeter for performance testing as it can simulate high concurrency scenarios and evaluate the system's response time and throughput. The population size is set to 30, the maximum number of iterations is 50, the learning factors are 2.05, and the inertia weight is 0.9. For the GA population size of 100, the maximum number of iterations is 20, the crossover rate is 0.8, and the mutation rate is 0.01. The experimental environment for the research includes the use of Intel Core i7-9700 CPU, 32GB RAM, and Windows 10 Professional 64 bit operating system, developed using MATLAB R2020b and Python 3.8, and automated testing, performance evaluation, and security checks implemented using tools such as Selenium, JMeter, and OWASP ZAP. The testing platform takes an online shopping system as an example, covering functional modules such as user login, product search, shopping cart operation, and order payment. The evaluation indicators

include functional coverage (≥ 90%), state coverage (≥ 85%), transition coverage (≥ 80%), and path coverage (≥ 75%), as well as error detection rate, algorithm performance, and stability. The experimental design covers the complexity stratification of test cases, parameter settings for PSO and GA, number of repeated experiments, and verifies the significance of the results through charts and statistical analysis. Generate a test report, record the discovered defects, and ultimately complete software testing. When conducting software testing, it is necessary to ensure the stability and reliability of the system under various input conditions. Simultaneously, it is necessary to include functional testing, performance testing, and security testing. And use automated testing tools to perform testing. Regularly conduct code reviews and test case reviews, and use static analysis tools to check code quality. Randomly select 50 software test data from the currently selected ones for testing. The selection of 50 test data is based on statistical sample size calculation, ensuring that the error range is within 10% at a 95% confidence level. This sample size can meet the testing requirements while also being within resource constraints. Complexity refers to the comprehensive difficulty of test cases in terms of functional coverage, operational steps, data input, and expected results. According to the complexity of the test cases, they are divided into three layers: simple, medium, and complex. Each layer selects 20%, 50%, and 30% of the test cases. The goal of this test is to verify the performance and stability of the system under high load conditions. The testing scope includes the login module, data processing module, and report generation module. The main method of calculating the coverage is obtained by comparing the actual number of covered combinations with the target number of combinations, so that in the coverage test using CCTL is obtained as shown in Table 1. The target coverage combination refers to the coverage of all possible parameter combinations and functional modules, while the actual coverage combination refers to the parameter combinations and functional modules actually covered through test cases. Use case coverage focuses more on the coverage of user scenarios and interactions, focusing on how users use the software. Functionality coverage focuses more on the coverage of the software's functionality, and is concerned with the specific operations that the software can perform. The software being tested refers to the user login module of the online shopping system. The "CA (33, 2; 2)" in Table 1 indicates that under three constraint conditions, the target combination quantity is 80, and the actual number of combinations covered is 78.

Table 1: Analysis of CCTL method testing software coverage

Software under testing	Number of constraints	Number of target combinations	Number of target combinations after unconstrained	Constrain the number of target combinations before validation
CA (33, 2; 2)	3	80	78	78
CA (33, 2; 2)	6	125	135	135
CA (53, 2; 2)	6	224	224	224
CA (53, 2; 2)	3	600	456	456
CA (83, 2; 2)	2	900	894	894
CA (83, 2; 2)	10	1654	1645	1645

Note: Shows the number of combinations of objectives under different number of constraints, etc. CA stands for 'Clock-Controlled Computation Tree Logic Algorithm'

As shown in Table 1, when the coverage rate of software testing is analyzed, it is found that the target array and the number of constraints after removing the constraints have the same value as the number of coverage combinations verified, which is visible of the software testing. The use of the method is able to achieve 100% coverage, which shows that the current method of testing the different software is able to achieve the number of constraints and time of the selection of software constraints; at the same time, the high coverage rate indicates that the method of testing is better. The similarity between the number of unconstrained combinations and the number of combinations before validation is due to the fact that the test case generation algorithm has already considered most of the constraint conditions when generating test cases. After removing the constraints, if the actual number of generated test case combinations does not change significantly, then the test coverage rate can reach 100%. To compare the generation time and data size of the current system method for software testing, the number of populations and the number of iterations were set to 150 populations and 20 iterations, respectively, as shown in Figure 8. When generating test cases, the average time for particle swarm optimization algorithm to generate test cases is longer than that of genetic algorithm. Therefore, it is said that the test cases generated by the particle swarm algorithm require a longer period of time.

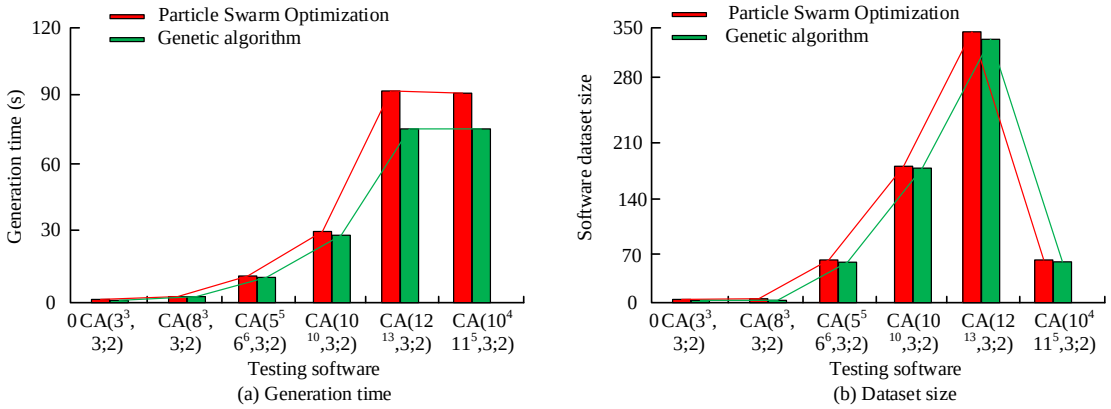

Figure 8: Comparison of testing time and data scale for different system software

Description: Compare the current system approach with other algorithms in terms of software test generation time and data size

Figure 8(a) shows that in the analysis of the CCTL method of the two algorithms for software testing generation time is different from the genetic algorithm method in the generation of the moment of generation of the event is significantly higher than the particle swarm algorithm, which indicates that in the generation of software data processing particle swarm algorithm of the generation of a longer period of time to test the software is significantly faster than the genetic algorithm, as can be seen from Figure 8(b) particle swarm algorithm of the From Figure 8(b), it can be seen that the particle swarm algorithm is significantly higher than the genetic algorithm in generating the number of coverage combinations, which indicates that the size of the data generated by it is larger, and the CCTL method is more important to generate the data coverage combinations, while the genetic algorithm is more important to generate and analyse the time moments. To test the effect of the current build system test software, the software parameter position information, speed information, analog, and other parameters to test, as shown in Table 2. The target parameter speed reflects the speed at which the software processes data or tasks, while the azimuth velocity reflects the efficiency of the system in multi-dimensional task scheduling. These parameters directly affect testing efficiency and accuracy. Higher speeds and azimuth velocities can improve testing efficiency. At the same time, by precisely controlling the parameter range, it can better simulate actual scenarios and improve the accuracy of test results. The maximum and minimum values of the target distance are based on the possible maximum and minimum shopping cart distances in the system. The injection time is the time interval used to simulate actual user operations, ensuring that test cases can cover system behavior at different time points.

Table 2: Test results of parameter position information, speed information, and analog parameters

Attribute name	Maximum value	Minimum value	Injection time
Target distance (m)	10000	0	[0 ms, 100 ms]
Target coordinates X (m)	10000	0	[0 ms, 100 ms]
Target coordinates Z (m)	10000	0	[0 ms, 100 ms]
Target coordinates Y (m)	10000	0	[0 ms, 100 ms]
Target direction speed (rad/s)	20	0	[150 ms, 250 ms]
Target speed (m/s)	80	0	[150 ms, 250 ms]
A1	12	0	[0 ms, 10 ms]
A2	12	0	[0 ms, 10 ms]
B1	15	0	[30 ms, 40 ms]
B2	15	0	[30 ms, 40 ms]
C1	28.5	0	[50 ms, 60 ms]
C2	28.5	0	[50 ms, 60 ms]

Note: Lists the software parameter information in the test

Table 2 shows that the size of the maximum and minimum values in the target position of the parameter are the same, the maximum value is 10000 m, the minimum value is 0 m, and the range of the injected moments of the position are in [0 ms, 100 ms]. The range of the moment of the position information of the particle parameter are in

[150 ms, 250 ms], the maximum value of the velocity of the target parameter is 80 m/s, and the minimum value is 0 m/s. The maximum value of the azimuthal velocity of the target is 20 rad/s, and the minimum value is 0 rad/s. The maximum value of the analogue of the software test and the different analogues are different but the minimum value is 0, and at the same time, their moment ranges are not the same. This shows that when the moment and parameter determinations are made for the software, the values of the parameters are different but some of the parameters have the same range of values. In order to verify the effectiveness of the current CCTL method for software testing, the test cases are set to 4700, with 100% coverage, using 46 parameters for analysis, and the system is analysed and tested using the big data platform, obtaining a partial test result graph as shown in Table 3. Safety value is an indicator to assess the performance and stability of software during testing, which is used to determine whether there is any potential security risk in the software. Fault information location is used to determine the exact location and cause of faults in the software by analysing the test results, helping developers to quickly diagnose and fix problems.

Table 3: Software test data results

Combination test data results
2100 16.907 14.25 2100 16.907 14.25 2100 16.907 14.25
2101 16.574 14.25 2101 16.574 14.25 2101 16.574 14.25
2102 14.00 15.75 2102 14.00 15.75 2102 14.00 15.75
2103 14.00 0.75 2103 14.00 0.75 2103 14.00 0.75
2104 17.702 7.291 2104 17.702 7.291 2104 17.702 7.291
2105 24 -5.162 2105 24 -5.162 2105 24 -5.162
2106 17.707 5.52 2106 17.707 5.52 2106 17.707 5.52
2107 17.207 14.25 2107 17.207 14.25 2107 17.207 14.25
2104 24 - 2104 24 - 2104 24 -
2104 14.25 - 2104 14.25 - 2104 14.25 -
2107 17.207 12 2107 17.207 12 2107 17.207 12
2103 3.5345 6.471 2103 3.5345 6.471 2103 3.5345 6.471
2107 17.207 -0.75 2107 17.207 -0.75 2107 17.207 -0.75
2108 2 -15.75 2108 2 -15.75 2108 2 -15.75
2109 10.327 0 2109 10.327 0 2109 10.327 0

Note: Provides the results of combined test data, showing the test results of different combinations of test cases

As shown in Table 3, the results of the software testing in the simulation test and fault results of the analysis and testing, and then in the results of the test to obtain the safety value, obtain the safety value of the test software fault information positioning, the test fault positioning in the results of the analysis of the region to display, and then the positioning results obtained and the simulation of the data parameters for the combination of the results of the positioning and fault results for the combination of the analysis of the results of the conclusions obtained were used to reflect the test software use cases and parameters of the direct combination of coverage and software testing. The above test results show that the current method can be tested and analyzed on the parameter combinations, and therefore verify the feasibility of the software testing method. Table 3 shows the actual test data results, which are generated based on the parameters defined in Table 2. Each set of data in Table 3 corresponds to a different combination and value of the parameters defined in Table 2. For example, the data set 2100 may indicate that for a particular combination of parameters, the test result is 16.907, 14.25, etc. The data set 2101 corresponds to the test result for another combination of parameters, and so on. The first set of data represents the execution results of the test cases. The second set of data represents performance indicators such as response time, throughput, etc. The third set of data represents the results of safety or fault detection. Therefore, the data is divided into three groups. In analysing the test results, the data were first pre-processed and then statistical metrics such as mean, standard deviation and variance were calculated for each algorithm. And key performance indicators were also calculated and finally visualised and analysed using bar charts and line graphs. In comparing the test results of different algorithms, LSTM and LAR are chosen as benchmark algorithms. To ensure the fairness of the comparison, all algorithms are tested on the same test dataset and test environment. In addition, the study verified whether the differences between the different algorithms were statistically significant using statistical methods such as t-test and ANOVA. To test the accuracy of the test parameters of the current method, the test results of the parameters were analyzed and compared with the test results of other algorithms, namely, long short-term neural networks (LSTM), and logistic regression algorithm (LRA) as shown in Figure 9. LSTM can help identify and predict potential issues in software testing by analyzing system logs, user behavior data, and historical

testing data, thereby improving testing coverage and efficiency. LSTM and LRA were chosen as comparison algorithms for the study because LSTM has a significant advantage in dealing with time series data by capturing long-term dependencies in the data. LRA was chosen because the algorithms provide a simple linear benchmark that helps to understand the underlying linear trends in the test data. LSTM is effective in analysing parameter variations in the test, while LRA provides a simple linear benchmark that helps to understand the underlying linear trends in the parameters.

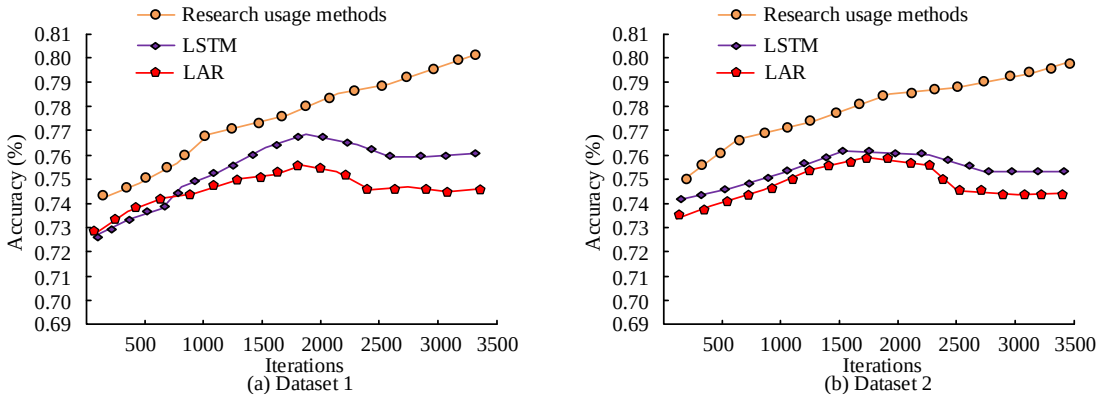

Figure 9: Comparison of software testing accuracy with different algorithm logics added
Description: Demonstrates the change in algorithm accuracy after adding different algorithm logic

As shown in Figure 9, after adding different algorithmic logic to the CCTL, the accuracy of the LSTM and LAR algorithms first increases with the number of iterations and then tends to stabilize, while the accuracy of CCTL combines PSO and GA approaches is in the increasing stage. Accuracy continues to increase over time, showing better learning ability and adaptability. This indicates that the research usage method is more effective in testing the accuracy of software, while the other two algorithms have the highest accuracy, at 76% and 75% respectively. The accuracy of the research usage method is higher, at 5% and 6% higher than the LSTM and LAR methods, respectively. To test the algorithmic stability of the proposed method, the loss functions of the three algorithms were compared and tested, as shown in Figure 10.

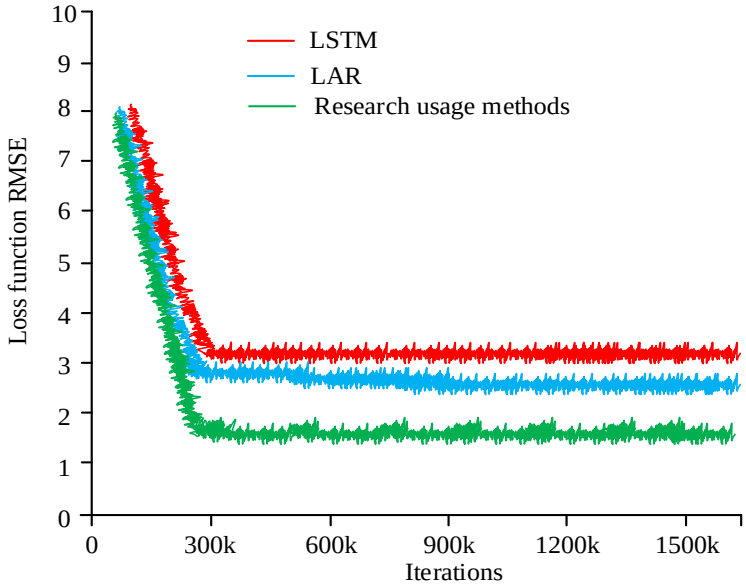

Figure 10: Comparison of loss functions among three algorithms
Description: compares the change in loss function of the three algorithms during the iteration process

Figure 10 shows that the loss function of the three algorithms decreases and then gradually stabilizes throughout the algorithm as the number of iterations increases. The loss function value of the research use method dropped to a minimum of 1.5 loss function at 300 k iterations, the LSTM algorithm dropped to a loss function of 3 at 300 k, and the LAR algorithm dropped to a minimum loss function of 3.2 at 320 k iterations. Loss function is an important tool in machine learning to measure the prediction error of a model, the smaller its value the smaller and more stable the model error. The research use model loss function value drops to 1.5 after

300,000 iterations and outperforms other models, which indicates that the research use model learns data features better during training, makes more accurate predictions, and performs better when dealing with complex data. This shows that the algorithmic logic of the research method has the smallest loss function value, and the algorithmic model is more stable. To test the effectiveness of different algorithm models in software testing, a comparative analysis was conducted on the Pairwise Testing (PT) model, Ant Colony Optimization (ACO), Reinforcement Learning Algorithm (RLA), and Particle Swarm Optimization (PSO) algorithm models, as shown in Table 4.

Table 4: Comparison of test results of different algorithm software

/	Test result					
Model	ACO	PT	PSO	GA	RLA	CCTL
Error detection rate (%)	94.35	92.65	91.54	91.58	92.35	96.54
Functional coverage (%)	92.61	91.26	90.36	91.32	92.67	94.85
Status coverage rate (%)	88.67	89.65	93.51	91.54	89.35	94.68
Conversion coverage (%)	81.26	90.35	92.54	84.65	83.51	93.59
Path coverage (%)	82.36	83.65	86.57	89.65	90.13	91.35

Note: Compares the performance of different algorithm models in software testing

From Table 4, it can be seen that after using different algorithms for software testing, it is evident that the CCTL model performs better in terms of testing results. In the comparison of software testing performance, the error detection rate of the CCTL model reached the highest level of 96.54%, which is about 5.00% higher than that of the PSO model. In the comparison of algorithm coverage, the CCTL model achieved good coverage levels under different coverage tests, with the model reaching the highest value of 94.85% in functional coverage analysis. It can be seen that the coverage and error detection rate of the model used in the study have a high level, and the actual software testing effect of the model is good. Unified testing environment, selection of unified performance indicators, and data alignment for performance research of different algorithms that have not been tested. Statistical tests show that the CCTL model performs the best on all indicators. Error detection rate is a key indicator for measuring the performance of software testing models, reflecting the model's ability to detect defects. A high error detection rate means that the model can more effectively detect software defects, thereby improving software quality and reliability. The high error detection rate of CCTL model indicates that CCTL combined with PSO and GA can generate high coverage test cases more comprehensively, effectively reducing software failure risks and improving user satisfaction. When comparing algorithm coverage, the CCTL model performed well in the function, state, transition and path coverage tests. This indicates that the CCTL model, by introducing time constraints and optimising parameter combinations, can generate high-coverage test cases more comprehensively and effectively improve the comprehensiveness and accuracy of software testing. This indicates that the CCTL model combined with PSO and GA can more effectively generate high coverage test cases, resulting in better performance in detecting defects and covering software features. Table 5 shows a summary comparison of the test results of the different methods.

Table 5: Summary comparison of tests for different methods of testing

Method	Accuracy	Efficiency	Coverage Rate	Capability to Handle Large Datasets	Real-Time Parameter Constraint Handling Capability
Bat Search Algorithm	72.35%	Moderate	79.84%	Limited	Weaker
Random Forest	88.67%	High	88.67%	Better	Average
PSO	91.32%	High	90.36%	Better	Strong
GA	90.36%	Moderate	90.47%	Average	Strong
Proposed Method (CCTL+PSO/GA)	5%-6% higher than PSO/GA	High	100%	Strong	Very Strong

Note: Comparison of test results of different methods

As can be seen from Table 5, the CCTL+PSO/GA method used in the research is significantly better than other methods in different test results, which has the strongest ability to deal with large-scale datasets, and the best ability to deal with real-time parameter constraints, which suggests that the model used in the research has a better practical application in the testing of different methods.

4.2 Discussion

In Raamish et al.'s study, in order to test the quality and performance of the software and improve its reliability, a ramp up algorithm based on LSTM and BrainStorm optimization and post acceptance was used. The new algorithm can be used for software fault detection. Compared with traditional methods, the new method can effectively improve the effectiveness of software detection [27]. However, this method only analyzes and detects faults and defects detected by software, and cannot improve the performance and stability of software detection. Oleshchenko's research found that software testing can consume a significant amount of cost and time during software development. However, using the KNN algorithm to train and test software development data can greatly reduce software testing time and cost. From this, it can be seen that using clustering algorithms for software data analysis and cost control is an important direction in the software testing process [28]. Build a new model for software testing in this study. After 300,000 iterations, the PSO algorithm generates a significantly higher number of test case coverage combinations than the GA algorithm. The PSO algorithm achieves a functional coverage of 90.36% compared to the GA algorithm's 91.32%.The PSO algorithm achieves an error detection rate of 91.54%, which is slightly lower than that of the GA algorithm's 91.58%.The CCTL method is able to, through the introduction of temporal constraints more accurately simulate the temporal characteristics of the software in actual operation. Meanwhile, the CCTL method is better at generating high-coverage test cases after combining the PSO and GA algorithms. When comparing the software production time of models, particle swarm optimization algorithm has a higher coverage combination than genetic algorithm in testing the performance of models with different iteration times. This indicates that particle swarm optimization algorithm has better performance in data coverage combination processing and higher performance in improving the coverage combination set of the model. Compared with KNN, CCTL is more efficient in dealing with complex time series data; compared with LSTM, CCTL is more flexible in dealing with nonlinear data; compared with BrainStorm optimisation, CCTL is more comprehensive in dealing with complex test scenarios. These advantages enable the CCTL method to demonstrate higher accuracy and efficiency in software performance testing, while reducing cost and improving software reliability. This may be because particle swarm optimization algorithms are easier to integrate data. When comparing the accuracy of different algorithm models, the study found that the accuracy of LSTM and LAR models were 5% and 6%, respectively. This may be due to the combination of genetic algorithm and particle swarm optimization algorithm used in the study. The types of constraints selected for research are closely related to the actual situation, aiming to reflect the various scenarios that software testing may encounter and ensure that test cases cover all key operations and interactions of the software. Although the experimental results showed high test coverage and accuracy, some test cases failed due to parameter configuration errors, environmental dependencies, resource limitations, concurrency conflicts, algorithm limitations, or insufficient test data. This indicates that the test cases further reveal the shortcomings of the testing strategy and algorithm performance, and points out the necessity of enhancing the robustness of the test cases. In software performance testing, PSO and GA optimization algorithms significantly improve the comprehensiveness and accuracy of testing by generating high coverage test cases. Algorithms not only improve testing efficiency and reduce resource consumption, but also lower testing costs. In addition, high-quality test cases generated by algorithm models can effectively detect software defects, improve software reliability and stability.

When comparing the loss functions of different models, the changes in the loss functions of the models show a trend of first decreasing and then approaching equilibrium. This may be because as the number of iterations increases, the functional loss of the model also increases, but when the model reaches a certain value, it begins to stabilize. The loss function value of the model used in the study is relatively small, which may be due to the improved algorithm performance after adding different algorithms to the model. When comparing the performance of different models, it was found that the algorithm using this model performed better, with the highest error detection rate of 96.54%. Compared with other algorithms, the algorithm used in this study has higher coverage and error rates. This may be due to the current algorithm model performing better in software testing. Has shown certain efficiency and accuracy in selecting regression test cases, but has limitations in handling large-scale datasets and real-time parameter constraints. The current research method uses PSO and GA optimization algorithm logic to significantly improve test coverage and outperforms bat search algorithm in terms of functional coverage. The random forest algorithm performs the best in predictive performance, but its real-time parameter processing ability in software testing is average. The method currently used in the research not only has a 5% -6% higher accuracy, but also demonstrates a very strong ability to handle real-time parameter limitations. PSO is faster in some cases, mainly because it can quickly find the optimal solution through the collaboration of individuals and groups. Meanwhile, GA simulates the genetic and mutation mechanisms in natural selection, making it suitable for global search, but its convergence speed may not be as fast as PSO. By combining the advantages of PSO and GA, the study not only improved the coverage and accuracy of testing, but also maintained efficient testing performance in environments with real-time parameter changes. It also provides new ideas and tools for software testing in more complex environments in the future. In future research, researchers may further explore how to optimize PSO and GA algorithms to improve performance in specific situations. At the same time, the proposed methods may stimulate the development of new testing methods, and with the improvement of algorithm performance, future software testing may become more automated and

intelligent, thereby improving testing efficiency and reducing costs.

In summary, in the comparison of different algorithm models, it is found that the use of models has better software testing performance. At the same time, the use of algorithms in software testing can effectively analyze and test software, and its effect can reach

5 Conclusion

This research mainly focuses on the current problem of lack of stability and performance of software testing, and proposes a new software testing system based on the CCTL method, first analyzing the component use case generation of software testing. Subsequently, the system is transformed and analyzed using the particle swarm algorithm and genetic algorithm logic, and a system model is built for software testing. The method used in the study is based on CCTL combined with PSO and GA. The loss function value of the method is significantly lower than other methods, indicating that it can better learn data features during the training process and make more accurate predictions. The new methodology achieves 100 per cent test coverage. Meanwhile, the new method effectively selects the number of constraints. In terms of algorithm performance, GA excels in generating test moments, while PSO is more advantageous in handling complex parameter combinations and large-scale data. These advantages enable the new method to excel in generating high-coverage test cases, significantly improving the comprehensiveness and accuracy of the tests. During the testing process, parameter settings significantly affect the generation of test cases and test results. The range of values of the algorithm parameters has a significant effect on the test results. For example, an increase in the number of particles and population size can enhance the search ability but increase the running time; an increase in the learning factor and crossover rate can make the search direction clearer but may fall into the local optimum. By reasonably adjusting these parameters, the speed and coverage of test case generation can be optimised and the test results can be improved. In addition, the effect of different parameter settings on the testing effect showed that the parameter values were different, but the range of values is similar. Simultaneously, the new method can effectively obtain the safety value and fault location information of the software. Meanwhile, the accuracy of other traditional methods is as high as 76% and 75%, whereas the accuracy of the research use method is higher, outperforming the accuracy of the LSTM and LAR methods by 5% and 6% respectively. Also the loss function of the algorithm used in the study is 1.7 and 1.5 lower than the loss functions of the other two algorithms, which shows that the method is more stable. From this, it can be seen that researching usage methods has better testing effects in software testing, and at the same time, studying the testing error rate, coverage rate, and accuracy of using models in different model comparisons has a high level. It can be seen that although this study has achieved a lot of results, it still needs to be improved, first of all, the algorithm needs to be further improved subsequently when it is constrained to the combination of cases, and the data used for the study was small, so larger data sets will need to be analysed in subsequent studies. The current research is mainly conducted in specific testing scenarios and may not fully cover all complex situations in practical applications. Therefore, further research is needed to expand the testing scenarios to verify the universality of the method. Although the methods used in the study have shown improvements in accuracy and stability compared to other methods, their stability and accuracy still need further validation in larger scale data and more diverse testing environments. The PSO and GA algorithms in the current study have limitations although they perform well. Therefore, future research will improve these algorithms by introducing hybrid strategies, dynamically tuning the parameters and optimising the cross-variance operation. The current research focuses on specific scenarios such as online shopping system, data processing module and report generation module. Future work will expand the test scenarios to validate the stability and accuracy of the new methods using larger scale data and more diverse test environments.

Data availability statement

The datasets used and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Conflicts of interest

The authors declare that this article is free of any conflicts of interest.

Author contributions

All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Yuan Sun, Md Gapar Md Johar, Jacquline Tham. The first draft of the manuscript was written by Yuan Sun, Md Gapar Md Johar, Jacquline Tham. All authors read and approved the final manuscript.

Funding

No funding was received.

References

[1] Alhroob A, Alzyadat W, Imam A T, Jaradat Ghaith M. The genetic algorithm and binary search technique in the program path coverage for improving software testing using big data. Intelligent Automation and Soft Computing, 2020, 26(4): 725-733. http://dx.doi.org/10.32604/iasc.2020.010106

[2] Han X, Yu T, Yan G. A systematic mapping study of software performance research. Software: Practice and Experience, 2023, 53(5): 1249-1270. https://doi.org/10.1002/spe.3185

[3] Murthy M S N, Suma V, Chandrappa C N, Shankar M M. Factors influencing effectiveness of testing applications in cloud using regression testing: a

statistical analysis. International Journal of Advanced Intelligence Paradigms, 2020, 17(1/2): 109-126. https://doi.org/10.1504/ijaip.2020.108770

[4] Guo S, Wang J, Xu Z, Huang L. Feature transfer learning by reinforcement learning for detecting software defects. Software: Practice and Experience, 2023, 53(2): 366-389. https://doi.org/10.1002/spe.3152

[5] Chen J. Construction and Application of an Economic Intelligent Decision-making Platform Based on Artificial Intelligence Technology. Informatica. 2024, 48(9). https://doi.org/10.31449/inf.v48i9.5705

[6] Kaur A, Agrawal A P. Performance comparison of Bat search and Cuckoo search using software artefact infrastructure repository and regression testing. International Journal of Advanced Intelligence Paradigms, 2021, 18(2): 99-118. https://doi.org/10.1504/IJAIP.2021.112899

[7] Chen J, Hu H, Yu D. Characterising and detecting methods to be benchmarked under performance unit test. International Journal of Software Engineering and Knowledge Engineering, 2022, 32(9): 1279-1305. https://doi.org/10.1142/S0218194022500486

[8] Khurshid S, Iqbal J, Malik I A, Yousuf B. Modelling of NHPP based software reliability growth model from the perspective of testing coverage, error propagation and fault withdrawal efficiency. International Journal of Reliability, Quality and Safety Engineering, 2022, 29(6): 10-28. https://doi.org/10.1142/s0218539322500139

[9] Qian J, Zhou X, Zhou H. Prioritising test scripts for the testing of memory bloat in web applications. IET Software, 2022, 16(3): 317-330. https://doi.org/10.1049/sfw2.12057

[10] Bugden W, Alahmar A. The safety and performance of prominent programming languages. International Journal of Software Engineering and Knowledge Engineering, 2022, 32(5): 713-744. https://doi.org/10.1142/S0218194022500231

[11] Jiang M, Chen T Y, Wang S. On the effectiveness of testing sentiment analysis systems with metamorphic testing. Information and Software Technology Information and Software Technology, 2022, 150(10): 1-11. https://doi.org/10.1016/j.infsof.2022.106966

[12] Hao T, Elith J, Lahoz-Monfort J, Guillera-Arroita G. Testing whether ensemble modelling is advantageous for maximising predictive performance of species distribution models. Ecography, 2020, 43(4): 549-558. https://doi.org/10.1111/ecog.04890

[13] Hosseini S M J, Arasteh B, Isazadeh A, Mohsenzadeh M, Mirzarezaee M. An error-propagation aware method to reduce the software mutation cost using genetic algorithm. Data Technologies and Applications, 2021, 55(1): 118-148. https://doi.org/10.1108/DTA-03-2020-0073

[14] Zeb A, Din F, Fayaz M, Mehmood G, Zamli K Z. A systematic literature review on robust swarm intelligence algorithms in search-based software engineering. Complexity, 2023, 2023(1): 4577581-4577582. https://doi.org/10.1155/2023/4577581

[15] Pan R, Ghaleb T A, Briand L. Atm: Black-box test case minimization based on test code similarity and evolutionary search. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023, 14(5): 1700-1711. https://doi.org/10.48550/arXiv.2210.16269

[16] Omar H K, Frikha M, Jumaa A K. Improving Big data recommendation system performance using NLP techniques with multi attributes. Informatica. 2024, 48(5). https://doi.org/10.31449/inf.v48i5.5255

[17] Sornkliang W, Phetkaew T. Performance analysis of test path generation techniques based on complex activity diagrams. Slovenian Association Informatika, 2021, 45(2): 231-242. https://doi.org/10.31449/inf.v45i2.3049

[18] Bednárek D, Kruliš M, Yaghob J. Letting future programmers experience performance-related tasks. Journal of Parallel and Distributed Computing, 2021, 155(C): 74-86. https://doi.org/10.1016/j.jpdc.2021.04.014

[19] Ke S Z, Huang C Y. Software reliability prediction and management: a multiple change-point model approach. Quality and Reliability Engineering International, 2020, 36(5): 1678-1707. https://doi.org/10.1002/qre.2653

[20] Bi W, Yu F, Cao N. Wei H, Cao G, Han X, Sun L, Higgs R. Research on data extraction and analysis of software defect in IoT communication software. Computers, Materials, and Continuum, 2020, 65(2): 1837- 1854. https://doi.org/10.32604/cmc.2020.010420

[21] Li L, Xu L, Cui H, Abdelkareem M A A. Validation and optimization of suspension design for testing platform vehicle. Shock and Vibration, 2021, 2021(7): 1-15. https://doi.org/10.1155/2021/7963517

[22] Shao Y, Liu B, Wang S, Xiao P. A novel test case prioritization method based on problems of numerical software code statement defect prediction. Eksploatacja i Niezawodnosc - Maintenance and Reliability, 2020, 22(3): 419-431. https://doi.org/10.17531/ein.2020.3.4

[23] Serat Z, Fatemi S A Z, Shirzad S. Design and economic analysis of on-grid solar rooftop PV system using PVsyst software. Archives of Advanced Engineering Science, 2023, 1(1): 63-76. http://dx.doi.org/10.47852/bonviewAAES32021177

[24] Garmaki M, Gharib R K, Boughzala I. Big data analytics capability and contribution to firm performance: the mediating effect of organizational learning on firm performance. Journal of Enterprise Information Management, 2023, 36(5): 1161-1184. https://doi.org/10.1108/JEIM-06-2021-0247

[25] Jalil S, Rafi S, LaToza T D, Moran K, Lam W. Chatgpt and software testing education: Promises & perils. In2023 IEEE international conference on software testing, verification and validation workshops (ICSTW), 2023, 16(6): 4130-4137. https://doi.org/10.48550/arXiv.2302.03287

[26] Weber M, Kaltenecker C, Sattler F, Apel S, Siegmund N. Twins or false friends? a study on energy consumption and performance of configurable

software. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023, 14(5): 2098-2110. https://doi.org/10.1109/ICSE48619.2023.00177

[27] Raamesh L, Jothi S, Radhika S. Enhancing software reliability and fault detection using hybrid brainstorm optimization-based LSTM model. IETE Journal of Research, 2023, 69(12): 8789-8803. https://doi.org/10.1080/03772063.2022.2069603

[28] Oleshchenko L. Software testing errors classification method using clustering algorithms. International Conference on Innovative Computing and Communication, 2023, 17(10): 553-566. https://doi.org/10.1007/978-981-99-3315-0_42

Appendix

Item	Details
Experimental Purpose	To evaluate the performance of the software performance testing system based on Clock-Controlled Computation Tree Logic (CCTL)
Experimental Environment	Intel Core i7-9700 CPU @ 3.20GHz, 32GB RAM, Windows 10 Professional 64-bit
Software Tools	MATLAB R2020b, Python 3.8 with necessary libraries
Experimental Steps	Setup environment, write test cases, apply PSO and GA, use automation tools, record results, generate reports
PSO Parameters	Number of particles: 30, Maximum iterations: 50, Acceleration coefficients: 2.05, 2.05, Inertia weight: 0.9
GA Parameters	Population size: 100, Crossover rate: 0.8, Mutation rate: 0.01, Selection method: Roulette wheel selection
CCTL Parameters	Time interval: [0 ms, 250 ms], Event expressions: Defined based on software functionality requirements
LSTM and LAR Parameters	LSTM hidden units: 128, LAR order: 4
Test Case Data	Derived from functional requirements documents of online shopping systems and user operation logs
Performance Data	Generated under various loads using JMeter tool
Security Data	Generated through security scans with OWASP ZAP tool
Software Defect Data	Collected from previous software testing and maintenance records