DLINE JOURNALS

Journal of Digital Information Management

ISSN: 0972-7272

JDIM 2025: 23 (1)

Performance Evaluation of Software in Large Data Environments Utilizing Time-Managed Computation Tree Logic

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
mdgapar@126.com

ABSTRACT

The research aims to solve the problems of unstable performance parameters and insufficient coverage in software testing and proposes a big data platform software performance testing system based on the clock-controlled computation tree logic method. The particle swarm algorithm finds the optimal solution through the movement and cooperation of particles in the search space. The 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 research uses the algorithmic logic of the particle swarm and genetic algorithms to confirm the software testing system's moment parameters and other information. At the same time, an algorithmic model researches the joint coverage and the use of the system's value, and finally, the big data platform is used to analyze the research system. The innovative combination of the CCTL method and optimization algorithm in the research has improved the accuracy and stability of software testing. 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 the genetic algorithm and particle swarm algorithm reach 90.36% and 91.32%, respectively. The accuracy of software testing for researching usage methods is 5% and 6% higher than testing 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 is 0 rad/s. The system can determine the various parameters of the software, and at the same time, if the software test results on the test results are typical, fault analysis can be completed typically; the performance of the use of algorithms is also better than other algorithms, and the study of the use of algorithms with a higher degree of stability. It can

be seen that the system and methods used in this research are better than traditional methods, and the test results in software testing have improved, providing a research direction for software testing after the research.

Subject Categories and Descriptors: [K.6.3] Software Management; [E. Data] Trees; [F.4.1] Logic and constraint programming

General Terms: Tree Logic, Software Testing, Data Environments, Computational Logic

Received: 3 September 2024, Revised 22 November 2024, Accepted 4 December 2024

Keywords: Software Testing, Particle Swarm Algorithm, Bell-controlled Computational Tree Logic Method, Genetic Algorithm, Testing Effectiveness

Review Metrics: 0/6; Review Score: 4.64; Inter-reviewer Consistency: 81.35%

DOI: https://doi.org/10.6025/jdim/2025/23/1/26-46

1. Introduction

In the 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 key to 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 complex problem that still needs to be solved [3]. 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]. The 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 fundamental 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 identify and analyze big-data platform performance bottlenecks and potential problems more effectively. 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. Therefore, to improve the accuracy and efficiency of software performance testing, this study innovatively proposes a new method based on CCTL to enhance the efficiency and accuracy of software performance testing in big data platforms. Firstly, the new model utilizes the algorithm logic of particle swarm optimization and genetic algorithm to improve and analyze the process to enhance the accuracy and stability of software testing. 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 big data platform software performance testing. 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 effectively reduce the number of required test cases and improve the testing efficiency in regression testing. Among them, the cuckoo search algorithm is slightly better regarding performance parameters [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 [7]. To address the parameter estimation problem 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 [8].

Qian et al. propose a method for prioritizing test scripts to address the memory bloat problem in web applications and 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 effectively speeds up testing and improves the efficiency of detecting memory bloats [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 security and performance by testing the code and analyzing the available information. The study results show that the Rust performs best in 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 affect MT's effectiveness. 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]. A study was conducted using a large eucalyptus species presence-absence dataset to compare the predictive performance of an ensemble species distribution model with that of a single model. 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 study showed that the ensemble models performed well in some tests but did not consistently 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 a genetic algorithm to reduce the cost of mutation detection. The research 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 and improve the accuracy and reliability of software testing. It can be seen that using a 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 a 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]. To explore the potential application of LLM in software testing, Wang et al. comprehensively reviewed 102 related studies, analyzed the application of LLM in tasks such as test case preparation and program repair, and validated LLM technology using big data. The research results indicate that LLM has great potential in software testing, effectively verifying model performance using big data [16].

In summary, existing research has made significant progress in 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 regression testing efficiency, 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. 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. It uses 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.

Secondly, a software testing system based on the CCTL algorithm model is built using the algorithm model. Then, based on the constructed model, software testing is implemented. Finally, the algorithm model constructed was tested for its actual effectiveness through experiments.

3.2 Building Software Testing Models

In the software for testing, software parameters of the moment input will have certain requirements; it must be input in a specific time parameter to make the whole operation 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, the software has three types of input time-space: 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.

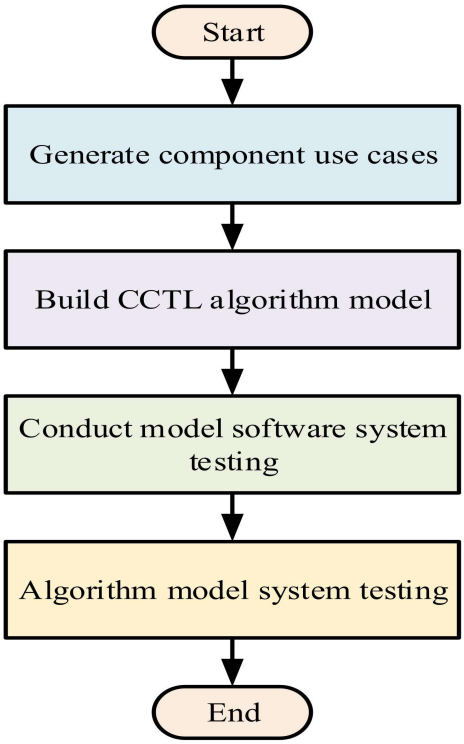

Figure 1. it can be seen that the first step is to analyze and generate test cases for the software

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 essential 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. 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].

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

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]:

$$EX_{t_0,t_1}\phi \to EX_{t_2,t_3}\psi \quad (2)$$

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 induced as n, and the expression is shown in Eq. (3).

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

In Eq. (3), ω,ζ indicates the input conditions for different parameters. The remaining parameters were the same as those described. 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. Their principle is to transform the models and convert the models that appear to conflict with valid combination methods. However, the problem with this method is that when significant 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 analysis. At this time, it is essential to use generative algorithms to research and analysis 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 is analyzed at this time. 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 and output the optimal value of the current parameters to achieve the analysis of the parameters of the judgement. Figure 2 shows the flow of the moment-combination generation algorithm.

As shown in Fig. 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 to select a better individual for constraint eva

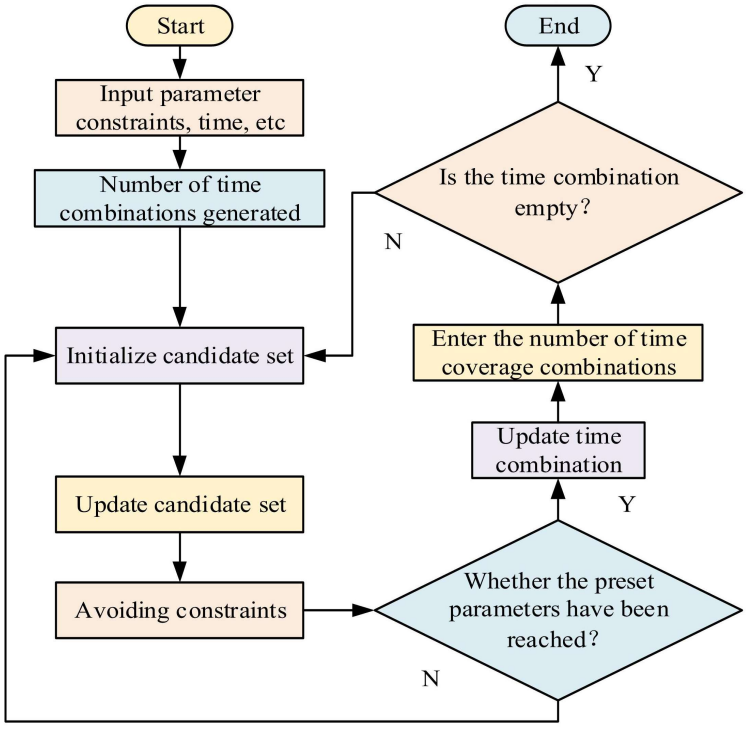

Figure 2. Time combination generation algorithm process

sion. Then, it is judged whether the current parameters under test reach the pre-set parameter data. 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 reinitialize the candidate collection. 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 the genetic algorithm and the particle swarm of the 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 the genetic algorithm and particle swarm algorithm, add excellent particles from the particle swarm algorithm to the population of the genetic algorithm, and add excellent chromosomes from the genetic algorithm to the particle swarm of the particle swarm algorithm. Finally, after the iteration, 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, ensuring the population's diversity. Secondly, 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. 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, a combination function is generated through target coverage combination, which includes combinations between parameters, 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. At the same time, to conclude 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).

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

In Eq. (4), CT_{p_1,p_2}^{n} 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 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 particle's position coordinate information at different moments. 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).

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

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]:

$$p_i = \frac{s_i}{\sum_{i=1}^{k} s_i} \quad (8)$$

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 a performance indicator function used to evaluate each individual. In the optimization process, the fitness value of each individual determines its probability of being selected for the next generation. Individuals with higher fitness values have a higher likelihood 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 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 number of gene positions is exchanged. 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 an optimal selection of the current parameters. Because the data selected in the time selection of software parameters are only valid for some 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.

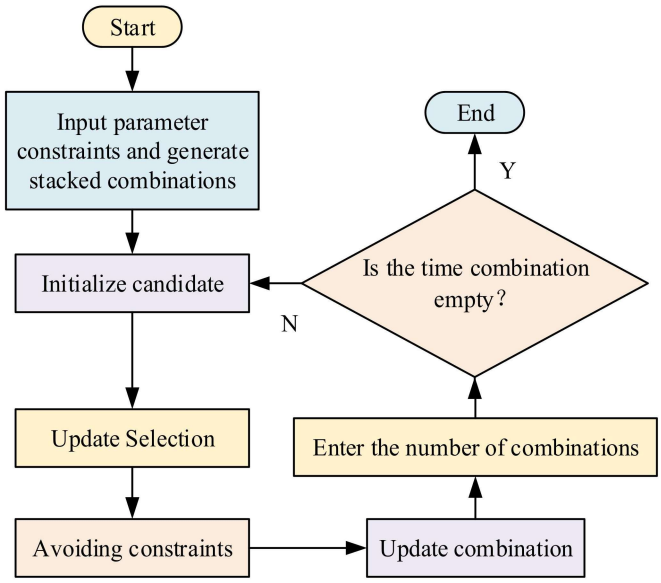

Figure 3. Joint algorithm flow combining value and time

Figure. 3 shows that the algorithm will first input parameters and constraints during testing. 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 develop the final coverage combination through subsequent selection and optimization steps. Secondly, based on the current candidate individuals and coverage combinations, select new candidates to optimize the coverage combination, ensuring

that the selected individuals meet the constraints. When choosing 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 candidate initialization and selection round. 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, it needs to be added to the set of parameters values 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. The particle swarm algorithm and genetic algorithm are also used in selecting the combination of the union for the change in the same way as described above. 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 software test's parameter analysis and moment selection in the CCTL method are completed by generating and overriding the target parameters.

3.4 Analysis of Software Testing Model System

The testing tool software system mainly consists of components; its main function is to load the model after completing the parsing to obtain the protocol's data information and then match the strength constraints and other requirements using case analysis. A use case analysis of the software testing tool is shown in Fig. 4.

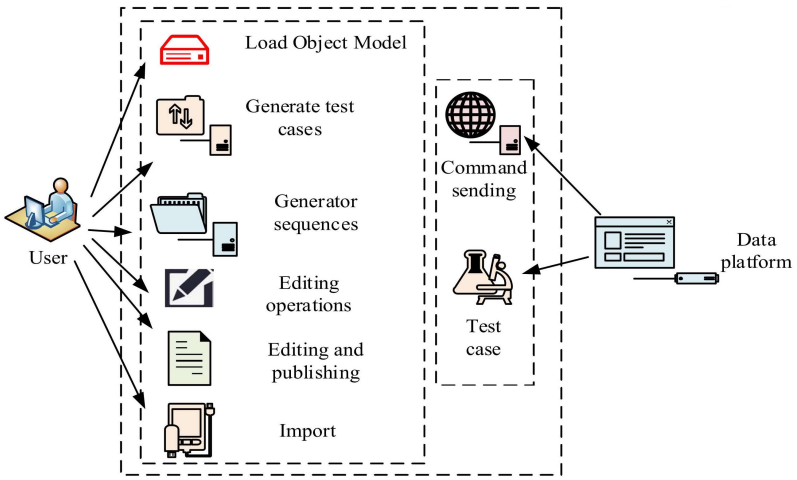

Figure 4. Use case analysis of software testing tools

As shown in Fig. 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. It is necessary to add a specific analysis of the system to the model's dynamic analysis system, as shown in Fig. 5.

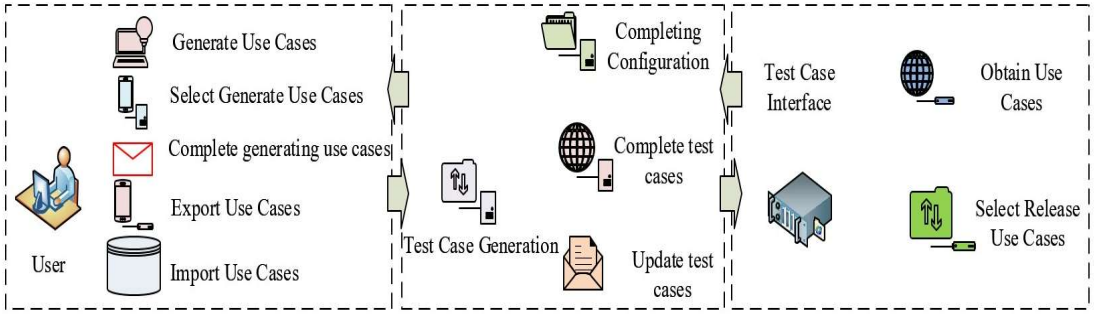

Figure 5. Example of model dynamic analysis

Figure. 5 shows that in the testing phase, the user must analyze the system software or model in the component according to the object model, and input the information into the test case parameters 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 parameter's value at the moment and setting the current moment and constraints, etc., and then completing the dynamic model. Construction of the dynamic model. Simultaneously, the user needs to deploy the front information in advance in the test, as shown in Fig. 6 for the parameter sequence image.

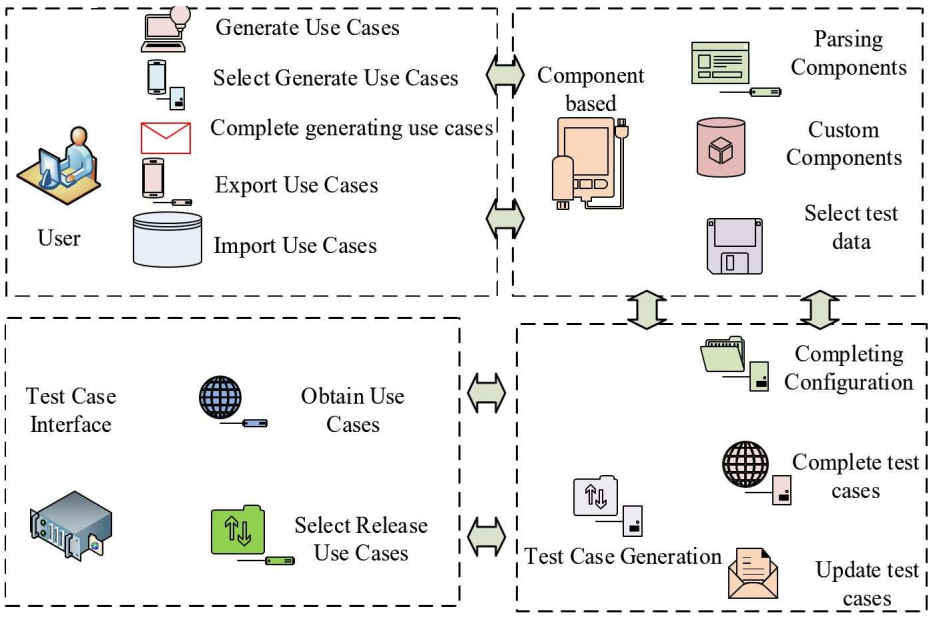

Figure 6. Example of deployment pre-parameter sequence process

As shown in Fig. 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. The user is reminded of the completion of the generation after the end of the generation. The data information is then exported for other operations, and the test data needs to be constantly updated when the test cases are generated. Simultaneously, updating the use cases when generating test data is necessary. The entire software testing system platform is shown in Fig. 7.

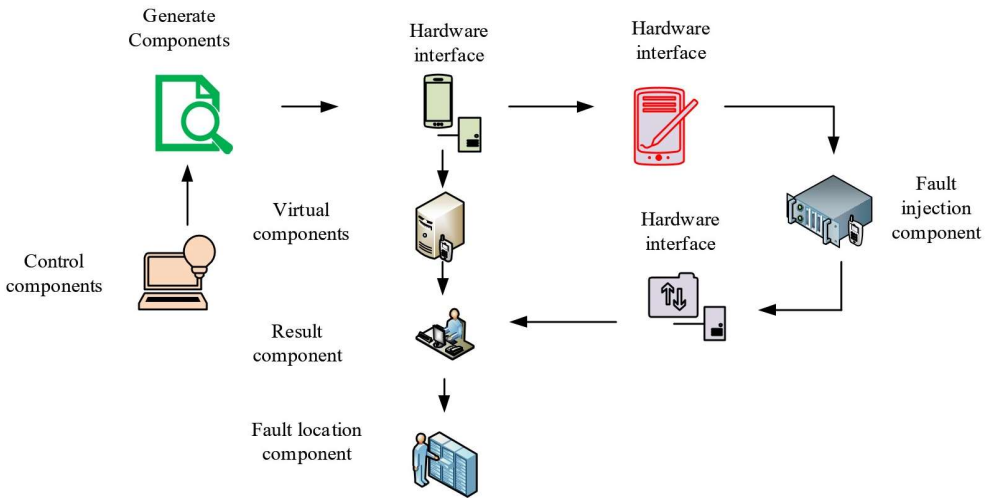

Figure 7. System platform for software testing

As shown in Fig. 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 and verify the system's fault-handling capability and robustness. During software testing, control components to generate and distribute test cases to the generating components. Generate test cases using the 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. The result component collects and processes the test results during the testing process. 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 better for the current model.

4. Results and Discussion

4.1 Results

Research the use of an online shopping system program for software testing. Firstly, the system must verify user login, product search, shopping cart functionality, and order payment. Next, configure the web and database servers 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. Automation tools such as Selenium are used to perform functional testing of the web interface, JMeter is used for performance testing, and OWASP ZAP is used for security testing. Finally, the execution status of all test cases will be recorded, and the failed cases will be analyzed. Generate a test report, record the defects discovered, and 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 required to include functional testing, performance testing, and security testing. Automated testing tools are used to perform testing. Regularly conduct code 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. 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. This test aims to verify the performance and stability of the system under high-load conditions. The testing scope includes the login, data processing, and report generation modules. The primary method of calculating the coverage is obtained by comparing the actual number of covered combinations with the target number of combinations so that the coverage test using CCTL is received, as shown in Table 1.

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 (3^3, 2; 2)	3	80	78	78
CA (3^3, 2; 2)	6	125	135	135
CA (5^3, 2; 2)	6	224	224	224
CA (5^3, 2; 2)	3	600	456	456
CA (8^3, 2; 2)	2	900	894	894
CA (8^3, 2; 2)	10	1654	1645	1645

Table 1. Analysis of CCTL method testing software coverage

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 in the software testing. The method can achieve 100% coverage, which shows that the current process of testing the different software can reach 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. 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 Fig. 8.

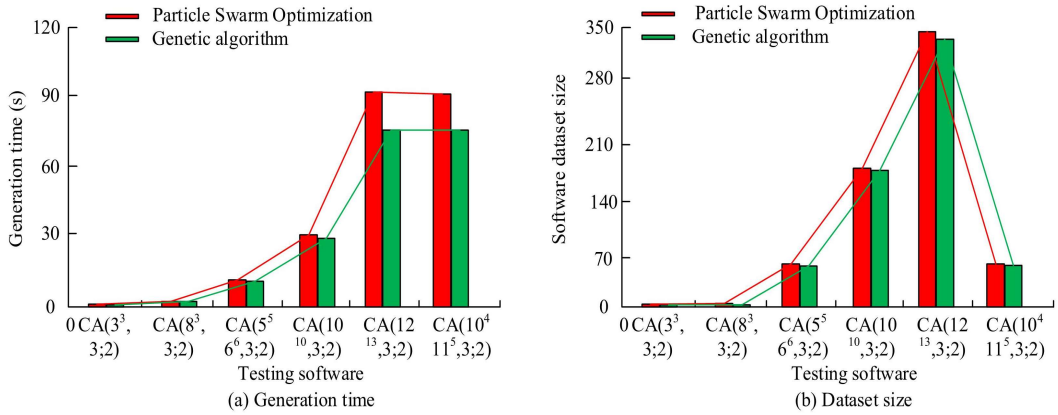

Figure 8. Comparison of testing time and data scale for different system software

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 more extended period to test the software is significantly faster than the genetic algorithm, as can be seen from Fig. 8(b) particle swarm algorithm of the From Fig. 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 more prominent. The CCTL method is more critical for creating the data coverage combinations, while the genetic algorithm is more important for developing and analysing 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.

Table 2 shows that the size of the maximum and minimum values in the target position of the parameter is the same, the maximum value is 10000 m, the minimum value is 0 m, and the range of the injected moments of the position is 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 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. Still, some of the parameters have the same range of values. 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.

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]

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

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, get 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. 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 Fig. 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. LAR can help monitor system performance in real time and detect and respond to abnormal situations in software testing by establishing performance benchmark models and fault detection models. LSTM is suitable for scenarios that require capturing long-term dependencies. This makes it particularly outstanding in handling non-stationary

time series data. LAR is ideal for linear time series data analysis and simple prediction tasks, but its ability to handle nonlinear and complex dependencies is limited.

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

Table 3. Software Test Data Results

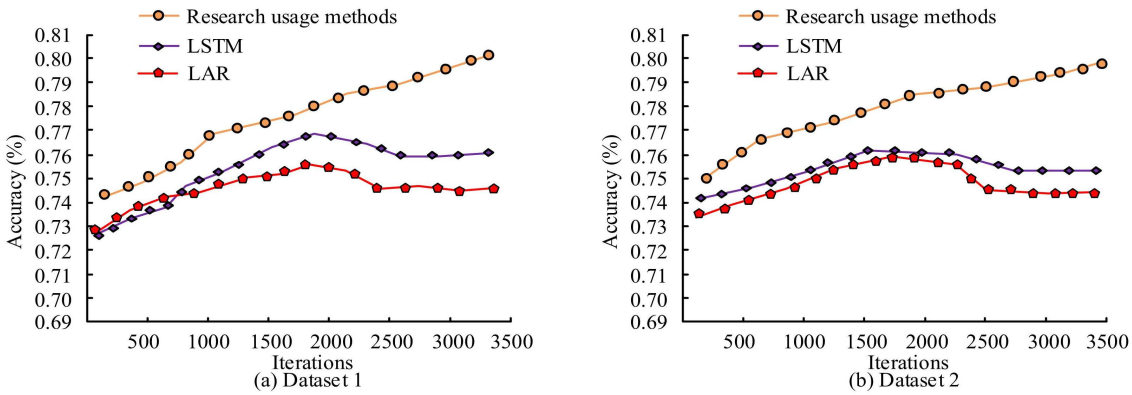

Figure 9. Comparison of software testing accuracy with different algorithm logics added

As shown in Fig. 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 the research-use method is in the increasing stage. This shows that the research-use method is more effective in testing the software's accuracy. At the same time, the other two algorithms have the highest accuracy of 76% and 75%, and the research-use method has a higher accuracy of 5% and 6%. In comparison, the accuracy of the research method was 5% and 6% higher than that of testing. To test the algorithmic stability of the proposed method, the loss functions of the three algorithms were compared and tested, as shown in Fig. 10.

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 3 at 300 k, and the LAR algorithm dropped to a minimum loss function of 3.2 at 320 *k* iterations. 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.

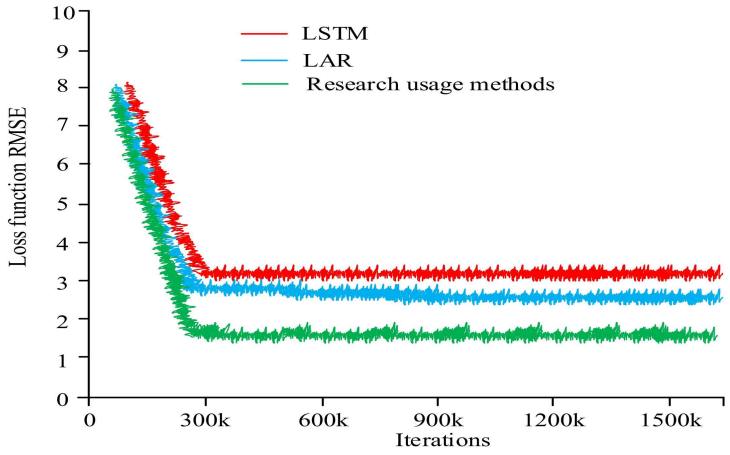

Figure 10. Comparison of loss functions among three algorithms

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

Table 4. Comparison of Test Results of Different Algorithm Software

Table 4 shows that after using different algorithms for software testing, the CCTL model performs better in terms of testing results. In comparing 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 comparing algorithm coverage, the CCTL model achieved good coverage levels under different coverage tests, 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 are high, and the actual software testing effect of the model is good.

4.2 Discussion

In Raamish et al.'s study, a ramp-up algorithm based on LSTM and BrainStorm optimization and post-acceptance was used to test the quality and performance of the software and improve its reliability. 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 L's research found that software testing can consume significant time and money 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 essential direction in the software testing process [28]. Build a new model for software testing in this study. When comparing the software production time of models, the particle swarm optimization algorithm has a higher coverage combination than the genetic algorithm in testing the performance of models with different iteration times. This indicates that the particle swarm optimization algorithm has better performance in data coverage combination processing and higher performance in improving the coverage combination set of the model. This may be because particle swarm optimization algorithms are more straightforward when integrating data. When comparing the accuracy of different algorithm models, the study found that the accuracy of LSTM and LAR models was 5% and 6%, respectively. This may be due to the combination of the genetic algorithm and particle swarm optimization algorithm used in the study.

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, possibly due to the improved algorithm performance after adding different algorithms to the model. When comparing the performance of various 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.

In summary, comparing 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 software testing's lack of stability and performance. It 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. Finally, the performance of the system and software testing effect was analyzed, and the results showed that the new method achieved 100% test coverage, adequate selection of the number of constraints and testing time of the software. The genetic algorithm generates test moments better than the particle swarm algorithm. At the same time, the latter is better in terms of the size of the data and the number of combinations to be covered. In addition, the effect of different parameter settings on the testing effect showed that the parameter values differed, but the range of values was similar. Simultaneously, the new method can effectively obtain the software's safety value and fault location

information. Meanwhile, the accuracy of other traditional methods is as high as 76% and 75%. In contrast, 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 enhanced subsequently when it is constrained to the combination of cases, and the study uses less data subsequently needs to be analyzed on a larger dataset. 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.

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

Funding

No funding was received.

References

[1] Alhroob, A., Alzyadat, W., Imam, A. T., Jaradat, G. M. (2020). The genetic algorithm and binary search technique in the program path coverage for improving software testing using big data. *Intelligent Automation and Soft Computing, 26*(4), 725–733.

[2] Han, X., Yu, T., Yan, G. (2023). A systematic mapping study of software performance research. *Software: Practice and Experience, 53*(5), 1249–1270.

[3] Murthy, M. S. N., Suma, V., Chandrappa, C. N., Shankar, M. M. (2020). Factors influencing effectiveness of testing applications in cloud using regression testing: A statistical analysis. *International Journal of Advanced Intelligence Paradigms, 17*(1–2), 109–126.

[4] Guo, S., Wang, J., Xu, Z., Huang, L. (2023). Feature transfer learning by reinforcement learning for detecting software defects. *Software: Practice and Experience, 53*(2), 366–389.

[5] Harry, J. R. (2021). MATLAB guide for analyzing countermovement jump strategies and performance over time. *Strength & Conditioning Journal, 43*(5), 44–53.

[6] Kaur, A., Agrawal, A. P. (2021). Performance comparison of Bat search and Cuckoo search using software artefact infrastructure repository and regression testing. *International Journal of Advanced Intelligence Paradigms, 18*(2), 99–118.

[7] Chen, J., Hu, H., Yu, D. (2022). Characterising and detecting methods to be benchmarked under performance unit test. *International Journal of Software Engineering and Knowledge Engineering, 32*(9), 1279–1305.

[8] Khurshid, S., Iqbal, J., Malik, I. A., Yousuf, B. (2022). 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, 29*(6), 10–28.

[9] Qian, J., Zhou, X., Zhou, H. (2022). Prioritising test scripts for the testing of memory bloat in web applications. *IET Software, 16*(3), 317–330.

[10] Bugden, W., Alahmar, A. (2022). The safety and performance of prominent programming languages. *International Journal of Software Engineering and Knowledge Engineering, 32*(5), 713–744.

[11] Jiang, M., Chen, T. Y., Wang, S. (2022). On the effectiveness of testing sentiment analysis systems with metamorphic testing. *Information and Software Technology, 150*(10), 1–11.

[12] Hao, T., Elith, J., Lahoz-Monfort, J., Guillera-Arroita, G. (2020). Testing whether ensemble modelling is advantageous for maximising predictive performance of species distribution models. *Ecography, 43*(4), 549–558.

[13] Hosseini, S. M. J., Arasteh, B., Isazadeh, A., Mohsenzadeh, M., Mirzarezaee, M. (2021). An error-propagation aware method to reduce the software mutation cost using genetic algorithm. *Data Technologies and Applications, 55*(1), 118–148.

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

[15] Pan, R., Ghaleb, T. A., Briand, L. (2023). ATM: Black-box test case minimization based on test code similarity and evolutionary search. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE, 14)*(5), 1700–1711.

[16] Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., Wang, Q. (2024). Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering, 20*, 3501–3502.

[17] Sornkliang, W., Phetkaew, T. (2021). Performance analysis of test path generation techniques based on complex activity diagrams. *Slovenian Association Informatika, 45*(2), 231–242.

[18] Bednárek, D., Kruliš, M., Yaghob, J. (2021). Letting future programmers experience performance-related tasks. *Journal of Parallel and Distributed Computing, 155*(C), 74–86.

[19] Ke, S.-Z., Huang, C.-Y. (2020). Software reliability prediction and management: A multiple change-point model approach. *Quality and Reliability Engineering International, 36*(5), 1678–1707.

[20] Bi, W., Yu, F., Cao, N., Wei, H., Cao, G., Han, X., Sun, L., Higgs, R. (2020). Research on data extraction and analysis of software defect in IoT communication software. *Computers, Materials, and Continuum, 65*(2), 1837–1854.

[21] Li, L., Xu, L., Cui, H., Abdelkareem, M. A. A. (2021). Validation and optimization of suspension design for testing platform vehicle. *Shock and Vibration, 2021*(7), 1–15.

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

[23] Serat, Z., Fatemi, S. A. Z., Shirzad, S. (2023). Design and economic analysis of on-grid solar rooftop PV system using PVsyst software. *Archives of Advanced Engineering Science, 1*(1), 63–76.

[24] Garmaki, M., Gharib, R. K., Boughzala, I. (2023). Big data analytics capability and contribution to firm performance: The mediating effect of organizational learning on firm performance. *Journal of Enterprise Information Management, 36*(5), 1161–1184.

[25] Jalil, S., Rafi, S., LaToza, T. D., Moran, K., Lam, W. (2023). ChatGPT and software testing education: Promises & perils. *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW, 16)*(6), 4130–4137.

[26] Weber, M., Kaltenecker, C., Sattler, F., Apel, S., Siegmund, N. (2023). Twins or false friends? A study on energy consumption and performance of configurable software. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE, 14)*(5), 2098–2110.

[27] Raamesh, L., Jothi, S., Radhika, S. (2023). Enhancing software reliability and fault detection using hybrid brainstorm optimization-based LSTM model. *IETE Journal of Research, 69*(12), 8789–8803.

[28] Oleshchenko, L. (2023). Software testing errors classification method using clustering algorithms. *International Conference on Innovative Computing and Communication, 17*(10), 553–566.