Computers, Materials & Continua
DOI: 10.32604/cmc.2022.024692
Article

Tech Science Press

Modified Harris Hawks Optimization Based Test Case Prioritization for Software Testing

Manar Ahmed Hamza[1,*], Abdelzahir Abdelmaboud[2], Souad Larabi-Marie-Sainte[3], Haya Mesfer Alshahrani[4], Mesfer Al Duhayyim[5], Hamza Awad Ibrahim[6], Mohammed Rizwanullah[1] and Ishfaq Yaseen[1]

[1]Department of Computer and Self Development, Preparatory Year Deanship, Prince Sattam bin Abdulaziz University, AlKharj, 16278, Saudi Arabia
[2]Department of Information Systems, College of Science and Arts, King Khalid University, Mahayil Asir, 62529, Saudi Arabia
[3]Department of Computer Science, College of Computer and Information Sciences, Prince Sultan University, P.O.Box No. 66833, Rafha Street, Riyadh, 11586, Saudi Arabia
[4]Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O.Box 84428, Riyadh, 11671, Saudi Arabia
[5]Department of Natural and Applied Sciences, College of Community-Aflaj, Prince Sattam bin Abdulaziz University, 16278, Saudi Arabia
[6]College of Computer at Al-Gunfudah, Umm Al-Qura University, Al-Gunfudah, 24382, Saudi Arabia
*Corresponding Author: Manar Ahmed Hamza. Email: ma.hamza@psau.edu.sa
Received: 27 October 2021; Accepted: 05 January 2022

Abstract: Generally, software testing is considered as a proficient technique to achieve improvement in quality and reliability of the software. But, the quality of test cases has a considerable influence on fault revealing capability of software testing activity. Test Case Prioritization (TCP) remains a challenging issue since prioritizing test cases is unsatisfactory in terms of Average Percentage of Faults Detected (APFD) and time spent upon execution results. TCP is mainly intended to design a collection of test cases that can accomplish early optimization using preferred characteristics. The studies conducted earlier focused on prioritizing the available test cases in accelerating fault detection rate during software testing. In this aspect, the current study designs a Modified Harris Hawks Optimization based TCP (MHHO-TCP) technique for software testing. The aim of the proposed MHHO-TCP technique is to maximize APFD and minimize the overall execution time. In addition, MHHO algorithm is designed to boost the exploration and exploitation abilities of conventional HHO algorithm. In order to validate the enhanced efficiency of MHHO-TCP technique, a wide range of simulations was conducted on different benchmark programs and the results were examined under several aspects. The experimental outcomes highlight the improved efficiency of MHHO-TCP technique over recent approaches under different measures.

Keywords: Software testing; harris hawks optimization; test case prioritization; apfd; execution time; metaheuristics

1 Introduction

The increasing importance of software testing adds significant value to the quality of software and indirectly towards social development and national economy. Inappropriately-developed and tested software might lead to serious issues, especially in issues concerning national security, etc., Further, it also brings high maintenance costs and huge amount of property loss [1]. Software testing is considered as a significant process to assure the trustworthiness and consistency of a software. Particularly, Automated Software Testing (AST) can improve testing efficiency, shorten testing time, and greatly reduce the manual workforce in this domain. Over the last few years, AST has gained a considerable interest in the field of both academics and industry. In general, test data automatic generation is a challenging and key task in AST [2]. A relevant test dataset can increase fault exposure probability and can efficiently reduce the testing execution time. In addition to these two problems discussed above, coverage ratio is a crucial index to estimate the quality of test dataset [3]. In comparison with functional testing, structural testing is a cost-efficient process in fault detection of programs. So, this process is extensively studied and applied.

Current software testing methods are mostly used for deterministic programs [4]. Indeed, there are several kinds of uncertainties experienced in real-time programs, for instance fuzziness/randomness, which suggest that the behaviour of these programs are not certain. Particularly, in case of a program with uncertainty and when a program is repeatedly run with a similar test data point, it might cover different statements or even different outputs, or sometimes traverse different paths too. In this scenario, prior test adequacy condition remains inappropriate. Now, the outcomes from few studies focused on testing a program with non-determinism. However, some of the studies focus on programs that is composed of randomness. Alternatively, the programs with randomness exist extensively in real-time applications like network software, game software, and Windows operating system [5].

Although software testing is recurrently performed, it is mostly performed in a hurried manner, owing to fixed resources and time constraints. As a result, it can be described that Test Case Prioritization (TCP) application can improve test feasibility from ST activity [6]. But, the studies in literature employed prioritization on test cases that passed with test case selection. Two researchers namely, Harold and Rothermel evaluated the concept in a wide context and proposed a novel method. TCP is an NP-Hard problem since it needs to check each feasible permutation sequence. Similarly, the rapid expansion of software makes one to resort to metaheuristics optimization algorithm to resolve the problems within a certain period of time [7]. Hence, the authors started looking into nature on how nature maintains an optimal balance. At present, several attempts are being made to find an optimum ordering through nature-inspired algorithm.

With this motivation, the current study designs a modified Harris Hawks Optimization-based TCP (MHHO-TCP) technique for software testing. The aim of the proposed MHHO-TCP technique is to maximize APFD and minimize the overall execution time. In addition, MHHO algorithm is also designed to boost both exploration and exploitation abilities of conventional HHO technique. The design of MHHO technique for TCP in software testing shows the novelty of current work. In order to validate the enhanced efficiency of MHHO-TCP technique, a wide range of simulations was conducted upon different benchmark programs and the results were investigated under several aspects.

Rest of the paper is arranged as follows. Section 2 briefs about related works, Section 3 introduces the proposed model, Section 4 discusses about performance validation, and Section 5 concludes the study.

2 Related Works

Spieker et al. [8] presented Retecs, a novel methodology to manually learn TCP and perform CI selection for the purpose of reducing round-trip time among developers' feedback on failed test cases and code commits. Retecs technique use RL algorithm for both selection and prioritization of test cases based on failure history, duration, and last implementation status. Panwar et al. [9] proposed CS and M-ACO algorithms to conclude the test cases in an enhanced order under time-constraint environments. Since this method is dependent on individual parameters, cuckoo search algorithm is preferred which is different from other optimization methods and is very effective and easy to perform.

Miranda et al. [10] presented the FAST family of TCP methods which radically changes the landscapes through borrowing algorithms, widely used in big data, to find relevant items. FAST technique provides scalable similarity-based TCP in white-box and black-box methods. The experimental results from practical Java and C subjects show that the fastest member of the family outperformed other black-box methods in terms of efficacy without considerable impact on efficiency. Further, it also outperformed white-box approach. Ali et al. [11] proposed a relevant solution and examined the prevailing problems towards regression testing in agile practices. The presented method has 2 stages while in initial phase, test case is prioritized through clustering while this test case changes frequently. If it is a tie, then the test case is prioritized according to their corresponding coverage criteria and failure frequencies. In the next phase, test cases with coverage criteria/high frequency of failure are elected. The presented method was authenticated through experimental research on three industrial subjects.

Hajjaji et al. [12] presented a similarity-based prioritization that is effectively employed on product samples. In this method, different products are gradually chosen based on the feature to be tested next, so as to increase the feature interaction coverage as soon as possible in product-wise testing. Xing et al. [13] adapted an advanced SI method—Artificial Fish School Algorithm to resolve the TCP problems. Particularly, the coding algorithm of artificial fish school was developed along with test case set; effective execution time and the average percentage of test-point coverage were chosen for optimization of strategies such as tail-chase, cluster, and forage behaviours of artificial fish school; the optimum solutions were attained through population iterations.

Gokilavani et al. [14] presented a multi-objective-based TCP and test case selection for distributed cloud platforms. Resemblance-Based Cluster Head (RBCH) method was presented in this study to select the CH based on total similarity among other test cases. Sivaji et al. [15] proposed a new African Buffalo-based Convolution Neural Slicing (AB-CNS) algorithm to reduce both resource and time consumption, when implementing regression testing. At first, the presented approach was implemented in test case minimalization task. It was prioritized using the FF of AB-CNS. Khalilian et al. [16] introduced an improved version of the methods discussed above in two ways. Firstly, the authors introduced a novel prioritization formula by gaining a variable coefficient as per the available historical efficiency data that act as feedback from previous test session. Next, a set of exhaustive research works has been carried out in the evaluation of system performances.

3 The Proposed Model

In this study, a novel MHHO-TCP technique is designed for TCP on software testing. The aim of the proposed MHHO-TCP technique is to maximize APFD and minimize the overall execution time. In addition, MHHO algorithm is designed to boost both exploration and exploitation abilities of conventional HHO algorithm.

3.1 Design of MHHO Algorithm

HHO is a metaheuristic technique and is applied as a competent solution to difficult issues. HHO is simulated by the attitude of Harris hawks, an intelligent bird. This species follows a process that empowers them to catch a prey, even there are chances for the prey to escape. Being a procedure modelled under mathematical process, it allows their computational execution. HHO has a stochastic technique which explores difficult search spaces for finding the optimum solution. The fundamental steps of HHO are attained under different conditions of energy. The exploration stage is inspired from the process where Harris's hawk cannot track the prey exactly. The hawks arbitrarily perch at distinct places and wait for its prey utilizing two operators that are chosen on the fundamental of probability q as provided in Eq. (1), where $q < 0.5$ implies that the hawks perch at the place of another population member and its prey (for instance, rabbit). When $q \geq 0.5$, the hawk is at arbitrary place nearby the population range. In order to facilitate the reason of HHO, a record of symbols utilized from this technique is determined as follows.

The exploration stage is determined as follows

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1|X_{rand}(t) - 2r_2X(t)| & q \geq 0.5 \\ (X_{rabbit}(t) - X_m(t)) - r_3(LB + r_4(UB - LB)) & q < 0.5 \end{cases} \tag{1}$$

The average place of Hawks X_m is demonstrated as given herewith

$$X_m(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t) \tag{2}$$

where $X(t)$ illustrates the places from iteration to all Hawks, t and N identifies with the entire number of Hawks. The average place is attained by utilizing distinct techniques though this is the simplest rule. There is a need to achieve optimum transition from exploration to exploitation. At this point, shift is expected amongst the varying inspired exploitative performances that depend upon the escaping energy factor, E of the prey that reduces dramatically in the escaping performance.

$$E = 2E_0\left(1 - \frac{t}{T}\right) \tag{3}$$

where, E_0, and T imply the primary escape energy and the maximal amount of iterations correspondingly.

Soft besiege is a vital phase in HHO which can be demonstrated, when $r \geq 0.5$ *and* $|E| \geq 0.5$. In this condition, the rabbit has enough energy. If it follows, the rabbit carries out an arbitrary misleading shift to escape. However, during the metaphor, it could not do so. Besiege stage is determined based on the subsequent rule.

$$X(t+1) = \triangle X(t) - E|JX_{rabbit}(t) - X(t)| \tag{4}$$

$$\triangle X(t) = X_{rabbit}(t) - X(t) \tag{5}$$

where $\triangle X(t)$ signifies the variance place of the vector to every rabbit and current place from the iteration t, and $J = 2(1 - r_5)$. The rabbits exhibit unplanned jumping capability throughout the escape stage. The J value differs arbitrarily from all iterations to represent the performance of rabbit. In extreme siege phase, if $r \geq 0.5$ *and* $|E| < 0.5$. The prey becomes tired and does not possess escape strength anymore [17]. During this analysis, the present place is modified as given herewith.

$$X(t+1) = X_{rabbit}(t) - E|\triangle X(t)| \tag{6}$$

As in the performance of hawks from real life, it slowly performs an optimum dive towards the prey, when it is required to capture a particular prey under competitive conditions.

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X(t)| \tag{7}$$

Soft besiege, projected from the preceding in Eq. (7), is carried out from progressive quick dive, only *if* $|E| \geq 0.5$ but $r < 0.5$. During this analysis, the rabbit possess sufficient energy to escape. So, it is implemented as a soft siege previously while the attack comes as a surprise. Fig. 1 demonstrates the flowchart of HHO technique.

$$Z = Y + S \times LF(D) \tag{8}$$

where S stands for arbitrary vector sized $1 \times D$ and LF represents Levy Fight purpose which is utilized in Eq. (9):

$$LF(x) = 0.01 \times \frac{u \times \sigma}{|v|^{\frac{1}{\beta}}},\ \sigma = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}} 1^{\frac{1}{\beta}} \right) \tag{9}$$

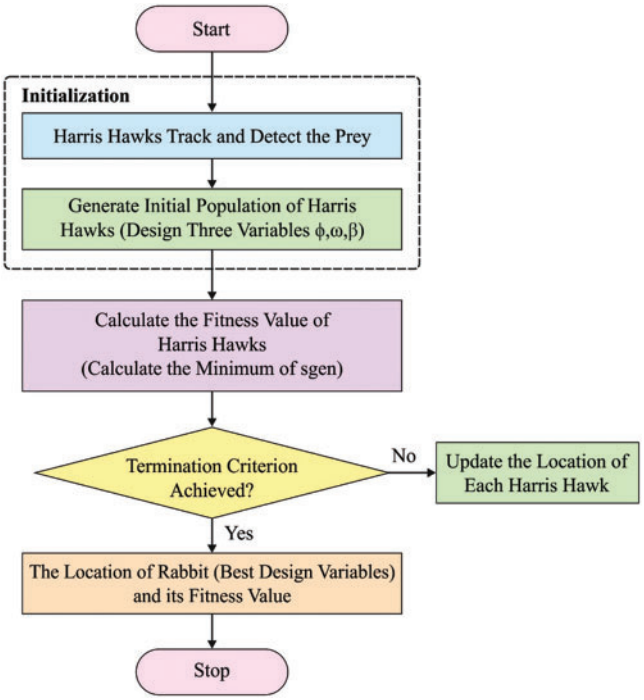

Figure 1: Flowchart of HHO

At this point, u, v refer to arbitrary values amongst 0 and 1, β signifies the default constant which is fixed at 1.5. The last phase from the procedure is to update the places of hawks based on the equations given below.

$$X(t+1) = \begin{cases} y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{cases} \tag{10}$$

where y and Z refer to the attained outputs utilized in Eqs. (7) and (8).

In this progressive fast dive, HHO represents the already hard-pressed one, where it occurs *if* $|E| < 0.5$ and $r < 0.5$. At this point, the strength of rabbits to escape is insufficient. So, hard siege was proposed before many surprise attacks were made for catching as well as killing the prey. During this phase, the Hawks seek to reduce different distances between their prey and their average place.

$$X(t+1) = \begin{cases} Y & if\ F(Y) < F(X(t)) \\ Z & if\ F(Z) < F(X(t)) \end{cases} \tag{11}$$

The values of y and Z are presented with the help of novel rules in Eqs. (12) and (13), where $X(t)$ demonstrates the value attained, utilizing Eq. (2).

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X_m(t)| \tag{12}$$

$$Z = Y + S \times LF(D) \tag{13}$$

In literature [18], it is demonstrated that HHO outperformed many advanced techniques and it addresses many optimization problems. But, a new adapted MHHO method is presented in current research work to promote search efficiency of HHO so as to attain fast convergence speed, robustness, and better solution accuracy. It is not challenging to define the modifications in E which could generate a major difference in simulation result. Thus, a novel E upgrading system is deployed in MHHO for promoting both exploitation and exploration abilities of HHO. The complete modifications made so far are given herewith.

Generally, two optimization approaches are followed in each metaheuristic algorithm such as exploitation and exploration. Exploration represents a global search in searching space; and exploitation represents the local optimum solution. From an algorithmic point of view, E is a connection between exploitation and exploration. When $|E| \geq 1$, the Harris hawks adapt a global search method to search for prey. On the other hand, when $|E| > 1$, local searching model is employed to hunt the prey. Moreover, when $|E| < 0.5$, 'Hard besiege' strategy is performed; when $1 \geq |E| \geq 0.5$, 'Soft besiege' system is elected by the hawks. The drawbacks of the original E upgrade system are stated.

Based on the last value of E_1, under the approach 6, is not zero. In approach 6, after the iteration, consider that the prey must have energy to escape and so the performances of the model can be improved. In the next half of iteration, $|E|$ could never be more than 1 in 1, 2, and 4 strategies. But, in the 1st half of the iteration, approach 4 is extremely exploratory, approach 1 follows, and the last is approach 2.

$$E_1 = 1 - \frac{t}{T} \tag{14}$$

$$E_1 = \left(1 - \frac{t}{T}\right)^2 \tag{15}$$

$$E_1 = \left(1 - \frac{t}{T}\right)^{\frac{1}{2}} \tag{16}$$

$$E_1 = \frac{1}{2} \sin\left(2\pi \times \frac{t}{T}\right) + \left(1 - \frac{t}{T}\right) \tag{17}$$

$$E_1 = \frac{1}{2} \sin\left(\pi + 2\pi \times \frac{t}{T}\right) + \left(1 - \frac{t}{T}\right) \tag{18}$$

$$E_1 = e^{-\frac{t}{T}} \tag{19}$$

$$E = 2E_0 \times E_1 \tag{20}$$

whereas E_0 represents an arbitrary value in the range of $(-1, 1)$; *sin* and *cos* represent sine and cosine functions, correspondingly [19]. The amount of iterations is represented by t while T is the maximal amount of iterations. e denotes the base amount of exponential functions, and the value is around 2.71828. The major time complexity of MHHO and HHO is based on three methods namely, fitness evaluation, hawk updating, and random initialization. Thus, it is formulated as follows.

$O(MHHO) = O(random\ initialization) + O(fitness\ evaluation) + O(hawk\ updating)$
$= O(N) + O(T \times N) + O(T \times N \times Dimension)$
$= O(N \times (1 + T \times (1 + Dimension)))$
$= O(HHO)$

In which N represents the population size, T denotes the overall number of iterations and Dimension denotes the amount of decision parameters.

3.2 Application of MHHO Algorithm for TCP

Assume that there are five test cases to be prioritized while the possibility of having the optimal prioritized arrangement is (1/5!) factorials as shown in Fig. 2. Every test case performs as firefly agents, when the distances among every test case represent the attractiveness functions amongst firefly agents. To identify the best-prioritized arrangement, the proposed method is utilized by string metrics as a FF. In this work, the real outcome of the distance estimated for every standard program is not demonstrated. Because, the amount of test cases and their corresponding content is too lengthy and huge in size to be sufficiently demonstrated. In fact, to demonstrate how distance is estimated in this study, five dummy test cases were generated. 'Edit distance' includes the evaluation of string characters, in which the quantified values represent the minimum number of replacements, insertions, and deletions to convert 1st into 2nd string. Edit distance is set to include only the least conversion of 1st into 2nd string.

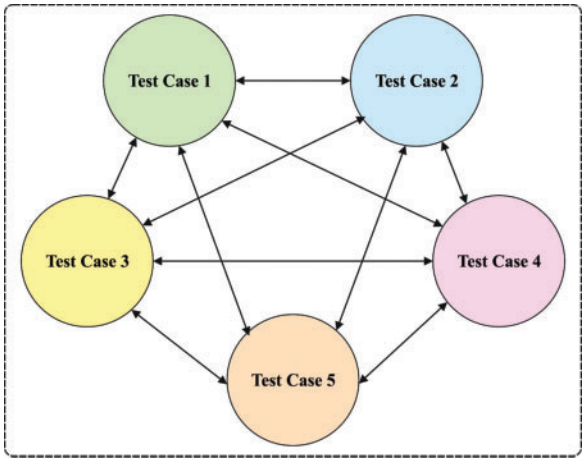

Figure 2: Test cases

In TFIDF, term frequency, tf, first computation is initiated with the help of elected string/term frequency in a document. In other words, tf represents the number of terms t that exists within d document. While, the equation for inverse document frequency, idf handles the importance of terms in a pool of documents. Then, TFIDF is formulated as follows.

$$TFIDF(t) = t/T \times log^{N}/n_{t} \tag{21}$$

In which: T = term frequency in each document; t = term frequency in one document; N = number of term that exist all over the documents; and nt = amount of documents has t term. These 2-string metrics are later utilized in the calculation of weight and distance of dummy test case table [20]. TF-IDF is a standout system among commonly-used term weighing systems in data retrieval methods. Based on this ability, TF-IDF is frequently used in research works. The weight amongst test cases is employed to assign brightness amongst firefly agents in MHHO-TCP.

The election of the next test case, in movement upgradation, depends on the weights of test cases that perform as brightness over distances among the test case, in which the maximum value gets elected as the following move. In arithmetical equation, the brightness of test case on distance estimation is represented as follows.

$$\frac{Weight\ of\ current\ Test\ case}{Distance\ to\ Next\ \ Test\ case} \tag{22}$$

As the weights of each dummy test case remains the same, the shortest distance is the only metric taken into account, because of the analogous weights of each dummy test case. But, in benchmark program/actual case study, the same weight problem reduces and the distance becomes heavily dynamic. Next the test case distance is evaluated by edit distance with the help of TFIDF string metrics. The total distance traveled denotes the prioritized test cases. The short distances of whole sequences of the test case are taken into account as the optimal distance. The optimal ones are chosen based on MHHO-TCP technique.

4 Experimental Validation

The proposed MHHO-TCP model was validated for its performance under different aspects and the results are discussed in this section. The results were investigated under varying iterations and benchmark functions namely, Gzip, Grep, TCAS, and CSTCAS.

Tab. 1 and Fig. 3 shows the results from detailed Average Percentage of Faults Detected (APFD) analysis accomplished by MHHO-TCP model against existing methods under Gzip benchmark function. The figure shows that Greedy technique accomplished a poor performance with least APFD values. Simultaneously, PSD technique gained somewhat improved APFD values. Moreover, LBS and FA techniques reached moderately closer APFD values. However, the proposed MHHO-TCP model outperformed existing techniques and achieved the maximum APFD values under all iterations.

Table 1: Results of the analysis of MHHO-TCP techniques in terms of APFD under Gzip dataset

Number of iterations	MHHO-TCP	FA	PSD	LBS	GREEDY
1	95.243	94.984	93.923	93.900	92.226
2	95.090	94.654	94.029	94.678	92.367

(Continued)

Table 1: Continued

Number of iterations	MHHO-TCP	FA	PSD	LBS	GREEDY
3	95.161	94.630	93.876	93.723	93.098
4	95.467	95.232	93.487	95.255	92.957
5	95.373	95.031	94.135	94.619	92.226
6	95.397	95.232	92.921	93.640	92.426
7	95.184	94.996	94.831	94.925	93.428
8	95.031	94.654	94.218	94.395	93.216
9	95.444	95.243	93.982	94.477	92.249
10	95.432	95.161	94.760	95.137	93.122
11	95.326	94.418	93.758	93.640	93.428
12	95.491	95.361	93.699	94.902	93.216
13	95.232	94.666	93.275	93.628	94.076
14	95.291	94.854	94.406	94.619	92.992
15	95.408	95.043	93.004	94.454	93.393
16	95.432	95.161	93.593	93.699	92.167
17	95.408	94.913	93.876	94.654	93.063
18	95.727	95.573	94.619	94.277	92.414
19	95.715	95.538	94.124	94.772	93.369
20	95.456	94.890	94.430	94.996	94.230
21	95.538	94.654	93.911	93.805	93.122
22	95.467	95.232	93.499	95.279	92.957
23	95.349	95.055	94.194	94.630	92.261
24	95.408	95.125	92.992	93.676	92.403
25	95.326	94.937	94.866	94.890	93.381
26	95.514	95.385	93.699	94.925	93.204
27	95.408	94.666	93.251	93.617	94.076
28	95.585	95.349	93.640	94.960	93.263
29	95.385	94.654	93.216	93.581	94.124
30	95.397	94.819	94.430	94.666	92.921

Tab. 2 and Fig. 4 shows the results achieved by the presented technique in a detailed APFD analysis against existing approaches under Grep benchmark function. The figure demonstrates that Greedy method accomplished the least performance with minimum APFD values. In addition, PSD technique gained slightly improved APFD values. Moreover, LBS and FA methods attained moderately closer APFD values. However, the presented technique outperformed the existing methods with maximum APFD values under all iterations.

Fig. 5 offer a detailed overview of the results from APFD analysis of MHHO-TCP model against existing methods under TCAS benchmark function. The figure shows that Greedy system accomplished the least performance with low APFD values. Also, PSD technique gained somewhat enhanced APFD values. Moreover, LBS and FA techniques reached moderately closer APFD values. However, the presented technique outperformed the existing techniques with maximum APFD values under all iterations.

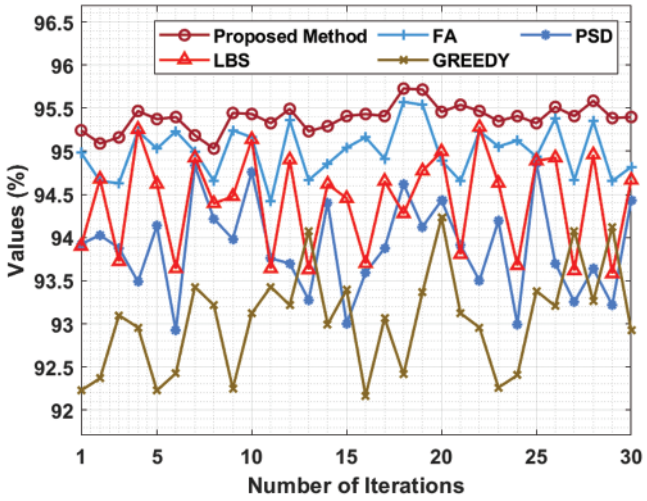

Figure 3: APFD analysis of MHHO-TCP method under Gzip dataset

Table 2: Results of the analysis of MHHO-TCP techniques in terms of APFD under GREP dataset

Number of iterations	MHHO-TCP	FA	PSD	LBS	GREEDY
1	95.511	95.057	94.603	95.188	94.447
2	95.894	95.751	94.842	94.435	92.474
3	95.715	95.332	93.777	93.861	92.294
4	95.535	94.961	94.495	94.818	93.048
5	95.751	95.499	93.933	95.045	93.335
6	95.631	95.296	94.925	95.284	93.275
7	95.392	94.734	93.108	94.567	93.323
8	95.559	95.308	94.375	93.765	92.546
9	95.643	95.428	93.586	95.476	93.048
10	95.523	94.794	94.112	94.854	92.534
11	95.822	95.679	94.232	94.985	93.538
12	95.476	95.081	94.052	94.794	93.227
13	95.476	95.200	93.108	94.638	93.490
14	95.260	94.782	93.395	93.801	94.172
15	95.093	94.567	93.885	93.777	93.550
16	95.535	95.380	94.064	94.579	92.318
17	95.607	95.069	94.985	95.009	93.502
18	95.476	95.165	94.304	94.770	92.318
19	95.392	94.794	94.017	93.861	93.156
20	95.547	95.320	93.096	93.765	92.450
21	95.619	95.428	93.574	95.368	93.060
22	95.404	94.770	94.124	94.854	92.522
23	95.822	95.691	94.244	94.973	93.526
24	95.667	95.057	94.029	94.806	93.203

(Continued)

Table 2: Continued

Number of iterations	MHHO-TCP	FA	PSD	LBS	GREEDY
25	95.571	94.997	94.531	94.830	93.024
26	95.703	95.523	93.849	95.057	93.359
27	95.559	95.188	93.108	94.603	93.538
28	95.380	94.758	93.347	93.706	94.220
29	95.416	94.543	93.945	93.909	93.562
30	95.583	95.452	94.100	94.638	92.306

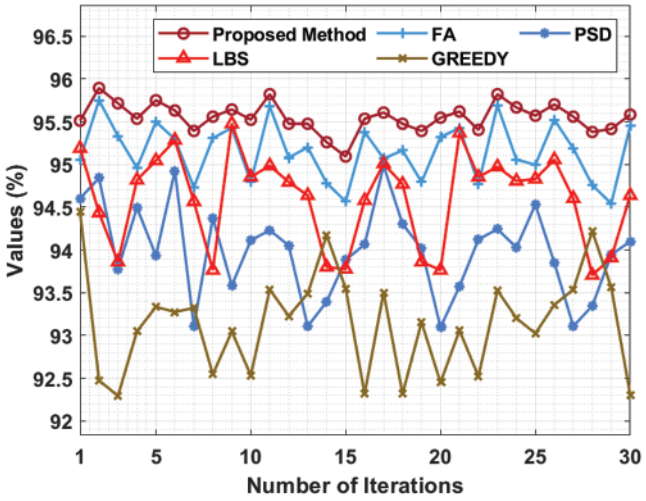

Figure 4: APFD analysis of MHHO-TCP method under Grep dataset

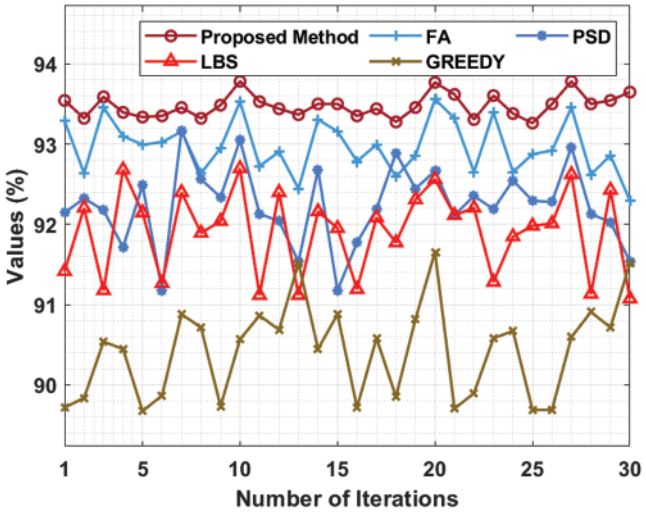

Figure 5: APFD analysis of MHHO-TCP method under TCAS dataset

Fig. 6 shows the results achieved by MHHO-TCP model from a detailed APFD analysis conducted against existing methods under CS-TCAS benchmark function. The figure shows that Greedy technique accomplished a poor performance with low APFD values. At the same time, PSD technique reached somewhat enhanced APFD values. Moreover, LBS and FA methods reached moderately closer APFD values. Finally, the presented technique outperformed all other existing techniques with the highest APFD values under all iterations.

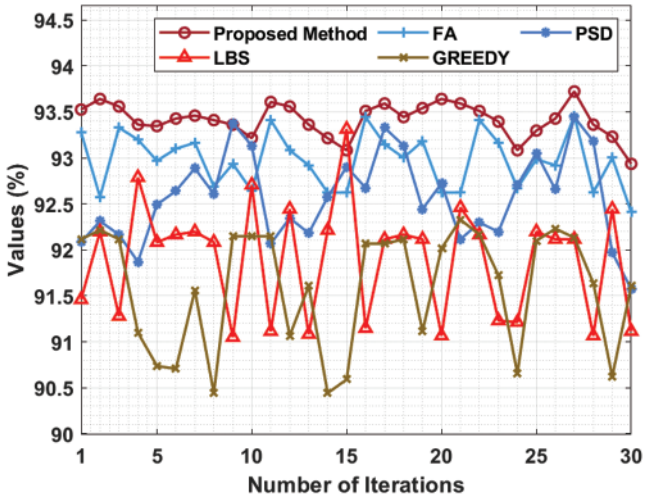

Figure 6: APFD analysis of MHHO-TCP method under CS-TCAS dataset

Tab. 3 and Fig. 7 portrays the outcomes achieved by MHHO-TCP model in average execution time analysis against existing techniques under various benchmark functions. The results showcase that the proposed MHHO-TCP model achieved the least average execution time compared to other approaches under all benchmark functions. For instance, with GZIP benchmark function, MHHO-TCP model attained a low average execution time of 2.103 mts whereas Firefly, PSO, LBS, and Greedy techniques took high average execution times such as 3.053, 4.980, 2.960, and 3.637 respectively. At the same time, with GREP benchmark function, the presented technique attained a minimal average execution time of 2.693 mts, whereas Firefly, PSO, LBS, and Greedy techniques consumed high average execution times namely, 3.647, 5.677, 3.780, and 3.950 respectively. Moreover, with TCAS benchmark function, the proposed MHHO-TCP method achieved the least average execution time of 5.283 mts, whereas other techniques such as Firefly, PSO, LBS, and Greedy obtained the maximal average execution times such as 6.460, 13.293, 6.627, and 7.617 correspondingly. Furthermore, with CS-TCAS benchmark function, MHHO-TCP approach reached a lesser average execution time of 8.230 mts, whereas Firefly, PSO, LBS, and Greedy techniques accomplished superior average execution times namely, 9.627, 19.980, 9.960, and 10.793 respectively.

An extensive mean APFD analysis was conducted between MHHO-TCP method and existing techniques and the results are shown in Tab. 4 and Fig. 8. The outcomes show that MHHO-TCP approach produced an effective outcome with increased mean APFD values. For instance, with Gzip benchmark function, MHHO-TCP model attained an increased mean APFD of 95.390, whereas firefly, PSO, LBS, and Greedy techniques resulted in reduced mean APFD of 95.004, 93.888, 94.414, and 93.046 respectively. Moreover, with CSTCAS benchmark function, the presented system attained a high mean APFD of 93.413, whereas firefly, PSO, LBS, and Greedy methodologies produced low mean APFD values such as 92.973, 92.571, 91.897, and 91.591 correspondingly.

Table 3: Execution time analysis results of MHHO-TCP technique against existing approaches

Methods	Average time execution (min)			
	GZIP	GREP	TCAS	CS-TCAS
MHHO-TCP	2.103	2.693	5.283	8.230
Firefly	3.053	3.647	6.460	9.627
PSO	4.980	5.677	13.293	19.980
LBS	2.960	3.780	6.627	9.960
Greedy	3.637	3.950	7.617	10.793

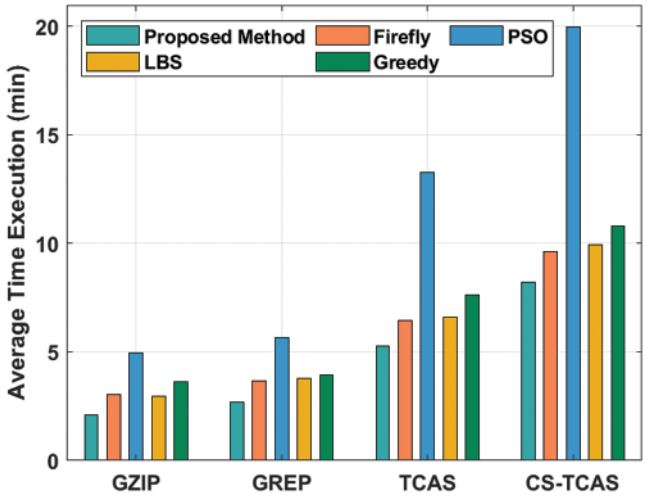

Figure 7: Average execution time analysis results of MHHO-TCP method

Table 4: Mean APFD analysis results of MHHO-TCP method against existing approaches

Mean APFD				
Methods	Gzip	Grep	TCAS	CSTCAS
MHHO-TCP	95.390	95.552	93.479	93.413
Firefly	95.004	95.153	92.972	92.973
PSO	93.888	93.992	92.238	92.571
LBS	94.414	94.601	91.910	91.897
Greedy	93.046	93.146	90.433	91.591

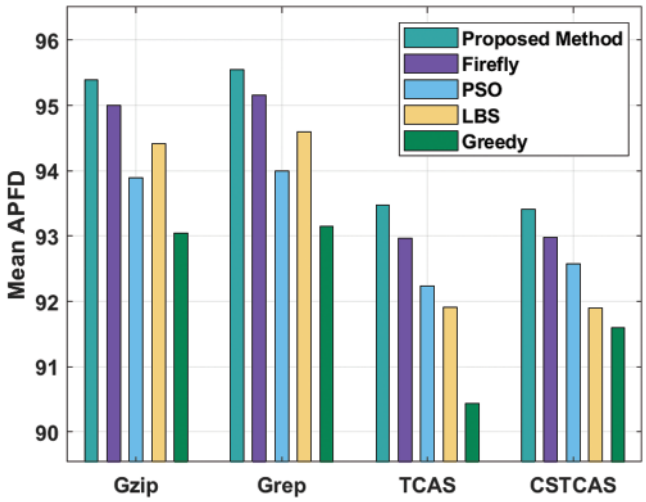

Figure 8: Mean APFD analysis results of MHHO-TCP method against existing approaches

5 Conclusion

In this study, a novel MHHO-TCP technique is designed for TCP on software testing. The aim of the proposed MHHO-TCP technique is to maximize APFD and minimize the overall execution time. In addition, MHHO algorithm is designed to boost both exploration and exploitation abilities of conventional HHO technique. The design of MHHO technique for TCP in software testing helps in achieving the maximum APFD and minimum execution time. In order to validate the enhanced efficiency of MHHO-TCP technique, a wide range of simulations was conducted upon different benchmark programs and the results were examined under several aspects. The experimental outcomes demonstrate the increased efficacy of MHHO-TCP technique against recent approaches under different measures. In future, HHO algorithm can be utilized replacing other metaheuristic algorithms to further boost the overall performance.

Acknowledgement: The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges (APC) of this publication.

Funding Statement: The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work under Grant Number (RGP.1/127/42). Princess Nourah bint Abdulrahman University Researchers Supporting Project Number (PNURSP2022R237), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] D. Hao, L. Zhang and H. Mei, "Test-case prioritization: Achievements and challenges," *Frontiers of Computer Science*, vol. 10, no. 5, pp. 769–777, 2016.

[2] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, H. N. A. Hamed and M. D. M. Suffian, "Test case prioritization using firefly algorithm for software testing," *IEEE Access*, vol. 7, pp. 132360–132373, 2019.

[3] B. Balakiruthiga, P. Deepalakshmi, S. N. Mohanty, D. Gupta, P. Pavan Kumar *et al.,* "Segment routing based energy aware routing for software defined data center," *Cognitive Systems Research*, vol. 64, pp. 146–163, 2020.
[4] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu *et al.,* "To be optimal or not in test-case prioritization," *IEEE Transactions on Software Engineering*, vol. 42, no. 5, pp. 490–505, 2016.
[5] V. Porkodi, A. R. Singh, A. R. W. Sait, K. Shankar, E. Yang *et al.,* "Resource provisioning for cyber–physical–social system in cloud-fog-edge computing using optimal flower pollination algorithm," *IEEE Access*, vol. 8, pp. 105311–105319, 2020.
[6] W. Jun, Z. Yan and J. Chen, "Test case prioritization technique based on genetic algorithm," in *2011 Int. Conf. on Internet Computing and Information Services*, Hong Kong, China, pp. 173–175, 2011.
[7] S. K. Lakshmanaprabu, S. N. Mohanty, S. S. Rani, S. Krishnamoorthy, J. Uthayakumar *et al.,* "Online clinical decision support system using optimal deep neural networks," *Applied Soft Computing*, vol. 81, pp. 105487, 2019.
[8] H. Spieker, A. Gotlieb, D. Marijan and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proc. of the 26th ACM SIGSOFT Int. Symp. on Software Testing and Analysis*, Santa Barbara CA USA, pp. 12–22, 2017.
[9] D. Panwar, P. Tomar and V. Singh, "Hybridization of Cuckoo-ACO algorithm for test case prioritization," *Journal of Statistics and Management Systems*, vol. 21, no. 4, pp. 539–546, 2018.
[10] B. Miranda, E. Cruciani, R. Verdecchia and A. Bertolino, "FAST approaches to scalable similarity-based test case prioritization," in *Proc. of the 40th Int. Conf. on Software Engineering*, Gothenburg Sweden, pp. 222–232, 2018.
[11] S. Ali, Y. Hafeez, S. Hussain and S. Yang, "Enhanced regression testing technique for agile software development and continuous integration strategies," *Software Quality Journal*, vol. 28, no. 2, pp. 397–423, 2020.
[12] M. A. Hajjaji, T. Thüm, M. Lochau, J. Meinicke and G. Saake, "Effective product-line testing using similarity-based product prioritization," *Software and Systems Modeling*, vol. 18, no. 1, pp. 499–521, 2019.
[13] Y. Xing, X. Wang and Q. Shen, "Test case prioritization based on artificial fish school algorithm," *Computer Communications*, vol. 180, pp. 295–302, 2021.
[14] N. Gokilavani and B. Bharathi, "Multi-objective based test case selection and prioritization for distributed cloud environment," *Microprocessors and Microsystems*, vol. 82, pp. 103964, 2021.
[15] U. Sivaji and P. S. Rao, "Test case minimization for regression testing by analyzing software performance using the novel method," *Materials Today: Proceedings*, pp. S2214785321009792, 2021.
[16] A. Khalilian, M. A. Azgomi and Y. Fazlalizadeh, "An improved method for test case prioritization by incorporating historical test case data," *Science of Computer Programming*, vol. 78, no. 1, pp. 93–116, 2012.
[17] E. H. Houssein, M. E. Hosney, M. Elhoseny, D. Oliva, W. M. Mohamed *et al.,* "Hybrid Harris hawks optimization with cuckoo search for drug design and discovery in chemoinformatics," *Scientific Reports*, vol. 10, no. 1, pp. 14439, 2020.
[18] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja *et al.,* "Harris hawks optimization: Algorithm and applications," *Future Generation Computer Systems*, vol. 97, pp. 849–872, 2019.
[19] Y. Zhang, X. Zhou and P. C. Shih, "Modified harris hawks optimization algorithm for global optimization problems," *Arabian Journal for Science and Engineering*, vol. 45, no. 12, pp. 10949–10974, 2020.
[20] J. H. Kwon, I. Y. Ko, G. Rothermel and M. Staats, "Test case prioritization based on information retrieval concepts," in *2014 21st Asia-Pacific Software Engineering Conf.*, Jeju, South Korea, pp. 19–26, 2014.