Effective Test Cases Generation with Harmony Search and RBF Neural Network

Hemant Kumar
Department of Computer Science
Babasaheb Bhimrao Ambedkar Central University, India
hemant20192@gmail.com

Vipin Saxena
Department of Computer Science
Babasaheb Bhimrao Ambedkar Central University, India
profvipinsaxena@gmail.com

Abstract: *Software testing is one of the integral activities during development of software products. Generation and selection of the test cases either in static or dynamic form play pivot role for ensuring the quality of software products. There are numerous approaches in the literature for automatic generation of test cases but coverage criteria and fault detection rate are prominent metrics for checking the effectiveness of the software products during testing phase of software development. In the present, a new Harmony Radial Testing (HRT) is proposed by combining the concepts of Harmony Search Algorithm (HSA) and Radial Basis Function-Neural Network (RBF-NN) approaches. The main objective of the proposed HRT method is to generate automatic test cases by considering the criteria of branch coverage with improvement in the Maximum branch Coverage (MaxC), Average Coverage (AC) and Average Percentage Fault Detection (APFD) rates. The proposed approach combined with the Radial Basis Function (RBF), denoted as a HRT approach. The proposed approach is used to optimize harmony search over the randomly selected sample test cases, training the RBF-NN to simulate the fitness function. Seven Python codes have been tested through proposed approach and computed results are compared with Primal-Dual Genetic Algorithm (PDGA), Simple Genetic Algorithm (SGA) and random methods. It is observed that the proposed HRT algorithm optimizes consistently yielded reliable results, which may be used in future for enriching the software testing process by the software industries.*

Keywords: *Algorithm optimization, automated testing, test cases, hybrid algorithm and software testing efficiency.*

Received December 25, 2023; accepted August 26, 2024
https://doi.org/10.34028/iajit/21/5/2

1. Introduction

Through the effective software testing strategies, software development team may lead to produce the high quality of the software products in the optimized time frame assigned by the project leader during the testing phase. Software testing is a vital activity for ensuring the reliability and as well as for the correctness of the complex software code. There are numerous kinds of the software testing strategies like analytical, model-driven, methodological, regression and many more which consume test cases for evaluating the performance of the software code. For different kinds of testing strategies, automatic generation of the effective test cases is a challenging talks and it is necessary to execute the code for producing the results in the optimized time. In the present work, the challenges like coverage criteria and early fault detection of code are considered by combining the Harmony Search Method (HSM) alongwith Radial Basis Function-Neural Network (RBF-NN) in the hybrid form and a new algorithm Harmony Radial Testing (HRT) is investigated for generation of the effective test cases. The aim of the proposed algorithm is not only to streamline the test cases generation but also to enhance the diversity and efficacy of the generated test cases.

Let us introduce a word 'Harmony' which contents of the chords, played alongwith melody. It was investigated in the year 1902 and elements of harmony are well described in [2]. On the basis of elements of harmony, a search algorithm was proposed in the year 2001 and research work was published in the year 2016 [12]. It was based on the random search, rules were based on the Harmony Memory (HM) and pitch was adjusting the operators. Later, it became as the heuristic optimization technique which strikes a balance between exploration and exploitation and finally adopted in observing the complexities of the software systems. On the other hand Radial Basis Function is a real valued function which was first reported in the literature in the year 1988 [4]. The real values depend on the difference between the inputs and some fixed points [16].

In the present work, HAS is complemented through the capabilities of RBF-NN approach for generation of quality test cases based on the learned patterns, further refining the search for the solution, reflective of desired program's behavior. The proposed methodology is encapsulated in the form of system model for new HRT approach as shown in Figure 1.

The model seeks to optimized test coverage, ensuring through testing of critical aspects of program functionality. The approach is considered as hybrid approach which begins by initializing the environment of the target code. A grid of potential solutions is generated and RBF-NN is trained to assess the fitness function. Thereafter, HAS optimizes the test cases in the iterative manner for the target code which shows the

efficacy of the proposed approach for generation of effective test cases. Section 2 describes overview of the proposed approach, section 3 describes formation of HRT, section 4 describes results and discussion while last section 5 describes conclusions and future direction for further use of HRT hybrid algorithm.

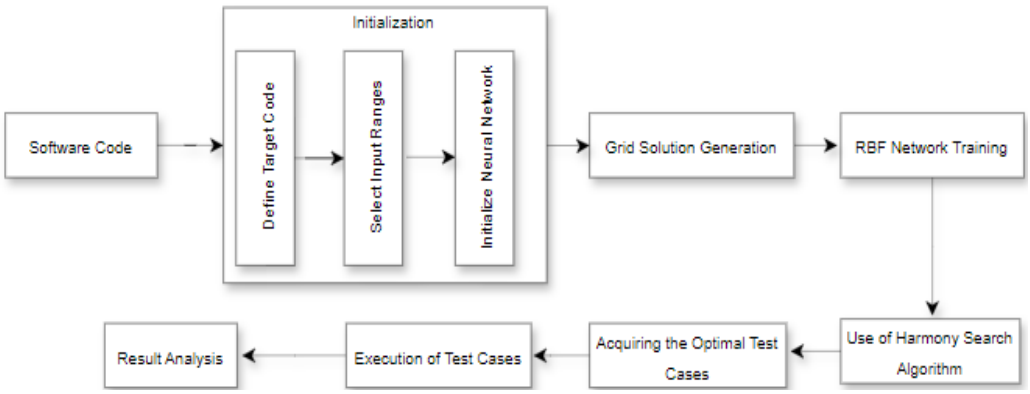

Figure 1. A system model for HRT.

2. Overview

The generation of test cases even for the small software code is a very complex activity and generated test cases must be optimized for ensuring the highly reliable software system. It is observed from the recent research that the generation of test cases based on branch coverage forms a foundation of the presented approach. Let us first describe the test case in the next sub-section.

2.1. Test Case

It is well explained as inputs to be used to produce expected outcomes. This term was first coined in the literature in the year 1896 through Plessy v. Ferguson legal case. Later on, this terminology was first used in the software testing in the year 1957 by Charles Baker [11]. Later on, this concept is used by many researchers and scientists for the testing of the software code and for ensuring the quality of the software code. There are two categories of test cases, one is certainty of test cases and another is uncertainty of test cases and both categories are covered under automatic generation of test cases. Further prioritization of test cases is another important activity during software testing and which saves the time of testing. It is done especially for uncertainty of the test cases [37]. The process of prioritization of test cases contains initial analysis to identify the variables based on the software code. The systematic generation of a diverse set of test cases spanned a wide spectrum of scenario. Mahalakshmi *et al*. [24] introduced a method for test case generation using named entity recognition. The approach focused on automating the identification of key entities within use cases, which were then used to create a scenario matrix for generating test cases. The named entity recognition system, trained on features extracted from use cases, enhanced the efficiency of test case generation, reducing the need for manual tagging and minimizing errors. The method was domain-independent and adaptable, offering a streamlined process for early-stage testing in software development.

From time to time, different approaches of software testing were appeared in the literature, but for sake of completeness of the proposed approach, only related software testing approaches are described below in brief.

2.1.1. Data-Flow Oriented Testing

It is based upon the theory of control flow and used to detect illogical variables that interrupt flow of data for producing the correct output. Therefore, anomalies in the flow of data among the various modules of the software code may be detected. For example, if variables are used without correct declaration of the variables then errors among the modules associated with flow of data may be detected. Further paths within the software code may be checked for correct flow of data via execution of various paths over the test cases, hence also called as path-oriented testing. Ji *et al*. [19] proposed this methodology for generation of test cases which involved leveraging neural networks for enhancing the testing efficiency and coverage criteria within the data-flow oriented scenario. Bao *et al*. [3] generated path-oriented test cases based on adaptive genetic control and applied over the six industrial software codes and found that the novel approach enhances the quality of software products. Lin and Yeh [22] also generated automatic test cases through path testing using the concept of Genetic Algorithm (GA). A concept of Harmony distance used for generation of effective test cases which produces highly reliable software products. Su *el al*. [34] also emphasized search-based algorithm, addressing the search-based algorithm, addressing the crucial problem of Path Coverage (PC) in the Automatic Test Case Generation (ATCG) by introducing Hypercube-based Learning (HBL) and Tailored Hypercube Based Learning (THBL) which are employed hypercube through an opposition based learning strategy. Experimental research demonstrated the higher PC with fewer test cases and optimized running time.

2.1.2. Metamorphic Testing

It is a property based testing strategy for generation of test cases with addressing test oracles which are written for determining the expected results over selected test cases to check whether the output based on test oracle is matching with the output generated through test case. On this ground, Chen *et al*. [6] presented a pioneer metamorphic testing approach with main aim to generate subsequent test cases through metamorphic relations. It is an inventive strategy for systematic derivation of new test cases, thereby making substantive combination to the progression of software testing techniques. Sun *et al*. [36] have explored this testing approach to construct follow-up test cases from existing source test cases which are generated associated path constraints symbolic execution. The path distance among test cases guided the prioritization of source test cases, hence enhancing the efficiency of the software products.

2.1.3. Regression Testing

It is a type of testing which works over the functional test cases and misbehaves over the nonfunctional test cases. In this testing, software code is slightly modified and checking over the generated test cases whether the code is giving the expected outcome or not. Solanki *et al.* [33] produces experimental analysis for Ant Colony Optimization (ACO) through regression testing approach. Test cases are prioritized against Meta heuristic techniques and improves sources of the quality and diversity of the food.

2.1.4. State Based Testing

It is type of testing which works on the transition of the states and one state is treated as a module of the software and tested over the test cases which flow from one state to another state. In this reference, Pradhan *et al*. [30] proposed an algorithm for state-based test case generation from diverse coverage criteria, by transforming state chart diagrams into a State Chart Intermediate Graph (SCIG). The study introduced efficient algorithms for Round Trip Path (RTP) and All Transition Pair (ATP) criteria, showcasing insights from case studies on Stack Operation and Vending Machine Automation (SOVMA) systems. Experimental findings revealed ATP challenges, resource consumption in All Transition (AT), and RTP's efficiency in addressing transition explosion, contributing insights to semi-HBL in Model-Based Testing (MBT).

2.1.5. Fuzzy Testing

It is a dynamic testing approach and investigated by Barton Miller in the year 1980. It considers coverage criteria and behavior aspect of software code and it is much faster for uncovering the bugs for automatic generation of the test cases. Hasan *et al*. [15] reviewed research work on software testing, considered various approaches of testing and observed that the fuzzy logic enhanced software quality using operational profiles, while fault propagation path design predicted the defects during testing. ATCG reduced required tests for large software programs. The entire review offers valuable insights into diverse test cases generation methods and input on software performance, contributing to a nuanced uncertainty of effective strategies in the Software Development Life Cycle (SDLC).

2.1.6. Hybrid Testing

It is a combination of two or more than two testing approaches. Lakshminarayana and SureshKumar [20] introduced a hybrid approach based on fitness function for optimizing the software test cases. The computed results are compared with Particle Swarm Optimization (PSO), Cuckoo Search (CS), Bee Colony Algorithm (BCA), and Firefly Algorithm (FA) and achieving 65% of success rate. The application is performed over Automated Teller Machine (ATM) which outperforms over the above mentioned algorithms and completes ATM withdrawal operations in just 16.4 seconds, thereby suggesting the approach for applicability in the software testing, especially for banking industries. Sulaiman *et al*. [35] focused on the increasing implementation of optimization algorithms for test case generation in MBT for Software Product Line (SPL). The demand for optimal test case results with a balanced trade-off between cost and effectiveness motivated hyper-heuristic test case generation approach in MBT for SPL, termed Improvement Selection Rules-Modified Choice Function (ISR-MCF) which incorporated three search operators like Non-Dominated Sorting Genetic Algorithm 2 with Low-Level Heuristic (NSGA-2-LLH), Strength Pareto Evolutionary with 2 Low-Level Heuristic (SPEA-2-LLH), and Particle Swarm Optimization with Low-Level Heuristic (PSO-LLH). The evaluation, conducted on a test model, demonstrated that ISR-MCF with NSGA-2-LLH outperformed existing rules in terms of minimization measures (test suite size and execution time) and maximization measures (coverage criteria). De Santiago Junior *et al*. [8] introduced the "many-objective perspective" to enhance Graphical User Interface (GUI) test cases generation, combining search-based optimization with MBT. Meta and hyper-heuristics were employed for both code-driven and use case-driven GUI testing, translating C++ source code into Event Flow Graphs (EFG) and creating EFG's directly via use cases. The evaluation included 32 problem instances, assessing three multi-objective evolutionary algorithms and six selection hyper-heuristics. Authors measured performance through five indicators and a new Multi-Metric Indicator (MMI), highlighted the superior performance of meta-

heuristics, especially Non-dominated Sorting Genetic Algorithm (NSGA)-2, with the choice function hyper-heuristic proving the most effective.

2.2. Tools and Models for Testing

For testing the software, technologists also developed testing tools from time to time. One of such tools is the Detection and Refactoring Tool (DaRT) which is used to address redundant test cases generation in the SDLC by identifying and refactoring the code in small-lazy classes, small methods and duplicate classes [17]. The above tool has android based application whose main aim is to reduce test case generator redundancy through structure and modifications without affecting external functionality. The tool has a notable 28% reduction in the generated test cases and upto 5% improvement in branch coverage. Clark *et al*. [7] also examined the role of agent-based models in simulating complex phenomena and supporting decisions, despite potential consequences from software faults. Positing five research questions addressed the key aspects of test case generation in agent-based models. From an initial search yielding 464 results, the study identified 24 primary publications, utilizing taxonomy to summarize advanced techniques for test cases generation. Results indicated that, while many techniques effectively tested functional requirements at agent and integration levels, few addressed society-level behaviour, and most did not encompass non-functional requirements or "soft goals".

2.3. Grid Search

Grid search is a systematic method used for exploring a specified input space by generating a grid of potential solutions and evaluating each one to find the optimal or near-optimal solutions. The process is particularly useful in scenarios where the input space is continuous and multidimensional, and it can be applied to various optimization problems, including hyper-parameters tuning in machine learning models. It consists of the following steps:

Step 1. Define input ranges and grid resolution.

a) There are ranges of values for each feature or dimension in the input space. For example, if we have two features, then ranges might be (0,1000) (0,1000) for both.
b) The grid resolution is the number of evenly spaced values to generate within each input range. A higher resolution means more values and a finer grid.

Step 2. Generate values for each dimension.

Using a function like numpy. linspace, generate evenly spaced values within each defined input range. For example:

- For a range of (0, 1000) (0, 1000) with a resolution of 20, the values might be [0, 52.63, 105.26, …, 947.37, 1000][0, 52.63, 105.26, …, 947.37, 1000].

Step 3. Generate the Cartesian product.

- The Cartesian product of the lists of values from each dimension is computed to create all possible combinations of the values across dimensions. This results in a comprehensive set of grid solutions that cover the entire input space.

Step 4. Evaluate each solution.

- Each point in the generated grid is evaluated using a target function (e.g., the Greatest Common Divisor (GCD) function in this case). The evaluation results are then adjusted for normalization and scoring using a specified criterion, such as a power function to compute a coverage score.

Hence, grid search is a kind of process of the search for selection of the certainty of the test cases and generally the approach of search has vast application in the machine learning. On the basis of above, a grid search is shown below in the Figure 2.

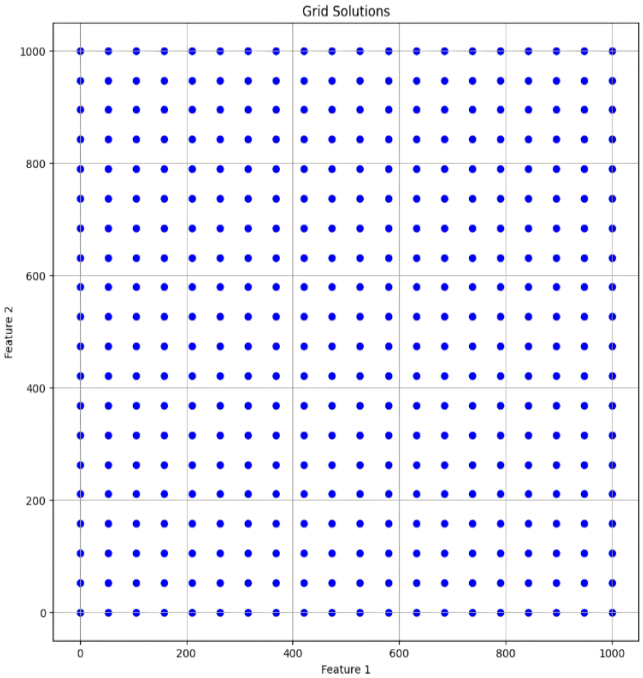

Figure 2. Grid search under HRT.

In this reference, Pontes *et al*. [29] conducted a literature review and proposed an optimized Multi-Layer Perceptron (MLP) network for predicting Average Surface Roughness (ASR) in machining processes. The tuning algorithm, incorporating Design Of Experiments (DOE) techniques, significantly reduced roughness prediction errors, providing an effective method for systematically designing Artificial Neural Network (ANN) models. The concept is applied to two machining processes, the method identified network topologies with substantial reductions in training and testing, resulting in an 82.3% and 71.5% reduction in prediction error compared to original ANN

models and a 9.7% and 46.3% reduction compared to ANNs optimized by a computational tool. The results demonstrated a significant reduction in the dispersion of prediction errors as compared to networks proposed by case studies and a computer package. Liashchynskyi and Liashchynskyi [21] compared three widely used algorithms for hyper-parameter i.e. optimization-grid search, random search, and GA-specifically focusing on application in Neural Architecture Search (NAS). The primary objective was to use the algorithms for constructing a convolutional neural network, and the experimental evaluation was carried out on the Canadian Institute For Advanced Research (CIFAR-10) dataset. The comparison was based on the execution time of the algorithms and the accuracy of the resulting models, providing insights into the performance differences among the approaches. The findings and analyses contributed to the understanding of the effectiveness of the hyper-parameter optimization algorithms in the context of NAS.

2.4. Radial Basis Function (RBF)

It is a mathematical function which takes real valued inputs and produces real valued outputs by considering the distance of real valued inputs from some fixed points as shown in the Figure 3 [9].

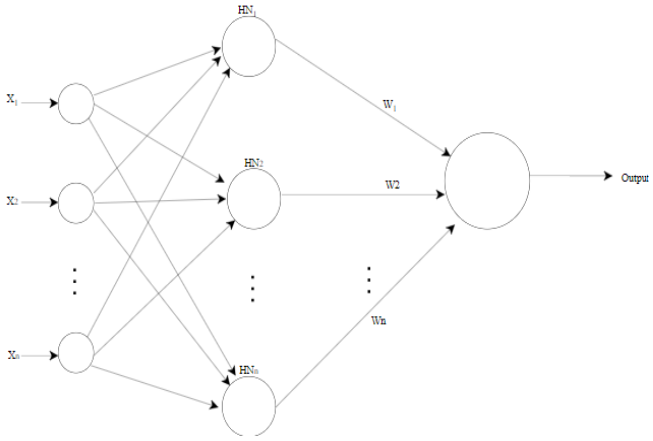

Figure 3. A concept of RBF_NN.

On the basis of definition of RBF, networks are designed using RBF which are known as RBF-NN. Scientists and researchers used the concept of RBF in the field of neural networks. Let us describe some of the important references. Musavi *et al*. [28] employed the RBF technique for training RBF classifiers, addressing objectives related to efficient clustering and determining kernel function widths. The study outlined techniques and conducted empirical tests, confirming the effectiveness of the proposed approach in terms of processing speed and scalability for nonlinear patterns. The research provided valuable insights for enhancing RBF efficiency in interpolation and classification, offering solutions for improved performance in diverse applications. Further, Mulgrew [27] investigated the application of neural networks, specifically general and radial basis functions, with an emphasis on adaptive equalization and interference rejection problems. The article deliberated on the utilization of neural-network-based algorithms, aiming to strike a balance between performance and complexity in the domain of adaptive equalization. The study encompassed a thorough examination of the application of radial basis functions to address challenges associated with adaptive equalization and interference rejection. The results underscored the effectiveness of neural-network-based algorithms, particularly in achieving a favourable trade-off between performance and complexity within the realm of adaptive equalization. Buhmann [5] undertook an exhaustive examination of RBF methods, highlighting the modern applications for approximating multivariate functions, especially in situations where grid data is unavailable. Buhmann [5] focused on current survey of recent advancements, explicating the theoretical underpinnings of RBF techniques and introducing novel categories of RBF's. Particular emphasis is placed on recent discoveries concerning convergence rates in RBF-based interpolation, progress in approximations on spheres, and the efficient numerical computation of inter-polants for large datasets. Schwenker *et al*. [32] investigated learning algorithms for RBF networks, categorizing RBF training into one, two, and three-phase schemes. Two phases RBF learning involves initially training the RBF layer, adjusting centers and scaling parameters, followed by adapting output layer weights. Numerical experiments demonstrated enhanced performance of RBF classifiers through a third back propagation-like training phase, termed three-phase learning, allowing the use of unlabeled training data. Support Vector (SV) learning in RBF networks, a distinct approach, represents a specialized form of one-phase learning. Numerical experiments comparing classifiers showed superior performance of RBF classifiers trained through SV and three-phase learning over two-phase learning. Liu *et al*. [23] used the concept of RBF for generation of test cases through GA.

2.5. Harmony Search

This kind of the search is used for finding the exact or approximate solution of the problem which is inspired by the music and called as meta-heuristic approach of searching the optimized solution. In the present work, this approach is used for efficient generation of automatic test cases which shall be used for enhancing the productivity and reliability of the software projects. A concept of HS is shown below in the Figure 4 [25].

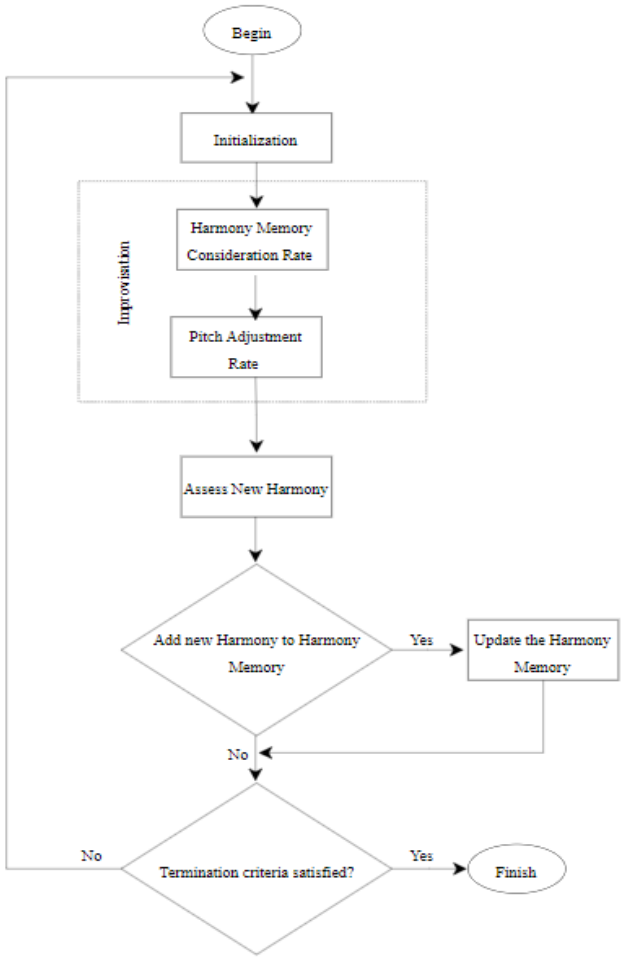

Figure 4. A concept of harmony search.

The details of the Harmony Search Algorithm (HSA) shall be discussed in the subsequent section; however some of the important references are added for clarity of the algorithm. Geem *et al*. [12] explored various optimization algorithms to solve problems in different fields. Traditional techniques, such as Linear Programming (LP), Non-Linear Programming (NLP), and Dynamic Programming (DP), played significant roles, but the limitations led to the exploration of heuristic optimization approaches like simulated annealing, tabu search, and evolutionary algorithms. The study introduced harmony search, a novel heuristic algorithm inspired by the improvisation of music players. The algorithm's performance was demonstrated through applications to a traveling salesman problem and a least-cost pipe network design problem, highlighting its effectiveness in addressing optimization challenges. The study contributed to the broader understanding of heuristic optimization algorithms and provided insights into the potential of new approaches like harmony search. Dubey *et al*. [10] systematically reviewed the HSA, a recently developed meta-heuristic known for its efficient optimization across diverse real-life problems. The paper provided a comprehensive overview of HSM, covering its natural inspiration, conceptual framework, control parameters, and mathematical foundations. It explored HSM's improvement and hybridization with other meta-heuristics, emphasizing its broad applicability in engineering, networking, scheduling, classification, bioinformatics, and more. The study also analyzed HSM variants, including binary, chaotic, multi-objective, and hybridized versions, highlighting the strengths and weaknesses. The findings are based on an analysis of over 100 papers, underscored HSM's adaptability and robustness. The conclusion discussed the future research directions, focusing on HSM's potential through parameter adjustments and hybridization with different algorithms.

Further, Qin *et al*. [31] conducted a systematic review on the HSA and its variants, a novel meta-heuristic inspired by musicians' adjustments. The study covered the basic principle, impact of improvements, and characteristics of different HS variants, analysing the applications. Approximately 100 papers were reviewed, revealing a focus on parameter enhancement and integration with other meta-heuristic algorithms for HS improvements. The primary application domain was engineering optimization and authors emphasized the algorithm's growing real-world applications. Ghiduk and Alharbi [14] also reviewed search-based algorithms in software engineering, focusing on comparing GAs and HSM for test data generation. The study assessed the efficiency in terms of time performance, significance of generated test data, and adequacy for testing criteria. Results showed that HSM's significant speed advantage over GAs, supported by t-Test analysis. Jalila and Mala [18] conducted a literature review on early-stage software testing, emphasizing the need for techniques enabling automated test case generation in initial software development phases and proposed a framework using formal specifications in Object Constraint Language (OCL) for automated test data generation, featuring a novel fitness function, Exit-Predicate-Wise Branch Coverage (EPWBC), and employing the HSA to optimize the test case generation process. Experimental results demonstrated the framework's superiority over other OCL-based test case generation techniques, showcasing the effectiveness of OCL-based testing with the HS algorithm in achieving extensive test coverage and an optimal test suite for improved system quality. Muazu and Maiwada [26] conducted a literature review on pairwise testing, an approach that tests all possible combinations of parameter values to achieve comprehensive coverage. The optimization of generating efficient test suites with minimal size is treated as a search problem, and the HSA is applied to address it. The research introduced PWiseHA, a pair-wise software testing tool developed with an optimization focus using the HSA. Results from PWiseHA demonstrated competitive performance compared to existing pairwise testing tools. Alsewari *et al*. [1] conducted a literature review on combinatorial test case generation strategies, specifically focusing on t-way testing strategies. The paper introduced a novel

approach, the General T-way Harmony Search-based Strategy (GTHS), utilizing the HSA to generate test lists. The chosen algorithm aimed to balance intensification and diversification. Experimental results, benchmarking GTHS against existing optimization-based strategies, demonstrated competitive performance, and particularly supporting high combination degrees ($t \leq 12$). Ghiduk and Alharbi [13] investigated the performance of GA's and HS algorithms in test data generation, comparing the ability and speed. The study empirically compared HSA and GAs, assessing time performance, significance of generated test data, and adequacy to satisfy a given testing criterion. Results indicated that HSA was significantly faster than GA's, supported by a p-value of 0.026, while no significant difference was observed in generating adequate test data, with a p-value of 0.25. The findings contributed to understanding the comparative efficiency of HSA and GAs in the test data generation process.

2.6. Research Gaps

On the basis of exhaustive literature survey, it is observed that different combinations of existing testing approaches in hybrid form may produce better outcome in the optimized time frame and generation of the effective test cases via HSM with RBF-NN is still missing in the literature. The type of hybrid approach is used for automatic generation of test cases by combining the concepts of HS with RBF-NN, called as the HRT approach. The proposed work has the following key contributions:

a) HSA integration: integration of the HSA introduces an adaptive and balanced approach to automated test case generation, effectively optimizing the input space.
b) RBF-NNs for fitness evaluation: utilization of RBF-NN for dynamic test case quality evaluation and guidance enhances the adaptability and efficacy of the HSA.
c) Dynamic fitness landscape adaptation: the model dynamically adapts over multiple iterations, refining its strategy based on changing fitness landscapes. This dynamic adaptation accommodates evolving program complexities.
d) Testing across diverse program types: demonstrating versatility, the model proves effective across various program types, ensuring adaptability in diverse testing scenarios.
e) Enhanced test coverage and quality: the combined use of harmony search and RBF-NN's aims to enhance test coverage, providing high-quality test cases through an iterative refinement process.
f) Automated optimization of test case generation: the model automates test case generation, reducing manual effort. Harmony search explores the solution space, while RBF-NN's provides automated evaluation, making the testing process more efficient and adaptive.
g) Practical application demonstration: practical applications illustrate the model's efficacy in generating test cases covering critical program functionality. These applications span mathematical algorithms, control flow-intensive functions, and complex arithmetic operations.

3. Methodology

In this section, the proposed methodology is described on the basis of framework given in the figure 1 for the automated generation of test cases by integrating HS and RBF-NN techniques. The primary objective is to methodically traverse the input space of a designated program, characterize its intricate behaviour through machine learning, and leverage a heuristic search algorithm to iteratively enhance the quality of generated test cases.

3.1. Test Generator Initialization

The followings are considered for initialization of the test cases generation:

- Target Program: a software code, for which test cases will be generated, is defined as a target program. In the present work, seven Python target programs are selected which are frequently used by the researchers and scientists however the presented approach shall be applicable for other Python scripts.
- Input Ranges: the ranges which are considered for the selected target programs are defined as input ranges. This range is varying from one target program to another based of the involvement of parameters.
- Instantiate RBF-NN: an instance of RBF-NN provides code uses the MLPRegressor class from scikit-learn with specific configurations, such as one hidden layer with 10 neurons and logistic activation.

3.2. Grid Solution and Evaluation

A set of grid solutions is generated by covering the specified input ranges. The algorithm uses the numpylinspace function to create evenly spaced values within the defined input ranges, resulting in a grid of solutions, then evaluated each solution in the generated grid using the target program as GCD function. The results are adjusted for normalization and scoring, using a power function to compute a coverage score.

In the proposed work, the steps for generating the grid solutions are given below:

Grid_Solution()
Input
Feature Ranges: [(0,1000), (0,1000)]
Grid Resolution: 20
Output
Grid Solutions: set of solutions covering the input space
Begin

Initialize Grid Solutions as an empty list
for each dimension in Feature Ranges do
values = linspace(dimension[0], dimension[1], Grid Resolution)
Append values to Grid Solutions alongwith the current dimension
end for
Generate Cartesian Product of Grid Solutions
return Cartesian Product as Grid Solutions
End

The above Grid_Solution() function generates a set of solutions that span the input feature space. It begins by defining the input feature ranges from 0 to 1000 for each dimension. The grid resolution is set to 20, meaning each dimension will be divided into 20 evenly spaced points. The function initializes an empty list called Grid Solutions. It then iterates over each dimension in the feature ranges, using the linspace function to generate evenly spaced values between the minimum and maximum of each range. These values are appended to Grid Solutions for each dimension. Once the values for all dimensions are gathered, the function calculates the Cartesian product of these values, which results in all possible combinations of the grid points across the dimensions. The Cartesian product represents the set of grid solutions that cover the entire input space. Finally, the function returns this Cartesian product as the output.

3.3. RBF Neural Network Training

In this activity, the RBF-NN is trained using the generated solutions and the corresponding evaluations. This step involves for creating a training set with features (input solutions) and targets (normalized GCD scores) and using it to fit the neural network. The steps are summarized below:

RBF_NN_Train()
Input
Program: target program for training
Population: set of solutions for training
Output
RBF-NN: trained neural network
Begin
Features = Extract_Features_From_Population(Population)
Targets = Evaluate_Solutions(Population)
rbf_nn = Initialize_RBF_NeuralNetwork()
rbf_nn.Train(Features, Targets)
End

The above RBF_NN_Train() function is designed to train a Radial Basis Function Neural Network (RBF-NN) using a set of solutions from a target program. The process begins by taking the target program and a population of solutions as input. First, the function extracts features from the population using Extract_Features_From_Population(Population), which processes the population to generate relevant input features. Next, it evaluates these solutions using Evaluate_Solutions(Population) to produce the corresponding target values. An RBF Neural Network (rbf_nn) is then initialized with Initialize_RBF_NeuralNetwork(). The extracted features and their corresponding target values are used to train the RBF-NN via rbf_nn Train(Features, Targets). Once trained, the function outputs the trained RBF Neural Network, ready for further tasks.

3.4. Harmony Search Algorithm

When RBF-NN is trained then, HSA is applied to further refine the test cases. The algorithm initializes a memory of solutions, iteratively generates and evaluates new solutions, updates the memory based on fitness, and adjusts mutation and crossover probabilities. The reason for selection of said approach is that it produces faster results in comparison of other optimization algorithms [13].

The steps of HSA are given below:

Harmony_Search()
Input
Program: target program to be tested
Feature Ranges: [(0,1000), (0,1000)]
Max Iterations: 100
Initial_Mutation_Probability: 0.2
Initial_Crossover_Probability: 0.7
Output
Best Solution: optimal solution found by harmony search
Begin
Harmony Memory Initialization()
best_solution = None
best_score=0
mutation_prob = Initial_Mutation_Probability
crossover_prob = Initial_Crossover_Probability
iteration = 0
while iteration < Max Iterations do
new_solution = Generate_Random_Solution_From_Grid()
Add new_solution to Harmony Memory
Sort Harmony Memory based on Fitness_Function
Select Top Solutions from Harmony Memory
if Fitness Function(new_solution) >best_score then
best_solution = new_solution
best_score = Fitness_Function(new_solution)
end if
*mutation_prob = max(0.01, mutation_prob * 0.95)*
*crossover_prob = min(0.9, crossover_prob * 1.05)*
iteration = iteration + 1
end while
returnbest_solution
End

The above RBF_NN_Train() function is designed to train a (RBF-NN) using a set of solutions from a target program. The process begins by taking the target program and a population of solutions as input. First, the function extracts features from the population using Extract_Features_From_Population(Population), which processes the population to generate relevant input features. Next, it evaluates these solutions using Evaluate_Solutions(Population) to produce the corresponding target values. An RBF Neural Network (rbf_nn) is then initialized with Initialize_RBF_NeuralNetwork(). The extracted features and their corresponding target values are used to train the RBF-NN via rbf_nn Train(Features, Targets). Once trained, the function outputs the trained

RBF Neural Network, ready for further tasks.

3.5. Harmony Radial Testing

The concept of RBF-NN is integrated with the harmony search and expanded in the form of HRT which is given below:

HRT()
Input:
- *GCD_function: A function that calculates the Greatest Common Divisor (GCD) of two integers.*

Output:
- *A test case (pair of integers) that maximizescoverage score for the GCD_function.*

Parameters:
- *feature_ranges: A list of tuples defining the minimum and maximum values for each input feature (integer);*
- *max_iterations: The maximum number of iterations for the HSA;*
- *initial_mutation_prob: The initial probability of mutating a solution during an iteration;*
- *initial_crossover_prob: The initial probability of performing crossover between solutions during an iteration;*
- *num_iterations (for run_experiment): The number of independent experiments to run.*

Algorithm Steps:

1. *Initialization:*
   - *Define feature ranges for input integers;*
   - *Create an RBF Neural Network (NN) for fitness estimation;*
   - *Initialize a Harmony Memory (HM) with 5 randomly chosen test cases from the defined grid.*
   - *Set best_solution and best_score to None;*
   - *Set initial values for mutation_prob and crossover_prob.*
2. *Harmony Search Loop (for each iteration):*
   - *Generate a new random test case;*
   - *Add the new solution to the HM;*
   - *Select the top 5 solutions (based on fitness) from the HM as the new HM;*
   - *Update best_solution and best_score if the new solution has a higher fitness score;*
   - *Adjust mutation_prob and crossover_prob using decay and amplification factors.*
3. *Fitness Function:*
   - *Use the trained RBF NN to predict the coverage score for a given test case.*
4. *Training RBF NN (within run_experiment):*
   - *Generate a set of initial test cases using generate_grid_solutions;*
   - *Train the RBF-NN using the generated test cases and their corresponding coverage scores (obtained using evaluate_solution).*
5. *Run Experiment :*
   - *Perform multiple independent runs of the HSA;*
   - *Return a list containing information about the best solution found in each run.*
6. *Selection and Output:*
   - *Find the best test case (highest coverage score) across all experiment runs.*
   - *Print details about the chosen test case and its coverage score.*

The HRT() function is an optimization algorithm designed to find a test case that maximizes the coverage score programs such as triangle, GCD, bessel, calday, numbers, remainderSth, and Complex, using harmony search. function using harmony search. It starts by defining the input parameters, including the feature ranges for the integer inputs, maximum iterations, initial mutation and crossover probabilities, and the number of independent experiments to run. The process begins with initialization: it defines the feature ranges, creates a RBF-NN for fitness estimation, and initializes HM with 5 randomly chosen test cases from a predefined grid. Both best_solution and best_score are set to none, and initial values for mutation and crossover probabilities are established. During each iteration of the harmony search loop, a new random test case is generated and added to the HM. The memory is then sorted based on the fitness scores, and the top 5 solutions are retained. If the new test case has a higher fitness score than the current best, it updates the best_solution and best_score. Mutation and crossover probabilities are adjusted over time to balance exploration and exploitation. The fitness function used in this process is based on the trained RBF-NN, which predicts the coverage score for each test case. Before running the main optimization, the RBF-NN is trained with a set of initial test cases generated by generate_grid_solutions, with their coverage scores obtained from evaluating the GCD function. The function executes multiple independent runs of the HSA (based on num_iterations), recording the best solution found in each run. After all experiments, the test case with the highest coverage score across all runs is selected as the final result, and details of this test case, including its coverage score, are printed. This method effectively integrates machine learning with optimization techniques to ensure comprehensive testing of the GCD function.

3.6. ExperimentalExecution

In this, a series of experiments is conducted by executing the HRT algorithm for a specified number of iterations. New solutions are generated in each iteration, evaluated using the RBF-NN, and the memory is updated to keep the best solutions.

3.7. Results Analysis

The results of the experiments are analyzed to identify the best-generated test cases. The analysis includes computing and comparing coverage scores obtained from the HSA. The best test case and its coverage score are reported.

3.8. Evaluation of the Solution

Individual solutions are evaluated using the target program and compute coverage scores. This step is essential for both the initial grid solutions and the

solutions generated by the HRT algorithm. The methodology involves for initializing the test case generator with the target program and input ranges, generating an initial set of solutions using a grid, training an RBF-NN to approximate the program's behavior, and refining the solutions using the HRT algorithm. The experiments are conducted to find the best test case based on coverage scores, and individual solutions are evaluated using the target program.

4. Results and Discussion

The HRT algorithm reveals the key parameters which are influencing its performance. The population size is considered as 50 which determine candidate solutions per iteration, crucial for comprehensive exploration. A larger population enhances thorough search. The maximum iterations is considered as 100, influences overall runtime by specifying the number of iterations. Initial mutation probability is taken as 0.2, introduces randomness, aiding effective exploration, while initial crossover probability is 0.7, promote feature sharing. Collectively, above-mentioned parameters shape the behaviour of HSA and efficacy in search and optimization tasks.

4.1. Test Case Generation and Verification of Branch Coverage

In the context of generating branch coverage test cases, the experiment utilized seven Python programs to assess the viability and efficacy of the test cases generation method as proposed in this paper, which leverages the HSA and RBF-NN. The Python programs employed in this experiment include GCD, triangle, remainderSth, and numbers. To elucidate the advantages of the proposed test case generation algorithm defined HRT, we will compare it against three alternative methods for branch coverage test case generation: the adaptive GA as Primal-Dual Genetic Algorithm (PDGA), traditional GA as Simple Genetic Algorithm (SGA), and random test generation method as random. The assessment metrics for the experiment encompass the average branch coverage rate, Maximum branch Coverage (MaxC) rate, and average convergence algebra. It is crucial to scrutinize the results to ensure the accuracy and reliability of the information provided. The branch coverage rate is calculated as in the Equation (1) [23]:

$$t = \frac{n}{m} \times 100 \tag{1}$$

where, *t* symbolizes the branch coverage rate, n denotes the count of evaluation results, and m represents the aggregate of judgment results. The standardized parameters across the four branch coverage test case generation methods, dictate a fixed population size of 50 and a maximum threshold of 100 evolutionary iterations. To mitigate the potential impact of stochastic variables on experimental outcomes, each method undergoes 50 iterations for every program subject to analysis. The evaluative metrics for this experiment encompass the average branch coverage rate, the MaxC rate, and the average convergence algebra. This methodological stringency ensures a thorough and quantitatively precise assessment of the performance attributes and reliability metrics inherent in the methods under examination. The average branch coverage rate *Ac* is calculated as in Equation (2) [23]:

$$Ac = \frac{\sum_1^n t\ (1,2, \ldots . , n)}{n} \tag{2}$$

where, *Ac* represents the average branch coverage rate and t represents the branch coverage rate of the program under test after it is executed in the algorithm.

MaxC rate is the highest branch coverage rate achieved by the program under test after multiple executions in an algorithm. The proposed HRT method is evaluated using existing techniques such as PDAG [3], SGA [22], and the random approach.

The results from 50 runs of each of the four algorithms for the seven tested programs are tabulated in the Table 1, presenting both the Average Coverage (AC) rate and the MaxC rate.

The branch coverage rates of the four algorithms are used on program p1 GCD are shown in the Table 1. Notably, all four algorithms have MaxC of 100%; however, the random method is unable to reach an average branch coverage rate higher than 80%. For program p2 (triangle), the random algorithm fails to meet the 80% requirement for branch coverage, whereas the HRT based test case generation techniques produce a MaxC rate of 100%.

Table 1. Comparison of branch coverage under HRT with existing approaches.

	Random		SGA		PDAG		HRT	
Programs	**Ac%**	**MaxC%**	**Ac%**	**MaxC%**	**Ac%**	**MaxC%**	**Ac%**	**MaxC%**
P1 (GCD)	77.50	100.00	100.00	100.00	100.00	100.00	100.00	100.00
P2 (triangle)	55.60	75.36	93.80	100.00	100.00	100.00	100.00	100.00
P3 (beseel)	52.81	76.19	95.86	100.00	100.00	100.00	100.00	100.00
P4 (calday)	67.39	91.91	94.27	100.00	100.00	100.00	100.00	100.00
P5 (remainderSth)	65.89	83.33	94.65	100.00	100.00	100.00	100.00	100.00
P6 (numbers)	45.68	72.10	90.61	97.43	98.23	100.00	100.00	100.00
P7 (complex)	48.21	63.27	92.86	92.33	100.00	100.00	100.00	100.00

More specifically, the random method only reaches 55.60%, whereas the SGA algorithm obtains an excellent average branch coverage rate of 93.80%. Programs p3, p4, and p5 demonstrate that the PDGA,

SGA, and HRT algorithms get 100% MaxC, while the random algorithm obtains 76.19%, 91.91%, and 83.33% for the comparable programs. The SGA and Random algorithms only achieve 100% branch coverage on the maximum levels, only the PDGA and HRT algorithms in program p7 achieve 100% branch coverage on both the average and maximum levels. The comparison demonstrates that the SGA algorithm consistently outperforms the Random approach in terms of average and MaxC rates. With program p6, the proposed HRT technique achieves a remarkable 98.23% average branch coverage for the PDGA algorithm, as well as 100% maximum coverage and average branch coverage. When the number of branches reaches a certain point, the PDGA algorithm notably becomes unstable, while the proposed HRT technique shows relative stability.

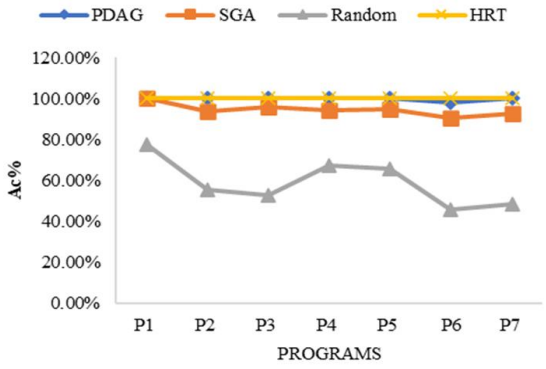

Figure 5. Comparison of average coverage rate.

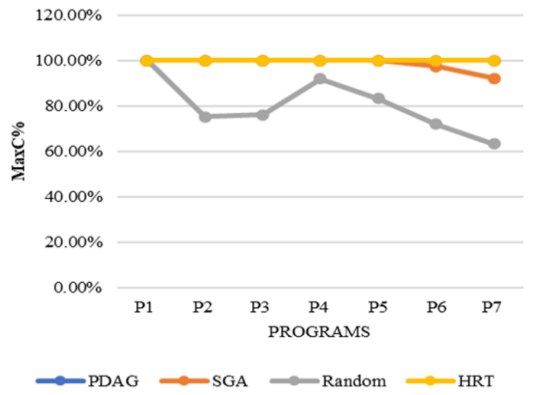

Figure 6. Comparison of maximum coverage rate.

The highlights as shown in the Figures 5 and 6, represent the performance of HRT for test case generation. The HRT algorithm, which is new to this study, is more effective than the other three approaches. The software achieves a notable feat of 100% coverage in both average and maximum metrics, which are derived from the six previously described occurrences. This result validates both the technical viability and efficacy of the methods described in this research.

4.2. Average Percentage Fault Detection Rate

Average Percentage Fault Detection Rate (APFD) is a pivotal metric in software testing, quantifies the average effectiveness by considering both efficacy and efficiency for individual test cases. It evaluates a test case's ability to detect faults and sequencing, optimizing testing effectiveness. This aids in prioritizing test cases and refining fault detection strategies, providing valuable insights for enhancing software testing protocols. The APFD is computed using in Equation (3):

$$APFD = 1 - \frac{Tf_1 + Tf_2 + \cdots + Tf_n}{nm} \quad (3)$$

where, Tf1, Tf2, …, Tfn denote the fault detection positions, representing the sequential order in which each fault is identified by the test cases within the test suite, n signifies the total number of faults present in the system, m represents the total number of test cases within the test suites. Table 2 presents an in-depth comparative analysis of four optimization algorithms-GA, ACO, Bee Colony Optimization (BCO), and HRT with a focus on their APFD [33].

Table 2. Comparison of APFD under HRT with exiting approaches.

	APFD			
Programs	**GA**	**ACO**	**BCO**	**HRT**
Triangle	0.88	0.93	0.95	1.0
GCD	-	-	-	1.0
Beseel	-	-	-	1.0
Calday	-	-	-	1.0
Numbers	-	-	-	1.0
remainderSth	-	-	-	1.0
Complex	-	-	-	1.0

The APFD scores serve as quantitative indicators of the algorithms' efficacy in fault detection within the context of software testing. GA exhibited a commendable APFD score of 0.88, underscoring its robust performance in fault identification. ACO surpassed GA, demonstrating a higher APFD score of 0.93, indicative of a more proficient fault detection capability. BCO outperformed both GA and ACO, achieving an APFD score of 0.95. However, the HRT showcased unparalleled effectiveness by achieving a perfect APFD score of 1.0. This exceptional performance suggests that HRT, leveraging the HSA in conjunction with RBF-NN, excelled in fault detection with unparalleled precision and completeness compared to its algorithmic counterparts. The incorporation of the HSA and the utilization of an RBF-NN in HRT contribute to its superior fault detection capabilities, making it a compelling choice for optimizing fault detection strategies in the realm of software testing.

4.3. Strengths and Limitations of HRT

The followings are the strengths of the presented approach:

a) Enhanced Branch Coverage.

- Focused Testing: the HRT approach is designed to address the challenges of branch coverage in software testing, ensuring that more branches of the software code are tested, leading to more

thorough testing and detection of potential errors.

b) Efficiency in Test Case Generation.

- Automatic Generation: by automating the generation of test cases, the HRT approach significantly reduces the manual effort and time required, which is typically resource-intensive and subjective.
- Optimization: harmony search optimizes the selection of test cases by exploring the input space more effectively compared to random or less sophisticated methods.

c) Adaptive Learning.

- RBF-NN: the integration of RBF-NNs allows for adaptive learning, which helps in simulating the fitness function dynamically. This improves the accuracy and relevance of the generated test cases based on the evolving software code.

d) Comparative Performance.

- Benchmarking: the document provides empirical results comparing the HRT approach with other methods like PDGA, SGA, and random methods. The HRT approach shows superior performance in these comparisons, as depicted in the form of tables and graphs.

e) Improved Software Quality.

- Systematic Coverage: the method ensures systematic coverage of the software, contributing to higher reliability and correctness of the software systems by uncovering defects that might be missed by less thorough testing methods.

The followings are the limitations of the presented approach:

a) Complexity in Implementation.

- Integration Challenges: combining harmony search with RBF-NNs may be complex. Implementing this hybrid model requires advanced understanding and expertise in both heuristic optimization and neural networks.
- Parameter Tuning: the success of the approach heavily relies on appropriate parameter tuning for both harmony search and the RBF-NN which can be time-consuming and require expert knowledge of datasets.

b) Resource Intensive.

- High Computational Demand: training the RBF-NN and running the HSA on large and complex datasets can be computationally intensive. This might necessitate high-performance computing resources, which may not be available in all environments.
- Memory and Processing Power: significant memory and processing power are required, which could be a limitation for smaller organizations or projects with limited computational resources.

c) Scalability and Adaptability.

- Scalability Issues: while the approach works well for the tested Python codes, its scalability to very large and complex software systems is not yet fully validated.
- Adaptability to Different Domains: the approach may need customization and adaptation to work effectively across different domains and types of software, which could limit its general applicability.

d) Empirical Validation.

- Limited real-world testing: although the document presents comparative results, the HRT approach requires more extensive empirical validation across a wider range of real-world applications to establish its robustness and reliability fully.
- Potential overfitting: there's a risk that the RBF-NN might over fit to the specific examples used in training, which could limit its effectiveness on unseen or significantly different software.

5. Conclusions and Future Scope

From the above work, it is concluded that HRT algorithm which is the integration of the HSA with RBF-NN is a very useful for automatic generation of test cases with outstanding performance over the software codes. In this work seven software codes, consistently achieved 100% for both the maximum branch coverage rate and average branch coverage rate. In contrast to previous algorithms such as Random, SGA, and PDGA, HRT achieved full branch coverage and could automatically produce complete test cases. Since it is a new approach of testing and outperforming over the Python software codes, therefore, it can also be applied for the various software codes developed during the phases of software development by the software industries. The HSA with RBF-NN optimization consistently yielded reliable results, demonstrating the usefulness of HRT as a tool for enhancing software testing processes. The presented technique improved the test coverage for the software codes as 100% which was not through other algorithms. Strengths and weaknesses are also described in the work. The HRT approach offers significant benefits, particularly in enhancing branch coverage and automating the test case generation process, thus improving software reliability and efficiency. However, in future, its complexity, resource intensity, need for extensive empirical validation, and potential issues with scalability and adaptability pose challenges that need to be addressed to fully realize its potential across diverse software development scenarios.

References

[1] Alsewari A., Poston R., Zamli K., Balfaqih M., and Aloufi K., "Combinatorial Test List Generation based on Harmony Search Algorithm," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, no. 6, pp. 3361-3377, 2020. https://doi.org/10.1007/s12652-020-01696-7

[2] Aristoxenus., Macran., and Stewart H., *Aristoxenus Harmonika Stoicheia. The Harmonics of Aristoxenus*, Oxford, Clarendon Press, 1902. https://archive.org/details/aristoxenouharmo00arisuoft/aristoxenouharmo00arisuoft/

[3] Bao X., Xiong Z., Zhang N., Qian J., Wu B., and Zhang W., "Path-Oriented Test Cases Generation Based Adaptive Genetic Algorithm," *PloS One*, vol. 12, no. 11, pp. 1-17, 2017. https://doi.org/10.1371/journal.pone.0187471

[4] Broomhead D. and Lowe D., "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, vol. 2, pp. 321-355, 1988. https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1988-Broomhead-CS.pdf

[5] Buhmann M., "Radial Basis Functions," *Acta Numerica*, vol. 9, pp. 1-38, 2000. https://doi.org/10.1017/S0962492900000015

[6] Chen T., Cheung S., and Yiu S., "Metamorphic Testing: A New Approach for Generating Next Test Cases," *arXiv Preprint*, vol. arXiv:2002.12543, pp. 11, 2020. https://doi.org/10.48550/arXiv.2002.12543

[7] Clark A., Walkinshaw N., and Hierons R., "Test Case Generation for Agent-Based Models: A Systematic Literature Review," *Information and Software Technology*, vol. 135, pp. 106567, 2021. https://doi.org/10.1016/j.infsof.2021.106567

[8] De Santiago Junior V., Ozcan E., and Balera J., "Many-Objective Test Case Generation for Graphical User Interface Applications Via Search-Based and Model-Based Testing," *Expert Systems with Applications*, vol. 208, pp. 118075, 2022. https://doi.org/10.1016/j.eswa.2022.118075

[9] Ding S., Xu L., Su C., and Jin F., "An Optimizing Method of RBF Neural Network based on Genetic Algorithm," *Neural Computing and Applications*, vol. 21, no. 2, pp. 333-336, 2012. DOI:10.1007/s00521-011-0702-7

[10] Dubey M., Kumar V., Kaur M., and Dao T., "A Systematic Review on Harmony Search Algorithm: Theory, Literature, and Applications," *Mathematical Problems in Engineering*, vol. 2021, pp. 1-22, 2021. https://doi.org/10.1155/2021/5594267

[11] GeeksforGeeks, History of Software Testing, https://www.geeksforgeeks.org/history-of-software-testing/, Last Visited, 2024.

[12] Geem Z., Kim J., and Loganathan G., "A New Heuristic Optimization Algorithm: Harmony Search," *Simulation*, vol. 76, no. 2, pp. 60-68, 2001. https://doi.org/10.1177/003754970107600201

[13] Ghiduk A. and Alharbi A., "Generating of Test Data by Harmony Search Against Genetic Algorithms," *Intelligent Automation and Soft Computing*, vol. 36, no. 1, pp. 647-665, 2023. https://doi.org/10.32604/iasc.2023.031865

[14] Ghiduk A. and Alharbi A., "Generating Test Data using Harmony Search Versus Genetic Algorithms," *Intelligent Automation and Soft Computing*, vol. 36, no. 1, pp. 647-665, 2023. https://www.techscience.com/iasc/v36n1/50002/html

[15] Hasan D., Hussan B., Zeebaree S., Ahmed D., Kareem O., and Sadeeq M., "The Impact of Test Case Generation Methods on the Software Performance: A Review," *International Journal of Science and Business*, vol. 5, no. 6, pp. 33-44, 2021. https://ijsab.com/volume-5-issue-6/3860

[16] Hassoun M., *Fundamentals of Artificial Neural Networks*, MIT Press, 1995. https://books.google.jo/books/about/Fundamentals_of_Artificial_Neural_Networ.html?id=Otk32Y3QkxQC&redir_esc=y

[17] Ibrahim R., Ahmed M., Nayak R., and Jamel S., "Reducing Redundancy of Test Cases Generation Using Code Smell Detection and Refactoring," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 3, pp. 367-374, 2020. https://doi.org/10.1016/j.jksuci.2018.06.005

[18] Jalila A. and Mala D., "Automated Optimal Test Data Generation for OCL Specification with Harmony Search Algorithm," *International Journal of Business Intelligence and Data Mining*, vol. 16, no. 2, pp. 231-259, 2020. https://doi.org/10.1504/IJBIDM.2020.104743

[19] Ji S., Chen Q., and Zhang P., "Neural Network-Based Test Case Generation for Data-Flow-Oriented Testing," *in Proceedings of the IEEE International Conference on Artificial Intelligence Testing*, Newark, pp. 35-36, 2019. https://doi.org/10.1109/AITest.2019.00-11

[20] Lakshminarayana P. and SureshKumar T., "Automatic Generation and Optimization of Test Case Using Hybrid Cuckoo Search and Bee Colony Algorithm," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59-72, 2021. https://doi.org/10.1515/jisys-2019-0051

[21] Liashchynskyi P. and Liashchynskyi P., "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS," *arXiv Preprint*, vol. arXiv:1912.06059, pp. 1-11, 2019. https://doi.org/10.48550/arXiv.1912.06059

[22] Lin J. and Yeh P., "Automatic Test Data Generation for Path Testing Using GAs," *Information Sciences*, vol. 131, no. 1-4, pp. 47-64, 2001. https://doi.org/10.1016/S0020-0255(00)00093-1

[23] Liu Z., Yang X., Zhang S., Liu Y., Zhao Y., and

Zheng W., "Automatic Generation of Test Cases Based on Genetic Algorithm and RBF Neural Network," *Mobile Information Systems*, vol. 2021, no. 1, pp. 1-9, 2022. https://doi.org/10.1155/2022/1489063

[24] Mahalakshmi G., Vijayan V., and Antony B., "Named Entity Recognition for Automated Test Case Generation," *The International Arab Journal of Information Technology*, vol. 15, no. 1, pp. 112-120, 2018. https://www.iajit.org/PDF/January%202018,%20No.%201/9172.pdf

[25] Manjarres D., Landa-Torres I., Gil-Lopez S., Del Ser J., Bilbao M., Salcedo-Sanz S., and Geem Z., "A Survey on Applications of the Harmony Search Algorithm," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 8, pp. 1818-1831, 2013. https://doi.org/10.1016/j.engappai.2013.05.008

[26] Muazu A. and Maiwada U., "PWiseHA: Harmony Search Algorithm for Test Suites Generation using Pairwise Techniques," *International Journal of Computer and Information Technology*, vol. 9, no. 4, pp. 91-98, 2020. https://doi.org/10.24203/ijcit.v9i4.23

[27] Mulgrew B., "Applying Radial Basis Functions," *IEEE Signal Processing Magazine*, vol. 13, no. 2, pp. 50-65, 1996. https://doi.org/10.1109/79.487041

[28] Musavi M., Ahmed W., Chan K., Faris K., and Hummels D., "On the Training of Radial Basis Function Classifiers," *Neural Networks*, vol. 5, no. 4, pp. 595-603, 1992. https://www.sciencedirect.com/science/article/abs/pii/S0893608005800383

[29] Pontes F., Amorim G., Balestrassi P., Paiva A., and Ferreira J., "Design of Experiments and Focused Grid Search for Neural Network Parameter Optimization," *Neurocomputing*, vol. 186, pp. 22-34, 2016. https://doi.org/10.1016/j.neucom.2015.12.061

[30] Pradhan S., Ray M., and Swain S., "Transition Coverage-Based Test Case Generation from State Chart Diagram," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 3, pp. 993-1002, 2022. https://doi.org/10.1016/j.jksuci.2019.05.005

[31] Qin F., Zain A., and Zhou K., "Harmony Search Algorithm and Related Variants: A Systematic Review," *Swarm and Evolutionary Computation*, vol. 74, pp. 101126, 2022. https://doi.org/10.1016/j.swevo.2022.101126

[32] Schwenker F., Kestler H., and Palm G., "Three Learning Phases for Radial-Basis-Function Networks," *Neural Networks*, vol. 14, no. 4-5, pp. 439-458, 2001. https://doi.org/10.1016/S0893-6080(01)00027-2

[33] Solanki K., Singh Y., and Dalal S., "Experimental Analysis of m-ACO Technique for Regression Testing," *Indian Journal of Science and Technology*, vol. 9, no. 30, pp. 1-7, 2016. DOI:10.17485/ijst/2016/v9i30/86588

[34] Su Q., Cai G., Hu Z., and Yang X., "Test Case Generation Using Improved Differential Evolution Algorithms with Novel Hypercube-Based Learning Strategies," *Engineering Applications of Artificial Intelligence*, vol. 112, pp. 104840, 2022. https://doi.org/10.1016/j.engappai.2022.104840

[35] Sulaiman R., Jawawi D., and Halim S., "Cost-Effective Test Case Generation with the Hyper-Heuristic for Software Product Line Testing," *Advances in Engineering Software*, vol. 175, pp. 103335, 2023. https://doi.org/10.1016/j.advengsoft.2022.103335

[36] Sun C., Liu B., Fu A., Liu Y., and Liu H., "Path-Directed Source Test Case Generation and Prioritization in Metamorphic Testing," *Journal of Systems and Software*, vol. 183, pp. 111091, 2022. https://doi.org/10.1016/j.jss.2021.111091

[37] Zhang M., Ali S., and Yue T., "Uncertainty-Wise Test Case Generation and Minimization for Cyber-Physical Systems," *Journal of Systems and Software*, vol. 153, pp. 1-21, 2019. https://doi.org/10.1016/j.jss.2019.03.011

Hemant Kumar received MCA degree from Subharti University in the year 2018 and presently pursuing a Ph.D. programme in Computer Science from Babasaheb Bhimrao Ambedkar University, Lucknow. He has published several research articles in the field of Software Testing and Machine Learning. His research interests are Software Engineering, Artificial Intelligence, Machine Learning, and Cyber Security.

Vipin Saxena received his Ph.D. degree from Indian Institute of Technology, Roorkee, Uttarakhand, India. Presently, he is working as a Professor in the Department of Computer Science, at Babasaheb Bhimrao Ambedkar University, Lucknow, India. He has 30 years of teaching and 33 years of research experience and published more than 250 research articles in International and National Journals and Conferences, authored 05 books in the field of Computer Science and Scientific Computing, attended 62 International and National Conferences, and received three National Awards for meritorious research work in the field of Computer Science.