Enhanced Multi-Verse Optimizer (TMVO) and Applying it in Test Data Generation for Path Testing

Mohammad Hashem Ryalat[1]*, Hussam N. Fakhouri[2], Jamal Zraqou[3], Faten Hamad[4], Mamon S. Alzboun[5], Ahmad K. Al hwaitat[6]

Department of Computer Science, Al-Balqa Applied University, Salt, Jordan[*1],
Department of Data Science & Artificial Intelligence, University of Petra, Amman, Jordan[2],
Department of Virtual and Augmented Reality, University of Petra, Amman, Jordan[3]
Department of Information Studies, Sultan Qaboos University, Muscat, Oman[4]
Department of Curricula and Instruction, Al al-Bayt University, Mafraq, Jordan[5]
Department of Computer Science, University of Jordan, Amman, Jordan[6]

***Abstract*—Data testing is a vital part of the software development process, and there are various approaches available to improve the exploration of all possible software code paths. This study introduces two contributions. Firstly, an improved version of the Multi-verse Optimizer called Testing Multi-Verse Optimizer (TMVO) is proposed, which takes into account the movement of the swarm and the mean of the two best solutions in the universe. The particles move towards the optimal solution by using a mean-based algorithm model, which guarantees efficient exploration and exploitation. Secondly, TMVO is applied to automatically develop test cases for structural data testing, particularly path testing. Instead of automating the entire testing process, the focus is on centralizing automated procedures for collecting testing data. Automation for generating testing data is becoming increasingly popular due to the high cost of manual data generation. To evaluate the effectiveness of TMVO, it was tested on various well-known functions as well as five programs that presented unique challenges in testing. The test results indicated that TMVO performed better than the original MVO algorithm on the majority of the tested functions.**

Keywords*—MVO; optimization; testing; swarm intelligence; multi-verse optimizer

I. Introduction

The term "optimization" describes the process of identifying the most optimal search solutions that are likely to resolve a particular issue. There is more than one conventional and meta-heuristic optimization strategy available. The standard techniques are gradient-based and have a faster execution time than convergence. On the other hand, these methods are not applicable to multimodal functions that are neither differentiable nor predictable. Thus, this technique does not allow for the discovery of the global optimal solution. Due to the fact that they start with only one point, it gets trapped at the local optimal value. There are many other search strategies that can be used to solve this problem; however, most of them require additional assistance that is based on exponential time, which makes them more time-consuming [1]. As a result, meta-heuristic optimization approaches have become the most widely used approach. Meanwhile, intelligent algorithms are increasingly used in the development of applications, testing, and the making of business decisions in today's world [2][3].

The use of meta-heuristics has been increasingly widespread over the past two decades. Computer researchers in a wide variety of domains are familiar with meta-heuristic techniques such as the Genetic Algorithm, multi-verse, and Particle Swarm Optimization, amongst others. Because of its ease of use, adaptability, and absence of approaches requiring derivation, meta-heuristic has garnered a lot of attention in recent years [4] [5]. Techniques for testing software include both black-box and white box testing. In the black-box method of software testing, the tester is only privy to the system's architecture. He or she is not privy to any information regarding the program's internal design and does not have access to the source code. Its purpose is to guarantee that the system accepts all of the necessary inputs in the way that was described and produces results that are accurate. White box testing, also known as structural testing, focuses on investigating the internal logic and structure of the source code being tested. During the structural test, each possible code path will be checked for a predetermined set of test information inputs. It is very important to select a diverse control flow way to test since there are a large number of paths for test succession, and performing the tests in succession can be difficult. Finding connections between system components, choosing those paths, creating test data for every path, and assessing test results are only a few examples of the many problem viewing paths involved in software testing [6].

The white box test criteria for software testing, such as branch coverage, focus on the process of locating a group of test cases that increases the likelihood of error discovery. Within the context of this approach, an experiment will serve as the indication that triggers the calling of the test routines with specific input group values. After that, those drivers will make a comparison between the output and the one that was relied upon. Utilizing known inputs that can be put to use but will ultimately prove to be impossible, allowing for an infinite supply of them. As a result, the primary focus of automated software testing is on the process of naturally locating the smallest set of inputs in order to broaden the scope of the test criteria [7]. When it comes to the process of developing test cases for critical path coverage testing, the concept of linear coded sequence is absolutely necessary. It is possible that the productivity of the development of all of these important paths

can be increased, and at the same time, it will be appropriate to create test cases with the assistance of a variety of testing tools so that those tested programs can be investigated. This can be done through all of the important paths [8].

It has been shown by the "No Free Lunch" theorem [9] that metaheuristics do not always succeed in solving optimization issues. Results show that metaheuristic optimization works well for one type of optimization issue but not another. For the aforementioned causes, it is important to create a more efficient optimization metaheuristic algorithm [10], [11].

The primary contribution of this study is the introduction of an improved version of the Multi-verse Optimizer, named the Testing Multi-Verse Optimizer (TMVO). Instead of focusing on a specific region, TMVO takes into account the mobility of the swarm and the average of the two best solutions across the universe. A mean-based algorithm model is employed to guide particle movement towards the optimal solution. TMVO's proposed movement equations enable effective space exploration and utilization, and also address the issue of poor convergence, providing an additional benefit by escaping local minima.

The second contribution of this study involves the application of TMVO algorithm, an enhanced swarm intelligence metaheuristic, to address the issue of single objective optimization in the automated generation of test cases for structural data testing, particularly path testing. Rather than automating the entire testing process, TMVO focuses on centralizing automated procedures for collecting testing data. The proposed TMVO achieves this goal by directing the swarm based on the past performance of the top three solutions discovered by the swarm. The population search history is also utilized to provide an alternative answer, which is the mean of the three best spots identified so far, thus improving the particles' ability to explore the space. This results in more opportunities for the swarm particles to be discovered and utilized, thereby increasing the likelihood of achieving a global optimum while avoiding a local minimum challenge. To overcome these challenges, the direction of particle flow is switched with each cycle.

Due to the absence of a universally applicable metaheuristic that can be used to address all optimization issues, and the fact that no metaheuristic has proven to be effective for solving all identified optimization problems, many swarm intelligence studies have focused on optimizing specific systems.

Route testing is a methodology for testing software that involves a search of the program domain for test cases that, when combined with the code, will cause the program to follow a specified path. Path testing is an optimization issue with no unique solution due to the unlimited number of possible pathways in a program. Consequently, it is only realistic to pick a fraction of these paths for testing. If the pathways to be tested have been clearly described and an adequate fitness function has been constructed, then TMVO might be used for this purpose. In this work, a test case is treated as a representative of a generation, with the chosen target route serving as the endpoint toward which the algorithm is directed.

This study aimed to address one of the most well-known problems in software testing by proposing an improved swarm intelligence metaheuristic method, called TMVO, to resolve the route testing problem. The TMVO method was created to address the aforementioned issues and proposed a better route for the swarm particles to follow, improving the movement strategy of a swarm of particles. To evaluate the algorithm's efficacy, a battery of benchmark functions was used, and its exploitation, exploration, global optimal solution, and best path-finding abilities were tested across these three domains. The results were compared to those of a popular metaheuristic technique, and several indicators, both visual and statistical, were used to assess the quality of the output. The proposed enhanced technique successfully solved the single-objective optimization issue in software testing.

The following goals have been set for this research; the first goal is to propose an improved MVO optimization method by averaging the best places in the search space, which is informed by the past motion of the particles. The second goal is to use the superior movement approach to increase the efficacy of swarm movement in path testing and test data collection. The third goal is to use the created metaheuristic to address the MVO premature, to converge problem and the local optima entrapment problem. The fourth goal is to compare the proposed enhanced method to existing optimization algorithms through empirical testing using standard benchmark functions and testing software.

In this work, the Testing Multi-Verse Optimizer (TMVO) is presented as an improved Multi-verse Optimizer. Instead than focusing on a single place, TMVO considers the swarm's mobility and the mean of the two best options in the universe. Using a mean-based algorithm model that has been suggested, particles will migrate toward the ideal solution. The recommended movement equations of TMVO ensure the effectiveness of space exploration and utilization. In addition to resolving the issue of poor convergence, it also escapes the local minimum.

This study makes a contribution through enhancing MVO in solving the problem of path testing by enhancing the test data generation. It also provides a comprehensive analysis of the algorithm's movement strategy, equations, pseudo-code, and parameters. When it comes to solving software testing issues, the algorithm offers a more effective path testing method for getting to best tested path. TMVO has been evaluated and validated in comparison to a number of well-established functions. In addition to this, it provides a solution for a problem involving a single optimization problem in software testing.

The remaining part of this study is organized as follows. The related works are reviewed in more detail in Section II. The methodology including different types of software testing, and the path coverage test is described in Section III. In Section IV, the experimental results and discussion are presented where Section V concludes this study.

II. Related Work

The Multi-Verse optimizer, often known as the MVO, was first suggested to be developed by Mirjalili and colleagues

[12]. They came up with an original algorithm that was inspired by nature and gave it the name Multi-Verse Optimizer (MVO). The white hole, the black hole, and the wormhole are the three natural phenomena that serve as the inspiration for this algorithm's design. The demand for these models arises from the requirement to independently carry out exploration, exploitation, and vicinity search. Biswas [13] presented an ant colony optimization (ACO)-based method that produces groups of ideal pathways and ranks them in order of preference. In addition, using these methodologies leads in the grouping of test data inside the area so that similarity may be used as input for the paths that are constructed. The proposed methods ensure comprehensive software coverage with little duplication of effort. In [14], the authors employed an approach dubbed "propagation error" to analyze the growth of defects. Through the development of test cases, we are able to activate seed faults and provoke associated potential issues. The testing procedure involves triggering and correlating these flaws. Clever algorithms are used in this method, with the aim of permanently designing test cases to disperse data about seed flaws. All faults and related defects that were before invisible are now easily discernible thanks to propagation routes.

Aspect oriented programming (AOP) is recommended by Jain et al. [15], [16] as a method for crawling into program modules without modifying their source code and component in order to investigate regions where faults are suspected to exist. AOP execution places an emphasis on making use of system cut points. In addition to this, it includes crucial code at each execution point for the purpose of testing. To improve the effectiveness of conventional random testing and random partition testing approaches, some researches suggested using Dynamic Random Testing, also known as DRT. The DRT is presented as a potential further improvement to the testing's viability. In order to decide on those upgrades for a testing profile that is more reasonable, it is necessary to have access to additional historical testing data along with an estimation of the rate at which defects are identified for each subdomain in real time, for example. This exemplifies one instance of the symbolization that the Java-based DSU system provides. In this approach, system tests that were developed for both older and newer versions of the program can be updated, and it purposefully tests whether or not an incremental upgrade can result in a failed test.

Testing software is widely regarded as an effective strategy for ensuring the quality of software in both the academic and commercial settings. The quality of the test data has an effect on the testing process and is also an essential component in determining how well software is tested. As a natural part of the software development life cycle, software testing may be carried out either automatically or manually as a matter of course. Both approaches have their advantages and disadvantages. The creation of test data is the initial step in the software testing process. In the testing process, there are a few various procedures that need to be carried out. These procedures include the development of test data, the prioritizing of test cases, and the reduction of test cases. The initialization of the test data is the method that is the most difficult aspect of testing in these methods. According to [17], there would be a variety of sub-tasks amongst test cases, test appropriateness, and test data [18].

Test cases are the conditions that are going to be set, and the analyzer is going to use those to determine whether or not the specified function fits in suitably. The gathering of test cases will ensure that the test is suitable. Test data are a special sort of data that is used for evaluating different software applications. They can be easily recognized from other types of data. In addition, it will serve as the feed for the system's input. It is possible that this will serve as the principal test for the data or the field validations for any software applications. Creating test data for very simple programs is not a tough undertaking. On the other hand, producing the data for extensive initiatives might be challenging [19]. There is a wide variety of software available that can be used to generate test data [20], including intelligent test data generators, test data generators that use path oriented principles, and test data generators that use goal oriented. Creating test data would involve the use of several methodologies, such as UML diagrams; nevertheless, the development of test data would be dependent on graphical user interfaces. The coverage-based testing methodology, which consists of a collection of conditions that absolutely need to fulfill all of the prerequisites, could be used to generate test data [21]. A wide variety of coverage strategies, including branch coverage, function coverage, and statement coverage, are all viable options.

However, there is no assurance that the flaws in the test data will be uncovered by every converge method. The offered strategies leverage objective function for test data creation. The test data that are generated as a result of the objective function give the best possible possibility for defect detection. The space and path disparity functions are the goal functions. In order to get the space disparity, we need to first measure the distance that separates each of the test suites. Next, we need to calculate the path disparity by working backwards from the branch condition through the control flow graph [22], [23]. Because product testing must take into account both the long term and the cost-benefit analysis, extensive testing may not be carried out. Since a wide variety of methods and resources are used to automate the processes [24], it's possible that the use of such mechanizations for testing has become essential as of late. Successful testing requires the identification of code routes, the creation of a test data suit for those paths, a testing procedure on the Software Under Test (SUT) using the data, an evaluation of the results, and the production of quality models.

Successful testing would examine as many test cases as possible that are similar to those already performed. As an added cost-cutting measure, it is important to prioritize paths with the expectation that the majority of errors will be found in the preliminary phases of the process, and to identify appropriate paths and test data from among the many possible options. Path testing is a very useful technique for finding bugs in software components [25], [26].

III. METHODOLOGY

In this section, we describe the procedures and techniques employed to carry out the study, including data analysis, and statistical methods. The research design and settings are also discussed in detail. This section provides a detailed account of

the methods used to answer the research questions and provides a clear understanding of how the research was conducted.

A. Types of Testing

It is of vital importance to clarify here the main types of testing since testing is used in this study to test the research hypothesis. The testing of software can be divided into two categories: static testing and dynamic testing.

In static software testing, the reviewer completes code reviews by walking through hypothetical inputs to the SUT while outwardly accompanying the real program flow. Static testing is a type of software testing. This method requires the reviewer to invest their time, and the reviewer themselves need to be an expert as well as possess the necessary skills to evaluate the code. It is possible to specify from these variables the paths that might not be executable. This is made possible by the enhancements to static testing that let the code be symbolically evaluated. This is done by gathering distinct paths and variables regarding code execution. This methodology could be used to aggregate these variables in order to provide a demand solver with the information it needs to decide which routes and paths were previously infeasible.

When performing dynamic testing, the SUT code may actually be executed using the test inputs that have been provided. The observed behaviors of the SUT are compared to its typical behaviors, and the test is either successful or unsuccessful depending on whether the observed behaviors match the technique that is relied upon to conduct the test. There are two different kinds of testing that may be done on dynamic systems: black-box testing and white box testing. The outcome of an output defect is what is understood to be a software defect [27].

In black-box testing, the system is evaluated without the tester having any prior knowledge of the system's underlying architecture. In black-box testing, the individual performing the testing does not have access to the program's source code. He or she needs knowledge regarding the modeling of the framework. In this section, the tester generally connects with the software through the user interface by providing inputs and testing outputs. However, the tester is not expected to have any prior knowledge regarding how to operate input. The accuracy of software objectives is checked for throughout the black-box testing process. These objectives can be tested using the inputs and outputs domain. This demonstrates that the program in question has both an input and an output; results from output failures are regarded to be software flaws [28].

Testing with a black-box can be used to identify problems with data structures, error functions, and interfaces. Black-box eliminates system techniques. It detects errors that are caused by faults in the software in order to find out what the problem is with the output. It is possible to use it to identify incorrect functions, which produced undesirable output at executed, inaccurate conditions. This is due to the fact that incorrect functions generate inaccurate outputs anytime they are put into action.

Testing procedures that provide information regarding the internal specification and design of the system are referred to as white-box testing. It is not unusual for this to be referred to as structural testing. It includes testing for anything to do with program logic, including testing for loops, testing conditions, and testing based on data flow. Even if there is only an incomplete software definition, this will assist in the discovery of flaws. The goal of white box testing is to ensure that each possible path in software has been explored by the test cases.

White box testers have access to the system's source code and are therefore familiar with its architecture. The tester begins by analyzing the source code, then uses the knowledge from the source code to generate a variety of test cases, and finally, particular code routes are utilized in order to achieve a desired amount of code coverage [29]. It is guaranteed by the test cases that each of the program's independent pathways has been followed at least once. Each internal data structure would be tested to ensure the system's dependability. Each loop is run until it reaches its boundaries while staying within its operational constraints. White-box testing is a technique that can be utilized by software engineers in the process of designing test cases. This technique involves practicing distinct paths within a module, practicing legitimate true and false decisions, executing loops at their limits and inside their operational limits, and practicing inner data structures to guarantee that they are correct. It would appear that test cases need to be modified whenever implementation is altered. In this article, we have simply utilized the black-box testing approach to evaluate the functionality of two separate lines based on different test cases utilizing BVA and Robustness testing. White box testing, on the other hand, covers testing the majority of the program's code. Changing the requirements under test conditions will help identify typographical problems [30].

B. Path Coverage Test

The testing technique known as "coverage basic path testing" refers to testing strategies that are designed to cover the fundamental path of the software. The test target is the fundamental flow of the program when it is executed using this method. After gathering test information for the program input space, taking those test data into consideration as input, and then eventually running the program, it carries out the fundamental path by running the program and executing it. The participation of the fundamental routes group is required in order to carry out the genuine testing technique. The following is a list of features that are shared by all fundamental paths: 1) Each and every path in the program is completely autonomous; 2) Each and every edge in the program is accessible; and 3) Any paths in the program that do not have a position with the path set can potentially be achieved through the use of paths linear operation in the fundamental path set. The fault propagation path is a way that will show the advancement of defects where mistakes originate in software nodes; they may gradually propagate on different nodes. This method will be referred to as the fault propagation path. During the procedure that is used to repair errors that have already been created, past errors will be used to determine which paths have the greatest potential for error propagation. This will help correct errors that have already been made. Inaccurate historical data will be used as a source of this knowledge, and it will be used to define these routes.

The MVO algorithm uses the expansion rate as the determining factor for the value of the function for each and every search. In addition, every particle in the search zone has a similar appearance to an elected solution as well as a variable in an elected solution. Greater expansion rates result in greater and lower possibilities of the existence of those hypothesized white holes and black holes, respectively. These higher expansion rates also bring search agents or universes with higher rates to transfer items through those white gaps. White holes are recommended as a result of reduced inflation rates, which also reduce the expansion rates that should be used to transport items into black holes. As a result, the probability of black holes is increased, and white holes are offered as a result. Wormholes, disregard the flatland rates; they would be the explanation for the arbitrary sending of the object to the best universes. The MVO algorithm contains a wheel choice component that can be used for scientific demonstrations of white holes and black holes, as well as the return of objects to the search area. The search agents are arranged in each iteration according to their expansion rates, and once a search agent is chosen, it must be assigned a white hole. These various characteristics of the universes are supported by MVO. It makes use of wormholes in order to transport irregular things through the search region, and it does so by exploiting those wormholes. These wormholes randomly switch the positions of those objects in the search region, preventing them from claiming their expansion rates in any scenario. Wormhole connections have to be helped along between our reality and the finest possible universe.

C. The proposed Multi-Verse Optimizer (TMVO)

This sub-section introduces the proposed TMVO, including the algorithm steps, pseudo-code, the strategy, TMVO's operations, and its parameters, and theoretical conclusion.

TMVO is a stochastic swarm optimization algorithm with a revolutionary exploration and exploitation movement approach for locating optimum solutions to optimization problems. TMVO is based on enhancing MVO movement strategy by taking the top three solutions in the swarm for the automatic development of test cases for structural data testing, particularly path testing. Since the original MVO algorithm lacked the ability to effectively cover both the exploration and exploitation stages of the search process, the TMVO algorithm was developed to solve this problem. In addition, TMVO addresses the premature convergence issue that arises with certain implementations of the MVO algorithm. TMVO algorithm advises focusing exploration and exploitation efforts on the following points: White holes would be a higher amount of time on make in the universes for secondary expansion rates, which they transmit items on distant universes. This is because white holes consume an inordinate quantity of matter and energy. In addition to this, assist them in improving their rates of expansion. Black holes would appear in universes with low expansion rates, and as a result, they provide a higher probability of items being accepted from other universes. This is because low expansion rates result in more compact universes. This adds another layer to the possibility of claiming an increasing inflation rate for universes that have a lower expansion rate. White and black hole tunnels have a tendency to transport from worlds the objects with rising expansion rates to the folks with low expansion rates; in this method, the general inflation rate concerning known universes will be moved forward across the span from those repetitions. Wormholes have a propensity to appear in any universe at random, regardless of the expansion pace, or something along those lines due to the many properties of. Through all of the repetitions, the universe remains preserved. If there is a sudden shift, white/black hole tunnels need universes, which will lead to an inquiry of the search space. Unanticipated progressions are also helpful in determining the ideal local solidity. Random wormholes re-expansion of the variables from variables of the universes around the finest result gained in this way in those course about iterations, thus ensuring that exploitation is performed around those the overwhelming majority guaranteeing area of the search region. WEP Adaptive values expansion will concentrate exploitation by using an optimization procedure. This is because the occurrence of wormholes in universes is a likelihood. TDR Adaptive values reduce the journey variable distance near the best universe. This is a method that expands the precision of a local search through iterations. The joining of those indicated by the algorithm is ensured by checking the exploitation of local search comparative of the amount derived from the number of iterations.

The following are the main steps involved in TMVO:

The first step, which is named initialization, involves initially populating the algorithm's parameters with random values. Second, the suggested motion equations will be used to iteratively improve upon these initial best guess answers.

The second step: the algorithm's designated equations are utilized to progressively enhance the outcomes until a stopping criterion is met.

The third step: The algorithm's optimal solution is determined by balancing the values of the goal function and comparing the resultant comparisons.

The pseudo-code for the TMVO algorithm is displayed in Fig. 1.

1. Define the set of all universes, U.
2. Define the set of all portfolio weights, w.
3. Define the set of all groups, G.
4. Initialize the current universe, u, and the current portfolio weights, w.
5. Evaluate the performance of the current portfolio, P, in the current universe, u.
6. For each group, g, in G:
   a. Select a subset of universes, U', from U that belong to group g.
   b. For each universe, u', in U':
      i. Calculate the portfolio weights, w', that maximize the expected return in universe u'.
      ii. Evaluate the performance of the portfolio, P', in universe u' using weights w'.
   c. Select the universe, u*, and the corresponding portfolio weights, w*, that result in the highest performance in the subset of universes.
7. Select the group, g*, and the corresponding universe, u**, and portfolio weights, w**, that result in the highest overall performance.
8. Update the current universe and portfolio weights to u** and w**, respectively.
9. Go to step 5 and repeat the process.

Fig. 1. TMVO pseudocode.

An exploration phase and an exploitation phase are separated by a population-based method, as we saw in the previous section. For MVO space exploration, it has employ white hole and black hole ideas. On the other hand, the wormholes help MVO make better use of the search spaces. We treat every possible answer as if it were its own world, with each variable representing a different type of thing that may be found in that universe. The value of the fitness function is used to determine the inflation rate that is applied to each solution. As time is a standard concept in both cosmology and multi-verse theory, we employ it throughout this study rather than iteration.

However as in MVO, the TMVO universes are optimized using the following criteria: If inflation rates are high enough, white holes are almost guaranteed to form. Black holes are less likely to form with greater inflation rates. Third, items in universes with a higher inflation rate are more likely to be sucked into white holes.

The number of items that enter the universe via black holes is larger in universes with a lower inflation rate. No matter the pace of inflation, things in all worlds may eventually make their way through wormholes to the best universe at random.

TMVO Pseudocode show that the normalized inflation rate serves as a roulette wheel for selecting and determining white holes. The likelihood of sending things through white hole or black hole tunnels increases as the inflation rate decreases. When solving the maximizing problems, -NI must be replaced with NI. Since the universes must swap things and experience sudden changes in order to traverse the search space, the exploration can be ensured using this approach.

The previously mentioned method allows for unabated object exchange across worlds. We assume that each universe is equipped with wormholes that allow its things to randomly travel through space, allowing for the preservation of cosmological variety while also allowing for the possibility of exploitation. Wormholes are capable of altering the objects of universes at random, regardless of their inflation rates. We assume that wormhole tunnels are constantly built between a universe and the best universe generated so far in order to supply local modifications for each universe and have a high likelihood of enhancing the inflation rate utilizing wormholes.

The suggested methods have varying degrees of computing complexity, which are determined by the number of iterations, the number of universes, the roulette wheel mechanism, and the universe sorting mechanism. Every iteration includes the process of sorting the universe, and we use the Quicksort algorithm, which, in the best case scenario, has a complexity of O(n log n), and in the worst case scenario, has a complexity of O(n2). The selection from the roulette wheel is carried out for each variable in each universe throughout the iterations, and its complexity ranges from O(n) to O(log n), depending on the implementation.

The followings are some observations that are concluded in order to gain an understanding of how the suggested algorithm could, in theory, have the ability to solve optimization problems:

- White holes are more likely to form in universes that have high inflation rates since this increases the likelihood that they will be able to transmit things to other universes and help those universes increase their inflation rates.
- Black holes are more likely to emerge in worlds with low inflation rates because these holes have a larger likelihood of receiving things from other universes since inflation rates are lower. This once more raises the possibility of increasing inflation rates for those universes that now have low inflation rates.
- The general or average inflation rate of all universes steadily improves over the course of the iteration process, as white and black hole tunnels tend to carry objects from universes with high inflation rates to those with low inflation rates.
- Because wormholes tend to form at random in any world, independent of the inflation rate, the variety of universes may be kept intact throughout the course of several iterations of the simulation.
- Wormhole and black hole tunnels need the sudden transformation of universes, which ensures the thorough investigation of the search space.
- Sudden shifts are helpful in resolving local optimalities that have stagnated.
- During the iterative process, wormholes randomly reposition some of the variables in the universes surrounding the best solution gained thus far. This facilitates exploitation all over the most promising area of the search space.
- The existence probability of wormholes in universes is gradually increased when adaptive WEP settings are used. As a result, the process of optimization places a strong emphasis on exploitation.
- To enhance the precision of local search during iterations, adaptive TDR values are used to decrease the variable's traveling distance around the best universe.
- By placing a greater emphasis on exploitation and local search in relation to the number of iterations, the suggested algorithm's convergence is ensured.

IV. Experimental Results and Analysis

A. Evaluation of TMVO over the Benchmark Functions

To test the performance of TMVO, experiments had been run over well-known benchmark functions that represent unimodal and multi-modal functions that have been used by many researchers [31][32][33].

The cost functions of the benchmark unimodal function (F1-F7) are displayed in Table I, and those for the multimodal functions (F8-F14) are displayed in Table II. In order to get reliable statistical findings, the experiment needs to be carried out n times before any meaningful conclusions can be drawn about the performance of meta-heuristic algorithms. Each run needs to be carried out until m numbers of iterations have been

completed, and this is for the purpose of verifying if the algorithm is stable. In most cases, the statistical and output metrics, such as the average, the standard deviation, as well as the minimum and maximum values, of the best solution in the most recent iteration are measured and registered for comparison studies of the algorithms. For the purposes of acquiring, recording, and verifying the outcomes of the TMVO algorithm, the exact same process and experimental approach have been adhered to throughout. In addition to computing the error, it is important to determine how much the findings deviate from the ideal value.

TABLE I. UNIMODAL FUNCTIONS MATHEMATICAL FORMULATION (F1-F7)

No.	Formula
F1	$f1(x) = \sum_{i=1}^{n} x_i^2$
F2	$f2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$
F3	$f3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_j \right)^2$
F4	$f4(x) = \max i\{\lvert x_i \rvert, 1 \le i \le n\}$
F5	$f5(x) = \sum_{i=1}^{n=1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
F6	$f6(x) = \sum_{i=1}^{n} (x_i + 0.5)^2$
F7	$f7(x) = \sum_{i=1}^{n} ix_i^4 + random(0,1)$

TABLE II. MULTIMODAL BASIC FUNCTIONS (F8-F14)

No.	Formula
F8	$f8(x) = \sum_{i=1}^{n} -x_i \sin\sqrt{\lvert x_i \rvert} * \sum_{i=1} ix_i^4 * random(0,1) *$
F9	$f9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$
F10	$f10(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{x=i}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$
F11	$f11(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
12	$f12(x) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n=1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10,100,4)$ $y_i = 1 + \frac{x_i + 1}{4}\, u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < a \end{cases}$
F13	$f13(x) = 0.1\left\{\sin^2 + \sum_{i=1}^{n}(x_i - 1)^2[1 + \sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\right\} + \sum_{i=1}^{n} u(x_i, 5,100,4)$
F14	$f14(x) = -\sum_{i=1}^{n}\sin(x_i).\left(\sin\left(\frac{ix_i^2}{\pi}\right)\right)^{2m}$, m=10

The average, on the other hand, compare the overall performance of the method. All of the statistical analyses that were carried out allow us to establish beyond a reasonable doubt that the results were not the product of random chance. In each of the algorithms, the population size was set at fifty, and the maximum number of iterations was set at one thousand. It is important to keep in mind, however, that the maximum number of iterations and the number of particles (possible solutions, for example) should be determined by experimentation when dealing with situations that occur in real life.

It is necessary to conduct tests a total of n times if one wishes to achieve reliable statistical findings from meta-heuristic algorithms. In addition, for the purpose of validating the consistency of the method, each iteration must be carried out until the m^{th} time. In order to create TMVO, report on its performance, and then validate its efficacy, the identical experimental process was carried out.

The effectiveness of the TMVO algorithm that was proposed has been assessed. It has been proved that there is a set of statistical measurements that includes the average, the standard deviation, the minimum, the maximum, and the error measurement. These measurements have been determined through the process of experimentation throughout the course of the twenty-three benchmark functions shown in the Tables (1-2).

The primary regulating parameters of these algorithms, the number of search particles and the maximum iteration, have been set to the values of 50 and 1000 respectively so that a fair comparison can be made between them. To achieve the highest possible level of performance, the settings for the various governing parameters of each algorithm are taken from the most recent version of the source code. Each of the algorithms is executed fifty times on each of the test functions, and the outcomes of these simulations are presented later in this study. It is important to note that the results of the algorithms are standardized in the range [0, 1] by employing the min-max normalization so that their performances may be compared across a variety of test functions.

We have evaluated the performance of TMVO on a set of well-known benchmark functions utilized by many researchers to measure the performance of optimization algorithms.

The benchmark sets for multimodal hybrid functions are categorized from function 15 to function 23 and the mathematical formulations for hybrid composition functions are shown in Table III.

The Lower Bound (LB), Upper Bound (UB), dimension (Dim), and F_{min} of the benchmark-evaluated functions are displayed in Table IV.

TABLE III. MULTIMODAL HYBRID FUNCTIONS (F8-F14)

Multimodal Functions Formula (F15- F23)
$f15(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 x_4}\right]$

$f16(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
$F17(X) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos X_1 + 10$
$f18(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]\, x\, [30 + (2x_1 - 3x_2)^2 x\, (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2]$
$f19(x) = -\sum_{i=1}^{4} C_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$
$f20(x) = -\sum_{i=1}^{4} C_i \exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$
$f21(x) = -\sum_{i=1}^{5} [(X - a_i)(X - a_i)^T + C_i]^{-1}$
$f22(x) = -\sum_{i=1}^{7} [(X - a_i)(X - a_i)^T + C_i]^{-1}$
$f23(x) = -\sum_{i=1}^{10} [(X - a_i)(X - a_i)^T + C_i]^{-1}$

TABLE IV. PARAMETERS OF THE EVALUATED FUNCTIONS F1-F23

Function	Dim	LB	UB	F_{min}
F1	30	-100.00	100.00	Zero
F2	30	-10.00	10.00	Zero
F3	30	-100.00	100.00	Zero
F4	30	-100.00	100.00	Zero
F5	30	-30.00	30.00	Zero
F6	30	-2400.00	2400.00	Zero
F7	30	-1.28	1.28	Zero
F8	30	-500.00	100.00	418.9829x5
F9	30	-5.12	5.12	Zero
F10	30	-32.00	32.00	Zero
F11	30	-600.00	600.00	Zero
F12	30	-2400.00	2400.00	Zero
F13	30	-2400.00	2400.00	Zero
F14	2	-5.00	5.00	1
F15	4	-5.00	5.00	0.00030
F16	2	-5.00	5.00	1.0316
F17	2	[-5,0]	[10,15]	0.398
F18	2	-2.00	2.00	3.00
F19	3	0.00	1.00	-3.86
F20	6	-5.00	5.00	-3.32
F21	4	0.00	10.00	-10.1532
F22	4	0.00	10.00	-10.4028
F23	4	0.00	10.00	-10.5363

TABLE V. COMPARISON BETWEEN THE TMVO AND MVO IN TERMS OF MEAN FITNESS VALUE

F#	Mean of TMVO	Mean of MVO	TMVO vs MVO
F 1	1.89618	20.8066	Better
F 2	1.482	5.2544	Better
F 3	8.9748	234.41	Better
F 4	1.4287	3.9419	Better
F 5	67.3715	843.6999	Better
F 6	6.5543	22.9003	Better
F 7	0.070841	0.86756	Better
F 8	-4864.7373	-7388.2747	No
F 9	25.6411	163.2599	Better
F 10	2.8175	4.5803	Better
F 11	0.91005	1.6356	Better
F 12	2.5573	2.4075	No
F 13	0.077985	0. 2351	Better
F 14	0.998	0.998	Equal
F 15	0.00044931	0.00058644	Better
F 16	-1.0316	-1.0316	Equal
F 17	0.3978	0.3978	Equal
F 18	3	3	Equal
F 19	-3.8628	-3.8627	Better
F20	-3.3214	-3.201	Better
F21	-10.0464	-2.6068	Better
F22	-10.3418	-9.8605	Better
F23	-5.1928	-5.1885	Better

The comparison between the proposed TMVO algorithm and the MVO algorithm in terms of mean fitness value is tabulated in Table V. Comparing TMVO algorithm with MVO over the tested functions F1-F23 showed that TMVO has very competitive results. In the unimodal functions (F1-F7) the TMVO has shown better results and outperformed MVO over all the seven functions. Regarding the multi-modal functions (F8-F12), TMVO was also achieved better mean fitness values than MVO except F8. Moreover, the proposed algorithm is competitive over the expanded multi-modal functions (F13, F14). The results have shown that when testing the algorithm TMVO over the multi-modal hybrid functions (F15-F23), the TMVO outperformed MVO in most cases and achieved the same fitness value in three cases.

The proposed TMVO achieved better fitness values in most cases due to the fact that TMVO offers additional exploration points inside the search space. TMVO takes the two best possible solutions and utilizes them to find a new solution at each iteration. It drives closer and closer to the global optimum by updating the current particle location to the position that is optimal between these two points.

B. Evaluation of TMVO in Test Data Generation for Path Testing

The experimental results testing is carried out on five benchmark programs, which are presented in Table VI. The fitness value is a numerical number that represents individual

quality in comparison with the existing local solution in order to seek for the optimum local solution that has the least amount of fitness value possible. The option that results in the lowest overall fitness value will be the one that we consider to be the most viable solution. The fitness value is computed by applying Korel's route distance relation to each variable. The fitness path distance is calculated by adding up each variable's fitness value at each point along the path. To start, a series of random test instances are generated so that the process can begin. Utilizing points that were picked at random allows for the improvement of the existing solution. Perform a calculation to determine the fitness value of each potential solution. Each swarm is assigned a fitness value, and then each swarm searches for the local minimum value within the search zone to see whether a higher value can be found.. If we can, the new value is saved, and the old value is replaced with it. Arrange candidate solutions in order of increasing fitness, beginning with the best. The onlooker phase begins with the most optimal solution to fitness. If the termination requirements are deemed to be complete, an onlooker local search will be issued; otherwise, it will be used to improve candidate solution fitness. In case that the phase is completed without satisfying the finishing conditions, the phase to replace sources that have reached the maximum number of tries will be initiated.

We utilized the five variables in Program1, which are (x, y, z, j, k). First, if j - 80 >= 0, the distance at variable j will be zero; if variable k - 70 >= 0, the distance at variable k will be zero; if variable x - 60 >= 0, the distance at variable y - 50 = 0; this is the Korel branch distance relation. Specific details and results including the fitness value are tabulated in Table VII. It is of vital importance to correctly interpret the values in Table VII. The letters A, B, C, D, and E represent the Korel's route branch distance of the variables j, k, c, y, and z respectively.

TABLE VI. BENCHMARK PROGRAMS USED AS CASE STUDIES

Program1	Program2	Program3

If (j >=80)
{….. }
Else if (k >= 70)
{….. }
Else If (x >=60)
{….. }
Else If (y>=50)
{….. }
Else If (z>=25)
{….. }
	while (j >=75)
	{….. }
	while (k >= 65)
	{….. }
	while (x >=55)
	{….. }
	while (y>=45)
	{….. }
	while (z>=35)
	{….. }
		If (j >=60)
		{….. }
		Else If (k >= 80)
		{….. }
		Else If (x >=55)
		{….. }
		if (y>=25)
		{….. }
		while (z>=45)
		{….. }

Program4	**Program5**	

If (j >=57)
{….. }
If (k >= 68)
{….. }
if (x >=34)
{….. }
	If (j >=45)
	{….. }
	Else If (k >= 30)
	{….. }
	while (x >=40)
	{….. }
	Else If (y>=35)
	{….. }	

TABLE VII. KOREL'S ROUTE BRANCH DISTANCES OF THE VARIABLES J, K, C, Y, AND Z ALONG WITH THE FITNESS VALUES (PROGRAM 1)

#	j	k	x	y	z	A	B	C	D	E	Fit.
1	91	50	75	100	54	11	0	15	50	29	105
2	89	64	84	68	66	9	0	24	18	41	92
3	81	87	71	82	87	1	17	11	32	62	123
4	62	72	89	52	99	0	2	29	2	74	107
5	84	56	72	86	79	4	0	12	36	54	106
6	70	91	84	92	59	0	21	24	42	34	121
7	77	67	71	72	88	0	0	11	22	63	96
8	76	97	61	84	65	0	27	1	34	40	102
9	53	65	72	79	85	0	0	12	29	60	101
10	90	56	80	80	70	10	0	20	30	45	105
11	96	66	62	53	75	16	0	2	3	50	71
12	99	63	61	88	85	19	0	1	38	60	118
13	55	87	90	55	85	0	17	30	5	60	112
14	78	95	60	72	93	0	25	0	22	68	115
15	68	98	63	93	93	0	28	3	43	68	142
16	76	97	85	69	51	0	27	25	19	26	97
17	99	57	79	84	68	19	0	19	34	43	115
18	59	92	85	75	84	0	22	25	25	59	131
19	100	93	100	59	82	20	23	40	9	57	149
20	59	67	72	94	76	0	0	12	44	51	107
21	67	87	62	58	59	0	17	2	8	34	61
22	100	80	76	90	69	20	10	16	40	44	130
23	66	78	95	58	82	0	8	35	8	57	108
24	50	54	54	66	86	0	0	0	16	61	77
25	63	78	89	98	51	0	8	29	48	26	111

The following equation is utilized to determine the fitness value that is used for the path of Program1. This value, which is the sum of the distances that were indicated before, is computed as follows:

$$F = (J - 80) + (K - 70) + (X - 60) + (Y - 50) + (Z - 25) \quad (1)$$

In Program2, we utilized the five variables (x, y, z, j, k). If the first variable (j) has a distance of zero, then the second variable (k) also has a distance of zero. (if (k) - 65>= 0), the third variable (x) has a distance of zero. if (x) - 55 >= 0, the fourth variable (y) has a distance of zero if (y) - 45 = 0, and the fifth variable (z) has a distance of zero if (z) - 35. The Korel's route branch distances of the variables in Program2 and the fitness values are displayed in Table VIII.

Eq. (2) has been used to get the fitness value that should be utilized for the path of program 2, which is 54. This value represents the sum of the distances that were indicated earlier.

$$F = (J - 75) + (K - 65) + (X - 55) + (Y - 45) + (Z - 35) \quad (2)$$

TABLE VIII. KOREL'S ROUTE BRANCH DISTANCES OF THE VARIABLES J, K, X, Y, AND Z ALONG WITH THE FITNESS VALUES (PROGRAM 2)

#	j	k	x	y	z	A	B	C	D	E	Fit.
1	86	84	52	94	78	11	19	0	49	43	122
2	92	57	59	86	100	17	0	4	41	65	127
3	74	54	62	63	83	0	0	7	18	48	73
4	86	57	94	75	71	11	0	39	30	36	116
5	66	96	83	52	81	0	31	28	7	46	112
6	70	98	69	58	95	0	33	14	13	60	120
7	81	88	92	92	80	6	23	37	47	45	158
8	53	69	86	91	100	0	4	31	46	65	146
9	53	81	64	74	68	0	16	9	29	33	87
10	89	79	53	82	94	14	14	0	37	59	124
11	51	58	88	97	84	0	0	33	52	49	134
12	78	66	99	54	84	3	1	44	9	49	106
13	86	61	66	73	89	11	0	11	28	54	104
14	100	57	98	51	57	25	0	43	6	22	96
15	77	68	78	63	100	2	3	23	18	65	111
16	91	80	92	68	57	16	15	37	23	22	113
17	78	76	82	62	68	3	11	27	17	33	91
18	86	65	60	67	56	11	0	5	22	21	59
19	72	73	72	67	72	0	8	17	22	37	84
20	87	73	100	84	67	12	8	45	39	32	136
21	61	64	92	61	50	0	0	37	16	15	68
22	95	50	69	78	68	20	0	14	33	33	100
23	62	53	52	51	83	0	0	0	6	48	54
24	52	91	54	60	73	0	26	0	15	38	79
25	94	96	83	80	100	19	31	28	35	65	178

In Program3, the five variables (x,y,z,j,k) are also employed to evaluate the proposed algorithm. The Korel branch distance relation states that if the value of the first variable, j, is greater than or equal to 60, then the value of the second variable, k, is greater than or equal to 80. If the value of the third variable, x, is greater than or equal to 45, the value of the fourth variable, y, is less than or equal to 75, and the value of the fifth variable, z, is 45. Table IX tabulates the outcomes when applying the TMVO over Program3. The symbols (A, B, C, D, E) represent Korel's Route Branch Distances of the variables (j, k, x, y, z) respectively.

TABLE IX. KOREL'S ROUTE BRANCH DISTANCES OF THE VARIABLES J, K, X, Y, AND Z ALONG WITH THE FITNESS VALUES (PROGRAM 3)

#	j	k	x	y	z	A	B	C	D	E	Fit.
1	58	85	77	72	75	0	5	22	47	30	104
2	94	70	83	83	91	34	0	28	58	46	166
3	83	90	88	76	93	23	10	33	51	48	165
4	70	55	87	96	100	10	0	32	71	55	168
5	86	94	64	60	93	26	14	9	35	48	132
6	77	90	60	79	72	17	10	5	54	27	113
7	73	51	64	73	68	13	0	9	48	23	93
8	66	50	86	83	64	6	0	31	58	19	114
9	79	75	53	61	76	19	0	0	36	31	86
10	97	63	76	54	91	37	0	21	29	46	133
11	87	86	67	84	93	27	6	12	59	48	152
12	75	83	65	71	53	15	3	10	46	8	82
13	80	91	97	64	95	20	11	42	39	50	162
14	90	72	81	62	63	30	0	26	37	18	111
15	71	100	54	81	64	11	20	0	56	19	106
16	79	81	84	66	61	19	1	29	41	16	106
17	71	55	71	64	93	11	0	16	39	48	114
18	66	88	78	75	71	6	8	23	50	26	113
19	52	88	80	63	82	0	8	25	38	37	108
20	79	76	69	71	68	19	0	14	46	23	102
21	75	84	61	88	50	15	4	6	63	5	93
22	56	67	83	95	72	0	0	28	70	27	125
23	79	98	69	61	60	19	18	14	36	15	102
24	53	100	53	84	57	0	20	0	59	12	91
25	58	80	75	65	95	0	0	20	40	50	110

The fitness value that was used for the path of program3 was 82, which is the sum of the distances that were indicated earlier and is computed using Equation 3 as follows

$$F = (J - 60) + (K - 80) + (X - 55) + (Y - 45) + (Z - 25) \quad (3)$$

TABLE X. KOREL'S ROUTE BRANCH DISTANCES OF THE VARIABLES X, Y, AND Z ALONG WITH THE FITNESS VALUES (PROGRAM4)

#	x	y	z	C	D	E	Fit.
1	100	52	61	43	0	27	70
2	64	97	77	7	29	43	79
3	66	99	66	9	31	32	72
4	92	85	80	35	17	46	98
5	89	91	62	32	23	28	83
6	67	88	65	10	20	31	61
7	80	50	71	23	0	37	60
8	93	63	65	36	0	31	67
9	86	72	53	29	4	19	52
10	99	98	80	42	30	46	118
11	56	56	72	0	0	38	38
12	53	82	54	0	14	20	34
13	95	82	55	38	14	21	73
14	56	70	96	0	2	62	64
15	93	55	76	36	0	42	78
16	56	80	78	0	12	44	56
17	55	56	60	0	0	26	26
18	100	95	51	43	27	17	87
19	52	80	55	0	12	21	33
20	53	100	94	0	32	60	92
21	93	61	64	36	0	30	66
22	96	58	82	39	0	48	87
23	90	77	60	33	9	26	68
24	70	64	75	13	0	41	54
25	88	84	97	31	16	63	110

In Program4, we employed three variables (x, y, z). If j - 60 >= 0, k - 80 >= 0, and x - 45 >= 0, then the distance between the first and third variables is zero, as predicted by the Korel branch distance relation. Refer to Table X. The symbols (C, D, E) represent Korel's Route Branch Distances of the variables (x, y, z) respectively.

Using Equation 4, we can determine that the fitness value for path of program4 is 33, which is the total of the distances we determined before.

$$F = (J - 57) + (K - 68) + (X - 34) \quad (4)$$

Four variables were employed which are j, k, x, and y in Program5. In the Korel branch distance relation, if the value of the first variable, j, is zero, then the value of the second, k, is also zero, and so on. If the value of the third variable, x, is also zero, then the value of the fourth one, y, is also zero. Table XI tabulates 25 different cases along with their fitness values.

Path 5 of Program5 uses a fitness value of 39, which is the total of the distances discussed before. The fitness value is calculated according to Equation5.

$$F = (J - 45) + (K - 30) + (X - 40) + (Y - 35) \quad (5)$$

TABLE XI. KOREL'S ROUTE BRANCH DISTANCES OF THE VARIABLES J, K,X, AND Y ALONG WITH THE FITNESS VALUES (PROGRAM5)

#	j	k	x	y	A	B	C	D	Fit.
1	93	95	73	52	36	27	39	0	102
2	64	64	93	64	7	0	59	0	66
3	89	50	90	68	32	0	56	0	88
4	74	73	96	79	17	5	62	10	94
5	89	53	70	50	32	0	36	0	68
6	54	76	98	88	0	8	64	19	91
7	55	97	73	76	0	29	39	7	75
8	98	61	81	91	41	0	47	22	110
9	99	84	88	52	42	16	54	0	112
10	80	73	62	77	23	5	28	8	64
11	54	64	89	82	0	0	55	13	68
12	55	69	59	98	0	1	25	29	55
13	88	52	61	92	31	0	27	23	81
14	50	59	99	64	0	0	65	0	65
15	71	58	55	83	14	0	21	14	49
16	64	100	89	92	7	32	55	23	117
17	98	56	84	57	41	0	50	0	91
18	91	74	97	52	34	6	63	0	103
19	64	75	55	73	7	7	21	4	39
20	89	83	56	95	32	15	22	26	95
21	75	85	64	56	18	17	30	0	65
22	66	75	51	89	9	7	17	20	53
23	80	52	79	94	23	0	45	25	93
24	99	70	90	89	42	2	56	20	120
25	70	86	66	52	13	18	32	0	63

V. CONCLUSION

In this study, Testing Multi-Verse Optimizer (TMVO), an improved Multi-Verse Optimizer, is presented. However, rather than focusing on a single place, TMVO considers the swarm's mobility and the mean of the two best solutions in the universe. Using a recently suggested mean-based algorithm model, particles will progress toward the ideal solution. TMVO's recommended movement equations ensure efficient space exploration and utilization. In addition, it eliminates the problem of low convergence and escapes the local minimum. TMVO has been applied for the generation of test data for software structural testing, specifically route testing, that takes use of the Multi-Verse optimization algorithm. The proposed algorithm has been exhaustively tested through the creation of test data for the path coverage criteria and its subsequent application to a set of test programs. Additionally, five distinct programs and codes have been utilized in order to complete this evaluation. The results showed that the algorithm was successful in finding the best tested path for the test data, which led to an improvement in performance. The performance of TMVO is tested over several well-known functions. The results have shown that TMVO outperform original MVO algorithm over most of the tested functions.

However, this study presented two contributions. Firstly, an improved version of the Multi-verse Optimizer called Testing Multi-Verse Optimizer (TMVO) was proposed, which considered the movement of the swarm and the mean of the two best solutions in the universe. The particles moved towards the optimal solution by using a mean-based algorithm model, which guaranteed efficient exploration and exploitation. Secondly, TMVO was applied to develop test cases for structural data testing, specifically path testing, in an automated manner. Instead of automating the entire testing process, the focus was on centralizing automated procedures for collecting testing data. Automation for generating testing data was becoming increasingly popular due to the high cost of manual data generation. To evaluate the effectiveness of TMVO, it was tested on various well-known functions as well as five programs that presented unique challenges in testing. The test results indicated that TMVO outperformed the original MVO algorithm on the majority of the tested functions.

Despite the success of TMVO, there are still several areas where the algorithm can be further developed and tested. This includes algorithmic parameter tuning where most optimization algorithms have several tuning parameters that need to be set for optimal performance. Future research can explore automated parameter tuning techniques such as machine learning algorithms to improve the performance of TMVO. In addition to that, testing TMVO on large-scale problems where researchers can focus on testing TMVO on large-scale optimization problems and analyzing its scalability and efficiency.

REFERENCES

[1] Shukri, S. E., Al-Sayyed, R., Hudaib, A., & Mirjalili, S. (2021). Enhanced multi-verse optimizer for task scheduling in cloud computing environments. Expert Systems with Applications, 168, 114230.

[2] Jamunaa, D., Mahanti, G. K., & Hasoon, F. N. (2022). Multi-verse optimization algorithm for optimal synthesis of phase-only reconfigurable linear array of mutually coupled parallel half-wavelength

dipole antennas placed at finite distances from the ground plane. Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical, 29(4), 1915-1924.

[3] Hamad, F., Al-Aamr, R., Jabbar, S. A., & Fakhuri, H. (2021). Business intelligence in academic libraries in Jordan: Opportunities and challenges. IFLA journal, 47(1), 37-50.

[4] Yadav, M., & Mishra, A. (2023). An enhanced ordinal optimization with lower scheduling overhead based novel approach for task scheduling in cloud computing environment. Journal of Cloud Computing, 12(1), 1-14.

[5] Ryalat, M. H., Dorgham, O., Tedmori, S., Al-Rahamneh, Z., Al-Najdawi, N., & Mirjalili, S. (2022). Harris hawks optimization for COVID-19 diagnosis based on multi-threshold image segmentation. Neural Computing and Applications, 1-19.

[6] Song, R., Zeng, X., & Han, R. (2020). An improved multi-verse optimizer algorithm for multi-source allocation problem. International Journal of Innovative Computing, Information and Control, 16(6), 1845-1862.

[7] Aljarah, I., Mafarja, M., Heidari, A. A., Faris, H., & Mirjalili, S. (2020). Multi-verse optimizer: theory, literature review, and application in data clustering. Nature-inspired optimizers: theories, literature reviews and applications, 123-141.

[8] Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: a systematic literature review. Empirical Software Engineering, 27(2), 29.

[9] Schaeffer, R., Khona, M., & Fiete, I. (2022). No free lunch from deep learning in neuroscience: A case study through models of the entorhinal-hippocampal circuit. bioRxiv, 2022-08.

[10] Fakhouri, H. N., Hamad, F., & Alawamrah, A. (2022). Success history intelligent optimizer. The Journal of Supercomputing, 1-42.

[11] Aldabbas, H., Asad, M., Ryalat, M. H., Malik, K. R., & Qureshi, M. Z. A. (2019). Data augmentation to stabilize image caption generation models in deep learning. Int J Adv Comput Sci Appl, 10(10), 571-9.

[12] Mirjalili, S., Mirjalili, S. M., & Hatamlou, A. (2016). Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Computing and Applications, 27, 495-513.

[13] Biswas, S., Kaiser, M. S., & Mamun, S. A. (2015, May). Applying ant colony optimization in software testing to generate prioritized optimal path and test data. In 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT) (pp. 1-6). IEEE.

[14] Kun, W., & Yichen, W. (2016, January). Software test case generation based on the fault propagation path coverage. In 2016 Annual Reliability and Maintainability Symposium (RAMS) (pp. 1-4). IEEE.

[15] Jain, M., & Gopalani, D. (2016, February). Aspect oriented programming and types of software testing. In 2016 Second International Conference on Computational Intelligence & Communication Technology (CICT) (pp. 64-69). IEEE.

[16] Ryalat, M. H. (2022, January). A New Algorithm to Find The K th Smallest Element in an Unordered List (Efficient for Big Data). In 2022 2nd International Conference on Computing and Information Technology (ICCIT) (pp. 51-56). IEEE.

[17] Fakhouri, S. N., Hudaib, A., & Fakhouri, H. N. (2020). Enhanced optimizer algorithm and its application to software testing. Journal of Experimental & Theoretical Artificial Intelligence, 32(6), 885-907.

[18] Hamad, F., Fakhuri, H., & Abdel Jabbar, S. (2022). Big data opportunities and challenges for analytics strategies in Jordanian Academic Libraries. New Review of Academic Librarianship, 28(1), 37-60.

[19] Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., ... & Wagner, S. (2022). Software engineering for AI-based systems: a survey. ACM Transactions on Software Engineering and Methodology (TOSEM), 31(2), 1-59.

[20] Li, Z., Li, T., Wu, Y., Yang, L., Miao, H., & Wang, D. (2021). Software defect prediction based on hybrid swarm intelligence and deep learning. Computational Intelligence and Neuroscience, 2021.

[21] Dorgham, O., Naser, M. A., Ryalat, M. H., Hyari, A., Al-Najdawi, N., & Mirjalili, S. (2022). U-NetCTS: U-Net deep neural network for fully automatic segmentation of 3D CT DICOM volume. Smart Health, 26, 100304.

[22] Rosales Muñoz, A. A., Grisales-Noreña, L. F., Montano, J., Montoya, O. D., & Perea-Moreno, A. J. (2022). Application of the Multiverse Optimization Method to Solve the Optimal Power Flow Problem in Alternating Current Networks. Electronics, 11(8), 1287.

[23] Pandya, S., & Jariwala, H. R. (2022). Single-and multiobjective optimal power flow with stochastic wind and solar power plants using moth flame optimization algorithm. Smart Science, 10(2), 77-117.

[24] Abualigah, L. (2020). Multi-verse optimizer algorithm: a comprehensive survey of its results, variants, and applications. Neural Computing and Applications, 32(16), 12381-12401.

[25] Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., & Abraham, A. (2022). A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. Engineering Applications of Artificial Intelligence, 111, 104773.

[26] Hejderup, J., & Gousios, G. (2022). Can we trust tests to automate dependency updates? a case study of java projects. Journal of Systems and Software, 183, 111097.

[27] Zheng, W., Shen, T., Chen, X., & Deng, P. (2022). Interpretability application of the Just-in-Time software defect prediction model. Journal of Systems and Software, 188, 111245.

[28] Rath, S. K., Sahu, M., Das, S. P., & Pradhan, J. (2022). Survey on Machine Learning Techniques for Software Reliability Accuracy Prediction. In Meta Heuristic Techniques in Software Engineering and Its Applications: METASOFT 2022 (pp. 43-55). Cham: Springer International Publishing.

[29] Rubert, M., & Farias, K. (2022). On the effects of continuous delivery on code quality: A case study in industry. Computer Standards & Interfaces, 81, 103588.

[30] Battina, D. S. (2019). Artificial intelligence in software test automation: a systematic literature review. International Journal of Emerging Technologies and Innovative Research (www. jetir. org| UGC and issn Approved), ISSN, 2349-5162.

[31] Rahkar Farshi, T., & Orujpour, M. (2021). A multi-modal bacterial foraging optimization algorithm. Journal of Ambient Intelligence and Humanized Computing, 1-15.

[32] Ahmed, R., Mahadzir, S., & Mohammad Rozali, N. E. (2022, November). A Meta Model Based Particle Swarm Optimization for Enhanced Global Search. In International Conference on Artificial Intelligence for Smart Community: AISC 2020, 17–18 December, Universiti Teknologi Petronas, Malaysia (pp. 935-944). Singapore: Springer Nature Singapore.

[33] Marco, R., Ahmad, S. S. S., & Ahmad, S. (2022). Bayesian hyperparameter optimization and Ensemble Learning for Machine Learning Models on software effort estimation. International Journal of Advanced Computer Science and Applications, 13(3).