Dynamic TWGH: Client-Server Optimization for Scalable Combinatorial Test Suite Generation

Heba Mohammed Fadhil\textsuperscript{1,2*}, Mohammed Najm Abdullah\textsuperscript{2} and Mohammed Issam Younis\textsuperscript{2}

\textsuperscript{1}University of Technology, Baghdad, Iraq
\textsuperscript{2}University of Baghdad, Baghdad, Iraq

Abstract. To ensure that a software/hardware product is of sufficient quality and functionality, it is essential to conduct thorough testing and evaluations of the numerous individual software components that make up the application. Many different approaches exist for testing software, including combinatorial testing and covering arrays. Because of the difficulty of dealing with difficulties like a two-way combinatorial explosion, this brings yet another problem: time. Using client-server architectures, this research introduces a parallel implementation of the TWGH algorithm. Many studies have been conducted to demonstrate the efficiency of this technique. The findings of this experiment were used to determine the increase in speed and compare it to the results obtained from the various methodologies. TWGH was shown to be a demonstration of scalability in studies involving speedups. When the recommended method was implemented, the rate of acceleration increased by eight.

I. Introduction

Combining different software and hardware systems has been used to test their success. Many articles and surveys have been made about this subject, giving answers at both important strategy levels and day-to-day action steps \cite{1,2}. But, many types of software use combinations for things like web apps, testing safety and security in IT. They also help with sorting methods and deciding what's most important first. In the last ten years, many systems and real-life tests have shown surveys of research on combinatorial interaction study. There are two main concerns: test size and time \cite{3,4}. In the world of checking how good a system is, test cases are often pictured as math things called covering arrays (CAs) and mixed covering arrays (MCAs). These help us figure out if we've tested everything properly. We need good and useful CA/MCA to get the best testing that works well with other things. This is a very difficult task. Guessing techniques like heuristics and metaheuristics can help fix these problems \cite{10,11}. There are a few methods called metaheuristic that can make and improve covering arrays (CA). In CA, it's a good idea to use methods like hill climbing (HC) \cite{7}, harmony search algorithm (HSA) \cite{8}, particle swarm optimization (PSO) \cite{9}, tabu search (TS) \cite{10}, ant colony optimization or ACO \cite{11}, simulated annealing (SA) \cite{12}.

These Metaheuristic algorithms can be further divided into two categories based on their various features: algorithms based on the population and single-solution algorithms. It's important to find a balance in order get these goals. A mix model puts together at least two plans. It makes both of them better so they can work well together to find balance \cite{13}. The whale optimization algorithm (WOA), grey wolf optimization (GWO) and harmony search algorithm (HAS) have been mixed together to create a new method called the three-part whale–grey wolf–harmony technique. This is used for things like increasing effectiveness while dealing with different types of problems.

* Corresponding author: heba@kecbu.uobaghdad.edu.iq
An opportunity for improvement was found when studying the well-known covering-array approaches. Many currently available constructors yield large covering arrays, whereas others focus on speed and scalability. Some more exist, but they tend to be slow and lack good scalability when producing small covering arrays. However, in practice, the optimal trade-off between efficiency and efficacy generally depends on the specific testing circumstances. Covering the array size takes priority over construction time when testing the selected configurations requires less time than computing the array. However, there are scenarios in which the covering array size is more crucial than construction time. Therefore, it is important for a covering array constructor to be adjustable such that the time spent building and the size of the resulting array can be balanced. A hybrid environment composed of a distributed client/server method was proposed in this thesis. In addition, in this hybrid platform, where the HSA algorithm is stored on the server, the WOA and GWO are employed as indivisible entities in the client, and a mapping method must be developed to ensure the load balancing and scalability of the TWGH.

There will be a few more sections to follow: Section 2 talks about what other research has been done in this area. Section 3 simply explains what a covering array is, and Section 4 quickly tells about the special problem-solving algorithms used. Section 5 explains the suggested way. A full explanation of the plan and data checking can be seen in Section 6. Section 7 is about looking at statistics. In the end, Section 8 explains what was found and lays out where it is going next.

II. Related Work

Recently, more and more experts have started using a mix of computer methods to find solutions called hybrid metaheuristic algorithms [14], [15]. Mixing methods can help fix many real-world and school problems easily. Many good mix metaheuristic methods were suggested for using in combinatorial testing. Mercan et al. [16] presented a hybrid that integrates the best features of two construction approaches for calculating covering arrays: A smart way of searching and a rule-based method are used to quickly cover most needed combinations. For the remaining difficult ones, they use an advanced criteria approach called CHIP. Agrawal et al. [17] made a mix whale optimization method. This was used by the writers who needed bat search (BS) and ant colony optimization (ACO). They took programs from software stuff storage to test their algorithm with regression tests cases choice methods. The study of the results showed that using this new plan made our test set much smaller with less mistakes. Nasser et al. [18] suggested using a flower pollination method along with a hill climbing plan. They created an approach called FPA-HC which can make smaller test suites for two way software testing tools. FPA-HC was checked against other ways, such as the first FPA. In the tests, FPA-HC showed good results compared to old two-way methods. They can give very strong outcomes too. Li et al. [19] suggested a different way to make better cover arrays. They changed the improved PSO by adding in an artificial cooling process like simulated annealing procedure. Using 16 old random strength covering arrays in an experiment, this method seemed to work better than the other six ways.

III. Combinatorial Testing

A combinatorial interaction test, also called two-way or pairwise testing, uses combo interactions to find and sort mistakes. Comb tests seem to be random or follow rules. These are called combo interactions or combos tests because each one makes its own list. They don't need to connect with anything else further. T-way testing, another name for CIT, shows how to use t. In each test case the number of connection factors needed is called 'interaction strength.' For example, in a table with 100 choices to make, each ask has two options possible. This makes it hard to check every possible mix of these settings, because there are $2^{100}$ different combinations [20], [21]. To show two-way testing, let's use the made up example of smart city design in Figure 1.
Figure 1. Smart City Planning [22].

Smart city planning is built on five main things: transport systems, online services, smart traffic control, watching healthcare payment and seeing water levels. These are important parts of making cities smarter. In the smart city planning example in Figure 2, they show MCA covered array. They put it like this: covering number, rows needed and points per row which is MCA \((N, 3, 3^{2^4})\) here.

<table>
<thead>
<tr>
<th>Smart Security</th>
<th>Smart Energy</th>
<th>Smart Health</th>
<th>Smart Mobility</th>
<th>Smart People</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCTV</td>
<td>Electricity</td>
<td>Government Hospitals</td>
<td>Public Transport</td>
<td>Wired</td>
</tr>
<tr>
<td>Sensors</td>
<td>Solar</td>
<td>Private Hospitals</td>
<td>Individual Vehicle</td>
<td>Wireless</td>
</tr>
</tbody>
</table>

Fig. 2. The three-dimensional mixed model in the evolution of the smart city system.

The three-dimensional mixed model in Figure 2 illustrates the evolution of the smart city system. The underlying basis assumes tactile force strength, denoted as \(t = 3\). Including all conceivable configurations of the smart city system, the comprehensive test method would require 48 test cases, estimated as multiples of...
3 × 2 × 2 × 2 but using a combinatorial test method similar to the T-way technique in 3-part tests. The number of test cases required is significantly reduced to only 17, covering different configurations all of which we discussed in the previous example well.

IV. **Heuristic and Meta-heuristic Algorithms**

Two commonly used methods for solving problems about the best way to do things are heuristic and meta-heuristic algorithms. These algorithms were built to give the best answer in a fair time. They assure you of this result. Some perfection problems need an answer that increases a lot with time as the size of problem cases expands. We can often find nearly perfect solutions to very hard problems by using shortcuts called heuristic methods. These give us near-perfect answers fast enough for most situations. Heuristics, first made in the 1970s, are now used to tackle many hard problems about making things better [23].

Even though big problem-solving strategies worked well at the start of this century, new and always changing ways have been successful lately. Three of these big solutions, HSA, GWO and WOA are talked about more below. Recently, they have been studied a lot by many researchers and talked about often. All these tools change to fix problems about picking the best features, and each tool depends on how it's own type of group behaves.

V. **Client/Server Architecture**

The client-server architecture, also known as the client-server model, involves separation of responsibilities between clients and servers in the same system or network. In this distributed architecture, a client sends requests to a server, which also distributes tasks between clients, and improves collaboration and resource participation. Effective client-server communication depends on compliance with both standard communication protocols that govern language and rules that govern communication. The Transmission Control Protocol (TCP) suite serves as the protocol of choice to facilitate communication between clients and servers. Once a connection is established, TCP handles message exchange, dealing with issues such as flow control, retransmission of corrupted or lost packets, and routing packet forwarding across networks.

In contrast, the Internet Protocol (IP) is a connectionless protocol, which means that packets traveling over the network do not communicate with each other [86, 87]. When a client wants to connect to a server, the first step is to request the server’s IP address from the Domain Name System (DNS). The DNS server responds to this query with the relevant IP address. The client then sends a request to the specified IP address, and the server responds with the requested information, including the specified port number of the application [86–88] Upon receipt of the response, the response packet is routed to the specific application one based on the associated port number, as shown in Figure 3.

![Fig. 3. Architecture of the Client-Server [26.]](image-url)
VI. Proposed work

The goal of metaheuristic algorithms is to find the optimal solution by using search agents to explore different possibilities. One challenge heuristic algorithms face is their ability to incorporate local optima. The metaheuristic algorithm addresses this issue with a comprehensive search and implementation plan. In the search phase, the search area is systematically searched to estimate possible solutions. Then, the implementation phase involves selecting the optimal solutions identified during the evaluation phase. Unlike the previous system that used three hybrid algorithms [27], the updated version adopts a client/server distributed architecture, and separates responsibilities. Both metaheuristic and hybrid algorithms have the same goal of managing local connectivity best practice, trying to increase performance.

The hybrid TWGH algorithm exploits the strengths of the Gray Wolf Optimizer (GWO) in the implementation phase and the Whale Optimizer (WOA) algorithm in the exploration phase to find optimal solutions. The Gray Wolf optimizer is known to efficiently extract optimal solutions from possibilities, and it meets key features that make it a valuable feature in a hybrid approach, if the hybrid TWGH algorithm aims to find a global solution best for real-world situations.

The TWGH estimation level reflects improvements in the convergence performance, focusing on the refinement of the method. This improvement is achieved by incorporating the spiral update equation from the whale optimizer algorithm into the gray wolf optimizer algorithm. Equations (1) and (2) show the modified tool in spiral and hunting positions.

\[
ps = e^{bk} \cos(2\pi k)D^l + X(i) \tag{1}
\]

\[
D^l = |\hat{c}_1 \cdot \vec{X}_a - ps| \\
D^p = |\hat{c}_2 \cdot \vec{X}_p - ps| \\
D^\delta = |\hat{c}_3 \cdot \vec{X}_\delta - ps| \tag{2}
\]

Where \( D^l(\vec{X} - \vec{X}) \) shows how far a whale is from its meal, and \( b \) is a constant used to describe the overall form of the logarithmic spiral. \( D^a, D^p, \) and \( D^\delta \) are the positions of the top three search agents, mean, and random integer \( l \) between \([-1, 1]\), respectively.

The client/server paradigm uses Python to communicate with remote applications. The server actively listens for connection requests from clients, either on the network or on the same device. The port number and IP address of the client server are required to establish a connection. The server receives the client request and responds by connecting to the network. Applications can be broken down into subtasks, allowing for decentralization. In some cases, the socket-exchanged data includes information that is processed by one or two algorithms implemented on the client side. Typically, the client initiates a request for information or actions from the server, performs the action, and retrieves corresponding data.

Once the connection is established, the client participates in WOA and GWO population insertion, fitness function calculation, and search operator calculation (see Figure 4). Both algorithms continuously update positions. The best solution that reaches the completed state is sent back to the server. Rather than maintaining a static structure, the HSA approach dynamically changes the parameters of HMCR and PAR to accelerate convergence. The HSA algorithm adjusts as necessary, recalculates the updated fitness score, and returns the best score to the client. When the final endpoint is reached, the session is terminated, and the result is returned and displayed on the server side.
This hybrid algorithm was developed during two decades of research into the development of combinatorial search algorithms. A comparative analysis of TWGH design with others as a way to evaluate TWGH for other methods, this study was conducted as a comparison of the time required for tests provided by TWGH, and the time required for different test deliveries by another well-known benchmark design. As a result, it can be directly compared with the results published for the methods in [16], [27], [28]. For each block, the researchers repeated the TWGH run 20 times to calculate the average and optimal test set sizes. The computer is powered by an Intel(R) Core (TM) i7-10750H CPU clocked at 2.60GHz and has 24GB of RAM. The operating system used is Windows 11 Home. In this experiment, speed was measured by the total amount of time spent consuming different populations of users. Speed is a metric used in computer programming to compare how two types of programs deal with the same problem effectively. The serial and parallel running times are $T_s$ and $T_p$ respectively, as can be seen in Eq. (3):

$$ Speedup = \frac{T_s}{T_p} $$

Several different parallel projects were performed to evaluate the performance of the client/server distributed framework. Additionally, a specific benchmark function was used for each experiment to establish the optimization method at multiple speeds. Figures 5–15 provide graphical representations of the speed profiles in the benchmark. A series of several previously considered TWGH benchmark projects were used to evaluate the effectiveness of the client/server distributed framework. The applications used to evaluate the performance of TWGH can be divided into four categories, namely those following: CPU Multiprocessor, CPU-GPU1 (with inactive operation), and CPU-GPU2 (with inactive operation).

As shown in the figures, the speedup rates obtained in each benchmark application are different from each other. This is due to the fact that the complexity of all benchmark functions falls into different areas as
mentioned earlier. The client/server distributed framework provides an average speed up of 8X compared to all other benchmarks (shown in Figure 15).

Fig. 5. Parallel Implementation Speedup Factor of t=2.

Fig. 6. Parallel Implementation Speedup Factor of t=3.
Fig. 7. Parallel Implementation Speedup Factor of $t=4$.

Fig. 8. Parallel Implementation Speedup Factor of $t=5$. 
Fig. 9. Parallel Implementation Speedup Factor of $t=6$.

Fig. 10. Parallel Implementation Speedup Factor of $t=7$. 
Fig. 11. Parallel Implementation Speedup Factor of t=8.

Fig. 12. Parallel Implementation Speedup Factor of t=9.
Fig. 13. Parallel Implementation Speedup Factor of $t=10$.

Fig. 14. Parallel Implementation Speedup Factor of $t=11$. 

**Execution Time for Different Configurations ($t=10$)**

**Execution Time for Different Configurations ($t=11$)**
Concluding from analysis, we find that the degree of parallel processing increases remarkably when dealing with small processor systems and neural networks a few parts of the solution but this effort deteriorates as the size of computational neurons increases. The main reason for this degradation is the fact that, when divided into sparsely populated compute nodes, each mapper processes only a small fraction of the total population. Array lengths are considered. Notably, the speed improves as the complexity of the benchmark task increases with time, indicating the effect of array dimensions on the optimization complexity and subsequent execution time.

VIII. Conclusions

In conclusion, the paper highlights the importance of the hybrid whale-Greywolf harmony approach, especially for coverage arrays (2 ≤ v ≤ 5, 2 ≤ p ≤ 12) and power (2 ≤ t ≤ 12). This hybrid approach proves to be a highly competitive technology, emphasizing the need for hybrid approaches to achieve the best results. Whale and Greywolf experiments were carefully planned and conducted two sets of hybridization (TWGH) tests to assess the effect of each decision on the final master plan. This paper introduces the parallel version of the TWGH algorithm, which uses a client/server framework. Several studies were conducted to demonstrate the effectiveness of this approach. Based on the experimental results, the average velocity was calculated and two-way statistics were compared between the methods. It is worth noting that the TWGH showed variability in the velocity tests. When implemented in a client/server configuration, the proposed method achieved a remarkable improvement of 8.51. This highlights the flexibility and efficiency of the parallel TWGH method in solving challenging optimization challenges.

References


