A Clustering-Based Approach to Identifying Potential Output Anomalies with Combinatorial Testing for Web Applications

Karen Wei and Shin-Jie Lee *

Posted Date: 13 December 2023

doi: 10.20944/preprints202312.0955.v1

Keywords: combinatorial testing; pairwise testing; hierarchical clustering; test oracle

Article

A Clustering-Based Approach to Identifying Potential Output Anomalies with Combinatorial Testing for Web Applications

Karen Wei and Shin-Jie Lee *

Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan

* Correspondence: jielee@mail.ncku.edu.tw

Abstract: The process of testing software is examining the artifacts and behavior of the software under test by validating and verifying it. Testing can provide valuable information about the software. Given an input to the system, there is a challenge of distinguishing correct from potentially incorrect behavior, which is the test oracle problem. Although the combinatorial testing approach can effectively reduce the number of test cases, the manual inspection of the outputs is still required. In this paper, we propose an approach to automatically classify the return pages with conducting combinatorial testing based on page similarities and hierarchical clustering, by which potential output anomalies can be identified. In the experiment, we apply our approach to a real-world e-commerce website, an anomaly can be found with the visualization of the clustering hierarchy, which suggests the potential of the applicability of the approach.

Keywords: combinatorial testing; pairwise testing; hierarchical clustering; test oracle

1. Introduction

Software testing plays a critical role in ensuring the reliability and functionality of software systems. The process of testing software involves a meticulous examination of its artifacts and behavior, aiming to validate and verify its functionality, which is pivotal in providing valuable insights into the software's performance and reliability.

A fundamental challenge in software testing is the test oracle problem, which involves distinguishing the correct behavior of a software system from potentially incorrect behavior. When confronted with various inputs, it becomes essential to accurately identify and validate the expected outcomes. Often, the manual inspection of the outputs is required, however, this is costly and unreliable. Automating test oracles will be required to ensure the quality of the test and its cost reduction [1].

To address the challenge of exhaustive test case generation for functional testing instead of security testing [2], the combinatorial testing approach is used, it is a testing method that can effectively reduce the number of required test cases. In [3], it is suggested that failures in a system are caused by interaction between relatively few parameters, and not every parameter contributes to failures. The most feasible t-wise testing is pairwise testing, where all possible pairs of parameter values are covered by at least one test [4]. Combinatorial testing has found extensive application in diverse software types, including LLM testing [5], software product line testing [6], autonomous driving systems [7] and multi-robot [8]. Although the combinatorial testing approach can effectively reduce the number of test cases, the manual inspection of the outputs is still required to ensure the system's correctness.

In this paper, we propose an approach that seeks to automate the classification of return pages, as shown in Figure 1, by conducting combinatorial testing based on page similarities and hierarchical clustering. By leveraging these techniques, our proposed approach identifies potential output anomalies, streamlining the testing process.

To evaluate the effectiveness of our proposed approach, we conducted an experiment on a real-world e-commerce website. Through the visualization of a hierarchical cluster, we successfully identified an anomaly, highlighting our approach's potential applicability in real-world scenarios.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

The objective of this paper is to present an automated approach to classifying return pages using combinatorial testing, page similarities, and hierarchical clustering. In the following sections, we will look into the details of our proposed approach, work related to techniques used in this paper, discuss the experimental setup and results, and finally conclude with a comprehensive analysis of our findings.

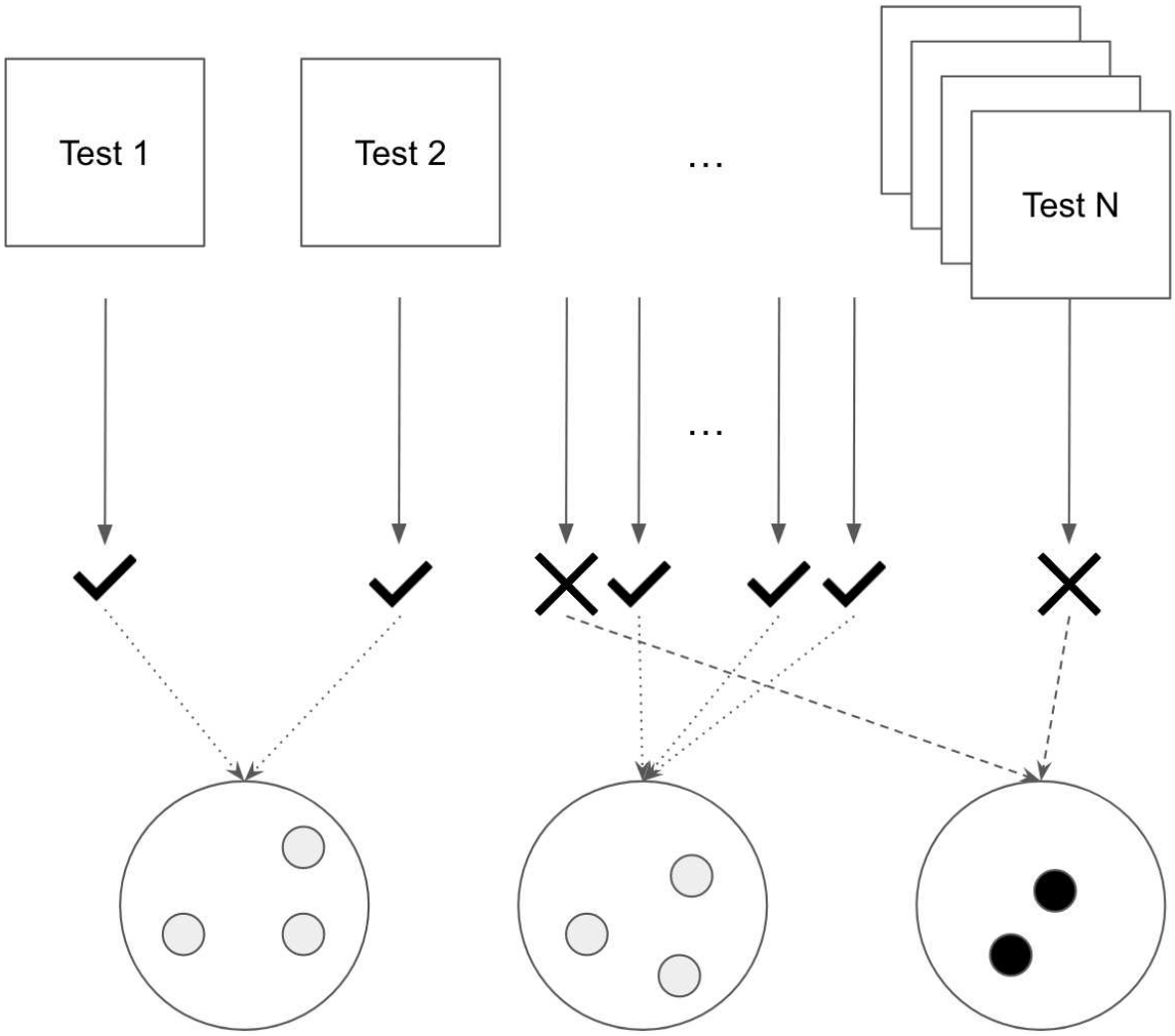


Figure 1. Clustering potential output anomalies.

2. The Proposed Approach

This section presents our proposed approach to tackle the test oracle problem by identifying potential output anomalies. The primary objective is to offer a comprehensive understanding of the proposed approach.

Four subsections will be discussed, beginning with the system architecture where we outline the overall framework involved. Next, we will thoroughly discuss the method employed, explaining the specific steps and procedures to conduct our proposed approach.

Following the discussion of the method used, we introduce the comparison algorithm that facilitates the comparisons between each page. The chosen algorithm plays a crucial role in determining page similarities and dissimilarities, aiding in the classification of return pages. Finally, we present a clustering algorithm that enables the detection of potential output anomalies.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

2.1. System Architecture

The implementation of our solution encompasses three fundamental components: web page surveying, preprocessing, and clustering. A web page is defined as a single document in a web application. Figure 2 shows a more detailed description of the system architecture.

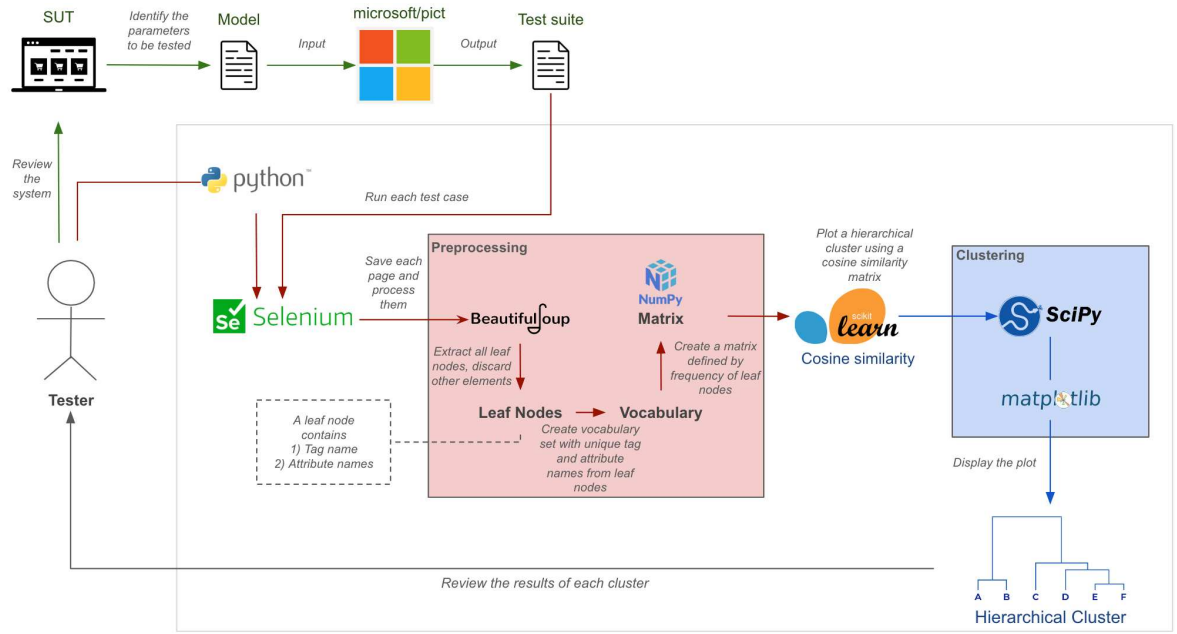


Figure 2. The system architecture where web page surveying is shown with green arrows, preprocessing with red arrows, and clustering with blue arrows.

Surveying the web page is the initial stage, where a detailed examination of the web page is conducted. This process involves identifying and collecting input parameters that need to be tested. These input parameters are compiled into a list with their inherent values, which is used to construct a model. The model facilitates the creation of a combinatorial test suite generated with PICT.

Microsoft's PICT (Pairwise Independent Combinatorial Testing) tool automatically generates test cases and test configurations. The model we applied in this paper is described in Microsoft PICT's documentation. A further and more complete definition and usage of a model can be found in [9].

Following web page surveying, the test suite first undergoes testing where each test case outputs a page source. A page source is a return page, or more specifically, an HTML DOM tree of the chosen web page (a test case) that has undergone testing. Then these page sources are preprocessed and used for extracting and refining essential features. The features are relevant to produce the proper input for cosine similarity, allowing us to produce an effective comparison by determining page similarities and dissimilarities based on relevant characteristics.

Finally, clustering is employed as the final component. In this paper, we use hierarchical clustering to group similar web pages together based on their similarities. This will reveal relationships and patterns among the return pages, identifying potential output anomalies from the expected behavior. Refer to Algorithm 1 for the pseudocode of the proposed solution just described.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

Algorithm 1: The Proposed Solution

Let N be the number of test cases;
Let $page\_source[1...N]$ be a new array where each index is a page source of a test case;
Let $soup[1...N]$ be a new array where each index is BeautifulSoup's parsed version of $page\_source[1...N]$;
Let $leaf\_nodes[1...N]$ be a new array of an array of leaf nodes where each index contains, a tag name, and a list of its attribute names taken from $soup$;
Let $vocabulary$ be a set of vocabulary of all the unique HTML tags and attribute names in $leaf\_nodes$;
Let $matrix$ be a matrix that represents the frequency of each tag and attribute name in the vocabulary;
Let $similarity\_matrix$ be the computed version of $matrix$ using cosine similarity;
Plot the resulting $similarity\_matrix$ using an hierarchical cluster;

2.2. Method

Firstly, Microsoft PICT is utilized to automatically generate a combinatorial test suite from the model created for the web page. After obtaining the test suite, Selenium is used to automate the testing process, while also capturing and saving the page sources.

Next, each page source is parsed with BeautifulSoup, a Python library designed for parsing HTML and XML documents. This library enables us to easily navigate the HTML DOM tree. Specifically, it allows for the extraction of the leaf nodes, which are nodes without any child elements, then discards any extraneous elements.

We specifically only extract leaf nodes from a page source since, generally the majority of visible elements on a web page are leaf nodes, and the rest are containers containing the leaf nodes. In the abstract, we are comparing visible elements on a web page rather than futile elements such as containers. As described later in the subsection "Comparing Algorithms", we chose an algorithm that does not consider the structure of the tree, and we will see more in-depth why it might be practical to extract the leaf nodes.

Leaf nodes is an array containing the list of tag and attribute names respective of each page. It encompasses two key components, the element tag name, and all the attribute names. However, the attribute values are excluded from this definition, as they can be dynamic or changing values, such as a unique identifier. Refer to Algorithm 2 for the creation of $leaf\_nodes$ and Figure 3 for a visual representation.

Algorithm 2: Construct Leaf Nodes Array

Input: $soup$, a collection of page sources parsed from BeautifulSoup;
Output: An array of leaf nodes called $leaf\_nodes$;
Let leaf_nodes[1..N] be a new array;
foreach $s \in soup$ **do**
    tag $\leftarrow$ tag name of s;
    attributes $\leftarrow$ a list of attribute names of s;
    $leaf\_nodes[i] \leftarrow$ (tag, attributes);
end

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

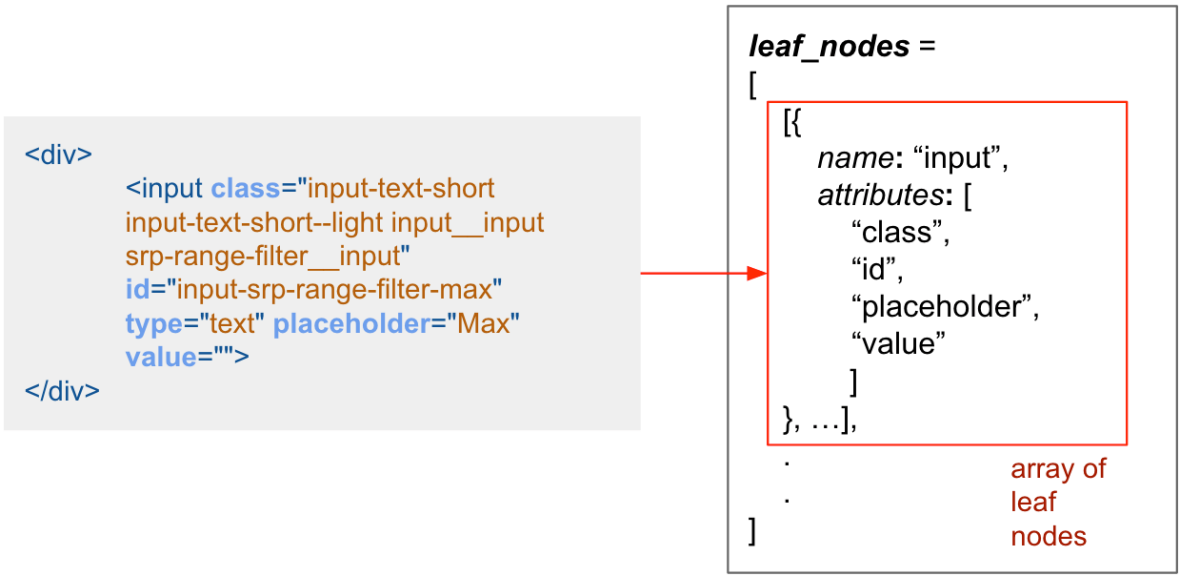


Figure 3. Leaf nodes from an HTML tree extracted into an array of leaf nodes.

Next, we create a vocabulary set, which will be used to create a dedicated matrix for our comparing algorithm. The vocabulary created here is compiled of unique combinations of tag names and attribute names extracted from every single page's leaf nodes. It can be referred to as the dictionary of the processed test suite.

A matrix is then created to store the frequency of each vocabulary for the returned page corresponding to every test case. Please refer to Algorithm 3 for a detailed explanation of how the matrix is constructed. This matrix serves as the input for scikit-learn's cosine similarity function. In order to gain a deeper understanding of cosine similarity and explore an alternative comparison algorithm we have considered, we will introduce a new subsection titled 'Comparison Algorithms'.

Algorithm 3: Construct a Matrix

```
Let matrix be a matrix that represents the frequency of each tag and attribute name in the
  vocabulary respective to each page
foreach page ∈ leaf_nodes do
    foreach element ∈ page do
        foreach tag ∈ element do
            foreach attr ∈ tag do
                Let i be the index of (tag, attr) in vocabulary;
                Increment matrix[page][i] by 1;
            end
        end
    end
end
```

2.3. Comparison Algorithm

In this subsection, we will introduce two comparison algorithms, cosine similarity, and Zhang and Shasha's (ZSS) tree edit distance algorithm.

Originally, it was decided to use ZSS's tree edit distance for the comparison algorithm to compute the similarity between every page, more information on the algorithm can be found in [10]. However, ZSS's own algorithm runs in $O(n^4)$ time for n nodes, making it very expensive for a big system. As a result, a new decision was made to use cosine similarity, running much faster with $O(n^2)$ time. Our

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

system needs to process many HTML DOM trees, which may make processing them costly, especially when n grows larger, hence the new decision for using cosine similarity.

Although tree edit distance has high costs, it is more accurate since it takes into account the structure of the tree for comparison. In contrast, cosine similarity does not consider the tree's structure when comparing two trees. Nonetheless, the final decision was to use cosine similarity for our comparison algorithm despite the algorithm not taking into consideration the structure of the tree.

Cosine similarity measures the cosine of the angle between two vectors, which can be interpreted as a measure of their alignment. Let t_1 and t_2 be vectors taken from $matrix$, where t_1 is created from t_1's leaf nodes and t_2 is created from t_2's leaf nodes. The formula for cosine similarity between t_1 and t_2 is $\cos(\theta) = \frac{\boldsymbol{t_1} \cdot \boldsymbol{t_2}}{\|\boldsymbol{t_1}\|_2 \|\boldsymbol{t_2}\|_2}$.

2.4. Clustering Algorithm

This subsection presents the clustering algorithm we applied in our proposed solution, which is SciPy's hierarchical clustering. We will provide an in-depth discussion on both the concepts and implementation of this algorithm.

Hierarchical clustering is a method of clustering analysis, its strategies fall into two categories: 1) Agglomerative, a "bottom-up" approach, and 2) Divisive, a "top-down" approach. One advantage to hierarchical clustering is that any measure of distance can be used, i.e., a matrix of distances, and observations are not required.

In our proposed approach, the agglomerative was used, each observation is its own cluster, and eventually, pairs of clusters are merged as they move up the hierarchy. In order to decide which cluster will be combined, a measure of similarity between clusters is required, this is known as cluster linkage.

The chosen linkage method in our approach is called single linkage, it measures the distance between the closest points in two clusters. Since our goal is to capture closely connected data points, the single linkage helps with the overall connectivity of these data points. Let C_1 and C_2 be two clusters and $dist(C_1, C_2)$ represent the distance between them. Let $C_1[i]$ be for all points i in cluster C_1 and $C_2[j]$ be for all points j in cluster C_2. Single linkage considers all possible distances between data points in C1 and C2 and selects the smallest distance as the distance between the clusters, as shown in Figure 4.

Finally, after obtaining the hierarchical cluster, we need a visualization of it to see the results more clearly, and we use Matplotlib's dendrogram for displaying the hierarchical cluster in our experiment.

In the following section, we will discuss the works related to our proposed solution, i.e., combinatorial testing on web applications, clustering to classify a system's response, and the test oracle problem.

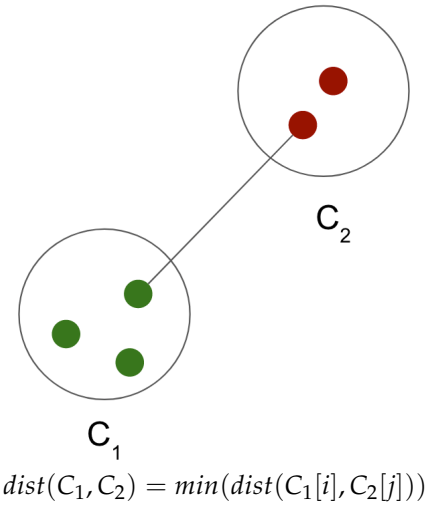


Figure 4. Single linkage clustering.

 doi:10.20944/preprints202312.0955.v1

3. Related Work

Combinatorial testing has been applied to be used to test web applications. This includes web security [11,12], additionally, it has also been used to build automated models for dynamic explorations of a website [13,14].

The use of clustering to automatically classify a system's response is demonstrated in [15], where their approach was to automatically classify the responses turned by web servers using data clustering techniques, and was based on a similarity approach.

Another research [16] discusses the use of clustering by grouping the resulting failures into smaller clusters, whereas successful results group into larger clusters. Instead of examining all the test outputs, the oracle problem may be reduced significantly by only examining the outputs of the smaller clusters.

Since the test oracle problem still exists within combinatorial testing, one of the most common approaches includes humans who analyze these results. Others are crash testing, assertions, and model-based test generation [4]. In addition, there are researches in automating the test oracle problem, in [1], they discuss the automation methods they used, which include N-Version Diverse Systems, and M-Model Program Testing, Decision Table, IFN Regression Tester, AI Planner Test Oracle, ANN Based Test Oracle, and Input/output Analysis Based Automatic Expected Output Generator.

In our research, we discussed an alternative approach to alleviating the current test oracle problem, more specifically, we identified potential output anomalies. Similar to [16] and [17], our goal is to separate failures and successes into different clusters. However, in our approach, multiple clusters are created, each with its own distinct behavior. This approach allows the tester to examine the clusters and their inputs to determine and identify the collection of inputs that triggers the system's different types of behavior. In the next section, we experiment with the proposed solution to investigate its applicability to web pages.

4. A Case Study

In this section, we experiment on a real-world e-commerce website to assess the effectiveness of our clustering-based solution. Our priority was to identify any bugs or abnormal behavior in the system by testing the website's combinations of input parameters and clustering any potential anomalies in the return pages.

Firstly, we surveyed the demonstration website, and four input parameters were selected to create our model, the inputs selected are circled in red with their names also labeled in red in Figure 5. They consist of a checkbox element, a select element, and two number text fields. We retained the already defined parameter's values as they were, for example, a checkbox has the values "unchecked" and "checked". Then we self-defined the values for inputs that did not have those values, such as the min and max number text fields. To reduce the infinite possibilities of inputs in the number text field, we set its parameter values to zero, a positive number, a negative number, and a special case known as a fuzz definition. A fuzz definition is an input that is invalid, malformed, or unexpected, which might allow the system to reveal software defects and vulnerabilities.

With the help of PICT, we were able to generate our pairwise tests suite which consists of 19 test cases, this process is demonstrated in Figure 6. Our test suite is a result of a pairwise combination, which means all possible pairs of parameter values are covered by at least one test. This test suite is turned into a Selenium test suite so the testing process can be automated. The result of the test process is an output of 19 page sources which are ready to be preprocessed.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

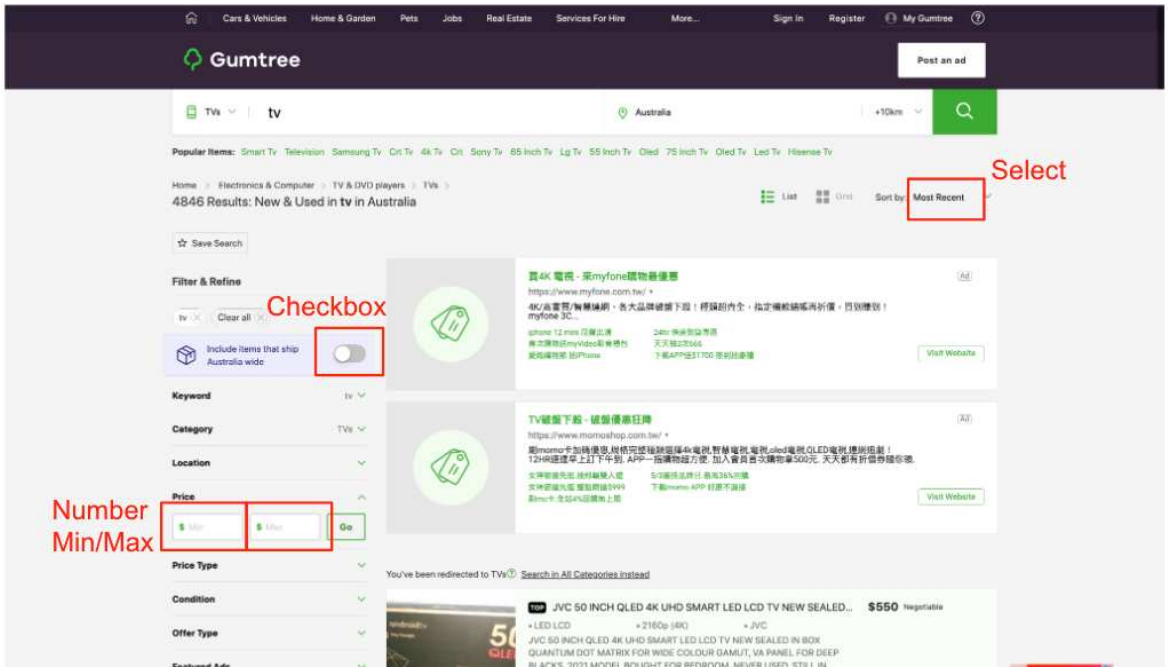
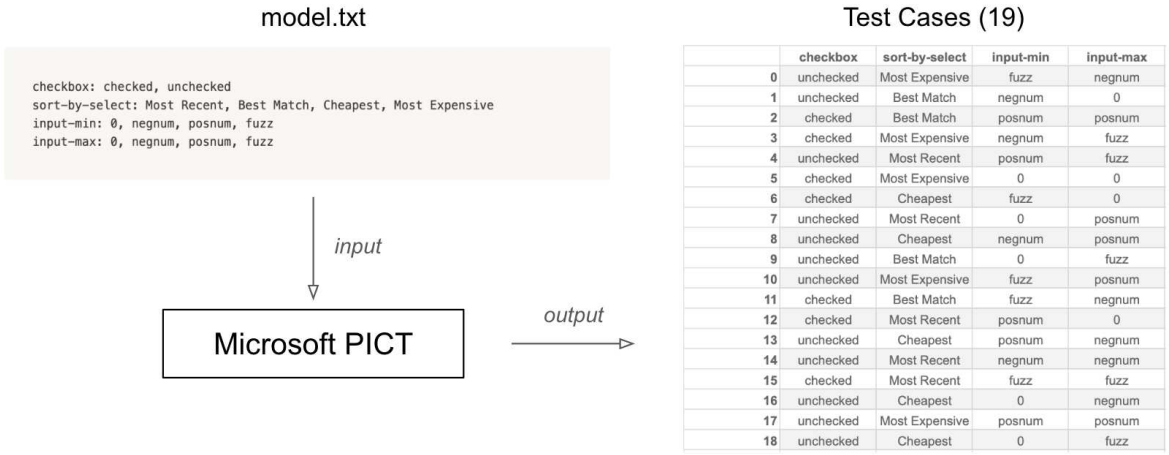


Figure 5. The demonstration website used in our experiment.



	checkbox	sort-by-select	input-min	input-max
0	unchecked	Most Expensive	fuzz	negnum
1	unchecked	Best Match	negnum	0
2	checked	Best Match	posnum	posnum
3	checked	Most Expensive	negnum	fuzz
4	unchecked	Most Recent	posnum	fuzz
5	checked	Most Expensive	0	0
6	checked	Cheapest	fuzz	0
7	unchecked	Most Recent	0	posnum
8	unchecked	Cheapest	negnum	posnum
9	unchecked	Best Match	0	fuzz
10	unchecked	Most Expensive	fuzz	posnum
11	checked	Best Match	fuzz	negnum
12	checked	Most Recent	posnum	0
13	unchecked	Cheapest	posnum	negnum
14	unchecked	Most Recent	negnum	negnum
15	checked	Most Recent	fuzz	fuzz
16	unchecked	Cheapest	0	negnum
17	unchecked	Most Expensive	posnum	posnum
18	unchecked	Cheapest	0	fuzz

Figure 6. The input and output from Microsoft PICT.

The page sources are then preprocessed using an HTML Parser, which is later on used to create leaf nodes, vocabulary, and a matrix like mentioned earlier. A similarity matrix is then created with the matrix from earlier calculated using cosine similarity. Now with the similarity matrix, we are able to create an agglomerative hierarchical cluster.

As shown in Figure 7, the page sources are preprocessed with an HTML Parser, which is used to then used to create leaf nodes, vocabulary, and a matrix. The resulting matrix creates a new matrix, which is the similarity matrix, a result of cosine similarity. With the similarity matrix, we are able to create an agglomerative hierarchical cluster.

Once the hierarchical cluster is created, the results obtained are shown in Figure 8. On the left side are two of the same zoomed-out hierarchical cluster, which is also the hierarchical cluster we get from the initial results. On the right side is the zoomed-in version of the original zoomed-out hierarchical cluster, where we can see more clearly that there are even more clusters within the big cluster seen in the zoomed-out version.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

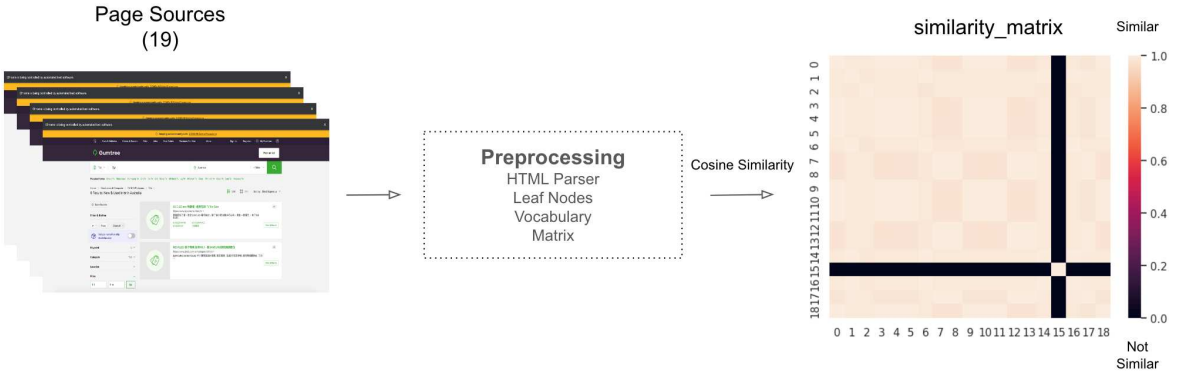


Figure 7. Preprocessing page sources into a similarity matrix with cosine similarity

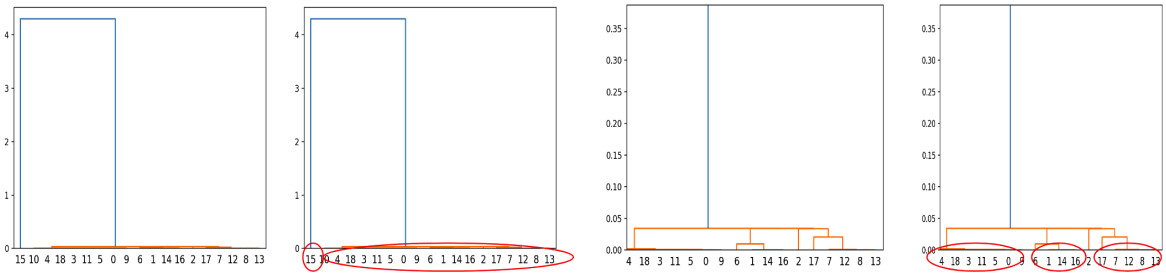


Figure 8. The hierarchical cluster, zoomed-out (left) and zoomed-in (right) versions.

Each of the red circles on both the zoomed-out and zoomed- in versions represents a cluster, they are there to aid to visualize more clearly how many clusters exist. In the zoomed- out version, it is evident that there are 2 clusters and that the cluster of test case number 15 is an anomaly in the test suite. Additionally, in the zoomed-in version, there are 3 more clusters inside the original big cluster. This can be seen more clearly with the hierarchical cluster heatmap in Figure 9. The legend on the heatmap representing the maxiumum value, 1.0, and the minimum value, 0.0, where 1.0 means identical, and 0.0 means no similarities. It is obvious that test case 15 shows 0.0 when compared to every other test cases, this means that 15 must be an anomaly.

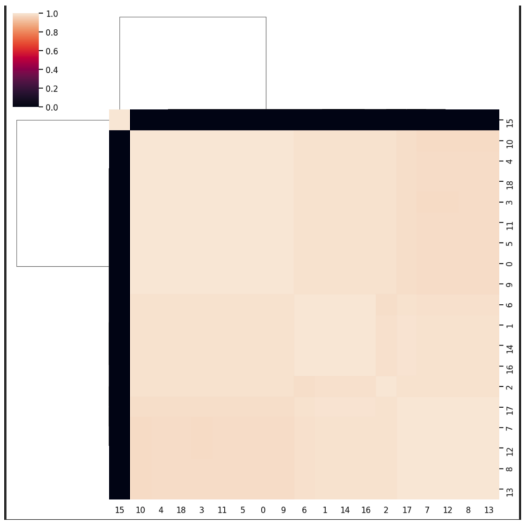


Figure 9. The hierarchical cluster heatmap.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

With no doubt, as seen in Figure 10, this page is an anomaly since the page only shows "HTTP Status 400 - Bad Request". The test case's similarity is so dissimilar to the other test cases, the distance between the rest of the test suite and this test case is so far off, it has become its own cluster.

The rest of the clusters, shown in Figures 11–13 are clusters with the remaining test cases. Based on human judgment, they all look identical to one another. However, visually, even with such irrefutable similarities, each one of the test cases still has different behaviors that can be detected causing them to separate into 3 different clusters. In cluster 1, all the test cases had shown 0 results on the return pages, whereas cluster 2 had 4, and cluster 3 had 4509 results. Despite the return pages' similarity with one another, there are still subtle differences between them that allow us to group them into their respective clusters where each cluster exhibits different behaviors.

Figure 10. The anomaly page.

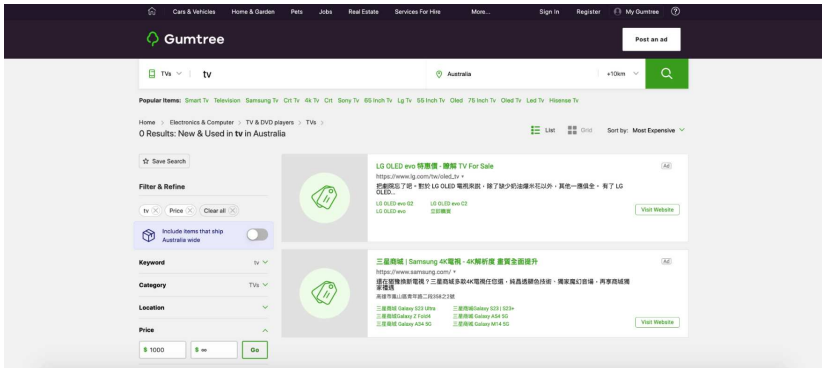


Figure 11. Example taken from cluster 1.

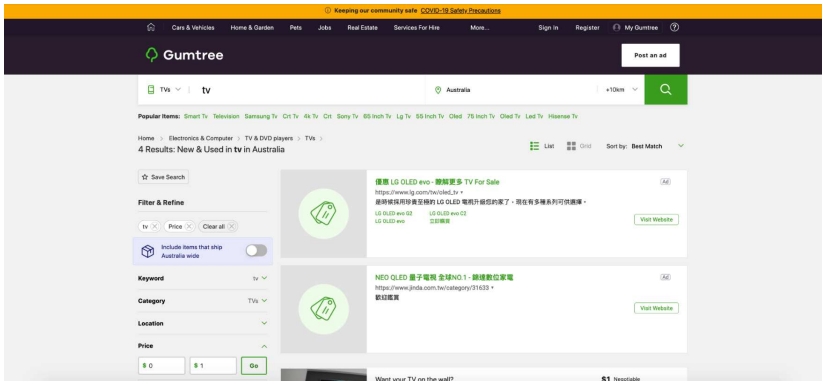


Figure 12. Example taken from cluster 2.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 December 2023 doi:10.20944/preprints202312.0955.v1

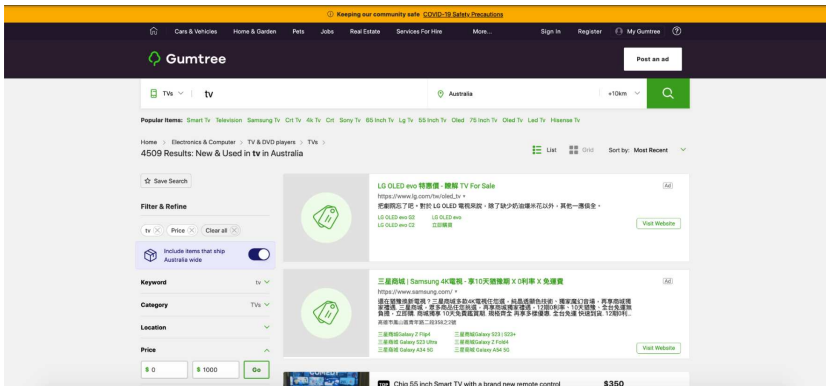


Figure 13. Example taken from cluster 3.

5. Discussion

In this paper, we proposed an approach to automatically classify the return pages by conducting combinatorial testing using cosine similarity and hierarchical clustering, and the objective was to identify potential output anomalies. In this section, we discuss the results of our experiment, highlight limitations, and suggest future research directions.

Firstly, our proposed approach successfully identified output anomalies with hierarchical clustering that clustered similar return pages together. The experiment results demonstrated that the clustering-based approach can effectively identify anomalous behaviors in the system that may be hard to distinguish through human analysis. This indicates the potential applicability of our cluster-based approach, and the findings in our experiment could have significant implications as it provides valuable information about the clustering of anomalies.

However, it is important to acknowledge the limitations of our study. The effectiveness of our approach heavily relies on the quality and which type of input parameter and values are used. In our experiment, we created a fuzz definition, which effectively found bugs in the system under test which resulted in the discovery of output anomalies. However careful consideration must still be given to the selection and generation of test cases to ensure comprehensive coverage. Additionally, the hierarchical clustering process is still influenced by the choice of algorithms and parameters, which may introduce biases or affect the accuracy of anomaly detection. These limitations highlight areas for further research and improvement.

One suggestion for future work is to address the challenge of interpreting clusters without a representation. Currently, when a cluster does not have a clear representative output, it becomes difficult to judge the overall behavior of the cluster as a whole. In such cases, it may be beneficial to explore techniques that enable the identification or generation of representative test cases for these clusters. By including a representative output within each cluster, the behavior and characteristics of the anomalous outputs within the cluster can be more distinct and easier to analyze.

6. Conclusion

Combinatorial testing covers all possible pairs by discovering flawed interactions between a few parameter values. Although this testing approach already effectively reduces the number of test cases, the test oracle problem still exists, and to alleviate this problem, we used a clustering-based method to group similar test cases together.

By utilizing clusters, we were able to associate return pages with their respective clusters based on their specific behaviors. This facilitated the investigation of each individual cluster, enabling us to identify clusters exhibiting abnormal behaviors. This approach greatly enabled the discovery of output anomalies, as return pages within the same cluster exhibited similar and distinctive behaviors.

In conclusion, our clustering-based approach for identifying potential output anomalies with combinatorial testing for web applications has demonstrated promising results. By leveraging

clustering algorithms, we have shown the ability to distinguish correct behavior from potentially incorrect behavior. Although there are limitations, our study opens avenues for further research and improvement. The findings have practical implications for enhancing the quality assurance processes of web applications and provide a foundation for future investigations in this domain.

Funding: This research is sponsored by the National Science and Technology Council under the grant NSTC 112-2221-E-006 -084 -MY2 in Taiwan.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shahamiri, S.R.; Kadir, W.M.N.W.; Mohd-Hashim, S.Z. A comparative study on automated software test oracle methods. 2009 fourth international conference on software engineering advances. IEEE, 2009, pp. 140–145.
2. Zhu, X.; Wen, S.; Camtepe, S.; Xiang, Y. Fuzzing: a survey for roadmap. *ACM Comput. Surv.* **2022**, *54*.
3. Cohen, D.M.; Dalal, S.R.; Fredman, M.L.; Patton, G.C. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* **1997**, *23*, 437–444.
4. Kuhn, D.R.; Kacker, R.N.; Lei, Y.; others. Practical combinatorial testing. *NIST special Publication* **2010**, *800*, 142.
5. Garn, B.; Kampel, L.; Leithner, M.; Celic, B.; Çulha, C.; Hiess, I.; Kieseberg, K.; Koelbing, M.; Schreiber, D.P.; Wagner, M.; Wech, C.; Zivanovic, J.; Simos, D.E. Applying Pairwise Combinatorial Testing to Large Language Model Testing. Testing Software and Systems; Bonfanti, S.; Gargantini, A.; Salvaneschi, P., Eds. Springer Nature Switzerland, 2023, pp. 247–256.
6. Sharafeldin, K.K.; Rabatul Aduni, B.S.; Mohammed Adam, K.A.; Jasni, M.Z.; Odili, J.B.; Al-Shami, S.A. Pairwise Test Case Generation using (1+1) Evolutionary Algorithm for Software Product Line Testing. *International Journal of Advanced Computer Science and Applications* **2023**, *14*.
7. Klück, F.; Li, Y.; Tao, J.; Wotawa, F. An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems. *Information and Software Technology* **2023**, *160*, 107225.
8. Nishitani, I.; Yamaguchi, T.; Hoxha, B. Checkmate: Fault Timing Localization for Multi-Robot Scenarios. 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), 2023, pp. 1–8.
9. Czerwonka, J. Pairwise testing in the real world: Practical extensions to test-case scenarios. Proceedings of 24th Pacific Northwest Software Quality Conference, Citeseer. Citeseer, 2006, pp. 419–430.
10. Zhang, K.; Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing* **1989**, *18*, 1245–1262.
11. Garn, B.; Kapsalis, I.; Simos, D.E.; Winkler, S. On the applicability of combinatorial testing to web application security testing: a case study. 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing, 2014, pp. 16–21.
12. Garn, B.; Lang, D.S.; Leithner, M.; Kuhn, D.R.; Kacker, R.; Simos, D.E. Combinatorially XSSing Web Application Firewalls. 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2021, pp. 85–94.
13. Qi, X.; Wang, Z.; Mao, J.; Wang, P. Automated testing of web applications using combinatorial strategies. *Journal of Computer Science and Technology* **2017**, *32*, 199–210.
14. Wang, W.; Sampath, S.; Lei, Y.; Kacker, R.; Kuhn, R.; Lawrence, J. Using combinatorial testing to build navigation graphs for dynamic web applications. *Software Testing, Verification and Reliability* **2016**, *26*, 318–346.
15. Dessiatnikoff, A.; Akrout, R.; Alata, E.; Kaâniche, M.; Nicomette, V. A clustering approach for web vulnerabilities detection. 2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing. IEEE, 2011, pp. 194–203.
16. Almaghairbe, R.; Roper, M. Building test oracles by clustering failures. 2015 IEEE/ACM 10th International Workshop on Automation of Software Test. IEEE, 2015, pp. 3–7.

17. Almaghairbe, R.; Roper, M. Separating passing and failing test executions by clustering anomalies. *Software Quality Journal* **2017**, *25*, 803–840.