Multiperspective Web Testing Supported by a Generation Hyper-Heuristic

Juliana Marino Balera(✉) and Valdivino Alexandre de Santiago Júnior

Instituto Nacional de Pesquisas Espaciais (INPE),
Av. dos Astronautas 1758, São José dos Campos, SP, Brazil
{juliana.balera,valdivino.santiago}@inpe.br

Abstract. Web interface testing is a sort of system testing level and it is laborious if accomplished manually, since it is necessary to map each of the elements that make up the interface with its respective code. Furthermore, this mapping makes test scripts very sensitive to any changes to the interface's source code. Approaches for automated web testing have been proposed but the use of hyper-heuristics, higher-level search techniques aiming to address the generalization issues of metaheuristics, for web testing are scarce in the literature. In this article we present a multi-objective web testing method, MWTest, which automates the generation of test cases based only on the URL of the web application and a new proposed generation hyper-heuristic, called GECOMBI. The GECOMBI hyper-heuristic takes into account combinatorial designs to generate low-level heuristics to support our goal. Moreover, the implementation of the MWTest method creates a Selenium test script quickly and without human interaction, exclusively based on the URL in order to support the automated execution of test cases too. In our evaluation, we compared GECOMBI to another generation hyper-heuristic, GEMOITO, and four metaheuristics (NSGA-II, IBEA, MOMBI, NSGA-III). Results show superior performance of GECOMBI compared to the other approaches.

Keywords: Web testing · Hyper-heuristics · Combinatorial designs · System testing

1 Introduction

Interface testing is an important part of the testing cycle of a web application. The goal is to identify failures from the externally visible behavior of the application [13]. However, this type of test depends on the mapping of each of the components of the interface to the source code and this is a cumbersome task if performed manually. Moreover, this mapping makes test scripts very sensitive to any changes to the interface's source code. Due to the rapid evolution of software nowadays, including web applications, following practices such as continuous integration (CI), effective test cases/data generation must resort to

O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13381, pp. 447–462, 2022.
https://doi.org/10.1007/978-3-031-10548-7_33

automated approaches to ensure that high quality web systems are produced. Therefore, manually generating and executing web test cases is not feasible in this context.

Testing of web applications has naturally been addressed by researchers where the most common types of testing are functional, security, usability, performance, compatibility, and structural testing [4]. However, one interesting direction is trying to address different perspectives, e.g., functional and non-functional, altogether to generate test cases. Hence, several different characteristics are considered together to generate the test suites (sequences of test cases)

Search-based software testing (SBST) is a subfield of search-based software engineering (SBSE) which is formed by the combination of software testing and optimization [6,18,25]. SBST is precisely suitable to deal with the previous multi-perspective test case generation approach. In SBST, testing a software system is formulated as an optimization problem and the main reasoning is that test objectives can be considered as objective functions, and hence optimization algorithms can be used to help in this regard.

As for SBST, metaheuristics such as evolutionary algorithms (EAs) (genetic algorithm (GA) [23,24]), particle swarm optimization (PSO) [21,30], simulated annealing (SA) [16] have been employed, indeed dominating the subfield [6]. However, researchers claim that metaheurisitics suffer from the lack of generalization. With the goal of tackling the generalization issue, hyper-heuristics [14,26] emerged as general-purpose high-level optimization methods controlling or building (components of) low-level (meta)heuristics (LLHs), and they have been preferred less than the metaheuristics for SBST [6].

Moreover, to the best of our knowledge, no previous work has used hyper-heuristics for test case generation for web interface systems following the multi-perspective point of view we have just mentioned above.

Currently, the most used technology for automating web interface testing is the Selenium framework. With Selenium, it is possible to automate the execution of test flows in several programming languages such as Java, Javascript, Ruby and Python.

In this article we present a Multi-objective Web Testing method, MWTest, which automates the generation of test cases based only on the URL of the web application and a new proposed generation hyper-heuristic, called GEneration hyper-heuristic via COMBInatorial designs (GECOMBI)(the code can be access in [1]). As its name implies, GECOMBI takes into account combinatorial designs [5] to generate LLHs to support our goal. Within our method, we used as LLHs four metaheuristics: Indicator-Based Evolutionary Algorithm (IBEA) [32], Metaheuristic for Many-objective Optimization based on the R2 Indicator (MOMBI) [17], Nondominated Sorting Genetic Algorithm-II (NSGA-II) [12] and III (NSGA-III) [11]. Moreover, the implementation of the MWTest method creates a Selenium test script quickly and without human interaction, exclusively based on the URL in order to support the automated execution of test cases too.

In our experiment, we then compared GECOMBI to these four meta-heuristics and one other generation hyper-heuristic: Grammatical Evolution hyper-heuristic for the Multi-objective Integration and Test Order problem (GEMOITO) [22]. Moreover, case studies come from five different web applications developed by the *Instituto Nacional de Pesquisas Espaciais* (INPE).

This article is organized as follows. Section 2 presents some relevant related studies. The MWTest method and GECOMBI hyper-heuristic are presented in Sect. 3. Experimental evaluation and results are in Sect. 4. Conclusions and future directions are in Sect. 5.

2 Related Work

Several approaches have been proposed for web applications testing [4,13]. We mention here some relevant studies related to ours.

2.1 Web Testing

The study proposed by [29] proposes the A POGEN tool. Using the Page Object pattern, the tool is able to automatically produce transformed Java page objects for Selenium WebDriver through a combination of clustering and static analysis. However, an approach possibly requires intervention, in addition, only the human elements that have an id are automatically identified.

The work proposed in [28] proposes a tool capable of creating a test specification at different levels of abstraction. To implement it, it is necessary to use the TTCN-3 language, which allows more robust test cases. The approach however, is dependent on human interaction and considerable knowledge of software testing techniques to be able to apply it.

The approach proposed in [9] allows non-functional requirements such as security to be explored in web test cases. However, the tool does not offer the generation of an automatic test script, and like the approaches mentioned above, it requires knowledge in software testing for your application.

The approach proposed in [20], like the previous approach, focuses on test cases based on non-functional requirements, specifically vulnerability. However, the approach does not support automated test case generation.

All the approaches mentioned above are very promising approaches for this aspect of software engineering. However, in general, it can be said that they do not offer full support to the developer who will use them, since it requires knowledge in software testing, or does not support the generation of automated test case scripts. Furthermore, some of the approaches did not unite the exploration of functional and non-functional requirements. The approach proposed in this work, unites all these differential in a single tool: automatic generation and without interaction of a test script and exploration of functional and non-functional requirements simultaneously.

2.2 Hyper-Heuristics

Hyper-heuristics are high-level optimization methods where the search is performed in the space of heuristics (or heuristics components) instead of being performed directly in the decision variable space (space of solutions) [14,26]. One domain to classify hyper-heuristics is in accordance with the nature of the heuristic search space which defines the characteristics of the search space. In the space of LLHs, there can be **selection hyper-heuristics**, which are methodologies designed to select an already existing set of LLHs, and **generation hyper-heuristics**, which are methodologies that generate new LLHs from other preexisting ones.

As recently reported, within SBST, more selection hyper-heuristics have been used compared to the generation ones [6], even if some authors argue that generation hyper-heuristics possess more features for greater level of generalization compared to the selection counterparts [10]. Hence, in this study we also aimed to realize the performance of a new proposed generation hyper-heuristics for web testing. However, we present below some studies relying on generation hyper-heuristics.

The GEMOITO geneation hyper-heuristic was proposed in [22]. GEMOITO was designed for the solution of the integration and test problems. The approach makes use of the grammatical evolution (GE) technique, explained in Sect. 3, and consequently, defines a specific grammar that contemplates several parameter values common to evolutionary algorithms (EAs).

In [15], authors presented an adaptation of the GEMOITO [22] hyper-heuristic, as we have just mentioned originally designed for the solution of integration and test order problems, to solve software production line (SPL) problems. This adaptation consisted in the change of the values of four parameters (mutation probability, crossover probability, mutation operator type, and crossover operator type) to values more appropriate to the class of target problems.

The differences between our research and all the previous ones are basically two. Firstly, no previous work has used a generation hyper-heuristic for test case generation for web interface systems. Since the claims of higher generalization capabilities of hyper-heuristics, we felt motivated to follow this direction. Secondly, the multi-perspective point of view, combining cost, functional, and non-functional properties (in this case, vulnerability of web applications) together is another interesting feature of our approach.

3 The MWTest Method

In this section we present our method, MWTest, whose main component is the GECOMBI generation hyper-heuristic. However, some important definitions are necessary before going on as shown in the sequence.

Definition 1. ***Abstract test case****: An abstract test case is one whose representation does not allow it to be effectively executed against the SUT. Such an abstract test case serves as a guide for generating the truly executable test case. Particular, in our case, an abstract test case is a sequence of vertices of the Event Flow Graph (EFG).*

Definition 2. ***Solution as a test suite with variable size****: A solution of a population created by an optimization algorithm is indeed a test suite, i.e. a sequence of abstract test cases. The number of abstract test cases a solution may have depends on the number of terminal vertices of the EFG, and which are present in the solution.*

Definition 3. ***Decision variable as test step****: A decision variable is one element of a solution. The value of the decision variable of a solution identifies a vertex of the EFG. It is therefore considered a test step of an abstract test case.*

From this point onward, unless otherwise noted, we will denote an **abstract test case** simply as a **test case** for simplicity. Basically, our method starts by handling a web site url which is its input. Hence, the method proposes the automated generation of an EFG [8] based only on the source code of the web application. Considering a set of objective functions (functional and non-functional properties) and this EFG, GECOMBI generates LLHs which are responsible for the creation of (abstract) test suites. Finally, the method proposes that one or more test suites are randomly selected to stimulate the web application.

In order to better explain our method, Fig. 1 shows one instance of the MWTest method. In other words, this is an actual implementation of our method, with a set of tools that we developed or adapted. We used this implementation to accomplish the experiment presented in Sect. 4.

Basically, our method (instance) starts by downloading a website's source code from its URL, using a crawler written in Python. This crawler relies on the BeautifulSoup [3] library. It takes as input the URL of the website being tested, downloads the source code, and stores it in a text file. This source code is just the client code, that is, the code downloaded by the browser when the user accesses the site.

The next step is the generation of the EFG model. For this, there is also a specialized module that receives as input the text file that contains the code downloaded in the previous step. From this text, the elements that correspond to the implementation of some component of the web interface are identified, such as a text box. After identifying all these components, the EFG model will be generated, which consists of a directed graph where each of its nodes correspond to elements of the web interface and the edges are the possible interactions between them. Nodes that do not have edges that depart from it are terminal nodes, corresponding to buttons to close or cancel something.

The main module of our approach, the GECOMBI hyper-heuristic. See detailed explanation of our hyper-heuristic in Sect. 3.1).

The set of LLHs that compose the GECOMBI hyper-heuristic will be executed and the product will be a test suite, where each position corresponds to an interaction with the target web interface. After that, an executable script will be generated using the Selenium framework. This script can be generated in different programming languages, depending on the tester's needs. This script is generated by the Selenium Generator module. When executing the generated script, it will simulate a user flow (defined by the test suite sequence) in a "fake" browser, Webdriver.

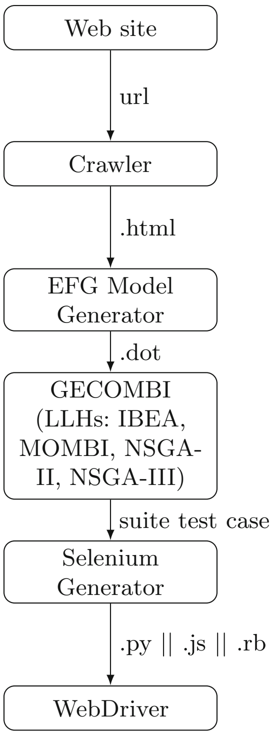

Fig. 1. An instance of the MWTest method

3.1 The Gecombi Hyper-Heuristic

In the context of generation hyper-heuristics applied to software testing, grammatical evolution (GE) [15] stands as the main option. GE is a type of genetic programming technique capable of generating new heuristics by means of a grammar, which defines the rules of adjustment of values for the configuration parameters of a generic EA, such as crossover probability, mutation probability.

However, the use of a grammar will eventually make a search limited since, regardless of the problem class, the parameters values will always be the same, which will require an adaptation of the grammar for each type of problem. In addition, GE allows all possibilities of parameters combinations to be explored but it does not define a systematic form of searching, which leads to the problem of the blast of combinations.

The basic idea of the GECOMBI generation hyper-heuristic is that the parameters of an EA do not act independently of each other, since they operate in the same population, i.e. the interaction of the effects of each configuration parameter tends to influence the quality of the solution as a whole. For example, it is not interesting to have both a very high crossover and mutation probabilities, since this can cause the population change to be so large that each offspring population will be totally different from the previous population, and hence the algorithm never converges.

Hence, one possible strategy for parameter tuning of an EA would be to try all interactions of values for these parameters, not considering parameter interactions that do not make sense. However, this is not achievable with an exhaustive approach (all combinations) because the number of combinations is generally very high.

On the other hand, combinatorial designs with constraints allows the evaluation of interactions between the different values of the metaheuristics configuration parameters, since the aim of the technique is to try to "group" all these interactions into a smaller set, not considering combinations that do not make sense (e.g. mutation and crossover probabilities simultaneously high). Within software testing, combinatorial designs are known as combinatorial interaction testing (CIT), a well-studied strategy to generate test cases/data [5].

The GECOMBI generation hyper-heuristic is split into two parts: generating LLHs and creating test suites. The generating LLHs phase is described in Algorithm 1. GECOMBI receives, among other inputs, the EFG related to the simple/smallest web application and a set of objective functions (Obj in Algorithm 1). Hence, we relied on the T-tuple Reallocation (TTR) [5] CIT algorithm to generate the t-tuples where each t-tuple is formed by a single value of each parameter. Here, a t-tuple is also known as a **configuration** (C_i in Algorithm 1) of an LLH. VP in Algorithm 1 means sets of sets of parameter values. Let us consider the following elements (sets) of VP: crossover_prob_val = {0.01, 0.9}, mutation_prob_val = {0.01, 0.9}, population_size_val = {100, 1000, 10000}.

Moreover, let the strength (t in Algorithm 1) for TTR be equal to two (pairwise interaction). All pairwise interactions are shown in Table 1. Note that lines 1 and 4 (gray) will not be considered (constraints), since they are combinations of values that are mostly inadequate in the context of EAs. This is done in line 4 ($removeConfigs$) in Algorithm 1.

The final set of configurations after line 4 is shown in Table 2. Thus, we have this set of configurations indicated as $\langle C_1, \cdots, C_m \rangle$ in Algorithm 1.

The set of LLHs (LLH; $n = |LLH|$) is also an input to the GECOMBI's generating LLHs phase. From lines 6 to 13 we define how to select the definite LLHs that will compose GECOMBI. Thus, we execute all LLHs considering all configurations, C_i, as defined for each LLH, and get the populations Pop_i, where $0 \leq i < n$. Since TTR's output can be really large, even if a lower strength is defined, the parameter β indicates a percentage of the best generated LLHs to be considered as the ones GECOMBI definitely indicate to create the test suites. To select the LLHs which will be indeed considered as the final ones, we use quality indicators (e.g. hypervolume [33] and IGD+ [19]. Hence, the LLHs with the best values in accordance with these quality indicator are the generated (i.e. selected) ones. The output of Algorithm 1 is precisely the LLHs generated by GECOMBI (set L) illustrated in Table 2.

The second phase is to create the test suites as shown in Algorithm 2. The set L of generated LLHs are input to this procedure as well as the EFG, Obj, and the population size ($pSize$). Note that the EFGs now are from any web application

Algorithm 1. GECOMBI: Generating LLHs

```
input: $EFG, Obj, VP, t, LLH, \beta$
output: $L$

1: $Prob \leftarrow adaptProblem(EFG,\ Obj)$
2: $M \leftarrow TTR(VP,\ t)$
3: $< C1,\ C2,\ ...,\ C_k > \leftarrow splitRows(M)$
4: $< C1,\ C2,\ ...,\ C_m > \leftarrow removeConfigs(< C1,\ C2,\ ...,\ Cn >)$
5: $Pop_h \leftarrow \emptyset$
6: $i \leftarrow 0$
7: while $i < n$ do $//n\ =\ the\ number\ of\ LLHs.$
8:     for $j\ \in Ci$ do $//\ all\ configurations\ for\ an\ LLH[i]$
9:         $Pop_i \leftarrow Pop_i\ \cup runLLH(LLH[i],\ j,\ Prob)$
10:    end for
11:    $i \leftarrow i\ +\ 1$
12: end while
13: $L \leftarrow generateLLHs(< Pop_1, Pop_2, \cdots, Pop_n >, \beta)$
14: return L
```

Table 1. Matrix with all pairwise interactions of parameters

i	***crossover_prob_val***	***mutation_prob_val***	***population_size_val***
1	0.01	0.01	
2	0.01	0.9	
3	0.9	0.01	
4	0.9	0.9	
5		0.01	100
6		0.01	1000
7		0.01	10000
8		0.9	100
9		0.9	1000
10		0.9	10000
11	0.01		100
12	0.01		1000
13	0.01		10000
14	0.9		100
15	0.9		1000
16	0.9		10000

and not only the smallest one. The set of test suites (TSs in Algorithm 2) is indeed the final population due to GECOMBI obtained by generating the nondominated solutions (procedure *generateND*), and by the adjustment of the population size in accordance with *pSize* (procedure *adjustPop*).

Table 2. Final set of configurations

Config	crossover_prob_val	mutation_prob_val	population_size_val
CONF_1	0.01	0.01	100
CONF_2	0.01	0.9	1000
CONF_3	0.01	0.01	10000
CONF_4	0.9	0.9	100
CONF_5	0.9	0.01	1000
CONF_6	0.9	0.9	10000

Algorithm 2. GECOMBI: Creating Test Suites

```
input: $EFG, Obj, L, pSize$
output: $TSs$

1: $Prob \leftarrow adaptProblem(EFG,\ Obj)$
2: $TSs \leftarrow \emptyset$
3: for $l \in L$ do
4:     $TSs \leftarrow TSs \cup runLLH(l, Prob)$
5: end for
6: $TSs \leftarrow generateND(TSs)$
7: $TSs \leftarrow adjustPop(TSs, pSize)$
8: return TSs
```

4 Experimental Design and Evaluation

In this section, we present the design and characteristics of the experiment we conducted to evaluate our method.

4.1 Objective, Algorithms and Quality Indicators

The objective of this evaluation is to identify which out of six optimisation algorithms is the best regarding test case generation for web applications. We considered our new proposed generation hyper-heuristic, GECOMBI, the GEMOITO hyper-heuristic, and the metaheuristics IBEA, MOMBI, NSGA-II, and NSGA-III. Note that these metaheuristics are used as LLHs of GECOMBI.

Note that each metaheuristic was configured as follows: SBX crossover with probability 0.0001, Polynomial mutation with probability 0.00125, population and archive size equal to 20. On the hand, the GEMOITO hyper-heuristic and GECOMBI had the same sets of values. As for GECOMBI, the parameter settings are shown later in Sect. 4.5.

As quality indicators to evaluate the performance of the algorithms, we used the hypervolume and IGD+. In our case, we took into the front-normalized values of these indicators. Note that as higher the (front-normalized) hypervolume value the better the algorithm, and as lower the (front-normalized) IGD+ value, the better the approach.

4.2 Research Question and Variables

We want to answer the following research question:

- **RQ_1** - Which of the six algorithms is the best regarding each quality indicator?

The independent variables are the optimisation algorithms. The dependent variables are the values of the quality indicators: hypervolume and IGD+.

4.3 Objective Functions

The objective functions we considered are described below:

- **test case consistency:** we defined no constraints to the problem instances. Hence, the algorithms have total flexibility to generate the test cases. But, it is usually necessary that the sequence of values of the decision variables within a (abstract) test case is consistent with the sequence of vertices (edges) of the graph, otherwise an inconsistent test case will be generated. Thus, we want to maximize the test case consistency. Note that by generating consistent test cases we are traversing correctly the EFG, and hence this can be seen as a function related to functional properties;
- **length of the test suite:** this objective function has as purpose to control the amount and the position in which the terminal vertices of the EFG appear in a solution. This is a cost measure where we want to minimize the amount of test cases in the test suite;
- **vulnerability:** this objective function is related to the vulnerability of web interface applications. One of the most well-known attacks is SQL Injection. In this attack, the text fields present in the interface are exploited, and SQL commands are inserted into them that can cause some damage to the website's database. We want to maximise the number of vulnerabilities in order to create more suitable test cases. This is a function related to non-functional properties (vulnerability).

4.4 Case Studies

Case studies are 18 case web interfaces whose source code is based on Javascript extracted from the web sites from the Instituto Nacional de Pesquisas Espaciais (INPE).

4.5 Generating LLHS

As for the GENERATING LLHs phase (see Sect. 3.1, Algorithm 1), we considered the smallest (less number of vertices in the corresponding EFG) web interface among all the 18 web interfaces. Parameters are described in Table 3. Altogether, 162 LLHs were generated by GECOMBI, and we set $\beta = 0.06$. Note that since this is a real problem, we created the True Known Pareto Front according to the execution of the LLHs. Hence, GECOMBI suggests the top 10 LLHs/configurations in accordance with the highest hypervolume values.

After identifying the top 10 LLHs, we moved on to the next step where we were able to compare GECOMBI to the other algorithms. It is very important to emphasize that the final population due to GECOMBI is adjusted to fit the size of the population of the metaheuristics run in isolation (see procedure *adjustPop* in Algorithm 2). Hence, the final population of GECOMBI is 20.

Table 3. Parameter settings for GECOMBI

Parameters	Values
Population size	20, 50
Crossover probability	0.0001, 0.009, 0.005
Mutation probability	0.0001, 0.009, 0.005
Crossover operator	*TwoPoint*, *SinglePoint*, *SBX*
Mutation operator	*Polynomial*, *Random*, *CDG*
Archive	(2,3)*population_size

4.6 Results

After the training of GECOMBI and the generation of LLHs in conjunction with the Pareto Frontier of the problem, the execution phase was started. For each of the web interface under test an EFG model was generated, and each of these models was submitted to GECOMBI so that the comparison of the results obtained was compared with the results of the Pareto Frontier obtained during the training phase. This goal was achieved by calculating hypervolume and IGD+ comparators.

After obtaining the data referring to the approach proposed in this work, the data for the comparison were obtained by submitting the same problem to the GEMOITO, NSGA-II, NSGA-III, IBEA and MOMBI were also run, configured according to the literature. The configuration of the native algorithms, we considered the parameter values for the solution of the *Traveling Salesman* problem, also implemented in the jMetal tool. The Traveling Salesman problem is of the same nature as the problem of generating test cases for web interface, so it was selected. In these applications, the city concept represents the test case, and the distance concept represents the cost of the test case.

Tables 4 and 5 bring the results of all 18 GUIs submitted. In all cases, the first column corresponds to the ID of each case study. The other column is the value obtained by the algorithm corresponding to the indicator referring to the table (i.e. hypervolume or IGD). The results obtained by the GEMOITO, NSGA-II, NSGA-III, IBEA, MOMBI and GECOMBI, although are very similar, in most of cases the GECOMBI algorithm obtained slightly better results. This is further evidence of the applicability of GECOMBI to real problems.

Table 4. Hypervolume values

Case study	NSGA-II	IBEA	MOMBI	NSGA-III	GEMOITO	GECOMBI
1	0.268782	0.120026	0.131513	0.257367	0.431783	**0.516108**
2	0.333638	0.180318	0.196101	0.459991	0.441912	**0.527794**
3	0.347497	0.193399	0.160884	0.515277	**0.521259**	0.508557
4	0.341984	0.256996	0.233039	**0.593170**	0.460526	0.543643
5	0.347971	0.220834	0.221348	0.512301	0.485573	**0.528132**
6	0.339228	0.203439	0.100345	0.434617	**0.468291**	0.383098
7	0.194871	0.109718	0.088416	0.297875	0.391903	**0.420904**
8	0.212474	0.196170	0.088853	0.529682	0.529376	**0.530784**
9	0.401964	0.193806	0.126963	0.563866	0.456579	**0.605975**
10	0.241677	0.209873	0.134434	0.523692	0.528122	**0.634823**
11	0.154823	0.079160	0.070553	0.214052	**0.440771**	0.358798
12	0.243821	0.158441	0.093224	0.478388	**0.517422**	0.324694
13	0.308754	0.248139	0.096751	**0.532646**	0.518316	0.432973
14	0.348418	0.192111	0.135408	**0.510069**	0.412745	0.428427
15	0.298458	0.173096	0.197651	**0.562706**	0.498127	0.459960
16	0.256699	0.188478	0.103281	0.437255	**0.453236**	0.386052
17	0.304820	0.115310	0.143298	0.241853	0.363048	**0.488897**
18	0.429585	0.264860	0.231269	0.454193	0.461678	**0.462882**

However, it is important to apply a statistical evaluation to verify if there is a significant difference between the results obtained by GECOMBI and the other algorithms compared. For this, we follow the experiment proposed in [7], for a statistical evaluation to provide greater confidence in results.

For the statistical verification, the values of the hypervolumes and IGD+ indices obtained by each algorithm were analyzed for each case study. Just as in [7], the first step is to verify the normality of the data. For this purpose, we apply the Shapiro-Wilk test [27] with significance level $\alpha = 0.05$. This test shows that the data is not normally distributed. In this way, the nonparametric Wilcoxon test (Signed Rank) [2] was applied, with significance level $\alpha = 0.05$. The results are presented in Table 6.

According to these results, some p-values are below 0.05, there is difference between the GECOMBI and other algorithms compered, with the advantage for the GECOMBI. In other cases, there is no difference between the GECOMBI and other algorithms compered, which leads to the conclusion that GECOMBI has the potential to be used as a solution as well as algorithms already established in the literature.

Just as the studie [31] that propose new hyper-heuristics aimed at solving problems in the context of Software Testing, use hypervolume as one of the evaluation metrics. In all cases, the relational experiments include the comparison of

Table 5. IGD+ values

Case study	NSGA-II	IBEA	MOMBI	NSGA-III	GEMOITO	GECOMBI
1	0.190867	0.264044	0.254507	0.196629	0.335454	**0.116704**
2	0.178105	0.226365	0.223408	0.132267	0.309283	**0.108458**
3	0.140721	0.194362	0.211330	**0.097364**	0.265463	0.099995
4	0.193979	0.209409	0.221629	**0.091531**	0.293613	0.109736
5	0.192311	0.229109	0.235112	0.137427	0.290119	**0.116768**
6	0.173407	0.242507	0.315417	**0.130213**	0.291789	0.184460
7	0.267496	0.323623	0.343992	0.277611	0.290227	**0.156939**
8	0.235092	0.249626	0.321397	**0.082650**	0.239363	0.120851
9	0.167383	0.236273	0.305631	0.106392	0.307430	**0.086991**
10	0.211442	0.237703	0.277094	0.088141	0.247280	**0.079455**
11	0.289887	0.368626	0.386215	0.254050	0.295715	**0.173608**
12	0.219030	0.272827	0.328390	**0.104665**	0.248476	0.176153
13	0.179412	0.202748	0.294480	**0.096213**	0.236766	0.143670
14	0.163677	0.236982	0.269477	0.107956	0.335256	**0.141035**
15	0.201289	0.245688	0.235549	**0.096922**	0.255825	0.143700
16	0.192261	0.215661	0.266578	**0.112177**	0.323162	0.135705
17	0.189252	0.297050	0.256142	0.203925	0.346667	**0.114139**
18	0.142135	0.186448	0.200636	0.137599	0.287251	**0.119112**

Table 6. *Wilcoxon* test

Comparation	*p-value* (Hypervolume)	*p-value* (IGD+)
GECOMBI ↔ GEMOITO	0.4951	7.629e–06
GECOMBI ↔ NSGAIII	0.5798	0.8317
GECOMBI ↔ MOMBI	7.629e–06	7.629e–06
GECOMBI ↔ IBEA	7.629e–06	7.629e–06
GECOMBI ↔ NSGAII	7.629e–06	1.526e–05

the proposed hyper-heuristic with other algorithms in the literature. In the same way that the experiment for evaluating GECOMBI showed very close results with the values of the algorithms compared, with the studies cited the same situation also occurred. This is further evidence that GECOMBI is a scalable solution for solving real problems.

5 Conclusions

This work proposed a multi-objective web testing method, MWTest, which automates the generation of test cases based only on the URL of the web application and a new proposed generation hyper-heuristic, called GECOMBI. The GECOMBI hyper-heuristic takes into account combinatorial designs to generate low-level heuristics to support our goal. Preliminary results were obtained with the preliminary version of GECOMBI based on the execution of 18 GUIs derived from the real case studys.

Experiments were performed for comparated the NSGA-II, NSGA-III, IBEA, MOMBI and GEMOITO algorithms, configured based on values obtained in the literature. The results sought to evaluate the potential to produce good GECOMBI solutions, as well as other approaches proposed in the literature. These preliminary results show that the application of GECOMBI to the 18 case studies were slightly better than the results found in the literature.

In addition, to verify that there is a statistically significant difference between the results obtained by the compared algorithms, a statistical evaluation was performed. This test showed in some cases, that there is a statistical difference between the results obtained by GECOMBI and the other algorithms compared.

Based on the results obtained, it is possible to conclude that GECOMBI has as much potential for solving problems as well as established approaches in the literature, since the experiments related to the 18 case studies had positive results. These results obtained are encouraging for GECOMBI to continue to be exploited. Future work includes the application of GECOMBI to generate test cases that explore non-functional aspects through the modeling of new objective functions, e.g. the ones that address usability and security. In addition, we intend to carry out rigorous experiments involving GECOMBI, comparing it with other hyper-heuristics, meta-heuristics, and even for many-objective problems.

References

1. Gecombi repository. https://github.com/BaleraJuliana/GECOMBI_code. Accessed 13 July 2019
2. The Wilcoxon signed-rank test. http://www.r-tutor.com/elementary-statistics/non-parametric-methods/wilcoxon-signed-rank-test. Accessed 13 July 2019
3. (2022). https://www.crummy.com/software/BeautifulSoup/bs4/doc/
4. Al-Ahmad, B., Al-Debei, K.: Survey of testing methods for web applications. Eur. Int. J. Sci. Technol. **9**(12), 1–22 (2020)
5. Balera, J.M., Santiago Júnior, V.A.: An algorithm for combinatorial interaction testing: definitions and rigorous evaluations. J. Softw. Eng. Res. Dev. **5**(1), 10 (2017). https://doi.org/10.1186/s40411-017-0043-z
6. Balera, J.M., Santiago Júnior, V.A.: A systematic mapping addressing hyper-heuristics within search-based software testing. Inf. Softw. Technol. **114**, 176–189 (2019). https://doi.org/10.1016/j.infsof.2019.06.012, http://www.sciencedirect.com/science/article/pii/S0950584919301430
7. Balera, J.M., Santiago Júnior, V.A.d.: An algorithm for combinatorial interaction testing: definitions and rigorous evaluations. J. Softw. Eng. Res. Dev. **5**(1), 10 (2017). https://doi.org/10.1186/s40411-017-0043-z
8. Banerjee, I., Nguyen, B., Garousi, V., Memon, A.: Graphical user interface (GUI) testing: systematic mapping and repository. Inf. Softw. Technol. **55**(10), 1679–1694 (2013). https://doi.org/10.1016/j.infsof.2013.03.004, http://www.sciencedirect.com/science/article/pii/S0950584913000669

9. Bozic, J., Wotawa, F.: Planning-based security testing of web applications with attack grammars. Softw. Qual. J. **28**(1), 307–334 (2020). https://doi.org/10.1007/s11219-019-09469-y
10. Burke, E.K., et al.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013). https://doi.org/10.1057/jors.2013.71
11. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014). https://doi.org/10.1109/TEVC.2013.2281535
12. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
13. Di Lucca, G.A., Fasolino, A.R.: Testing web-based applications: the state of the art and future trends. Inf. Softw. Technol. **48**(12), 1172–1186 (2006). https://doi.org/10.1016/j.infsof.2006.06.006, https://www.sciencedirect.com/science/article/pii/S0950584906000851
14. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. Eur. J. Oper. Res. **285**(2), 405–428 (2020). https://doi.org/10.1016/j.ejor.2019.07.073, https://www.sciencedirect.com/science/article/pii/S0377221719306526
15. Filho, H.L.J., Lima, J.A.P., Vergilio, S.R.: Automatic generation of search-based algorithms applied to the feature testing of software product lines. In: Proceedings of the 31st Brazilian Symposium on Software Engineering, SBES 2017, pp. 114–123. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3131151.3131152
16. Garvin, B.J., Cohen, M.B., Dwyer, M.B.: Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empir. Softw. Eng. **16**(1), 61–102 (2011). https://doi.org/10.1007/s10664-010-9135-7
17. Gómez, R.H., Coello, C.A.C.: MOMBI: a new metaheuristic for many-objective optimization based on the R2 indicator. In: 2013 IEEE Congress on Evolutionary Computation, pp. 2488–2495 (2013). https://doi.org/10.1109/CEC.2013.6557868
18. Harman, M., Jia, Y., Zhang, Y.: Achievements, open problems and challenges for search based software testing. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1–12, April 2015. https://doi.org/10.1109/ICST.2015.7102580
19. Ishibuchi, H., Masuda, H., Nojima, Y.: A study on performance evaluation ability of a modified inverted generational distance indicator. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 695–702. GECCO 2015, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2739480.2754792, http://doi.acm.org/10.1145/2739480.2754792
20. Jan, S., Panichella, A., Arcuri, A., Briand, L.: Search-based multi-vulnerability testing of xml injections in web applications. Empir. Softw. Eng. **24**, 3696–3729 (2019). https://doi.org/10.1007/s10664-019-09707-8
21. Mahmoud, T., Ahmed, B.S.: An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use. Expert Syst. App. **42**(22), 8753–8765 (2015). https://doi.org/10.1016/j.eswa.2015.07.029, http://www.sciencedirect.com/science/article/pii/S0957417415004893
22. Mariani, T., Guizzo, G., Vergilio, S.R., Pozo, A.T.R.: Grammatical evolution for the multi-objective integration and test order problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 1069–1076. GECCO 2016, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2908812.2908816

23. McCaffrey, J.D.: An empirical study of pairwise test set generation using a genetic algorithm. In: 2010 Seventh International Conference on Information Technology: New Generations, pp. 992–997, April 2010. https://doi.org/10.1109/ITNG.2010.93
24. Petke, J., Cohen, M.B., Harman, M., Yoo, S.: Practical combinatorial interaction testing: empirical findings on efficiency and early fault detection. IEEE Trans. Softw. Eng. **41**(9), 901–924 (2015). https://doi.org/10.1109/TSE.2015.2421279
25. Saeed, A., Ab Hamid, S.H., Mustafa, M.B.: The experimental applications of search-based techniques for model-based testing: taxonomy and systematic literature review. Appl. Soft Comput. **49**, 1094–1117 (2016). https://doi.org/10.1016/j.asoc.2016.08.030, https://www.sciencedirect.com/science/article/pii/S1568494616304240
26. Santiago Júnior, V.A., Özcan, E., Carvalho, V.R.: Hyper-heuristics based on reinforcement learning, balanced heuristic selection and group decision acceptance. Appl. Soft Comput. **97**, 106760 (2020). https://doi.org/10.1016/j.asoc.2020.106760, https://www.sciencedirect.com/science/article/pii/S1568494620306980
27. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**(3–4), 591–611 (1965)
28. Stepien, B., Peyton, L., Xiong, P.: Framework testing of web applications using TTCN-3. STTT **10**, 371–381 (2008). https://doi.org/10.1007/s10009-008-0082-1
29. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: APOGEN: automatic page object generator for web testing. Softw. Qual. J. **25**(3), 1007–1039 (2016). https://doi.org/10.1007/s11219-016-9331-9
30. Wu, H., Nie, C., Kuo, F.C., Leung, H., Colbourn, C.J.: A discrete particle swarm optimization for covering array generation. IEEE Trans. Evol. Comput. **19**(4), 575–591 (2015). https://doi.org/10.1109/TEVC.2014.2362532
31. Zamli, K.Z., Din, F., Kendall, G., Ahmed, B.S.: An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation. Inf. Sci. **399**, 121–153 (2017). https://doi.org/10.1016/j.ins.2017.03.007, http://www.sciencedirect.com/science/article/pii/S0020025517305820
32. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Yao, X., et al. (eds.) Parallel Problem Solving from Nature - PPSN VIII, pp. 832–842. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30217-9_84
33. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999). https://doi.org/10.1109/4235.797969