A Controlled Experiment for Combinatorial Testing

Juliana Marino Balera
Instituto Nacional de Pesquisas Espaciais
(INPE)
São José dos Campos, SP, Brazil
juliana.balera@inpe.br

Valdivino Alexandre de Santiago Júnior
Instituto Nacional de Pesquisas Espaciais
(INPE)
São José dos Campos, SP, Brazil
valdivino.santiago@inpe.br

ABSTRACT

In this paper, we present a controlled experiment for *combinatorial designs* algorithms aiming at software test case generation. We compare our recently proposed algorithm, TTR, to generate Mixed-Level Covering Array (MCA) with four other well-known combinatorial designs algorithms/tools regarding two aspects: cost in terms of the size of the set of test cases, and cost in terms of the time to generate the test suites. We used a set of 27 instances for this experiment. Results show that our algorithm was the best in terms of the size of the test suite, but was the poorest in terms of the time to generate the test cases. However, the not so good performance of our algorithm regarding the time to generate the test suite can be alleviated by the fact that TTR produces shorter set of test cases to be executed. We also made a comparison about the similarity of the test cases, i.e. to realize how similar are the test suites (test input data) of TTR compared with the other four algorithms/tools. We conclude that the TTR's test suite is not similar to any other test suites meaning that our algorithm has the potential to uncover different software defects by exercising different parts of the Software Under Test (SUT).

CCS Concepts

•Mathematics of computing → Combinatorial algorithms; *Statistical software;* **•Software and its engineering → Software testing and debugging;**

Keywords

Controlled Experiment; Combinatorial Designs; Software Testing

1. INTRODUCTION

Theoretically, to find all software defects it should be needed to test the Software Under Test (SUT) exhaustively [17]. However, in practice this is unfeasible because the number of possible combinations of the test input data is extremely high, particularly considering complex software. Therefore, the use of software testing techniques is necessary to select a subset within the total number of possible test cases [6] which has a high likelihood of revealing many of the software defects, turning the software testing process feasible.

Combinatorial designs are a set of methods for generating/selecting test cases that allow the selection of a relatively small set of test cases even if the input domain, and the number of subdomains after partitioning is large and complex [13]. These methods have proven to be effective in the discovery of defects due to the interaction of several input variables (factors).

One combinatorial designs technique, which appears to be one of the most used by the software testing community, is Mixed-Level Covering Array (MCA). In MCA, it is possible that factors (parameters) take levels (values) from different sets. It is a technique of unbalanced designs [13]. MCA is a matrix defined by MCA (N,$l_1^{k_1} l_2^{k_2} ... l_p^{k_p}$, t), where: N is the number of rows in the matrix; the k_1 factor has l_1 levels, ..., and the k_p factor has l_p levels. The parameter t corresponds to the degree of interaction of factors, known as the strength. In the context of software testing, each row of the MCA is a test case[1]. Therefore, an MCA is a test suite.

A recent study [11] presented a survey on the efforts related to more rigorous evaluations (controlled experiments, quasi-experiments [23]), in 25 years in the software testing community in Brazil, represented by the Brazilian Symposium on Software Engineering (SBES), and in the world, represented by the International Conference on Software Engineering (ICSE). The research considered, among other situations, the evolution of the application of assessment methods in articles related to software testing, and its rigor. Results show that there was an increase in the rate of scientific articles employing methods of evaluation through the years, however only one study, in each conference, reported application of controlled experiments, where the number of samples is considered large (> 10 participants), hypotheses are formulated, and a statistical evaluation of the results can provide greater confidence in them [23].

Based on this motivation to present more rigorous evaluations, this work aims to show the results of a controlled experiment to compare our efforts to generate MCAs, the algorithm T-Tuple Reallocation (TTR) [1], with four other well-known algorithms/tools for combinatorial testing: IPOG-F [8], jenny [9], IPO-TConfig [21], and PICT [4]. We carried

[1] However, for combinatorial designs, it is common to consider 1 "test case" as being only the test input data of this case. In this work, we will adopt this convention.

SAST, September 19-20, 2016, Maringa, Parana, Brazil
© 2016 ACM. ISBN 978-1-4503-4766-2/16/09...$15.00
DOI: http://dx.doi.org/10.1145/2993288.2993289

out two cost comparisons: size of the test suite and time to generate the test suites. Results showed that our algorithm, TTR, was the best in terms of the size of the test suite (smaller size), but had the poorest performance in terms of time to generate the test suites (greater time). However, the weakest performance in terms of time to generate the test suites can be offset by the fact that TTR generates a smaller number of test cases to run. A set of 27 instances/samples (composed of factors and levels that serve to generate an MCA) was used, with values of strength, t, ranging from 2 to 6.

In addition, we conducted a similarity analysis between the sets of test cases generated by TTR and the other four algorithms/tools. Our conclusion is that the TTR test suite is not similar to any other test suite, meaning that our algorithm has the potential to identify different defects by stimulating different parts of the SUT.

This paper is organized as follows. Section 2 presents the main concepts behind TTR. Section 3 details the planning of the experimental study. Experiment's results and analysis are presented in Section 4. Section 5 presents related work, and in Section 6 are the conclusions and future work.

2. OVERVIEW OF TTR

The TTR algorithm [1] aims at generating MCAs by the t-way testing technique. A high-level view of TTR[2] is presented in Algorithm 1. The general concept of TTR is to build an array of combinatorial designs through the reallocation of tuples from the matrix Θ to the matrix M, and then each relocated tuple should cover the greatest number of tuples not yet covered, considering a parameter called a *goal* (ζ). Also note that f is the submitted set of factors and t is the selected strength. In addition, we should recall that a *tuple* is a finite ordered list of elements. In the sequence, we will briefly comment on the main features of the algorithm.

Algorithm 1 The TTR algorithm

```
$\Theta \leftarrow constructor(f, t)$
$M \leftarrow calculateInitialSolution(\Theta)$
**while** $\Theta \neq \emptyset$ **do**
    $\zeta \leftarrow calculateZeta(M)$
    $(M, \Theta) \leftarrow main(M, \Theta, \zeta)$
**end while**
```

Constructor: this procedure aims to generate all tuples to be covered. They are produced as follows: all strength-based factor groups are created so that each group has the size of the strength, without repetition. After that, all combinations of levels corresponding to these factor groups are made. For example, let us consider two factors A and B with two levels each and $t = 2$ (strength). We will have a factor group "AB" with 4 tuples: (A1, B1), (A1, B2), (A2, B1) and (A2, B2). Each tuple is associated with a parameter called *flag*, which helps in the tuple's selection process for reallocation.

Initial Solution: this procedure concerns building an initial solution, which is made in the following way: among the factor groups generated by the *constructor* procedure, TTR selects the factor group that has the greatest amount of tuples. Each tuple is removed from the matrix Θ and reallocated in the matrix M, becoming then a test case. Thus, each test case covers at least one tuple and, hence, a factor group is covered completely.

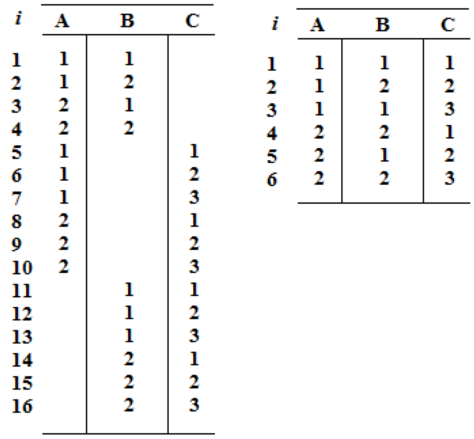
Goal: it is a value associated with each row of the matrix M, and is related to the "potential" coverage of each row of the array. They are calculated by simply combining the value of the strength, t, and the number of factors covered by each tuple at a certain time. For example, let us consider the following factors A, B and C with 2, 2 and 3 levels, respectively, and $t = 2$. There are three possible factor groups: "AB", "AC" and "BC". Each group will have $(2*2) = 4$, $(2 * 3) = 6$ and $(2 * 3) = 6$ tuples, respectively. The *calculateInitialSolution* procedure will reallocate the factor group "AC" into the matrix M, thus no tuple combination of "AC" need to be covered, which means that, from now on, each test case can cover up two tuples, because there exist two factor groups that have tuples not yet covered.

Main: this procedure transforms the matrix produced by the initial solution into the final solution (final set of test cases). After constructing Θ, the initial matrix M, and the calculation of each goal (ζ), the *main* procedure builds test cases to cover the greatest amount of tuples by means of the gradual reallocation of tuples from Θ to M. For each iteration, the factor group with more uncovered tuples is selected, and then each one of its tuples is temporarily combined with each row of the matrix M. Then it is calculated the amount of tuples not covered by this temporary row and compared with the value of goal for that row. If these values are equal, then the tuple is reallocated permanently to that row of M, otherwise it is compared with the next row of M until it fits somewhere. The procedure accomplishes this comparison until all tuples of that factor group are reallocated.

It is important to consider the case where a tuple is compared with all the rows of the matrix M, however, it does not fit in any place. For this case, TTR generates a "marking" and the algorithm proceeds to the next factor group. When this group is again selected by *main* as the group with the greatest amount of uncovered tuples, this marking indicates that the value of the goal must be reduced. This ensures that all tuples are covered by the algorithm.

Let us consider Figure 1 where we see factors A, B and C, with 2, 2 and 3 levels, respectively. Let us also consider pairwise design, $t = 2$. All tuples generated by the *constructor* procedure, i.e. the initial Θ, can be viewed on the leftmost matrix in Figure 1. The rightmost matrix is the final solution (matrix M). To achieve that, the *calculateInitialSolution* procedure selects the factor group with the greatest number of tuples, in this case "AC" corresponding to tuples 5-10 (6 tuples) in the initial Θ which are then reallocated into M. Note that factor group "BC" has also 6 tuples (tuples 11-16) and could be selected. TTR always chooses a factor group which has related the greatest amount of tuples according to the input of factors. Thereafter, the *calculateZeta* procedure finds the goals (ζ) in accordance with the number of factor groups that were not covered. At this point, all goals are 2 ("AB" and "BC" are not covered). Then, the *main* procedure searches the factor group that possesses the greatest amount of tuples not yet covered. Tuples 11-16 (due to factor group "BC") in the initial Θ are selected. Tuple 11 is combined with tuple 5 and this generates test case 1 in the matrix M. Thus, note that this process addresses two tuples (1 and 11) of the initial Θ and this is precisely equals to the goal (ζ). Then, it is not necessary to

[2] The current version of the algorithm is 1.1.



i	A	B	C
1	1	1	
2	1	2	
3	2	1	
4	2	2	
5	1		1
6	1		2
7	1		3
8	2		1
9	2		2
10	2		3
11		1	1
12		1	2
13		1	3
14		2	1
15		2	2
16		2	3

i	A	B	C
1	1	1	1
2	1	2	2
3	1	1	3
4	2	2	1
5	2	1	2
6	2	2	3

Figure 1: Example of TTR in operation. The leftmost matrix is the initial Θ and the rightmost matrix is the final M

perform any other action related to test case 1 in M. Next, tuple 12 is selected and the comparison process is repeated.

In version 1.0 of TTR, the implementation of the algorithm was insensitive to the order of input of factors and levels. In other words, the instances $6^1 7^1 2^1 4^1 3^1 7^1 4^1 9^1$ and $9^1 6^1 7^1 2^1 4^1 3^1 7^1 4^1$ always generated the same amount of test cases via TTR. Other solutions, such as IPOG-F and jenny, vary the amount of test cases according to the order of input of factors and levels. Despite insensitivity is something interesting, we noticed an even better improvement in TTR performance in terms of a smaller number of test cases generated, if, like the others, our algorithm was sensitive to the order of input of factors/levels. Thus, we created version 1.1 of TTR where sensitivity to the order of factors and levels is one of its features.

3. EXPERIMENTAL DESIGN

In this section, we present the description of the controlled experiment in regard to its **Design/Planning**.

3.1 Definition

The primary aim of this study is to evaluate two cost perspectives related to the generation of test cases via combinatorial designs/MCA considering our solution, version 1.1 of the TTR algorithm, and four other algorithms/tools proposed in the literature: IPOG-F [8], jenny [9], IPO-TConfig [21], and PICT [4]. The first definition of cost refers to the size of the test suites. The second definition of cost refers to the time to generate the test suite, based on each algorithm/tool. We should emphasize that this time is not the time to run the test suites derived from each algorithm. A third comparison has been done where we analyzed the similarity between the generated test suites. Here, the goal is to realize whether the test cases, generated by a certain approach, differ or are similar to test cases generated by another solution.

3.2 Context

The experiment was conducted by the researchers who defined it. The experimentation process proposed in [22] was used as the basis for the accomplishment of this controlled experiment, using the R [10] tool for cost analysis, and python [14] for the similarity analysis.

As our objective is basically to compare the cost of several solutions in the literature with our proposal, the set of samples is, in fact, formed by instances that will be submitted to the algorithms/tools for the generation of the matrices (test suites). Initially, we chose 33 test instances/samples (composed of factors and levels) with the strength, t, ranging from 2 to 6. However, only 27 out of the 33 instances could generate test cases for all algorithms: in 6 instances, IPO-TConfig failed to finish its execution, even after 24 hours of execution and, therefore, we discarded these 6 instances. Table 1 shows the 27 instances/samples used in this study.

It is important to characterize each instance/sample. Let us consider instance $i = 1$ in Table 1:

$$6^1 7^1 2^1 4^1 3^1 7^1 4^1 9^1, \quad strength = 2 \tag{1}$$

For example, in the context of unit test case generation for programs developed according to the Object-Oriented Programming (OOP) paradigm, this instance can be used to generate test cases for a class that had one attribute (factor) which could take 6 values (levels; 6^1), 1 attribute that could take 7 values (7^1), another attribute that could take 2 values (2^1), $\cdots$, 1 attribute that could take 9 values (9^1). In the system and acceptance testing context, this same sample could be used to identify test scenarios (test objectives) in a model-based test case generation approach [18, 19]. In both cases, the test suites must meet the criteria of pairwise testing (strength = 2), where each combination of 2 values of all factors must be covered.

3.3 Hypotheses

For the evaluation of cost related to the size of the test suites and the time to generate them, the following hypotheses were considered:

Table 1: Samples for the controlled experiment: Instances

i	Strength	Instance
1	2	$6^1 7^1 2^1 4^1 3^1 7^1 4^1 9^1$
2	2	$2^3 3^1 4^1 2^2 3^1$
3	2	$9^1 8^1 2^1 4^1 5^1 6^1 7^1 2^1$
4	2	$3^2 2^1 4^1 5^1 6^1 9^1 2^1$
5	2	$2^2 3^2 4^2 5^2 6^2 7^2$
6	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2$
7	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2$
8	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2 10^2 11^2$
9	3	$2^1 5^1 2^1 3^2 2^3$
10	3	$2^3 3^1 4^1 2^2 3^1$
11	3	$2^1 5^1 6^1 2^2 3^1 2^2$
12	3	$3^2 2^1 4^1 2^4$
13	3	$2^2 3^2 4^2 5^2$
14	3	$2^2 3^2 4^2 5^2 6^2$
15	3	$2^2 3^2 4^2 5^2 6^2 7^2$
16	4	$5^1 3^1 2^2 3^1 2^1 4^1 3^1$
17	4	$22^3 3^1 4^1 2^2 3^1$
18	4	$2^1 5^1 2^1 4^1 5^1 3^1 4^1 2^1$
19	4	$3^2 2^1 4^1 5^1 6^1 2^2$
20	4	$2^2 3^2 4^2$
21	4	$2^2 3^2 4^2 5^2$
22	5	$2^3 4^1 3^1 2^1 4^1 3^1$
23	5	$2^3 3^1 4^1 2^2 3^1$
24	5	$3^1 4^1 2^1 4^1 5^2 2^2$
25	5	$3^2 2^1 4^1 5^1 3^1 2^2$
26	5	$2^2 3^2 4^2 5^2$
27	6	$2^2 3^2 4^2$

- **Null Hypothesis H1.0:** There is no difference in cost (size of the test suites) between TTR and IPOG-F;
- **Alternative Hypothesis H1.1:** There is difference in cost (size of the test suites) between TTR and IPOG-F;
- **Null Hypothesis H2.0:** There is no difference in cost (size of the test suites) between TTR and jenny;
- **Alternative Hypothesis H2.1:** There is difference in cost (size of the test suites) between TTR and jenny;
- **Null Hypothesis H3.0:** There is no difference in cost (size of the test suites) between TTR and IPO-TConfig;
- **Alternative Hypothesis H3.1:** There is difference in cost (size of the test suites) between TTR and IPO-TConfig;
- **Null Hypothesis H4.0:** There is no difference in cost (size of the test suites) between TTR and PICT;
- **Alternative Hypothesis H4.1:** There is difference in cost (size of the test suites) between TTR and PICT;
- **Null Hypothesis H5.0:** There is no difference in cost (time to generate the test cases) between TTR and IPOG-F;
- **Alternative Hypothesis H5.1:** There is difference in cost (time to generate the test cases) between TTR and IPOG-F;
- **Null Hypothesis H6.0:** There is no difference in cost (time to generate the test cases) between TTR and jenny;
- **Alternative Hypothesis H6.1:** There is difference in cost (time to generate the test cases) between TTR and jenny.

With regard to the similarity analysis, the following hypotheses were considered:

- **Null Hypothesis H7.0:** There is no difference of similarity between the test suites of TTR and IPOG-F;
- **Alternative Hypothesis H7.1:** There is difference of similarity between the test suites of TTR and IPOG-F;
- **Null Hypothesis H8.0:** There is no difference of similarity between the test suites of TTR and jenny;
- **Alternative Hypothesis H8.1:** There is difference of similarity between the test suites of TTR and jenny;
- **Null Hypothesis H9.0:** There is no difference of similarity between the test suites of TTR and IPO-TConfig;
- **Alternative Hypothesis H9.1:** There is difference of similarity between the test suites of TTR and IPO-TConfig;

- **Null Hypothesis H10.0:** There is no difference of similarity between the test suites of TTR and PICT;

- **Alternative Hypothesis H10.1:** There is difference of similarity between the test suites of TTR and PICT.

3.4 Variables

Regarding the variables involved in this experiment, we can highlight the *independent variables* and *dependent variables*. The first type are those that can be manipulated or controlled during the process of trial and define the causes of the hypotheses [2]. For this experiment, such variables are the algorithm/tool for generating combinatorial designs/MCA, the instances/samples used, and the programming language in which the algorithms were implemented: Java (TTR, IPOG-F, IPO-TConfig), C ++ (PICT), and C (jenny). For the *dependent variables*, we can observe the result of manipulation of the *independent variables* [2]. For this study, we identified the number of generated test cases, the time to generate each set of test cases, and the result of the similarity analysis between the test suites.

3.5 Experiment's Description

Each one of the algorithms/tools was subjected to each one of the 27 test instances (see Table 1), one at a time. The output of each algorithm/tool, with the generated test suite according to each instance, was directed to a text file to be recorded. An important point to be stressed is that, beyond the TTR itself, we implemented a version of IPOG-F. The other tools (PICT, jenny, and IPO-TConfig) were already implemented and ready for use.

To analyze the cost considering the size of test suites, we simply verified the amount of generated test cases, i.e. the number of rows of the matrix M, for each instance/sample.

For the second perspective of cost, it is necessary to consider that we were not able to measure the time in some tools because we do not have their source code. Therefore, we could only obtain the time to generate test cases considering the tool that implements our own algorithm, TTR, our implementation of IPOG-F, and jenny. To accomplish this time measurement, we instrumented each one of the tools and measured the computer's current time before and after the execution of each algorithm. In all cases, we used a computer with an Intel Core(TM) i7-4790 CPU @ 3.60 GHz processor, 8 GB of RAM, running Microsoft Windows 7 Professional 64-bit operating system. The goal of this second analysis is to provide an empirical evaluation of the time performance of the algorithms.

For the two cost measures, we used an appropriate statistical evaluation checking data normality. Verification of normality was done in three steps: (i) by using the Shapiro-Wilk test [20] with a significance level $\alpha = 0.05$; (ii) by checking the *skewness* of the frequency distribution; and (iii) by using a graphical verification by means of a Q-Q plot [10] and histogram. Thus, we believe we have greater confidence in this conclusion on data normality compared to an approach that is based only on the Shapiro-Wilk test, considering the effects of polarization due to the length of the samples. As we will discuss in Section 4, in all cases and considering all the 6 first hypotheses, data were not normally distributed. Therefore, we applied the nonparametric Wilcoxon test (Signed Rank) [10] with significance level $\alpha = 0.05$. However, if the samples presented *ties*, we applied a variation of the Wilcoxon test, the Asymptotic Wilcoxon (Signed Rank) [10], suitable to treat ties with significance level $\alpha = 0.05$.

With respect to the similarity analysis, taking the TTR as a basis, we have developed a program in python where we looked for the differences between the test suites generated by TTR and every other algorithm/tool. The similarity analysis method is presented in Algorithm 2.

Algorithm 2 The similarity analysis method

```
obtain set $X = \{x \mid x = |TS_i^{TTR}|\}$
calculate set $L = \{l \mid l = 0.95x, \quad x \in X\}$
obtain set $Y = \{y \mid y = |TS_i^{TTR} \cap TS_i^{other}|\}$
$d_{LX} \leftarrow calculateEuclideanDistance(n, L, X)$
$d_{YX} \leftarrow calculateEuclideanDistance(n, Y, X)$
if $d_{YX} \leq d_{LX}$ then
    Test suites $X$ and $Y$ are similar
else
    Test suites $X$ and $Y$ are NOT similar
end if
```

Each element $x \in X$ is the amount of test cases generated via TTR for each instance i. After obtaining the set X, we calculate set L: a set of ideal values where each element, $l \in L$, is 95% of a respective value $x \in X$. The set Y refers to the cases that are common to two different approaches. Thus, each element $y \in Y$ is the amount of test cases generated by TTR which are common to other solutions, for each instance i. We then calculate the Euclidean distance in an n-dimensional space (n = 27 in this study) considering L and X, d_{LX}, and considering Y and X, d_{YX}. We conclude that two test suites are similar if $d_{YX} \leq d_{LX}$. Hence, the set of ideal values, L, helps to determine the maximum euclidean distance to consider a set Y (test cases due to other solutions which are common to TTR) similar to the set X (test cases due to TTR).

3.6 Validity

In this section, we discuss some aspects related to the validity of the experiment. Regarding the validity of the conclusion, it is important to consider the reliability of the metrics. In this study, the quantities of test cases were obtained in an automatic way by the algorithms/tools, and it is possible to achieve the same results, in the case of replication of this experiment by other researchers. To minimize the impact of the input order of factors and levels and get more consistent results for the statistical analysis, we generated test cases with 3 variations in the order of input of factors and levels, and took into account the average of these 3 values for the statistical tests.

We expect that the validity of conclusion concerning the time for the generation of test cases is also the same in case of replication of this experiment. Again, we executed each tool 3 times and considered the mean values for the statistical tests. However, we do expect that a replication of this study will provide different results of time simply because such results depend on the computer configuration (processor, memory, operating system) used to run the tools. But, for this time perspective, the performance of the best solution (jenny) was much better than the second best approach (IPOG-F), and considerably better than our algorithm which was the weakest. Thus, we dot not expect a different validity of conclusion.

We expect the same validity of conclusion, in case of replication, for the similarity analysis. The bottom of line is that the similarity analysis is based on the amount of generated test cases and, hence, the same reasoning previously presented related to the cost in the perspective of the size of the test suites applies here. We also generated test cases three times for each instance and all tools, but as we would like to see the common test cases (TTR and other algorithm/tool) we have made a separate analysis, per execution, rather than considering an average value of the findings. For all 3 evaluations (cost/size, cost/time, similarity), 10 hypotheses were statistically assessed by means of hypothesis tests and verification of similarity via Euclidean distance.

Threats to internal validity compromise the confidence in stating that there is a relationship between dependent and independent variables. There were no factors that interfere in this relationship because the participants, i.e. the samples/instances, were randomly selected, there were no unanticipated events to interrupt the collection of metrics once started, and the generation of test cases strictly followed the algorithms implemented in the tools. Likewise, the validity of construction was also assured since traditional combinatorial designs algorithms/tools were used to compare with our solution, the TTR algorithm.

Finally, threats to external validity compromise the confidence in asserting that the results of the study can be generalized to and between individuals, settings, and under the temporal perspective. Basically, we can divide threats to external validity into two categories: threats to population and ecological threats.

Threats to population refer to how significant is the sample set of the population used in the study. For our study, the strengths and the range of factors and levels that make up the instances are the determining points for the characterization of this threat. Note that for such a study, the possibility of combination of strengths and factors/levels is literally infinite. However, unlike other informal studies [3] that focus more on pairwise testing (strength = 2), 70% of strengths we used are greater than 2, and there are instances with up to 20 factors, a factor which can take up to 11 levels (see instance 8 in Table 1). We believe that our choice of the set of samples is more significant than in other studies in the literature.

Ecological threats refer to the degree to which the results may be generalized between different configurations. Test interaction effects (e.g. carry out a pre-test with the participants of the experiment) and the Hawthorne effect (due to the participants simply feel stimulated by knowing that they are participating in an innovative experiment) are some of the types of this threat. The participants in our experiment are the test instances and therefore this type of threat does not apply to our case.

4. RESULTS AND DISCUSSION

4.1 Costs: Size of the Test Suites and Time

As we have already stated, the experiment reported in this work aimed to evaluate two aspects related to cost taken into account algorithms/tools TTR (Version 1.1), PICT, jenny, IPOG-F and IPO-TConfig: amount of test cases generated and time for generating test cases. Additionally, a similarity analysis was performed in order to check how similar were the sets of test cases created via TTR compared with other algorithms/tools.

Hypotheses (Section 3.3) 1 to 4 relate to the first evaluation of cost, the amount of test cases, while hypotheses 5 and 6 refer to the second perspective of cost, time to generate the test cases. It is important to emphasize that this is the time only to derive the suite test. Hypotheses 7 to 10 are related to the analysis of similarity between the sets of test cases.

Considering the cost related to the amount of test cases, as described in Section 3.5, data normality has been determined via 3 methods: via Shapiro-Wilk test with a significance level $\alpha = 0.05$, by checking the skewness, and by using a graphical verification by means of a Q-Q plot and histogram. None of the data related to the four hypotheses were normally distributed. Therefore, we applied the nonparametric test Asymptotic Wilcoxon (Signed Rank) [10] with significance level $\alpha = 0.05$. Table 2 shows the results and the total average value ($\bar{x}$) regarding the number of test cases generated by the five solutions. Due to the fact that all algorithms are sensitive to the input order of factors and levels, each instance was considered three times: one in the order shown in Table 2, other with the first and the last factors swapped, and another with the second and penultimate factors swapped. Thus, the values that appear due to each solution is an average of these three submissions. For example, in instance/sample 1, the average of the three runs for jenny was 71.67 while TTR's average was 63. Table 3 shows the p-values related to this assessment.

According to these results, we noticed that all four null hypotheses (H1.0 to H4.0) were rejected because the p-values are below 0.05. So there is difference in cost, considering the amount of generated test cases via the algorithms/tools. As the total average value ($\bar{x}$) of TTR is smaller than any other average value due to other solutions, we conclude that TTR is the best alternative for generating a lower cost test suite.

With respect to the cost in the perspective of the time to generate the set of test cases, we followed the same procedure already described to verify data normality. As in the previous case, data related to the hypotheses 5 e 6 did not also present normal distribution. Hence, again we have made use of the nonparametric test Asymptotic Wilcoxon (Signed Rank) [10] with significance level $\alpha = 0.05$. Table 4 shows the results and the total average value ($\bar{x}$) related to the time (ms) to generate the test suites. Because the issue of sensitiveness to the input order of factors and levels, we generated test cases three times, and the value that appears in each row of Table 4 is an average of these three submissions. Table 5 shows the p-values related to this assessment.

These results show again that the two null hypotheses (H5.0, H6.0) were rejected because the p-values are below 0.05. So there is difference in cost, considering the time to generate the set of test cases via the algorithms/tools. The average value of TTR is the largest of all values and, unlike the first cost evaluation, TTR presented the weakest performance in this second assessment. One possible explanation for the poorest TTR's performance is the programming language used: TTR was implemented in Java and jenny was developed in C. It is a fact that the latter is a much more appropriate language regarding real time aspects. But IPOG-F was also implemented in Java. The explanation for IPOG-F surpasses TTR is that IPOG-F makes comparisons in order to find the best local solution (it is a greedy algorithm) only to a particular test case. After that, what it

Table 2: Cost related to the size of the test suites: results and mean value

i	Strength	Instance	*jenny*	IPO-TConfig	PICT	IPOG-F	TTR
1	2	$6^1 7^1 2^1 4^1 3^1 7^1 4^1 9^1$	71.67	72.00	76.00	89.67	63
2	2	$2^3 3^1 4^1 2^2 3^1$	14.33	15.00	13.33	17.33	12.67
3	2	$9^1 8^1 2^1 4^1 5^1 6^1 7^1 2^1$	79.00	78.00	74.33	93.67	72.00
4	2	$3^2 2^1 4^1 5^1 6^1 9^1 2^1$	55.33	55.33	54.00	65.33	54.00
5	2	$2^2 3^2 4^2 5^2 6^2 7^2$	62.00	64.00	55.33	119.33	54.67
6	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2$	84.33	85.33	73.33	159.00	73.33
7	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2$	104.67	109.67	93.33	208.33	95.67
8	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2 10^2 11^2$	156.33	167.33	145.00	318.00	185.00
9	3	$2^1 5^1 2^1 3^2 2^3$	52.67	52.00	49.33	59.33	51.67
10	3	$2^3 3^1 4^1 2^2 3^1$	44.67	45.00	39.00	46.33	38.00
11	3	$2^1 5^1 6^1 2^2 3^1 2^2$	96.33	99.00	90.33	116.00	90.00
12	3	$3^2 2^1 4^1 2^4$	42.00	39.67	41.33	46.00	36.67
13	3	$2^2 3^2 4^2 5^2$	132.00	133.67	122.33	155.00	108.67
14	3	$2^2 3^2 4^2 5^2 6^2$	252.33	260.33	229.00	315.33	217.33
15	3	$2^2 3^2 4^2 5^2 6^2 7^2$	431.00	381.33	385.33	473.33	371.67
16	4	$5^1 3^1 2^2 3^1 2^1 4^1 3^1$	253.00	255.33	242.67	265.00	226.33
17	4	$2^3 3^1 4^1 2^2 3^1$	107.33	107.67	106.67	114.00	100.00
18	4	$2^1 5^1 2^1 4^1 5^1 3^1 4^1 2^1$	460.33	463.00	431.00	513.00	324.00
19	4	$3^2 2^1 4^1 5^1 6^1 2^2$	449.00	453.67	359.67	493.33	396.33
20	4	$2^2 3^2 4^2$	164.00	169.00	150.00	166.33	145.67
21	4	$2^2 3^2 4^2 5^2$	502.67	517.00	464.33	564.67	439.33
22	5	$2^3 4^1 3^1 2^1 4^1 3^1$	413,33	418,67	397.33	448.33	366.67
23	5	$2^3 3^1 4^1 2^2 3^1$	234.00	234.67	232.00	267.00	234.67
24	5	$3^1 4^1 2^1 4^1 5^2 2^2$	1328.00	1331.00	1257.33	1432.67	1225.00
25	5	$3^2 2^1 4^1 5^1 3^1 2^2$	645.33	635.00	613.00	637.00	602.33
26	5	$2^2 3^2 4^2 5^2$	1440.00	1440.00	1440.00	1440.00	1440.00
27	6	$2^2 3^2 4^2$	576.00	576.00	576.00	576.00	576.00
$\bar{x}$			305.62	305.88	289.31	340.72	281.51

Table 3: Cost related to the size of the test suites: *Asymptotic Wilcoxon*

Hipothesis	*p-value*
1: TTR ↔ IPOG-F	$1.229e - 05$
2: TTR ↔ *jenny*	0.0001011
3: TTR ↔ IPO-TConfig	0.0001019
4: TTR ↔ PICT	0.0244

does is to fill the remaining tuples, with no concern about the amount of tuples that were covered at each iteration. This causes the IPOG-F algorithm to be faster from a certain point on, however, it produces a greater amount of test cases, compared with TTR, due to this feature.

Despite this time analysis demonstrated a worse performance of our algorithm, it is important to note that in a software testing process, the time to run the test suites, which mainly depends on the size of the sets of test cases, is much more relevant than the time to generate the set of test cases. So the weakest TTR's time performance can be compensated by the fact that our solution generates smaller sets of test cases than all the other solutions analyzed.

4.2 Similarity Analysis

Following the method described in Section 3.5, we performed an analysis to realize whether the sets of test cases generated by version 1.1 of TTR were similar to the test suites generated by IPOG-F, IPO-TConfig, jenny, and PICT. We also conducted 3 runs of each algorithm/tool but as the main idea is to calculate the euclidean distance by comparing the number of test cases generated by TTR (set X) with the number of test cases generated by other solutions that are common to the ones created via TTR (set Y), we considered each run separately.

In Table 6 we show the results of the first exection only. TTR column is the set X and the other columns are sets Y. Hence, for the first instance, jenny, IPO-TConfig and IPOG-F have no test case in common to TTR, while 11 test cases generated via PICT have also been derived via TTR. We calculated the set of ideal values, L, and the euclidean distances (d_{LX}, d_{YX}), as presented in Algorithm 2.

Based on Table 7, it is clear that all null hypotheses from 7 to 10 (H7.0 the H10.0) are rejected. In other words, the test suites generated via TTR are not similar to any of the sets of test cases generated by the other approaches. This can be an indication that TTR tends to stimulate different parts of the SUT in relation to the other algorithms/tools.

5. RELATED WORK

According to the literature review we did, we could not find any study that presents a controlled experiment considering combinatorial designs. In this section, we present some relevant studies that consider controlled experiments.

In [11], the authors presented a 25-year historical perspective of evaluation studies related to software testing and which were published in Brazil, represented by SBES, and in the world, represented by ICSE. Main results showed that the SBES community has considerably increased the efforts

Table 4: Cost related to the time (ms) to generate the test suites: results and mean value

i	Strength	Instance	*jenny*	IPOG-F	TTR
1	2	$6^1 7^1 2^1 4^1 3^1 7^1 4^1 9^1$	0.074	0.163	0.115
2	2	$2^3 3^1 4^1 2^2 3^1$	0.026	0.067	0.080
3	2	$9^1 8^1 2^1 4^1 5^1 6^1 7^1 2^1$	0.090	0.150	0.173
4	2	$3^2 2^1 4^1 5^1 6^1 9^1 2^1$	0.075	0.077	0.130
5	2	$2^2 3^2 4^2 5^2 6^2 7^2$	0.099	0.163	0.991
6	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2$	0.136	0.302	4.400
7	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2$	0.203	0.595	18.999
8	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2 10^2 11^2$	0.313	2.430	1240.164
9	3	$2^1 5^1 2^1 3^2 2^3$	0.075	0.131	0.947
10	3	$2^3 3^1 4^1 2^2 3^1$	0.055	0.157	0.732
11	3	$2^1 5^1 6^1 2^2 3^1 2^2$	0.098	0.275	1.783
12	3	$3^2 2^1 4^1 2^4$	0.079	0.121	0.731
13	3	$2^2 3^2 4^2 5^2$	0.126	0.355	4.601
14	3	$2^2 3^2 4^2 5^2 6^2$	0.347	2.550	172.537
15	3	$2^2 3^2 4^2 5^2 6^2 7^2$	0.944	18.313	34664.835
16	4	$5^1 3^1 2^2 3^1 2^1 4^1 3^1$	0.250	1.507	56.274
17	4	$2^3 3^1 4^1 2^2 3^1$	0.118	0.484	12.128
18	4	$2^1 5^1 2^1 4^1 5^1 3^1 4^1 2^1$	0.498	3.301	143.801
19	4	$3^2 2^1 4^1 5^1 6^1 2^2$	0.476	4.843	145.860
20	4	$2^2 3^2 4^2$	0.114	0.155	0.463
21	4	$2^2 3^2 4^2 5^2$	0.554	3.824	207.121
22	5	$2^3 4^1 3^1 2^1 4^1 3^1$	0.438	3.073	116.216
23	5	$2^3 3^1 4^1 2^2 3^1$	0.232	1.394	42.661
24	5	$3^1 4^1 2^1 4^1 5^2 2^2$	1.601	87.788	1271.550
25	5	$3^2 2^1 4^1 5^1 3^1 2^2$	0.709	18.094	316.005
26	5	$2^2 3^2 4^2 5^2$	0,723	0.424	0.193
27	6	$2^2 3^2 4^2$	0,376	0.106	0.129
$\bar{x}$			0.327	5.587	1423.097

Table 5: Cost related to the time (ms) to generate the test suites: *Asymptotic Wilcoxon*

Hipothesis	*p-value*
5: TTR ↔ IPOG-F	$3.204e - 07$
6: TTR ↔ *jenny*	$5.215e - 07$

in performing rigorous evaluations. However, the authors stated that these efforts are still low compared with the number of studies published in ICSE and which present rigorous assessments. Moreover, they concluded that only a single paper, in each conference, presented a controlled experiment. Thse two studies are described below.

In [2], a controlled experiment comparing cost and difficulty of satisfaction of the mutation analysis criteria was presented considering the Procedural and Object-Oriented Programming Paradigms. The authors assessed 32 programs developed in C and Java. They used two tools, Proteum [5] for C code, and MuClipse [15] for Java code. The experiment was conducted based on the approach proposed in [22]. Results showed that not only the cost but also the difficulty of satisfaction of Procedural Paradigm are greater than the Object-Oriented Paradigm.

A controlled experiment comparing the "What You See is What You Test" (WYSIWYT) testing methodology with an Ad Hoc approach was presented in [16]. Results showed that the subjects that used their methodology, WYSIWYT, performed significantly more effective testing, as measured by du-adequacy, and were much more efficient testers, as measured by speed and redundancy, than the ones that followed the Ad Hoc approach. Moreover, subjects that followed WYSIWYT are less overconfindent that the Ad Hoc ones. These results are interesting because they may indicate that it is possible to achieve some benefits of formal notions of testing without formal training related to testing.

Based on the lack of controlled experiments published in these two relevant conferences, we believe that, with this research, we give an interesting contribution towards a wider use of planned experiments in the context of software testing.

6. CONCLUSIONS

In this work, we presented a controlled experiment for evaluating cost in terms of the amount of created test cases and the time to generate the test suites between our solution, the TTR algorithm, compared with four other algorithms/tools found in the literature: IPOG-F, IPO-TConfig, jenny, and PICT. Results showed that TTR was the best in terms of the amount of generated test cases (smaller test suites), but had the poorest performance in terms of the time to generate the test cases. However, the weakest performance of our algorithm can be offset by the fact that it generates smaller sets of test cases, which demands less time for running the test suites. We also conducted a similarity analysis where we concluded that the set of test cases generated via TTR are not similar to those created by the other solutions. Thus, the TTR tends to explore different behaviors in comparison with the other algorithms/tools.

Future directions include optimizing the TTR algorithm

Table 6: Similarity analysis: sets X and Y of the first execution

i	Strength	Instance	*jenny*	IPO-TConfig	IPOG-F	PICT	TTR
1	2	$6^1 7^1 2^1 4^1 3^1 7^1 4^1 9^1$	0	0	0	11	63
2	2	$2^3 3^1 4^1 2^2 3^1$	0	0	0	4	12
3	2	$9^1 8^1 2^1 4^1 5^1 6^1 7^1 2^1$	0	0	2	10	72
4	2	$3^2 2^1 4^1 5^1 6^1 9^1 2^1$	1	1	1	15	54
5	2	$2^2 3^2 4^2 5^2 6^2 7^2$	2	4	3	49	56
6	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2$	8	3	4	57	75
7	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2$	1	2	2	52	100
8	2	$2^2 3^2 4^2 5^2 6^2 7^2 8^2 9^2 10^2 11^2$	26	2	2	106	185
9	3	$2^1 5^1 2^1 3^2 2^3$	15	13	14	29	53
10	3	$2^3 3^1 4^1 2^2 3^1$	12	7	14	15	36
11	3	$2^1 5^1 6^1 2^2 3^1 2^2$	11	5	5	35	90
12	3	$3^2 2^1 4^1 2^4$	8	3	7	17	39
13	3	$2^2 3^2 4^2 5^2$	7	11	8	25	109
14	3	$2^2 3^2 4^2 5^2 6^2$	9	4	5	39	217
15	3	$2^2 3^2 4^2 5^2 6^2 7^2$	2	2	3	65	372
16	4	$5^1 3^1 2^2 3^1 2^1 4^1 3^1$	62	42	48	89	226
17	4	$2^3 3^1 4^1 2^2 3^1$	23	18	22	40	99
18	4	$2^1 5^1 2^1 4^1 5^1 3^1 4^1 2^1$	48	29	49	146	413
19	4	$3^2 2^1 4^1 5^1 6^1 2^2$	73	58	59	146	397
20	4	$2^2 3^2 4^2$	44	39	68	80	146
21	4	$2^2 3^2 4^2 5^2$	58	46	63	168	445
22	5	$2^3 4^1 3^1 2^1 4^1 3^1$	104	90	100	180	366
23	5	$2^3 3^1 4^1 2^2 3^1$	66	75	67	99	233
24	5	$3^1 4^1 2^1 4^1 5^2 2^2$	102	0	254	446	1228
25	5	$3^2 2^1 4^1 5^1 3^1 2^2$	129	148	170	255	601
26	5	$2^2 3^2 4^2 5^2$	1440	1440	1440	1440	1440
27	6	$2^2 3^2 4^2$	576	576	576	576	576

Table 7: Similarity analysis: euclidean distances. Caption: max = maximum allowed euclidean distance

	max	IPOG-F	*jenny*	IPO-TConfig	PICT
Execution 1	115.63	1598.15	1507.63	1084.56	1390.59
Execution 2	113.70	1411.37	1457.43	975.65	1332.85
Execution 3	115.57	1499.67	1441.89	1096.42	1444.73

to try to improve further their performance in terms of generating a smaller test suite. In addition, we will perform another controlled experiment or quasi-experiment, considering this new version of TTR, addressing not only the cost in terms of the size of the test suites but also the cost in relation to the amount of computer memory required by the algorithms and the time to really execute the test suites. In addition, we will also consider, within this new controlled experiment or quasi-experiment, the effectiveness of the set of test cases via mutation analysis [6] [7] [12] since effectiveness is a very important question to be answered.

7. ACKNOWLEDGMENTS

This work is supported via grant 415.563.888-61 - *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).*

8. REFERENCES

[1] J. M. Balera and V. A. Santiago Júnior. T-tuple reallocation: An algorithm to create mixed-level covering arrays to support software test case generation. In O. Gervasi, B. Murgante, S. Misra, M. L. Gavrilova, A. M. A. Rocha, C. Torre, D. Taniar, and B. O. Apduhan, editors, *Computational Science and Its Applications – ICCSA 2015*, volume 9158 of *Lecture Notes in Computer Science (LNCS)*, pages 503–517. Springer International Publishing, 2015.

[2] D. N. Campanha, S. R. Souza, and J. C. Maldonado. Mutation testing in procedural and object-oriented paradigms: An evaluation of data structure programs. In *Software Engineering (SBES), 2010 Brazilian Symposium on*, pages 90–99. IEEE, 2010.

[3] J. Czerwonka. Pairwise testing combinatorial test case generation. http://www.pairwise.org/tools.asp. Accessed: 2016-08-14.

[4] J. Czerwonka. Pairwise testing in the real world: Practical extensions to test-case generators. In *Proceedings 24th Pacific Northwest Software Quality Conference*, pages 285–294, Portland, 2006.

[5] M. E. Delamaro. *Proteum - um ambiente de teste baseado na análise de mutantes.* PhD thesis, ICMC/USP, São Carlos, 1993.

[6] M. E. Delamaro, J. C. Maldonado, and M. Jino. *Introdução ao teste de software.* Campus-Elsevier, 2007. 408 p.

[7] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints

on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, April 1978.

[8] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn. Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5):287–297, 2008.

[9] B. Jenkins. Jenny: A pairwise tool. http://burtleburtle.net/bob/math/jenny.html. Accessed: 2016-06-06.

[10] M. Kohl. *Introduction to statistical data analysis with R.* bookboon.com, London, 2015.

[11] O. A. L. Lemos, F. C. Ferrari, M. M. Eler, J. C. Maldonado, and P. C. Masieiro. Evaluation studies of software testing research in Brazil and in the world: A survey of two premier software engineering conferences. *The Journal of Systems and Software*, 86(4):951–969, 2013.

[12] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Jozala. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *IEEE Trans. Softw. Eng.*, 40(1):23–42, Jan. 2014.

[13] A. P. Mathur. *Foundations of software testing.* Dorling Kindersley (India), Pearson Education in South Asia, Delhi, India, 2008. 689 p.

[14] E. Matthes. *Curso Intensivo de Python.* novatec, Brazil, 2016.

[15] J. Offutt, Y.-S. Ma, and Y.-R. Kwon. The class-level mutants of mujava. In *Proceedings of the 2006 International Workshop on Automation of Software Test*, AST '06, pages 78–84, New York, NY, USA, 2006. ACM.

[16] K. J. Rothermel, C. R. Cook, M. M. Burnett, J. Schonfeld, T. R. G. Green, and G. Rothermel. Wysiwyt testing in the spreadsheet paradigm: an empirical evaluation. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 230–239, 2000.

[17] V. Santiago, W. P. Silva, and N. L. Vijaykumar. Shortening test case execution time for embedded software. In *Proceedings of the 2nd IEEE International Conference SSIRI*, pages 81–88, 2008.

[18] V. A. Santiago Júnior. *SOLIMVA: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications.* PhD thesis, Instituto Nacional de Pesquisas Espaciais (INPE), 2011. 264 p.

[19] V. A. Santiago Júnior and N. L. Vijaykumar. Generating model-based test cases from natural language requirements for space application software. *Software Quality Journal*, 20(1):77–143, 2012. DOI: 10.1007/s11219-011-9155-6.

[20] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, 1965.

[21] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Testing of Communicating Systems: Tools and Techniques, IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems (TestCom 2000), August 29 - September 1, 2000, Ottawa, Canada*, pages 59–74, 2000.

[22] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction.* Kluwer Academic Publishers, USA, 2000.

[23] C. Zannier, G. Melnik, and F. Maurer. On the success of empirical studies in the international conference on software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 341–350, New York, NY, USA, 2006. ACM.