Revista Cubana de Ciencias Informáticas
Vol. 16, No. 2, Abril-Junio, 2022
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 85-100

Tipo de artículo: Artículo original
Temática: Pruebas de software
Recibido: 02/02/2022 | Aceptado: 22/03/2022

# Impact of the significance of input values and paths on the effectiveness of the automatically generated test suite

Impacto de la significación de valores de entrada y caminos en la efectividad de LA suit de pruebas generadas de forma automática

Perla Beatriz Fernández Oliva[1] https://orcid.org/0000-0002-3360-4447
Martha Dunia Delgado Dapena [2] https://orcid.org/0000-0002-2601-3462
Alejandro Miguel Güemes Esperón [3] https://orcid.org/0000-0001-9704-9449
Heydi Margarita Henry Chibas[4] https://orcid.org/0000-0002-6255-4311
José Miguel Loor Intriago [5] https://orcid.org/0000-0002-4752-1765

[1] Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría", Cujae. Calle 114, No. 11901 entre 119 and 127, Marianao, Código Postal: 19390, La Habana, Cuba. perla@ceis.cujae.edu.cu
[2] Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría", Cujae. Calle 114, No. 11901 entre 119 and 127, Marianao, Código Postal: 19390, La Habana, Cuba. marta@ceis.cujae.edu.cu
[3] Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría", Cujae. Calle 114, No. 11901 entre 119 and 127, Marianao, Código Postal: 19390, La Habana, Cuba. aguemes@tesla.cujae.edu.cu
[4] Facultad de Ingeniería Informática, Universidad Tecnológica de La Habana "José Antonio Echeverría", Cujae. Calle 114, No. 11901 entre 119 and 127, Marianao, Código Postal: 19390, La Habana, Cuba. hhenri@ceis.cujae.edu.cu

[5] Universidad Técnica de Manabí. Avenida Urbina y Portoviejo, Código Postal: 130105, Manabí, Ecuador. Jmloor@utm.edu.ec

*Autor para la correspondencia (perla@ceis.cujae.edu.cu)

## ABSTRACT

Software tests are fundamental in the reliability and quality of systems, contributing to their positioning in the market. Generating test data is a critical task, as extensive testing is costly in time and effort. An adequate design of the test cases, which contemplates a selection of adequate values, can detect a high number of defects. There are proposals for the automatic generation of test suites using metaheuristic algorithms with the fundamental objective of reducing the time associated with this process, facilitating its execution by developers and testers, and achieving high levels of coverage. However, there is no emphasis on effectiveness in detecting errors. This work contemplates an analysis of the impact of the significance in the input values and in the paths or scenarios for the detection of suit errors generated using metaheuristic algorithms. Proposals for the automatic generation of test suits by different authors are addressed and experiments based on two case studies are carried out. The results are compared between including or not in this algorithm the significance in the input values and in the paths or scenarios.

**Keywords:** software quality; software testing; lean test suites; metaheuristic algorithms.

## RESUMEN

Las pruebas de software son fundamentales en la confiabilidad y calidad de los sistemas, contribuyendo a su posicionamiento en el mercado. La generación de datos de prueba es una tarea decisiva, puesto que la realización de una prueba exhaustiva es costosa en tiempo y esfuerzo. Un diseño adecuado de los casos de prueba, que contemple una selección de valores adecuados, puede detectar un elevado número de defectos. Existen propuestas para la generación automática de suits de pruebas utilizando algoritmos metaheurísticos con el objetivo fundamental de disminuir los tiempos asociados a este proceso, facilitar su ejecución por parte de desarrolladores y probadores y lograr altos niveles de cobertura. Sin embargo, no se hace énfasis en la efectividad para detectar errores. Este trabajo contempla un análisis del impacto de la significación en los valores de entrada y en los caminos o escenarios para la detección de errores de suit generadas utilizando algoritmos metaheurísticos. Se abordan propuestas para la generación automática de suits de pruebas por diferentes autores y se realizan experimentos basados en dos casos de estudio. Se comparan los resultados entre incluir o no en este algoritmo la significación en los valores de entrada y en los caminos o escenarios.

**Palabras Clave:** calidad de software; pruebas de software; suites de pruebas reducidas; algoritmos metaheurísticos.

## Introduction

The effectiveness of the test cases is measured according to the number of errors that they managed to detect, for this, in the classic design of the test cases, techniques are proposed from the Software Engineering disciplines that are aimed at generating test cases that use different coverage criteria and that start from not testing all possible values, but rather a significant set of them that allows detecting the greatest possible number of errors in the operation of the software. These values that can be obtained from applying techniques such as equivalence partitions and limit values for the case of black box tests and the techniques of conditions and loops for white box tests help to obtain more effective test cases in the error detection if they are combined with coverage criteria of scenarios and paths respectively.

Software engineers often face concerns and imprecision of requirements. Perfect solutions are often impossible and the nature of the problems sometimes makes analytical algorithm definitions problematic. Like other engineering disciplines, software engineering is often concerned with near-optimal solutions or those that fall within a specified acceptable tolerance. It is precisely these factors that make search-based optimization techniques robust and easily applicable. Metaheuristic algorithms such as genetic algorithms (GA), simulated annealing and taboo search have been successfully applied to a number of engineering problems (Harman, 2001), with the aim of solving the combinatorial explosion of automatic test suit generation.

The proposals that address the issue of automatic test case generation focus on the generation of test data and paths that satisfy a given coverage criterion (Larrosa, 2019). Among the proposals studied is (Khan, 2019) (Khari, 2019) (Marino, 2019) (Sheoran, 2019), all use metaheuristic techniques to generate combinations of test values. It has been shown that the adoption of search-based algorithms in software testing research is one of the alternatives to deduce and generate test cases in an adequate and optimal way to propose tests and solve particularly issues related to combinatorial problems (Khari, 2019a) (Musa, 2019).

There are proposals for the automatic generation of paths and test values using metaheuristic algorithms with the fundamental objective of reducing the times associated with this process; facilitate its execution by developers and testers and; achieve high levels of coverage (Khari, 2019) (Khari, 2019a) (Sheoran, 2019) (Khan, 2019). However, there is no emphasis on effectiveness in detecting errors.

The hypothesis of this research is that if the significance of the input values and the paths or scenarios is taken into account, reduced test suites with a greater error detection capacity can be obtained, that is, more effective test suits.

The scope of this research includes an analysis of the impact of the significance of input values and the paths or scenarios in the detection of suit errors generated using metaheuristic algorithms.

The proposals for the automatic generation of test suits by different authors are addressed in this work and experiments are carried out by implementing a heuristic algorithm for the automatic generation of test suits. The results are compared between including or not in this algorithm the significance of input values and paths or scenarios.

## Related work

One of the techniques used to measure how effective the test case is the error-based technique: mutant analysis criterion, which allows estimating the effectiveness of each test case, based on the number of mutants killed during the test. The execution of the test case for each of the mutants (Khan, 2017).

It is a criterion that uses a set of slightly modified programs (mutants) obtained from a given program. The goal is to find a set of test cases capable of revealing the behavioral differences between the program and its mutants. When the mutant behaves differently from the program, it is said to be a dead mutant. Otherwise, if for the entire test set the behavior of the mutant is the same as that of the program, then the mutant is said to be a living mutant, and it must be enhanced to kill the living mutant. The mutation operators are the rules that define the alterations that must be applied in the original program and thus generate the mutants. The operators represent an implementation of a defect model and by using them it validates whether or not the program contains the type of error modeled by them.

An important point that should be highlighted is the objective measurement that the criterion yields and that is related to the adequacy of the test cases used to validate the programs. The metric used is the Mutation Score, which relates the number of mutants killed to the number of mutants spawned. According to (Khan, 2017), equation (1) is calculated as shown:

$$MS(\%) = \frac{dm}{m} * 100 \qquad (1)$$

Where:

MS: Mutation Score

dm: number of mutants killed by the test cases in the program

m: total number of mutants generated in the program

The higher the rating, the more suitable the set of test cases for the program being tested. The formula depends both on test cases that are capable of killing mutants, and on the ability to recognize equivalent mutants.

This way you can measure how effective the test cases are. If actual system failures are perceived as non-existent, then this approach is really no better than any random testing technique. However, if analysis and design models can provide insight into what is likely to go wrong, then they can find a significant number of errors with less effort (Pressman, 2010).

Mutation testing allows a more stable and reliable system to be obtained, high source program coverage is achieved, program mutants are thoroughly tested, software quality improves, and gaps in test data can be identified. However, complex mutations are difficult, time consuming and expensive to implement, testers must have programming skills to perform mutation testing, and automation is necessary for mutation testing as it is time consuming. The works listed below use optimization algorithms in different ways to solve the problem of combinatorial explosion in the automated generation of test suites.

In (Galeotti, 2017) the author adapts the EvoSuite tool (a tool that generates test cases for classes written in the Java language), using a novel approach in which he optimizes entire sets of test cases (test suit) that satisfy one or more criteria of coverage so that it takes into account the Enabling Preserving Abstractions (for its acronym in English EPA) associated with the component when generating test cases for it. The operation of EvoSuite is based on the genetic algorithm that, as part of the evaluation of the fitness of individuals (sets of test cases), makes use of fitness functions. These functions are determined by the line and branch coverage criteria, in addition the author in this work incorporated new criteria: EPATRANSITION and EPAERROR that makes use of the valid and invalid EPA transitions respectively. Experiments were carried out with the intention of testing the prototype, where it is shown that it is more effective than the original tool when generating test sets with a high coverage of EPA, making use of the mutation technique for the detection of faults; in some cases, the difference was almost 70%. He further shows that the prototype can be used effectively to detect differences between the code and its associated EPA for the subjects involved in the study. The tool not only allows detecting that there are differences, but also builds a test case in which it describes how to execute the invalid transition while showing what transition it is (Galeotti, 2017).

The author in (Kumar, 2016) analyzes the algorithms of artificial bee colony or Artificial Bee Colony in English (ABC), particle swarm or Particle Swarm Optimization Algorithm (PSO) and search in harmony or Harmony Search (HS) generating and optimizing cases of tests for an operation in ATMs or Bank ATM. The test cases generated using the HS algorithm are compared with the test cases generated by PSOA and ABC and it was found that ABC produces optimal results in less time and with more accuracy.

In (Pandey, 2017) he proposes an algorithm based on the behavior of the Firefly Algorithm (FA) for the optimization of the test suite. The experiments are carried out with a reference program and the results of the simulation with the proposed algorithm are compared with the ABC, the Ant Colony Optimization Algorithm (ACO) and the GA, showing that it surpasses them in terms of branch coverage in software testing.

According to (Millar, 2013) he presents a family of functional test case prioritization techniques that use the dependency information of a test suite to prioritize the suit, which is called dependency structure prioritization. Since these dependencies mirror the dependencies of the system itself, it is proposed that ordering test runs based on the complexity of the interactions can increase the failure detection rate, compared to arbitrary test orders. They present experimental results on six software systems built for industrial use and compared the technique with the raw and ordered test defined by test engineers with random prioritization, two coarse-grained techniques based on feature coverage, and greedy prioritization using information about known faults. The results indicate that the technique proposed in this work outperforms the engineering, random prioritization and coarse-grained techniques, but not greedy prioritization. This indicates that dependency structure prioritization is a promising approach to improve failure detection rate.

In accordance with (Puablo, 2017), the author presents a new strategy belonging to the group of approximate techniques that allows dynamically generating the appropriate test data set to test a program. The strategy used is based on knowledge of information from the structure of the program it is dealing with, to guide the search for new input data. The generator uses, to find the test cases that cover the final objective of the test, the Algorithm Based on Particle Swarm Optimization (PSO). As a suitability criterion, which is nothing more than the criterion that will be taken into account when determining how complete the test has been. Condition coverage was used, which requires that all simple conditions in a program take the two logical values: true and false. In the experimentation stage with the solution proposed in (Puablo, 2017), it was found that the proposal improves existing solutions, increasing the level of coverage and lowering the amount of process necessary to do so, but when the problem has a greater structural complexity and the coverage of the conditions is a finer search task, the PSO algorithm can exploit its full potential and this can be verified in the programs that served as an experiment, where the random utilization rate barely reaches 10%. A comparison was made with the algorithms used in the field of automatic generation.

The study of (Khari, 2019a) focuses on the behavior of the evaluation of metaheuristic algorithms, particularly: the Hill Climbing algorithm (HC), PSO, FA, the Cuckoo search algorithm (CS), Bat Algorithm (BA) and the ABC algorithm using its standard implementation to optimize the path coverage and branch coverage produced by the test data, whose objective is to find the most suitable algorithm to narrow future research in the field of test automation for the optimization-based path coverage approach. Each algorithm was implemented to generate automated test suites based on a program under test and indicates that ABC, BA and PSO generate more optimal test suites, while CS, HC and FA produce non-optimal test suites. On the other hand, BA, HC and ABC are the fastest algorithms with similar processing time; FA, PSO and CS are the slowest algorithms. Therefore, ABC and BA are presented as the most suitable algorithms for test suite generation, while PSO may be perfected in the future for this problem.

The work (Khan, 2017) proposes a hybrid algorithm, the hybrid genetic algorithm for the generation of test cases using test data flow approaching it to the mutation test. Use mutation tests with path tests. They designed a tool in the C# programming language to automatically generate a control flow graph (CFG) using a program in the C programming language to generate prime numbers between two numbers. All paths were found and some mutants were injected into this program loop and were modified. The main job of mutants is to identify the mutation score. 10 test suites of different sizes were generated, path tests and mutation analysis were applied, taking three different methods, and the full mutation score was calculated. A comparison was made with the random method, the genetic algorithm and the hybrid genetic algorithm, concluding that the hybrid algorithm is better in the search for mutants (Khan, 2017).

The study (Khan, 2019) proposes the generation of automatic test cases with the help of the genetic algorithm to reduce the data entry time. It proposes a method in which the work is divided into two parts: the tool for the generation of CFG and the automatic generation of test cases, thus proposing the algorithm for the generation of test cases that covers the maximum number of paths, as well as the parameters of the genetic algorithm. All the paths have been found with the help of a program in the C programming language. The proposed method with the genetic algorithm covers 100% of the paths with less number of iterations, but for a small program.

In the work of (Varshney, 2018) he proposed a hybrid algorithm adapting Particle Swarm Optimization (PSO) and Differential Evolution in English (DE) to generate test data for the data flow dependencies of a program with a search strategy of neighborhood and thus improve the search ability of hybrid algorithms. The fitness function is based on the concept of dominance relations and the distance of the branches. The measures considered are: mean number of generations and mean percentage coverage. The function of the hybrid algorithm is compared with DE, PSO, GA and Random Search. Over several experiments on a set of

benchmark programs, it is shown that the executed hybrid algorithm is significantly better than DE, PSO, GA and Random Search in generating data from data flow tests relative to the collected measurements.
Some proposals only take into account the execution time and the size of the suit and do not analyze which errors are detected, an important aspect to take into account to contribute to the quality of the project software under test. The works that make use of the black box prioritization technique and the white box mutation technique focus on the detection of errors that can serve as a guide to carry out an analysis of the effectiveness of test cases in software projects. However, the latter implies that it is expensive at runtime.

## Materials and Methods

To analyze the impact that the meaning of the input values and of the paths or scenarios can have in these heuristic algorithms, a set of Java software components were developed that use the BICIAN heuristic algorithm library (Diaz, 2015) (Larrosa, 2019). The GeVaUExt component is in charge of configuring the operators of the BICIAN library and defining the test suite generation problem, in order to obtain reduced test suites, including in the objective function the criteria of significance of input values and of pathways or scenes.

In order for the GeVaUExt component to generate the test cases taking into account the meaning of the values and the meaning of the paths or scenarios, the IGenerarValores interface must be implemented in order to be able to redefine the methods necessary for its correct operation. For this, it was necessary to implement five methods whose fundamental characteristics are presented below:

1. Method to obtain a random value of a specific variable (getVariableRandom) in the MIMiCodificacion class, since the values of the variables are of the SignificantValue type in which the real value of the variable and its meaning are stored.
2. Method to generate the initial state (generateRandomState) of the MiM1P in such a way that the state is a test suit, and not a test case as it was implemented in (Khan, 2019).

3. Method to generate a new state from another state (generateNewState) of the MiM1P class in which a new test case is created to add it to the corresponding path of the test suite until it reaches the size specified by the user having taken into account the variability of the values, that is, the values of the variables of the combinations that cover a path must vary, for example, if a path of the test suite contains the test case (1, 2 , 3) and it is created the test case (1, 2 , 4), this is exchanged for the other so that similar combinations do not exist. When the test suite meets the size then the test cases with lower heuristics are eliminated and new combinations are added.
4. Method to evaluate the generated state (Evaluation) of the MyObjectFunction class, in which it evaluates the best test suit taking into account the significance of the values and paths.
5. Method to configure the problem (configProblem) of the Tester class, which is in charge of calling the previous classes to be able to execute the metaheuristic algorithm.

To assess whether the significance of the values and paths or scenarios impacts the effectiveness of the test suits obtained, a case study was designed to measure the effectiveness of the implemented algorithms and the quality of the test set. For the case study, several suites are generated with GeVaUExt and the inclusion of the most significant values and paths is analyzed.

## Case study: Buy online

The case study is based on the fact that several customers can make purchases in online stores where they must register on the platform and then place several orders according to the quantity that exists in the store. For this case study, two functionalities were chosen. The description of each of the functionalities is as follows:

1. Reserve product request quantity where the user can successfully or not reserve a requested quantity depending on the actual quantity in the store.
2. Report the products in stock limits according to the minimum and maximum quantities.

The system generates significant values for each value of the variables applying test case design techniques depending on the equivalence class according to the domain it belongs to, some generated values belong to the numerical domain and others are string type. The variables and values with their significance were determined, as well as the techniques of equivalence partitions and limit values.

7 and 14 conditionals were defined by the user, respectively, to describe the scenarios of the two functionalities under analysis in study case. From there, the scenarios and the matrix corresponding to the truth values of the conditions in each were generated functionality. For the first 7 paths and for the second 12.
The case study has the following objective:

- Determine the effectiveness of the implemented algorithms to select the test cases with the most significant input values. For this, the following question is proposed.

Do the generated test suites contain the test cases with the most significant values?

## Results and Discussion

Five test suites were generated in five executions and with 2000 iterations, of the Hill Climber algorithm, for both functionalities using the implementations. Hill Climber was chosen for this first experimentation since it is a basic algorithm, easy to understand and most of the articles studied select this algorithm to obtain their results.

To answer the first question of the buy online case study: do the generated test suites contain the test cases with the most significant values?

The significant values of the paths were fixed with a value equal to 1.0 for both functionalities. Figure 1 shows in a graph the number of times that the values of the variables are repeated in each of the suits generated for the first functionality of study case.

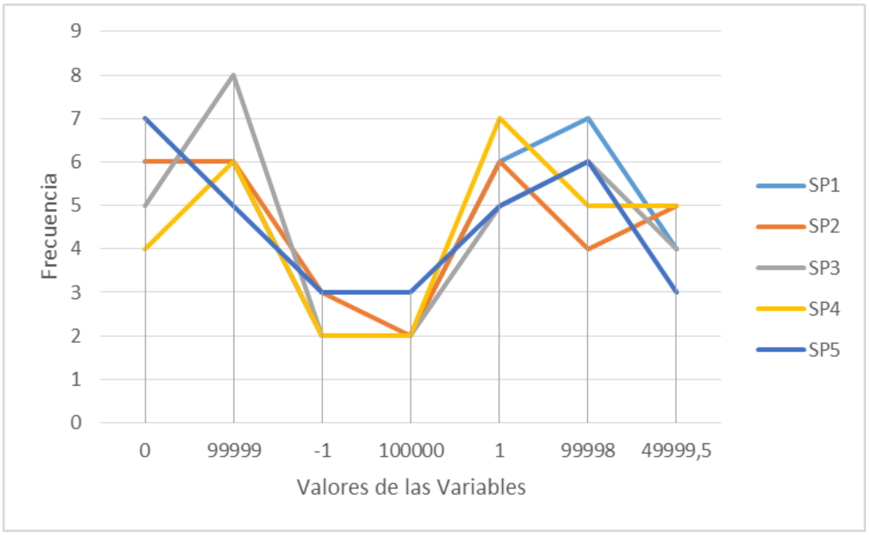

**Fig. 1-** Number of times the values are repeated in each test case to reserve the number of values by setting a single significance value of the paths.

In this case, the values of greatest significance are 0 and 99999 with significance 1.0 for both variables that have a numerical domain. These values are contained in each of the test suites, having a higher frequency in most of the suits, being very useful for error detection. The values 99998 and 1 tend to be repeated more in the first and fourth suit respectively, their significance is high with a value of 0.8 so it can also detect errors. The value 49999.5, which has less significance, tends to have a higher frequency than -1 and 100,000 since both values can only be contained in test cases that cover the first four paths or scenarios because they are outside the range of 0 and 99999 so the significant value of these values is not the correct one to obtain the test suite.

Figure 2 shows the graph with the number of cases of times that the values of the variables are repeated in each test case for the functionality of reporting the products in limits of existence.

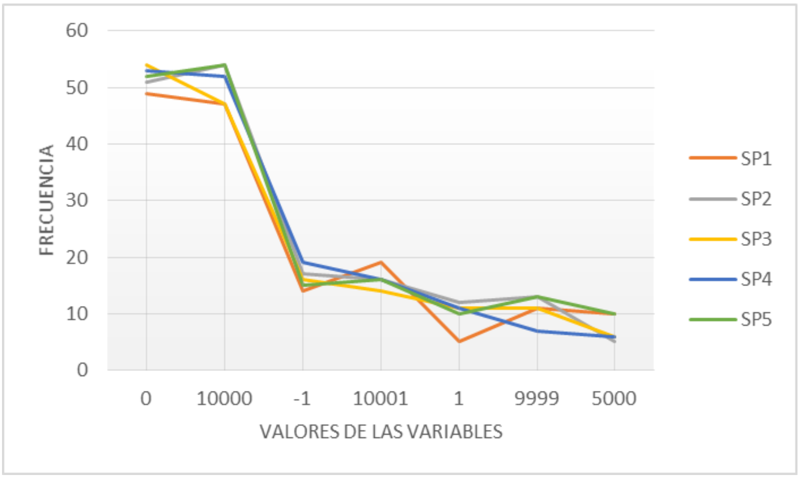

**Fig.2-** Number of times that the values are repeated to report the products in limits of existence, setting a single significance value of the paths.

The graph shows the values of the variables with numerical domain, where the 0 and 10,000 that have the highest significant value equal to 1.0 are contained in the test cases of each of the test suits, with a greater number of repetitions. While the values of the string domain variable with the significant value equal to 1.0 are also contained in the test cases more frequently than the catalog value with significance equal to 0.7.

## Conclusions

With the analysis carried out, it can be concluded that the inclusion of the criteria of significance of the input values and scenarios or paths combined with the use of the Hill Climber algorithm, directly impacts the automatic generation of more effective reduced test suits in detection of mistakes. However, this work can be perfected for improvement, so among the recommendations, the realization of experiments with other heuristic algorithms and with international library projects should be taken into account.

## References

Díaz, R., et al., Especificaciones Funcionales de la biblioteca BiCIAM. Universidad Tecnológica de La Habana, Facultad de Ingeniería Informática, 2015.

Galeotti, J.P., Generación Automática de Casos de Test para EPAs: Un enfoque basado en Algoritmos Genéticos, in Departamento de Computación. 2017, Universidad de Buenos Aires. p. 50.

Harman, M. and B. Jones, Search-based software engineering. Information and Software Technology, 2001.

Khan, R., A. Srivstava, and M. Amjad, Generation of Automatic Test Cases with Mutation Analysis and Hybrid Genetic Algorithm. 2017, IEEE.

Khan, R. and A. Kumar, Automatic Software Testing Framework for All defuse with Genetic Algorithm. 2019. 8(8).

Khari, M., Empirical Evaluation of Automated Test Suite Generation and Optimization. Arabian Journal for Science and Engineering, 2019.

Khari, M., et al., Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization. Soft Computing, 2019.

Kumar, R., et al., Automates Test Case Generation and Optimization: A Comparative Review. 2016: p. 14.

Larrosa, D., Extensiones a MTest.Search para la generación automática de pruebas unitarias en diferentes lenguajes, in Facultad de Informática. 2019, Universidad Tecnológica de La Habana "José Antonio Echeverría" CUJAE.

Marino, J. and V.A.d. Santiago, A systematic mapping addressing Hyper-Heuristics within Search-based Software Testing. 2019: p. 14.

Millar, T., Using Dependency Structures for Prioritization of Functional Test Suites. IEEE Transactions on Software Engineering, 2013.

Musa, J.a., R. Romli, and N. Yusoff, An Automatic on the Applicability of Meta-Heuristic Searching Techniques for Automated Test Data Generation in Automatic Programming Assessment. Baghdad Science 2019. 16.

Pandey, A. and S. Banerjee, Test Suite Optimization Using Chaotic Firefly Algorithm. International Journal of Applied Metaheuristic Computing, 2017. 8(4).

Pressman, R.S., Ingeniería de software. Un enfoque práctico. Séptima ed. 2010, México.

Puablo, J., Generación dinámica de casos de pruebas utilizando metaheurísticas, en la Facultad de Informática de La Plata. 2017, Universidad Nacional en Sistemas. p. 84.

Sheoran, S., N. Mittal, and A. Gelbukh, Artificial Bee Colony Algorithm in data flow testing optimal test suite generation. 2019.

Varshney, S. and M. Mehrotra, A Hybrid Particle Swarm Optimization and Differential Evolution based Test Data Generation Algorithm for Data-Flow Coverage Using Neighbourhood Search., in Informatics 42. 2018.

## Conflicto de interés

Los autores autorizan la distribución y uso de su artículo.

## Contribuciones de los autores

1. Conceptualización: Perla Beatriz Fernández Oliva y Martha Dunia Delgado Dapena.
2. Curación de datos: Heydi Margarita Henry Chibas.
3. Análisis formal: Perla Beatriz Fernández Oliva y Alejandro Miguel Güemes Esperón.
4. Adquisición de fondos: Martha Dunia Delgado Dapena.
5. Investigación: Perla Beatriz Fernández Oliva.
6. Metodología: Perla Beatriz Fernández Oliva y José Miguel Loor Intriago.
7. Administración del proyecto: Martha Dunia Delgado Dapena.
8. Recursos: Martha Dunia Delgado Dapena
9. Software: Alejandro Miguel Güemes Esperón y Heydi Margarita Henry Chibas
10. Supervisión: Martha Dunia Delgado Dapena.

11. Validación: Heydi Margarita Henry Chibas y Alejandro Miguel Güemes Esperón.
12. Visualización: Alejandro Miguel Güemes Esperón y José Miguel Loor Intriago.
13. Redacción – borrador original: Perla Beatriz Fernández Oliva.
14. Redacción – revisión y edición: Martha Dunia Delgado Dapena y Perla Beatriz Fernández Oliva

**Financiación**

Universidad Tecnológica de La Habana "José Antonio Echeverría", Cujae.