www.editada.org

International Journal of Combinatorial Optimization Problems and Informatics, 14(3), Sep-Dec 2023, 91-102. ISSN: 2007-1558. https://doi.org/10.61467/2007.1558.2023.v14i3.409

Non-bio-inspired Metaheuristics in Software Testing: A Systematic Literature Review

Ángel Juan Sánchez-García[1], Alfredo Delgado-Santiago[1], Marcela Quiroz-Castellanos[2], Xavier Limón[1] and Rocío Erandi Barrientos-Martínez[2]*

[1]Facultad de Estadística e Informática, Universidad Veracruzana, Xalapa, Veracruz, México.
[2] Instituto de Investigaciones en Inteligencia Artificial, Universidad Veracruzana, Xalapa, Veracruz, México.
dltunasd@gmail.com, {angesanchez, mquiroz, hlimon, rbarrientos}@uv.mx

Abstract. The software testing phase usually consumes a lot of the development of software projects time in order to find defects before release. Different strategies have been approached to optimize this phase of the testing stage. Metaheuristics are important in software testing due to their ability to find optimal or near-optimal solutions in complex situations. This research aims to analyze the current status of the application of metaheuristics that assist in software testing phase activities, specifically the most representative Non-bio-inspired algorithms (NBA) are surveyed, being Hill Climbing the most reported. The main activities of the software testing where NBA were implemented, were test case and test data generation and test case prioritization. It was concluded that NBAs used on their own are only viable in some activities of the software testing phase. As future work, it is proposed to investigate the use of hybrid algorithms and approaches in software testing phase.

Keywords: Metaheuristic, Software Testing, optimization, Systematic Literature Review

Article Info
Received 11 Sep, 2023
Accepted 11 Dec, 2023

1 Introduction

The need for software systems to be free of defects is increasing. To ensure the quality of the Software, a transcendental phase is the testing phase. Software testing is essential to ensure that the software meets the quality, reliability, and security requirements set by the customer (Rosenblatt, 1957). Testing also helps detect bugs and defects in software (Moshe et al.,1993), before it is released, reducing the risks and costs of software bugs. During the software development process, due to the time and cost constraints, it is not possible to test manually the software and fix the defects (Caruana, Lawrence, Giles, 2000).

As the years go by, the systems developed in software projects become more complex, and, consequently, so do the tests. In the testing phase, optimization algorithms have been used to generate test cases or identify defects. However, most of the strategies used are based on evolutionary algorithms or bio-inspired algorithms. There are studies on the use of metaheuristics at the software testing stage (Ritter, Iancu, Urcidet, 2003), approaches mainly inspired by nature. The approaches in these studies are commonly divided into evolutionary algorithms such as Genetic Algorithms (GA) (Ritter, & Sussner, 1996; Ritter, & Urcid, 2003) among others, as well as collective intelligence such as Particle Swarm Optimization (PSO) (Ritter, & Urcid, 2007; Sussner, 1998), Artificial Bee Colony (ABC) (Sussner & Campiotti, 2020) and Whale optimization (Sussner & Esmi, 2011; Sussner & Esmi, 2009) among others. However, although these studies show good results in Software Testing activities, there are several other alternatives which are not inspired by nature, but rather by physical phenomena and that could contribute to this field.

This paper is organized as follows: Section 2 describes background and related work. Section 3 details the method used to execute this Systematic Literature Review. In Section 4, the results obtained from this work are presented. In Section 5, a discussion of results is presented. Finally, Section 6 draws the main conclusions and proposes future work.

2 Related Work

Artificial Intelligence is a discipline that has supported each of the phases of software development, such as requirements, design, coding, testing and maintenance. Recently in Araújo, Oliveira, and Meira (2017), the authors carry out a Systematic Mapping Study focused on software testing, where the authors conclude the trend of the use of Artificial Intelligence in many of the problems of this Software phase.

In a manual search of related work, a Systematic Literature Review (SLR) on Search-Based Software Testing was found (Araújo, 2012). However, this study focused specifically on the generation of test cases, concluding that metaheuristics algorithms are indeed promising for solving a wide variety of test case generation problems (Araújo, 2012).

Another study Araújo & Sussner (2010) covers a range of studies from 2006 to 2017 on the application of Genetic Algorithms in the software testing stage. However, this review only focuses on one type of evolutionary algorithm and one software testing activity of the wide variety that exists in this stage, test case prioritization. As it can be seen, it is a very specific review, however, it contributes in the way that metaheuristics help software engineers and testers in this activity.

On the other hand, another study was found (Pessoa & Maragos, 2000), which covers a range of studies published between 2003 and 2016. It also focuses on Genetic Algorithms, but they are applied in another software testing activity, generation of test data.

In Araújo & Sussner (2010), authors present a systematic literature review of the Bio-inspired (and meta-heuristic) used in a particular area of software testing: Software Fault Prediction (SFP). They cover a range from 2007 to 2019 because they comment that in the 2000s, there is a rise in the application of search algorithms in software engineering. From the 34 selected studies, the authors found that the most used bioinspired metaheuristics in SFP are: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Bat Search (BS), Genetic Programming (GP) among others.

In Gómez-Flores & Sossa (2020), the authors aim to cover the last 40 years in software testing trends where there is a growing interest in the optimization and improvement of software testing. The authors classify the trends into methods such as: Test Generation, Empirical Evaluation, Fault Localization, Regression Testing, Mutation Testing, Program Analysis, Bug Reporting, Algorithm Optimization, Event Tracing, and Product Line Inspection, where it is seen that an important area is the use of optimization algorithms in software testing.

The scope of the systematic review reported in Arce, Zamora, and Sossa (2017) was test case selection in regression testing. Specifically, the authors looked for nature-inspired approaches that would help with the test case selection problem, where algorithms based on evolutionary algorithms, collective intelligence, and non-bioinspired algorithms were identified.

Another recent review focused on regression test case selection (TCS) is presented in Hernández, Zamora, Sossa (2018), in which test case prioritization (TCP) was also addressed. In this work that covers recent years (2018 - 2022), it is reported that for TCS the most reported approaches are based on multi-objective search, models and machine learning, where search algorithms are included again (bioinspired and non-bioinspired in TCS).

Listed below are some conclusions based on the results listed in Table 1 that justify performing an SLR in software testing based on non-bioinspired metaheuristics.

1. There are some SLRs in the area of software testing that address approaches to assist in this area but are not focused only on specific optimization algorithms.
2. Identified SLRs that address search problems in software testing have only focused on bio-inspired metaheuristics, such as evolutionary algorithms (usually genetic algorithms) and collective intelligence.
3. As for testing, SLRs focus on specific activities such as prioritization or selection of test cases. but they do not address all the activities of the testing phase.

Table 1. Related work summary.

Ref	Year	Coverage	Target Search	Applied area	Number of primary studies
Araújo, Oliveira, Meira	2022	2010 – 2022	Type of software test	Software testing	171
Araújo	2009	2007 – 2009	Search-based test generation	Software testing	68
Araújo	2019	2006 – 2018	Genetic algorithms	Test case priorization	20
Pessoa, & Maragos	2018	2003 – 2016	Genetic algorithms	Test data generation	68
Zamora, & Sossa	2020	2007 – 2019	Bio-inspired algorithm	Software fault prediction	34
Sossa & Guevara	2022	1980 – 2019	Software testing strategies	Software testing	14,684
Arce, Zamora, Fócil-Arias, & Sossa	2023	2007 – 2021	Nature inspired approaches	Test case selection	20
Arce, Zamora, Sossa, & Barrón	2023	2018 – 2022	Any technique	Test case priorization and selection	35

As can be seen in this section, although the use of metaheuristics in software testing seems very promising, even in the area of Search-Based Software Engineering, no review was found that focused on non-bio-inspired metaheuristics. Furthermore, the SLRs found are focused on a type of algorithm or on a specific test activity (generation of test cases or prioritization of test cases). Therefore, the purpose of this SLR is to complement the identified Systematic reviews with metaheuristics that are not bioinspired. With this, it will be possible to have a balance of both approaches, to describe advantages, disadvantages and possible combinations between them to improve the results obtained in the literature. In addition, it is intended to identify software testing activities (such as test case generation, branch coverage, defect identification, among others), where different search optimization approaches have been used. Finally, it is expected to identify benchmarks of various software testing activities, in which different optimization and search approaches can be tested to compare future contributions.

3 Research Method

The method proposed by (Zamora & Sossa, 2017), which was proposed to carry out SLR Software Engineering area, was selected for this work. The planning phase is presented below.

3.1 Research questions

The research questions that guide this study are described in Table 2.

Table 2. Research questions.

Research question	Motivation
RQ1.- What are the non-bio-inspired metaheuristics that have been reported at the software testing stage?	To identify the NBAs reported in the software testing phase
RQ2.- What are the main activities of the testing stage where non-bio-inspired metaheuristics have been applied?	It is important to identify in which software testing activities the algorithms reported in RQ1 have been used, in order to analyze the contributions
RQ3.- What are the advantages and disadvantages that have been found with the application of non-bio-inspired metaheuristics at the testing stage?	One of the objectives of this research is to describe the strengths and weaknesses of each approach to know in which activities they will be able to obtain better results
RQ4.- What types of benchmark problems have been used to test non-bio-inspired metaheuristics?	To know the benchmarks used to test the algorithms found, to identify their characteristics and compare results with the proposals generated in future work

3.2 Search strategy and data sources

This section shows the keywords and related terms that were used in the search string. The decision to include the terms "Path Algorithms", "Local Search", "Neighborhood Search", "GRASP" and "Simulated Annealing" was made because they were considered important search terms in the context of this SLR. The keywords are listed in Table 3.

Table 3. Keywords and synonyms identified.

Keyword	Related terms
Metaheuristic	Metaheuristics, meta-heuristic
Software engineering	-
Software testing	Testing
Benchmark	Benchmarks
Path algorithm	Trajectory algorithm
Local search	Explorative search
Neighborhood search	-
GRASP	Greedy Randomized Adaptive Search Procedure
Simulated annealing	-

The proposed search string from the key terms is described below.

("software testing" OR testing) AND ("software engineering") AND ("trajectory algorithm" OR "local search" OR "explorative search" OR "neighborhood search" OR "Greedy Randomized Adaptive Search Procedure" OR GRASP OR "simulated annealing" OR "tabu search") AND (benchmark OR benchmarks)

Due to Science Direct operators limit (OR and AND), the string was adapted for use in this research source, so that the string was as similar as possible to the main string. The following search string was used in Science Direct.

"software testing" AND "software engineering" AND ("trajectory algorithm" OR "local search" OR "explorative search" OR "neighborhood search" OR GRASP OR "simulated annealing" OR "tabu search")

Table 4 shows the databases used as source for this SLR.

Table 4. Data sources.

Database	Website
IEEXplore	https://ieeexplore.ieee.org/Xplore/home.jsp
ACM	https://dl.acm.org/
SpringerLink	https://link.springer.com/
ScienceDirect	https://www.sciencedirect.com/

1.3 Selection of primary studies

In this section, the criteria for the selection of primary studies are presented. The inclusion and exclusion criteria are presented in Table 5 and Table 6, respectively.

Table 5. Inclusion criteria.

ID	Description
IC1	The study was published between 2017 and 2023
IC2	Full access to the study
IC3	The title or abstract of the study contains the search term ‘software testing’ and its synonyms with another search term
IC4	Reading the abstract, the study hints at answering at least one research question

Table 6. Exclusion criteria.

ID	Description
EC1	Studies that are not written in the English language
EC2	Studies that are outreach articles, posters, books, chapters, presentations, abstracts, or tutorials
EC3	Duplicated studies

3.4 Selection procedure

The selection procedure was made up of the following four stages:

- Stage 1: Primary studies are filtered according to IC1 and IC2.
- Stage 2. The primary studies are removed according to EC1 and EC2.
- Stage 3. Primary studies are filtered according to IC3 and IC4.
- Stage 4. The primary studies are removed according to EC3.

4 Results

The search process was carried out according to the SLR planning, executing search string in each of the selected sources. As it is shown in Table 7, the greatest reduction of studies occurred during Stage 3 of the selection process due to the terms that appeared in the title or abstract of the papers.

Table 7. Application of inclusion and exclusion criteria by stage.

Database	First results	Stage 1	Stage 2	Stage 3	Stage 4
ACM	2,079	1,208	83	7	7
IEEEXplore	139	75	25	10	10
SpringerLink	2,408	2,103	76	3	3
ScienceDirect	329	144	119	5	5
Total	**4,595**	**3,530**	**303**	**25**	**25**

The list of references of the 25 primary studies selected for analysis can be found in [21] and the template for data extraction from each primary study can be found in [22]. Figure 1 shows the proportion of the type of paper found (journal paper or conference paper).

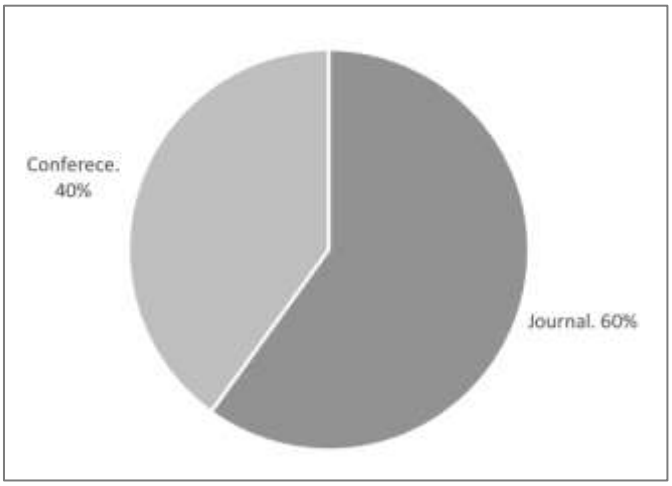

Figure 1. Distribution by publication type.

It is worth mentioning that no dominant journal or conference was found. That is, each primary study belongs to a different Journal or conference. Figure 2 shows the distribution of the years of publication of the selected studies, with 2018 being the most dominant year on this topic.

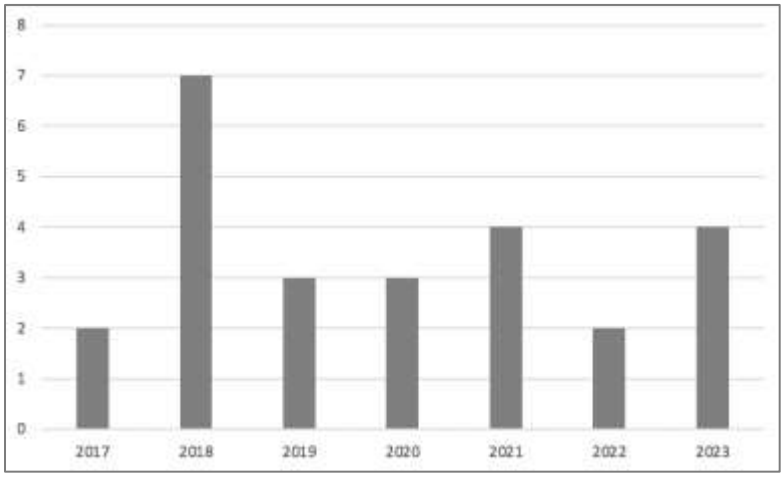

Figure 2. Distribution by year in selected databases.

Next, the report of the results obtained by answering each research question is presented.

4.1 RQ1. What were the non-bio-inspired metaheuristics that have been reported at the software testing stage?

Figure 3 shows the frequency of the algorithms found in this research. Several types of Non-bioinspired algorithms were identified: Hill climbing, Random search, Simulated annealing, Greedy Algorithm, LIPS (Linearly Independent Path-based Search), Neighborhood Search and Ls-Sampling, Gradient descendent and Tabu search, being Search based approaches the most studied in the Software testing stage.

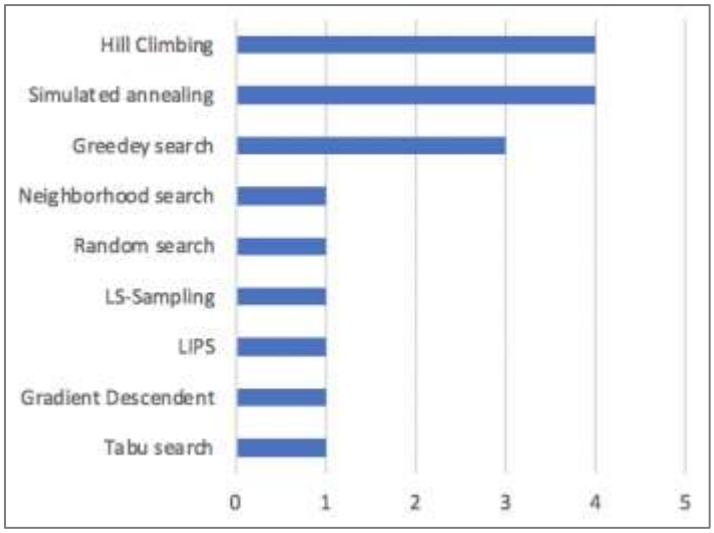

Figure 3. Frequency of reported algorithms.

Hill climbing tends to be used for use case prioritization by software engineering practitioners, whose require a good solution and not the best solution. This algorithm is one of the simplest to understand although it is mentioned that it is used in problems where there are not many local optima. In primary studies, simulated annealing is introduced for combinatorial optimization (such as test case prioritization) because by not always establishing the best options, it tends to escape local optima and allows a broader exploration of the solution space. Greedy search is presented as an option due to its simplicity and the obtaining of results in less time and with lower computational cost.

4.2 RQ2. What are the main activities of the testing stage where non-bio-inspired metaheuristics have been applied?

The purpose of this research question is to identify the different activities of the testing stage where the Algorithms detected in RQ1 were applied. Figure 4 shows the most addressed testing phase activities.

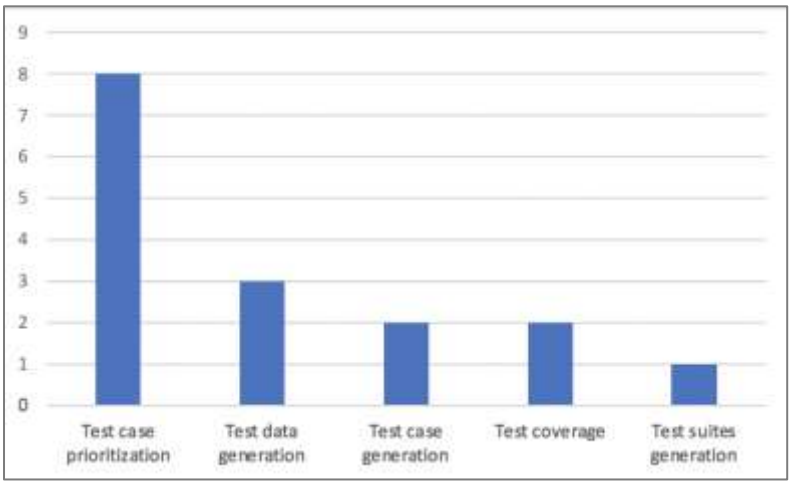

Figure 4. Activities of testing phase.

Four activities were reported for the software testing phase where NBAs assist; most of the algorithms were used for test case prioritization. This activity focuses on identifying and classifying test cases so that the most important and critical ones will be tested first. Test case prioritization is a challenging task, but it can be optimized using appropriate strategies. The key is to identify the critical test cases and prioritize them based on their importance to the business and their potential impact on the system. The algorithms found for this task were Hill Climbing (Arce, Zamora, Sossa, & Barrón, 2018; Arthur, & Vassilvitskii, 2007; Xiao, Rasul & Vollgraf, 2017), Greedy Algorithm (Arce, Zamora, Sossa, & Barrón, 2018; Xiao, Rasul, & Vollgraf, 2017), Random Search (Xiao, Rasul, & Vollgraf, 2017), Simulated Annealing (Arthur & Vassilvitskii, 2007) and Neighborhood search (Arthur & Vassilvitskii, 2007) were used for this activity.

On the other hand, test data generation and test case generation are two separate but related tasks. The main difference between the two tasks is that the first one focuses on the creation of specific data to be used in the tests, while the second one focuses on defining the test cases and the steps to follow to evaluate a specific functionality or characteristic of the product. system. Hill Climbing (Krizhevsky & Hinton, 2009) and Simulated Annealing (Krizhevsky & Hinton, 2009) algorithms were used for test data generation, and algorithms like LIPS Chollet(2015) was used in test cases generation. The findings of this research indicate that, compared with bio-inspired algorithms, NBAs algorithms did not perform well doing in this activity.

According to the findings NBAs were used to sort and group test cases into test suites. Only the use of LS-Sampling (Dua and Graff, 2017) was reported for this activity of the software testing stage. Search-based approaches showed the most versatility for performing software testing stage activities.

4.3 RQ3. What are the advantages and disadvantages found with the application of bio-inspired algorithms at the testing stage?

In primary studies, the algorithms were predominantly compared with Bio-inspired algorithms. The findings from these comparisons commonly indicated that NBA did not exhibit a noteworthy improvement when compared to Bio-inspired algorithms. Nonetheless, because of their simplicity, i.e., the small amount of code required for their execution, these algorithms are well-suited for specific or limited tasks.

Another advantage mentioned is the number of parameters to tune. In bioinspired algorithms (such as evolutionary algorithms and collective intelligence), there are parameters such as population size, number of iterations (or generations), selection of mutation

and crossover operators, among others. Some testers tend to prefer non-bioinspired metaheuristics to avoid investing effort in parameterizing the optimization algorithms.

However, in large systems according to the findings, it is more advisable to use multi-objective approaches due to the number of functionalities they cover. Most of the reported algorithms were used to test case prioritization.

4.4 RQ4. What types of benchmark problems have been used to test non-bio-inspired metaheuristics?

Most of the NBAs were tested with specific problems, i.e., problems proposed by the authors to simulate a real-life scenario (Arthur & Vassilvitskii, 2007; Xiao, Rasul, & Vollgraf, 2017; Chollet, 2015). Triangle was used in two studies (Xiao, Rasul & Vollgraf, 2017; Dua and Graff, 2017). One study was tested with the Corpus SF110 benchmark (Dua and Graff, 2017). The proportion of these benchmarks can be seen in Figure 5.

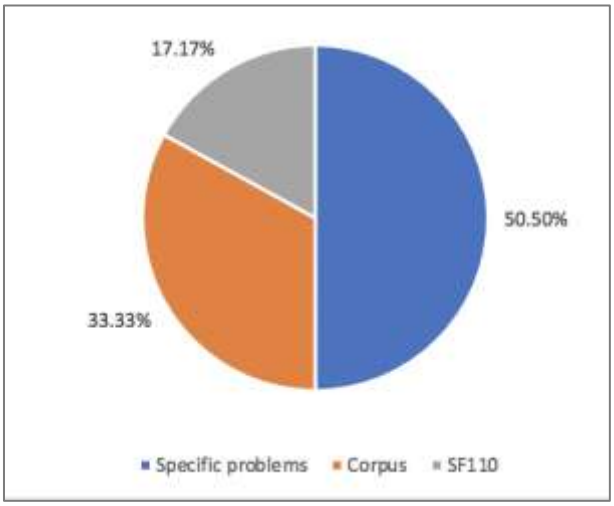

Figure 5. Reported benchmark to compare search algorithms.

5 Discussion

The present study was motivated by the following issues:

1. There is no SLR that presents non-bioinspired metaheuristics applied to software testing.
2. There is no SLR that summarizes the activities in the software testing phase, where non-bioinspired metaheuristics have been applied.
3. Most SLRs are based on bioinspired metaheuristics (such as evolutionary algorithms and collective intelligence).

Therefore, compared to the related work in Section 2, this SLR summarizes the non-bioinspired metaheuristics of software testing activities as well as the benchmarks where they have been tested.

With respect to Table 3, the most reported metaheuristics are Hill climbing, simulated annealing and Greedy search, due to their simplicity with respect to bioinspired metaheuristics. Figure 4 shows that test case prioritization is the activity in the testing phase most addressed by these metaheuristics is test case prioritization (which test case should be executed first or has a dependency on others) due it is a combinatorial optimization problem.

Finally, in the software industry, there is no outstanding benchmark for testing metaheuristics. That is, each organization uses its own systems to test the results of optimization algorithms.

6 Conclusions and future work

Metaheuristics are optimization techniques that are widely used in various areas of engineering, including software testing. In this paper, a Systematic Literature Review was performed in order to present the state of non-bioinspired metaheuristics in the area of Software testing. The importance of metaheuristics in software testing lies in their ability to generate optimal or near-optimal solutions in complex and dynamic situations, where other techniques cannot offer a satisfactory solution. Additionally, metaheuristics can help improve the efficiency and effectiveness of software testing by reducing the time and resources required to perform tests. These techniques can also help identify and fix defects more quickly, which can improve software quality and reduce costs associated with maintenance and repairs.

According to the information gathered in this SLR, non-bio-inspired metaheuristics mean a great improvement to the software testing process, however, bio-inspired algorithms are still superior in this aspect (Dua and Graff, 2017). In carrying out this research, the use of hybrid algorithms was detected, showing a significant improvement (according to the benchmarks) in the efficiency when performing activities in the testing process.

Research in the software testing phase is very extensive and there is continuous work in this regard. There are open problems such as the selection of test cases that must be executed in order not to repeat paths in a program or system. This is complicated because as the cyclomatic complexity increases in a function or procedure, the number of options grows considerably. In addition, there are numerous types of testing at different levels of the system and with different objectives (regression, white box, black box, unit, usability, security), which means that metaheuristics must be identified and selected at each type and level of testing. testing a system.

It is proposed as future work the research of hybrid algorithms and approaches, since according to studies (Arce et al., 2019; Zhang, 2000), they represent a significant improvement over the use of individual approaches.

References

Araújo, R. A. (2012). A morphological perceptron with gradient-based learning for Brazilian stock market forecasting. *Neural Networks*, 28, 61-81.

Araújo, R. A., & Sussner, P. (2010). An increasing hybrid morphological-linear perceptron with pseudo gradient-based learning and phase adjustment for financial time series prediction. In: Proceedings of the 2010 IEEE World Congress on Computational Intelligence, Vol. IJCNN, Barcelona, Spain, 2010, pp. 807–814.

Araújo, R. A., & Sussner, P. (2010). An increasing hybrid morphological-linear perceptron with pseudo gradient-based learning and phase adjustment for financial time series prediction. In: Proceedings of the 2010 IEEE World Congress on Computational Intelligence, Vol. IJCNN, Barcelona, Spain, 2010, pp. 807–814.

Araújo, R.A., Oliveira, A.L.I, Meira, S. (2017). A morphological neural network for binary classification problems. *Engineering Applications of Artificial Intelligence*, 65, 12–28.

Arce, F., Mendoza-Montoya, O., Zamora, E., Antelis, J.M., Sossa, H., Cantillo-Negrete, J., Carino-Escobar, R.I., Hernández, L.G., & Falcón, L.E. (2019). Dendrite Ellipsoidal Neuron Trained by Stochastic Gradient Descent for Motor Imagery Classification. In: Carrasco-Ochoa J., Martínez-Trinidad J., Olvera-López J., Salas J. (eds) Pattern Recognition. MCPR 2019. Lecture Notes in Computer Science, vol 11524. Springer, Cham.

Arce, F., Zamora, E., Fócil-Arias, C., & Sossa, H. (2019). Dendrite ellipsoidal neurons based on k-means optimization. *Evolving Systems*, *10*(3), 381-396. https://doi.org/10.1007/s12530-018-9248-6.

Arce, F., Zamora, E., Sossa, H. (2017). Dendrite Ellipsoidal Neuron. In: 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 795-802.

Arce, F., Zamora, E., Sossa, H., & Barrón, R. (2018). Differential evolution training algorithm for dendrite morphological neural networks. *Applied Soft Computing Journal*, *68*, 303-313. https://doi.org/10.1016/j.asoc.2018.03.033.

Arthur, D., & Vassilvitskii, S. (2007). K-means++: the advantages of careful seeding In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, pp. 1027-1035. Society for Industrial and Applied Mathematics, Philadelphia (2007).

Caruana,R., Lawrence,S., Giles, L. (2000). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS'00) (pp. 386-387). MIT Press, Cambridge, MA, USA.

Chollet, F. (2015). Keras. https://keras.io

Dua, D. and Graff, C. UCI machine learning repository (2017). http://archive.ics.uci.edu/ml

Gómez-Flores, W., & Sossa JH (2020). Towards Dendrite Spherical Neurons for Pattern Classification. In: Figueroa Mora, K., Anzurez-Marín, J., Cerda, J., Carrasco-Ochoa, J., Martínez-Trinidad, J., & Olvera-López, J. (eds) Pattern Recognition, MCPR 2020. *Lecture Notes in Computer Science*, vol 12088. Springer, Cham.

Hernández, G., Zamora, E., Sossa, H. (2018). Morphological-Linear Neural Network. 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Rio de Janeiro, 2018, pp. 1-6.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical Report (2009) https://www.cs.toronto.edu/ kriz/cifar.html

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6), 861-867.

Pessoa, L. F. C., & Maragos, P. (2000). Neural networks with hybrid morphological/rank/linear nodes: a unifying framework with applications to handwritten character recognition. *Pattern Recognition*, 33, 945–960.

Ritter, G. X., & Sussner, P. (1996). An introduction to morphological neural networks. In Proceedings of the 13th International Conference on Pattern Recognition, Vienna, Austria, (pp. 709-717).

Ritter, G. X., & Urcid, G. (2003). Lattice algebra approach to single-neuron computation. *IEEE Transactions on Neural Networks*, 14(2), 282-295.

Ritter, G. X., & Urcid, G. (2007). Learning in Lattice Neural Networks that Employ Dendritic Computing. In V.G. Kaburlasos & G.X. Ritter (Eds.), Computational Intelligence Based on Lattice Theory. Studies in Computational Intelligence (Vol. 67). Springer, Berlin, Heidelberg.

Ritter, G. X., Iancu, L. Urcidet, G. (2003). Morphological perceptrons with dendritic structure. In The 12th IEEE International Conference on Fuzzy Systems, FUZZ 2003, Volume 2, (pp. 1296-1301).

Rosenblatt, F. (1957). The Perceptron-a perceiving and recognizing automaton. Cornell Aeronautical Laboratory, Report 85-460-1.

Sossa, J. H., & Guevara, E. (2014). Efficient training for dendrite morphological neural networks. *Neurocomputing*, 131, 132-142 Sussner, P. (1998). Morphological perceptron learning. In Proceedings of the IEEE International Symposium on Intelligent Control, Gaithersburg, MD, (pp. 477482).

Sussner, P., & Campiotti, I. (2020). Extreme learning machine for a new hybrid morphological/linear perceptron. *Neural Networks*, 123, 288-298.

Sussner, P., & Esmi, E. (2009). Introduction to morphological perceptrons with competitive learning. In: Proceedings of the International Joint Conference on Neural Networks, Atlanta, GA, 2009, pp. 3024-3031.

Sussner, P., & Esmi, E. (2011). Morphological perceptrons with competitive learning: Lattice theoretical framework and constructive learning algorithm. *Information Sciences*, 181(10), 1929-1950.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. 2017, arXiv:1708.07747 (Preprint). https://github.com/zalandoresearch/fashion-mnist

Zamora, E., & Sossa, J. H. (2017). Dendrite morphological neurons trained by stochastic gradient descent. *Neurocomputing*, 260, 2017, 420-431.

Zhang, G. P. (2000). Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, Part C (Applications and Reviews), 30(4), 451-462, Nov. 2000.