Revista Brasileira de Computação Aplicada, July, 2025

DOI: 10.5335/rbca.v17i2.16291
Vol. 17, Nº 2, pp. 103–112
Homepage: seer.upf.br/index.php/rbca/index

ORIGINAL PAPER

CNOPsolution Tester: a web application for result validation in constrained optimization problems

Adrian García-López [1], Oscar Chávez-Bosquez [2], Betania Hernández-Ocaña [2]

[1]Instituto Tecnológico Superior de Comalcalco, Ingeniería en Sistemas Computacionales, [2]Universidad Juárez Autónoma de Tabasco, División Académica de Ciencias y Tecnologías de la Información

* betania.hernandez@ujat.mx

Received: 2024-09-19. **Revised:** 2025-07-25. **Accepted:** 2025-07-31.

Abstract

In numerical optimization, solutions to optimization problems are developed using methods or techniques that generate approximate results for complex problems, such as metaheuristics. When these techniques generate results, it is essential to verify whether these values meet the optimization problem requirements, for example, values of the variables within the minimum and maximum range and the satisfaction of the problem constraints. Doing it by hand can be very difficult, especially considering optimization problems have different characteristics. This paper presents the development of a web application called CNOPsolution Tester, which allows researchers and end-users to validate the values of decision variables based on their ranges and constraints. The application was designed based on diagrams created with the Unified Modeling Language and developed in JavaScript. We tested the web application with multiple users and 24 benchmark optimization problems to evaluate its usability, measuring the number of clicks needed to solve each problem. We also validated the functionality using 20% of the problems. The results demonstrate that the CNOPsolution Tester is an efficient tool and easy to use by any end-user, as shown by the response time and the number of clicks used on the application. We applied the Pearson correlation to identify the relationship between the number of clicks, problem variables, and constraints.

Keywords: Metaheuristics; Optimization Problems; Pearson Correlation; Tester.

Resumo

Na otimização numérica, as soluções para problemas de otimização são desenvolvidas usando métodos ou técnicas que geram resultados aproximados para problemas complexos, como as meta-heurísticas. Quando essas técnicas geram resultados, é essencial verificar se esses valores atendem aos requisitos do problema de otimização, por exemplo, valores das variáveis dentro do intervalo mínimo e máximo e a satisfação das restrições do problema. Fazer isso manualmente pode ser muito difícil, especialmente considerando que os problemas de otimização têm características diferentes. Este trabalho apresenta o desenvolvimento de uma aplicação web chamada CNOPsolution Tester, que permite a pesquisadores e usuários finais validar os valores das variáveis de decisão com base nos seus intervalos e restrições. A aplicação foi projetada com base em diagramas criados com a Linguagem de Modelagem Unificada e desenvolvida em JavaScript. Testamos a aplicação web com múltiplos usuários e 24 problemas de otimização de referência para avaliar sua usabilidade, medindo o número de cliques necessários para resolver cada problema. Também validamos a funcionalidade usando 20% dos problemas. Os resultados demonstram que o CNOPsolution Tester é uma ferramenta eficiente e fácil de usar para qualquer usuário final, conforme demonstrado pelo tempo de resposta e pelo número de cliques utilizados na aplicação. Aplicamos a correlação de Pearson para identificar a relação entre o número de cliques, variáveis do problema e restrições.

Palavras-Chave: Meta-heurística; Problemas de otimização; Correlação de Pearson; Testador.

1 Introduction

In the real world, different disciplines face complex problems that need to be optimized using some technique, model, paradigm, or tool to minimize or maximize a quality measure or objective function representing the problem to be solved. Any chosen approach is based on the values that the problems decision variables take, which must comply with certain linear/non-linear constraints and even limits on their values (Kumar et al., 2020). A complex problem can be modeled as a general problem of mathematical programming (Sarker and Newton, 2007) and can be represented as:

$$\text{minimize (or maximize): } f(\vec{x})$$

subject to

$$\begin{aligned} g_i(\vec{x}) &\leq 0, \quad i = 1, 2, ..., m \\ h_j(\vec{x}) &= 0, \quad j = 1, 2, ..., p \end{aligned}$$

where:

$$L_k \leq x_i \leq U_k, \quad k = 1, 2, ..., D$$

where, f denotes the objective function, $\vec{x}$ is the n-dimensional solution vector $\vec{x} = [x_1, x_2, x_3, ..., x_n]^T$, which $\vec{x} \in \mathbb{R}^n$. D is the number of design variables, m is the number of inequality constraints, and p is the number of equality constraints. If we denote F as the feasible region (where all the solutions that satisfy the problem are found) and S as the entire search space, then it should be clear that $F \subseteq S$ (Hernández-Ocaña et al., 2016), as shown in Fig. 1. This mathematical model is called Constrained Numerical Optimization Problems (CNOP)(Mezura-Montes and Coello, 2011).

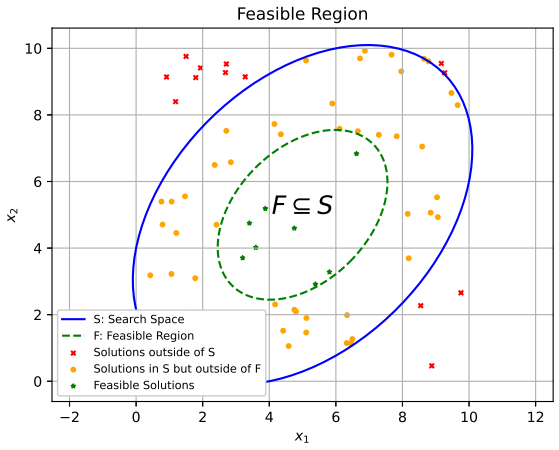

Figure 1: Feasible region F and search space S.

Searching for feasible solutions to CNOPs using classical algorithms becomes difficult, as most optimization problems are classified as NP-hard that cannot be solved in a polynomial time region, especially when they are highly constrained (Garey et al., 1976). An alternative for generating a single or a set of solutions that facilitate decision-making for the end user is to consider bio-inspired metaheuristics. These are based on population groups and designed to improve the search procedures for optimal solutions in reasonable times, implementing two search schemes: exploration (diversification of the population) and exploitation (intensification of the population) (Abdel-Basset et al., 2018).

Metaheuristics are classified according to the type of natural phenomenon they are based on. Among the two most popular classifications are:

- Evolutionary Algorithms (EAs) that emulate the process of natural evolution and the survival of the fittest of species (Eiben and Smith, 2003) and can be divided into four categories:
  - Evolutionary strategies (Rechenberg, 1989)
  - Evolutionary programming (Fogel, 2006)
  - Differential evolution (Storn and Price, 1997)
  - Genetic programming (Koza, 1992)
- Swarm Intelligence Algorithms (SIAs) that replicate the collaborative behavior of certain simple and intelligent species (Engelbrecht, 2005). Some examples of SIAs are:
  - Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995)
  - Artificial Bee Colony Optimization (ABC) (Karaboga and Basturk, 2007)
  - Bacteria Foraging Optimization Algorithm (BFOA) (Passino, 2002)
  - Ant Colony Optimization (ACO) (Dorigo et al., 1996)

Researchers have implemented different metaheuristics in various studies, providing detailed results for each decision variable and the solution to the problem. For example, in the research by Kumar et al. (2020), the authors suggest implementating various metaheuristics and constraint handling approaches and validate their proposals on a set of 57 real-world constrained optimization problems. Another relevant study is Barbosa et al. (2010), where the authors focus on the implementation of various evolutionary algorithms. The researchers present the test results using the function set of the CEC 2006 competition. The work mentioned in Brest et al. (2017) introduces a novel metaheuristic algorithm, experimenting with it on a set of problems associated with the reference functions of the CEC 2017.

When initiating research that involves applying novel methods, metaheuristics, or performance metrics to solve CNOPs, it becomes necessary to seek related works that address similar or identical problems the new research will tackle. This process is essential for validating the results or comparing the values generated by the method in execution. However, this search and validation of results entail a significant delay in the research time, as, for validation, common tools such as spreadsheets or programming languages are used to independently

evaluate the problem with the values generated by the technique used.

Also, values of interest in the literature must be verified in some cases, but the lack of an intuitive tool to perform these checks can cause additional delays. Not having a tool that allows for efficient evaluation of the feasibility of values about the constraints and limits of the variables also contributes to the complexity and slowness of the research process.

A proposed solution is designing and developing of a web application hosted on GitHub Pages[1]. This application will serve as a solution tester for CNOP values. Diagrams were created based on the Unified Modeling Language (UML) to identify the system needs, and the development was carried out in the JavaScript programming language.

The web application CNOPsolution Tester includes a set of CNOPs used in the study of García-López et al. (2023). Additionally, the CNOPsolution Tester allows the end-user to input an undefined problem within this set through a user-friendly graphical interface. As a result, the application displays the objective function value, the value of each constraint, and the feasibility of the decision variable values based on their minimum and maximum ranges.

2 Design and development

UML diagrams were first designed to develop the CNOPsolution Tester. This facilitates development by allowing the identification of system needs. Fig. 2 presents the use case diagram that defines the end user's navigation.

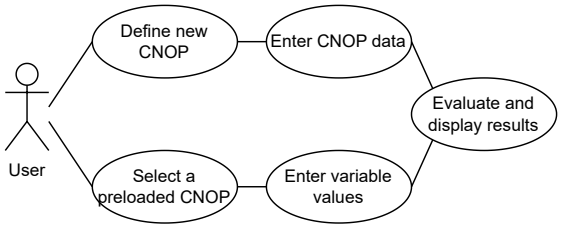

Figure 2: Use case diagram for CNOPsolution Tester.

Prior to developing the web application, a flowchart was designed to show the process between all parts of the CNOPsolution Tester. This diagram visualizes the sequence of steps necessary to evaluate a CNOP and verify the results, including the decision variables and constraints. Fig. 3 presents the flowchart of the CNOPsolution Tester.

[1]Static website hosting service: https://pages.github.com/

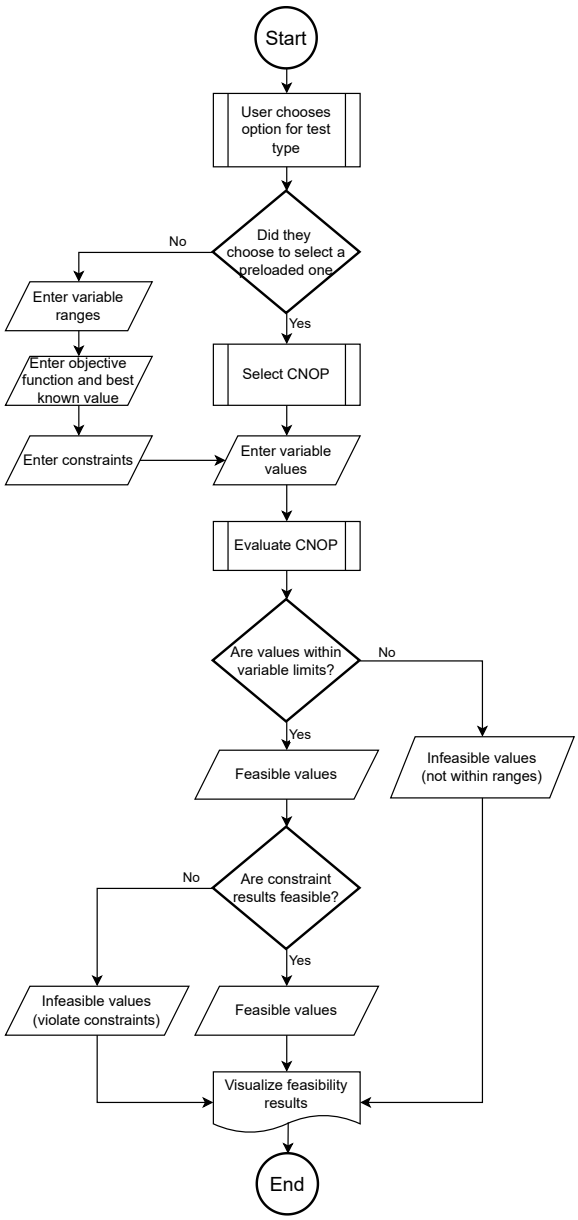

Figure 3: Flowchart of CNOPsolution Tester.

We implemented Algorithm 1 for CNOPsolution Tester, which represents the sequence of steps followed to evaluate a CNOP and verify the results of the entered solution vector $\vec{x}$. The algorithm begins with the selection of a preloaded benchmark CNOP or the definition of a new problem. If the user chooses a preloaded problem, the system allows viewing the problem information, such as the objective function $f(\vec{x})$, the constraints $g_i(\vec{x})$ or $h_j(\vec{x})$, and the variable limits $L_k \leq x_i \leq U_k$. For a new problem,

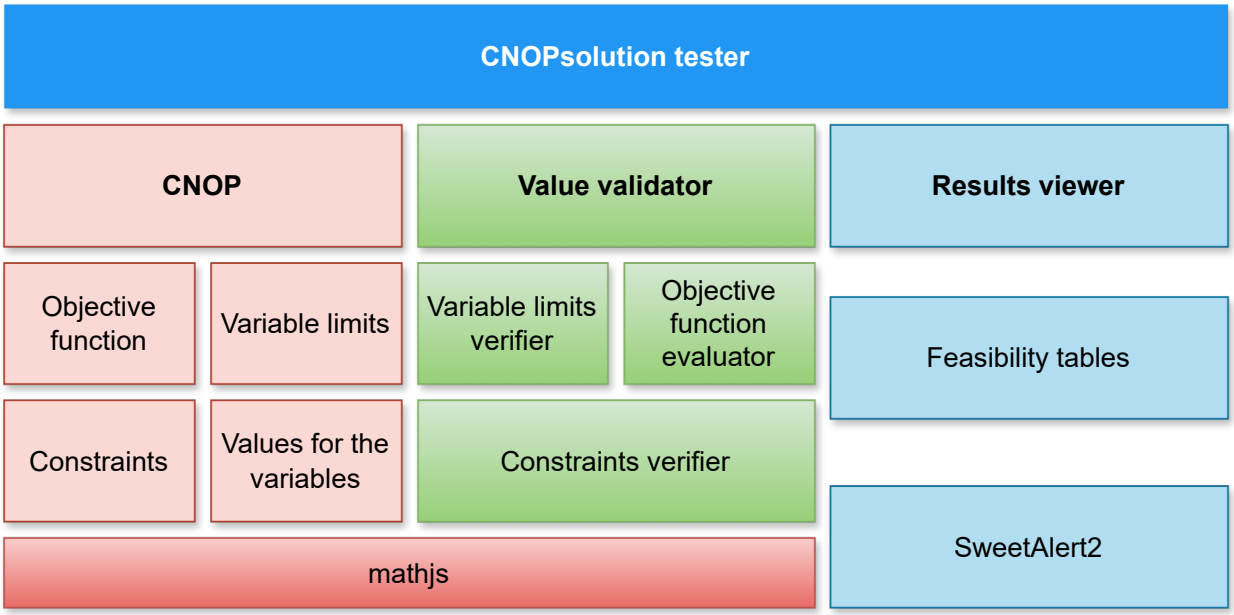

Figure 4: Architecture of CNOPsolution Tester.

the end-user must enter this information and enter the number of dimensions n of the problem.

Once the problem information is loaded or entered, n fields are enabled for the user to enter the values of the decision variables (solution vector $\vec{x}$). Finally, the value of the objective function is calculated, the constraints are evaluated, and it is verified that the values of the decision variables comply with the specified limits $L_k \leq x_i \leq U_k$.

Algorithm 1 General process of CNOPsolution Tester

1: **Start**
2: Choose: *preloaded problem* or *new problem*
3: **if** the option is *preloaded problem* **then**
4: Load $f(\vec{x})$, $g_i(\vec{x})$ or $h_j(\vec{x})$ and $L_k \leq x_i \leq U_k$ from the selected problem
5: **else**
6: Request the n-dimensional number of the problem
7: Request $f(\vec{x})$, $g_i(\vec{x})$ or $h_j(\vec{x})$ and $L_k \leq x_i \leq U_k$ from the new problem
8: **end if**
9: Enable n fields to enter the values of the decision variables $\vec{x}$
10: Calculate and display the value $f(\vec{x})$, $g_i(\vec{x})$ or $h_j(\vec{x})$
11: Verify and display the variable limits $L_k \leq x_i \leq U_k$
12: **End**

In software development, the architecture of a system is a fundamental component for defining the structure and behavior of the system, in addition to the components that make it up. Fig. 4 shows the high-level structure of CNOPsolution Tester, including the back-end and front-end elements.

3 Description and operation of CNOPsolution Tester

For the development of the CNOPsolution Tester web application, we used the high-level, interpreted, object-oriented, and cross-platform programming language JavaScript for the application's interactivity, being one of the most used and versatile client-side programming languages (Jansen, 2024). JavaScript has a wide ecosystem of numerous libraries and frameworks. For the front end, we used the HTML markup language for the structure of the web application and the CSS style language for the presentation, using the DashMin template[2].

To evaluate the CNOPs, we incorporated a mathematical expression evaluator to reduce the complexity of the implementation when evaluating the objective function and constraints, in this case, mathjs (de Jong and Mansfield, 2018), a JavaScript framework that allows performing mathematical operations at runtime from text strings. We used other frameworks for the development that facilitate the system's usability. These are Bootstrap[3] for designing the user interface, SweetAlert[4] for displaying alert messages, and jQuery[5] for manipulating

[2] DashMin – Free Bootstrap 5 Admin Dashboard Template. Available at: https://themewagon.com/themes/

[3] https://getbootstrap.com: Bootstrap is a popular front-end framework for designing responsive and mobile-first websites.

[4] https://sweetalert2.github.io/: SweetAlert is a JavaScript library for displaying beautiful and customizable alert messages in web applications.

[5] https://jquery.com: jQuery is a fast, small, and feature-rich

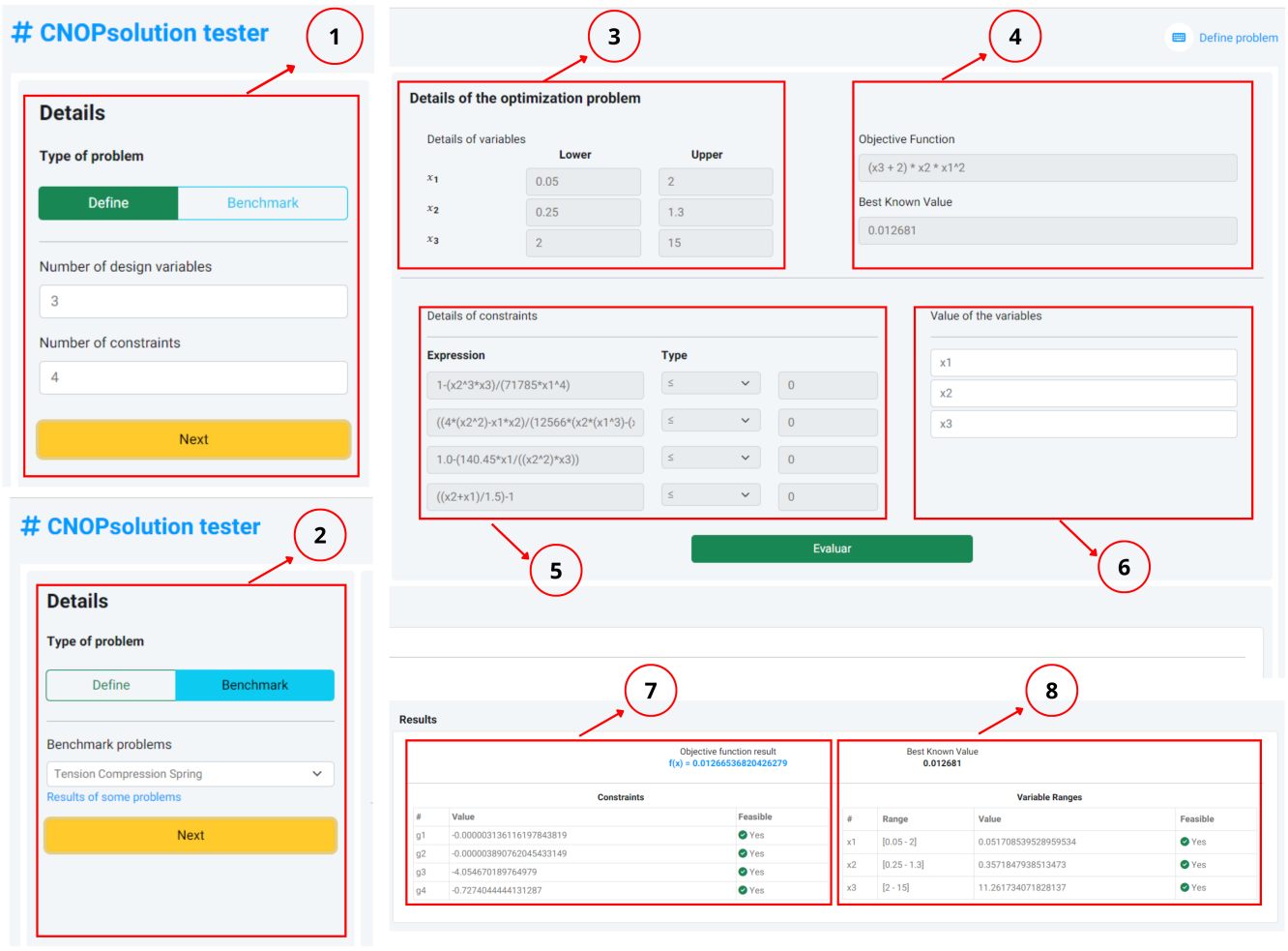

Figure 5: Modules of the CNOPsolution Tester web application.

the Document Object Model (DOM).

After completing the coding of the CNOPsolution Tester, the resulting web application is visualized in Fig. 5, where the User Interface (UI) of the web application is presented. The UI consists of several key elements:

- In Panel 1, by selecting the "Define" option, two fields allow the user to configure the definition of a new CNOP. These fields request the number of variables and the number of constraints. After entering this information, the fields are automatically generated according to the provided values.
- In Panel 2, selecting "Benchmark" allows the user to display only a selection component. This component allows one to choose among the optimization problems included in the CNOPsolution Tester.
- In Panel 3, the decision variable ranges are displayed or must be inserted, depending on the user's choice (either to create a new problem or select one included).
- Panel 4 allows the user to enter the objective function and the value of the best-known solution.
- In Panel 5, the CNOP constraints are configured, including equality or inequality constraints.
- The decision variable values are entered in Panel 6. These values will be evaluated in the objective function and constraints and verified to ensure compliance with the established minimums and maximums.
- Panels 7 and 8 present results, including the value of the objective function, the value of the constraints, the feasibility of each constraint, and the feasibility of the decision variable values based on their ranges.

Also, the CNOPsolution Tester contains the help component that accompanies and guides the end user through each section that must be entered for successful use.

CNOPsolution Tester is available on GitHub, a version control platform that facilitates open access to source code. The repository is publicly accessible at `https://github.com/garcialopez/CNOPsolution-tester`. Additionally, the application is hosted on GitHub Pages[6], a service designed

JavaScript library for simplifying HTML DOM tree traversal and manipulation, event handling, and animation.

[6]Free web hosting service: `https://pages.github.com/`

for static websites. To view and use the software, visit https://garcialopez.github.io/CNOPsolution-tester.

4 Experiments and Discussion

Evaluating the response capacity to user requests is necessary to deploy the CNOPsolution tester. It is important to measure the response time to evaluate its performance for the end-user. Therefore, tests were conducted using the Loaderio website[7], an online service used for load testing web applications. Loaderio allows simulating multiple users accessing the web application at the same time.

For these tests, we configured the connection of 250 clients over 1 minute. At the end of the test, the results presented in Table 1 were obtained.

Table 1: Performance test results with the Loader tool.

Response times	Bandwidth	Redirects
Mean: 4 ms	**Sent:** 3.89 MB	**Valid:** 15,000
Minimum: 4 ms	**Received:** 180.67 MB	**Invalid:** 0
Maximum: 86 ms	–	–

Results of the load test for 250 simultaneous connections on the CNOPsolution tester.

The results from Table 1 indicate that the web application correctly managed the load of 250 clients for 1 minute. The response times were fast, with an average of 4 milliseconds, showing a high response capacity. In addition, all redirects were valid, which allows us to understand the application's traffic management.

Measuring clicks in software is important to understand user interaction with the UI. In CNOPsolution Tester, when end-users choose to evaluate a preloaded CNOP, we identified the following formula that describes the total number of clicks: $n + 6$, where the first 4 clicks correspond to the selection of a CNOP, n represents the number of entered variables, and the last 2 clicks are for evaluation and result display. To perform a click simulation in the application, the first three preloaded problems have been used, the results are presented in Table Table 2.

Table 2: Number of click results on CNOPSolution Tester.

Problem	n	$n + 6$
Tension compression spring	3	9
Pressure Vessel	4	10
Design of a reinforced concrete beam	2	8

Fig. 6 shows the number of clicks on the preloaded CNOPs. For the Tension compression spring problem, 9 clicks were required to evaluate it, while for the Pressure Vessel problem, 10 clicks were needed. In contrast, for the Design of a reinforced concrete beam problem, 8 clicks were needed.

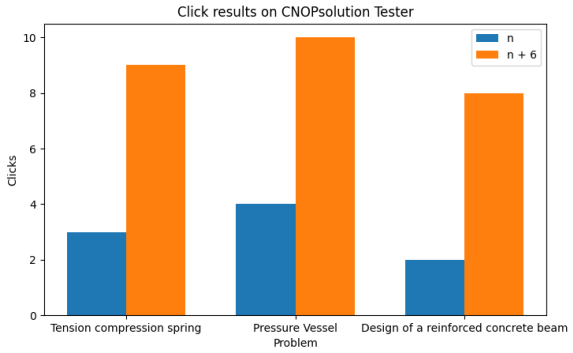

Figure 6: Number of clicks on the preloaded CNOPs.
The number of clicks is represented by the sum of the number of variables plus 6.

Table 3: Number of clicks in custom problems by the user.

Problem	n	m	p	$3n + 4m + 4p + 7$
g01	13	9	0	82
g02	20	2	0	75
g03	10	0	1	41
g04	5	6	0	46
g05	4	2	3	39
g06	2	2	0	21
g07	10	8	0	69
g08	2	2	0	21
g09	7	4	0	44
g10	8	6	0	55
g11	2	0	1	17
g12	3	1	0	20
g13	5	0	3	34
g14	10	0	3	49
g15	3	0	2	24
g16	5	38	0	174
g17	6	0	4	41
g18	9	13	0	86
g19	15	5	0	72
g20	24	6	12	151
g21	7	1	5	52
g22	22	1	19	153
g23	9	2	4	58
g24	2	2	0	21

When the user selects to enter a custom problem, more clicks are required to configure the input. We identified the formula for the total number of clicks as $3n+4m+4p+7$ where n is the number of variables, m is the number of inequality constraints, p is the number of equality constraints and 7 additional clicks correspond to the initial setup: defining how many variables and constraints there will be, entering the best known value, and clicking the buttons to perform the calculation, view the results, and export the visualization to PDF. To calculate the number of

[7]Load testing service: https://loader.io

clicks needed to evaluate problems entered by the user, we have used the problems from the Benchmark published in (Liang et al., 2006).

The results are presented in Table Table 3 and shows the number of clicks needed to evaluate the problems entered by the user. For example, for problem g01, evaluating the problem required 82 clicks, while for problem g02, 75 clicks were needed. On the other hand, for problem g03, 41 clicks were needed. The mean number of clicks for the problems entered by the user was 62 clicks, with a mean of 9 variables, 5 inequality constraints, and 2 equality constraints. Figure Fig. 7 shows the number of clicks in the problems entered by the user.

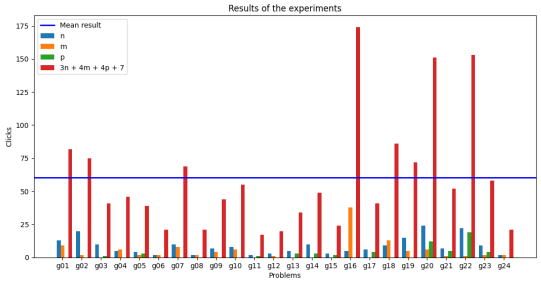

Figure 7: Number Number of clicks in custom CNOPs by the user.

A click is represented by the sum of the number of variables, inequality constraints, equality constraints, plus 7 additional configuration.

In Fig. 7, the blue line identifies the mean number of clicks for the problems custom by the user. However, we cannot conclude that the number of clicks is proportional to the number of variables, inequality constraints, and equality constraints. For this reason, we have calculated the Pearson correlation coefficient (r) (Pearson and Galton, 1895) to measure the relationship between the number of clicks and the number of variables, inequality constraints, and equality constraints. The formula to calculate the Pearson correlation coefficient is presented in Eq. (1).

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum_{i=1}^{n}(x_i - \bar{x})^2)(\sum_{i=1}^{n}(y_i - \bar{y})^2)}} \quad (1)$$

where x_i and y_i are the variables, $\bar{x}$ and $\bar{y}$ are the means of the variables, and n is the number of observations. This means the Pearson correlation coefficient measures the linear relationship with values between -1 and 1. A value of 1 indicates a positive correlation, a value of -1 indicates a negative correlation, and 0 indicates no correlation. By applying Eq. (1) to the data in Table 3, the results presented in Table 4 are obtained.

The correlation matrix in Table 4 shows the correlation coefficients between all combinations of pairs of the variables n, m, p, and *clicks*. The correlation between n and *clicks* is 0.692163, indicating a moderate positive correlation. The correlation between m and *clicks* is 0.652895, indicating a moderate positive correlation. This demonstrates that as m increases, so does *clicks*. The correlation between p and *clicks* is 0.541070, indicating a weaker positive correlation. This suggests that although p and *clicks* are correlated, the relationship is not as strong as with n or m. Fig. 8 shows the heat map of the Pearson correlation matrix to visualize the relationship between the variables and identify the relationship of the number of clicks with the variables n, m, and p.

Table 4: Pearson Correlation Matrix.

–	n	m	p	clicks
n	1.000000	0.007476	0.605063	0.692163
m	0.007476	1.000000	-0.186574	0.652895
p	0.605063	-0.186574	1.000000	0.541070
clicks	0.692163	0.652895	0.541070	1.000000

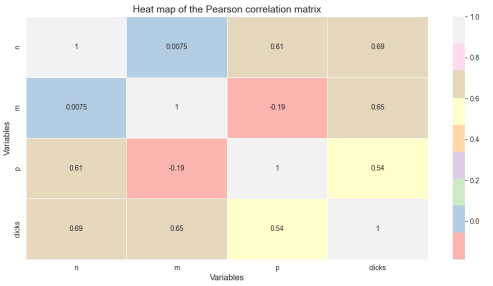

Figure 8: Heat map of the Pearson correlation matrix between all combinations of pairs of the variables n, m, p, and *clicks*.

When using software, the end-user expects the application to be easy to use or initially provided with a user guide. In CNOPsolution Tester, we have implemented a user guide for the end-user. The guide was implemented with Shepherd[8], a JavaScript framework that allows the creation of step-by-step guides for users. The user guide is activated when the user presses the **Help** button on the user interface. Figure Fig. 9 shows how the user guide.

Another critical aspect is correctly validating the decision variable values based on their ranges and constraints. For this, we choose a sample of 20% of the problems from the Benchmark published in (Liang et al., 2006). From the previously explored problems of CEC 2006, we have selected the problems g04, g05, g06, and g13 because they are problems with values close to the mean of variables previously explored in the click measure. The authors who designed the mentioned problems also provided the decision variables' values and the objective function's results.

[8]Shepherd.js – JavaScript library for guiding users through your app. Available at: https://shepherdjs.dev/

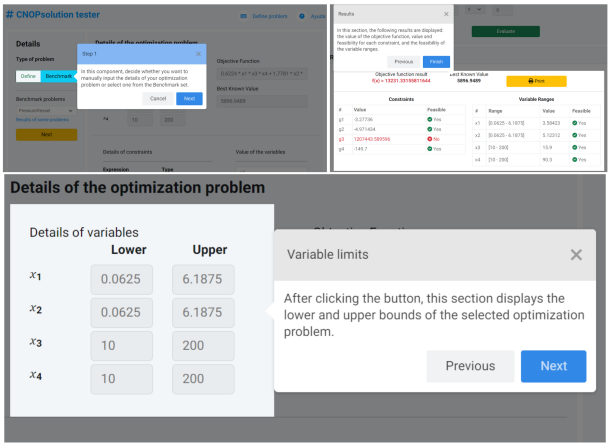

Figure 9: User guide in CNOPsolution Tester.
Pressing the **Help** button activates the user guide.

Following, we present the comparison and validation of the results provided by the authors and the values generated by the CNOPsolution Tester.

- **Problem g04:** The g04 problem is an optimization problem with 5 decision variables, 6 inequality constraints, and 0 equality constraints. The decision variable values provided by the authors are: $x = [78, 33, 29.99525602568159, 45, 36.775812905788 20]$. The objective function provided by the authors is -30665.5386717834 and the value generated by CNOPsolution Tester is -30665.538671783317. During validation, it is demonstrated that the decision variable values provided by the authors and the objective function are feasible, meet the problem constraints, and comply with the decision variable limits (see Fig. 10).

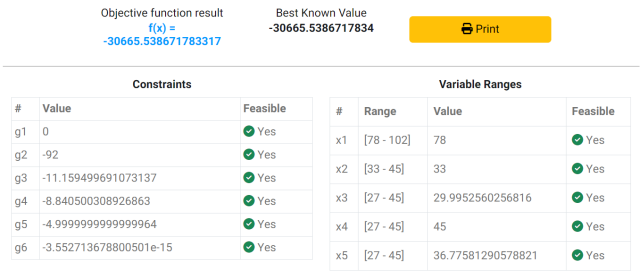

Figure 10: Validation of problem g04 in CNOPsolution Tester.

- **Problem g05:** The g05 problem is an optimization problem with 4 decision variables, 2 inequality constraints, and 3 equality constraints. The decision variable values provided by the authors are: $x = [679.945148297, 1026.066976000, 0.1188763690944, -0.396233485215]$. The objective function provided by the authors is 5126.4967140071, and the value generated by the CNOPsolution Tester is 5126.4967140071. The decision variable values provided by the authors indicate that the objective function is feasible and meets the constraints and limits of the decision variables (see Fig. 11).

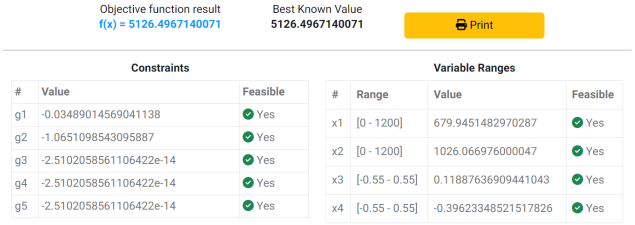

Figure 11: Validation of problem g05 in CNOPsolution Tester.

- **Problem g06:** The g06 problem is an optimization problem with 2 decision variables, 2 inequality constraints, and 0 equality constraints. The decision variable values provided by the authors are: $x = [14.095000000000064, 0.8429607892154795668]$. The objective function provided by the authors is -6961.8138755802, and the value generated by the CNOPsolution Tester is -6961.813875580138. The decision variable values provided by the authors indicate that the objective function is feasible and meets the constraints and limits of the decision variables (see Fig. 12).

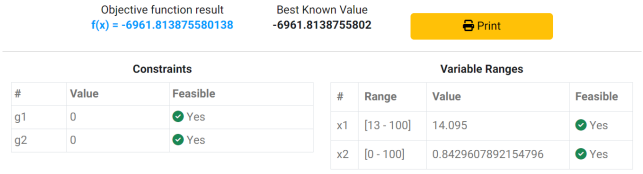

Figure 12: Validation of problem g06 in CNOPsolution Tester.

- **Problem g13:** The g13 problem is an optimization problem with 9 decision variables and 13 inequality constraints. The decision variable values provided by the authors are: $x = [-0.65777619242 79, -0.153418773482, 0.323413871675, -0.94625761165, -0.6577761943767, -0.7532134346326, 0.3234138741235, -0.3464629479623, 0.599794662852]$. The objective function provided by the authors is -0.8660254038 and the value generated by *CNOPsolution Tester* is -0.8660254037844387. The decision variable values provided by the authors indicate that the objective function is feasible and meets the constraints and limits of the decision variables (see Fig. 13).

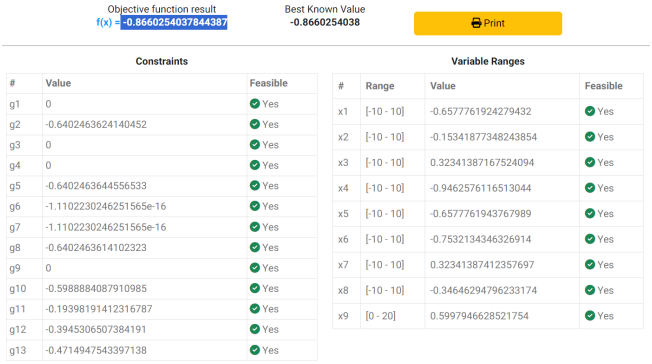

Figure 13: Validation of problem g13 in CNOPsolution Tester.

Based on what has been presented in this section, the CNOPsolution Tester is an efficient tool for validating the values of decision variables in terms of their ranges and constraints. For all tests performed with values existing in the literature, the results generated by the CNOPsolution Tester match the values provided by the authors. Additionally, the web application can manage the load of multiple users and provides a usage guide for the end-user.

5 Conclusion

In this work, we have presented CNOPsolution Tester, a web application that allows researchers and end-users to validate the values of decision variables based on their minimums, maximums, and constraints. We have designed and developed the web application to facilitate the validation of CNOP results and reduce result validation time. We have preloaded a set of CNOPs into the CNOPSolution Tester as if the end-user entered custom problems. We also host the web application on GitHub Pages for easy access, and multiple users have tested it to evaluate its performance. Additionally, we implemented a user guide for the end user, and it has been validated with problems from the literature.

We also highlight that, unlike previous works such as Kumar et al. (2020) and Brest et al. (2017), which focus on the development and benchmarking of metaheuristics, CNOPsolution Tester provides a dedicated tool for validating solution vectors against variable bounds and constraints. While some researchers may use general-purpose tools like spreadsheets (e.g., MS Excel) or custom scripts to manually verify constraints, such approaches are time-consuming and prone to error. CNOPsolution Tester addresses this gap by offering a user-friendly, web-based interface specifically designed for validating constrained optimization results efficiently.

The results demonstrate that the CNOPsolution Tester is an efficient tool for validating the values of decision variables based on their minimums, maximums, and constraints. It has been tested and evaluated with 250 end-users. Additionally, Pearson's correlation was implemented to identify the relationship of the number of clicks with the problem's variables and constraints.

6 Future work

In future work, the web application is planned to be expanded to include more optimization problems and add functionalities for evaluating multi-objective optimization problems.

Acknowledgments

To the SECIHTI (Ministry of Science in México) for supporting the Doctoral program in Computer Science at the Universidad Juárez Autónoma de Tabasco.

Declarations

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abdel-Basset, M., Abdel-Fatah, L. and Sangaiah, A. K. (2018). Chapter 10 - metaheuristic algorithms: A comprehensive review, *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, Academic Press, ., pp. 185–231. https://doi.org/10.1016/B978-0-12-813314-9.00010-4.

Barbosa, H. J. C., Bernardino, H. S. and Barreto, A. M. (2010). Using performance profiles to analyze the results of the 2006 cec constrained optimization competition, *IEEE Congress on Evolutionary Computation*, pp. 1–8. https://doi.org/10.1109/CEC.2010.5586105.

Brest, J., Maučec, M. S. and Bošković, B. (2017). Single objective real-parameter optimization: Algorithm jso, *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1311–1318. https://doi.org/10.1109/CEC.2017.7969456.

de Jong, J. and Mansfield, E. (2018). Math.js: An advanced mathematics library for javascript, *Computing in Science & Engineering* **20**(1): 20–32. https://doi.org/10.1109/MCSE.2018.011111122.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**(1): 29–41. https://doi.org/10.1109/3477.484436.

Eiben, A. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*, Springer-Verlag, Natural Computing Series. https://doi.org/10.1007/978-3-662-44874-8.

Engelbrecht, A. (2005). *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons.

Fogel, D. B. (2006). *Evolutionary computation: toward a new philosophy of machine intelligence*, 3 edn, Wiley-IEEE Press, Piscataway, NJ.

García-López, A., Chávez-Bosquez, O., Hernández-Torruco, J. and Hernández-Ocaña, B. (2023). Jmetabfop: A tool for solving global optimization problems, *SoftwareX* **23**: 101452. https://doi.org/10.1016/j.softx.2023.101452.

Garey, M., Johnson, D. and Stockmeyer, L. (1976). Some simplified np-complete graph problems, *Theoretical Computer Science* **1**(3): 237–267. https://doi.org/10.1016/0304-3975(76)90059-1.

Hernández-Ocaña, B., Pozos-Parra, M. D. P., Mezura-Montes, E., Portilla-Flores, E. A., Vega-Alvarado, E. and Calva-Yáñez, M. B. (2016). Two-swim operators in the modified bacterial foraging algorithm for the optimal synthesis of four-bar mechanisms, *Computational intelligence and neuroscience* **2016**: 17. https://doi.org/10.1155/2016/4525294.

Jansen, P. (2024). Tiobe index for november 2024, Online article. Available at https://www.tiobe.com/tiobe-index/.

Karaboga, D. and Basturk, B. (2007). Powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm, *Journal of Global Optimization* **39**(3): 459–471.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4, pp. 1942–1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968.

Koza, J. (1992). On the programming of computers by means of natural selection, *Genetic programming* .

Kumar, A., Wu, G., Ali, M. Z., Mallipeddi, R., Suganthan, P. N. and Das, S. (2020). A test-suite of non-convex constrained optimization problems from the real-world and some baseline results, *Swarm and Evolutionary Computation* **56**: 100693. https://doi.org/10.1016/j.swevo.2020.100693.

Liang, J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello, C. A. C. and Deb, K. (2006). Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization, *Technical report*, School of EEE Nanyang Technological University, Singapore.

Mezura-Montes, E. and Coello, C. A. C. (2011). Constraint-handling in nature-inspired numerical optimization: Past, present and future, *Swarm and Evolutionary Computation* **1**(4): 173–194. https://doi.org/10.1016/j.swevo.2011.10.001.

Passino, K. (2002). Biomimicry of bacterial foraging for distributed optimization and control, *IEEE Control Systems Magazine* **22**(3): 52–67.

Pearson, K. and Galton, F. (1895). Note on regression and inheritance in the case of two parents, *Proceedings of the Royal Society of London* **58**(347–352): 240–242. https://doi.org/10.1098/rspl.1895.0041.

Rechenberg, I. (1989). Evolution strategy: Nature's way of optimization, *Optimization: Methods and Applications, Possibilities and Limitations: Proceedings of an International Seminar Organized by Deutsche Forschungsanstalt für Luft-und Raumfahrt (DLR), Bonn, June 1989*, Springer, pp. 106–126. https://doi.org/10.1007/978-3-642-83814-9_6.

Sarker, R. A. and Newton, C. S. (2007). *Optimization Modelling: A Practical Approach*, 1 edn, CRC Press, Boca Raton. https://doi.org/10.1201/9781420043112.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, *Journal of global optimization* **11**(4): 341–359.