XLVI CILAMCE
VITÓRIA - ES - BRAZIL - 2025
XLVI IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING

Coupling visual and coded programming in computationally expensive optimization problem applied to an energy performance problem

Henrique A. Lopes[1], Iuri P. Vergínio[2], Rafael P. Garcia[2]

IT Dept.[1]; Dept. of Architecture and Urban Planning[2], Federal University of Viçosa
Viçosa, 36570-900, Minas Gerais, Brazil

Abstract. The demand for computational optimization tools in architecture has grown significantly in recent years. Among the most prominent tools is the Grasshopper plugin for Rhinoceros, which provides a visual programming environment for parametric design. However, the most significant advances in optimization are typically achieved through coded-based programming, relegation visual-based approaches to a second role. A recurring problem in architecture is determining the energy performance of buildings, specifically the one evaluated in this article, as an optimization problem that uses computer simulations to calculate its fitness function. This problem is formulated according to 4 variables: building orientation, transmittance of shading elements, and width of the openings to the east and west. The objective is to find values for these variables that minimize the building's energy consumption. In this work, we evaluate an implementation flow for coding in Python within the Rhinocero+Grasshopper visual programming platform to implement an optimization algorithm based on Differential Evolution. Given the high computational cost associated with executing the algorithm (mainly due to its dependence on simulations), an artificial neural network was implemented to predict the value of the fitness function of some solutions. For this coupling, we used the GHPythonRemote tool, which allows the integration of libraries developed in Python that are not natively available in Grasshopper, such as, for example, scikit-learn. Therefore, the purpose of this article is to demonstrate the optimization process through the coupling of visual and coded programming in a computationally expensive architectural optimization. The results obtained using this approach are compared to those from Galápagos, a native genetic algorithm-based optimization plugin in Grasshopper. The results demonstrate the potential of coupling by making the implementation of algorithms more flexible, in addition to contributing to improving the optimal solution of the problem and reducing the computational cost of the optimization process by allowing the application of machine learning methods.

Keywords: Visual and coded Programming, Differential Evolution, Grasshopper.

1 Introduction

In recent years, different programming languages have been developed and refined, each with their own characteristics and specific areas of application. Coded programming is defined by sequential lines of code that resemble natural language. This approach, although more technical, offers greater flexibility. It allows detailed control over the processes and the functions, the use of advanced libraries, and the creation of customized and complex solutions adapted to the specificities of the problems. Visual programming, on the other hand, is developed with plugins and components connected by "lines" that represent the data flow and the logic of the program. Visual programming is more accessible to professionals in fields such as design and architecture, since it emphasizes intuition and visualization of the process. It simplifies the process of creating and modifying algorithms, making it simpler and more accessible to professionals with less experience in coded implementations.

In the field of Architecture, visual programming has stood out for its ease of use and ability to integrate various design processes (Landim [1]). This trend is closely related to advances in parametric architecture (De Kestelier and Peters [2]). Tools that integrate visual programming into the design process, such as the Rhinoceros+Grasshopper interface, have become a reference due to their intuitive and powerful interfaces. However, more noticeable computational advances are found in coded languages, especially Python. This language

stands out for the large number of available libraries, such as those focused on machine learning.

Several studies in scientific literature have used visual programming in methodologies involving simulation and optimization applied to building performance issues, such as energy efficiency (Waibel et al. [3]; Turrin et al. [4]). However, these processes are notoriously known for being computationally expensive, mainly due to the use of simulations, which limit the scalability and applicability of the studies (Silva et al. [5]). An alternative to getting around this problem is to use tools that allow some solutions to be evaluated predictively instead of simulating them. Machine learning techniques, such as artificial neural networks (ANN), are used for this purpose and are freely available in Python libraries. The use of predictive techniques based on machine learning has proven increasingly effective in architecture, urban planning, and engineering, especially in improving processes that traditionally depended on complex and time-consuming computational simulations. For the energy performance of buildings, this type of technique has already been studied in works such as those by Melo et al. [6] and Lu et al. [7]. This type of approach allows architects to explore alternative and optimal design solutions, simultaneously evaluating them according to the building performance they wish to achieve.

This work aims to demonstrate how coupling visual and coded programming, particularly Python and Grasshopper, can improve both solution quality and computational efficiency in building energy optimization problems. The proposal includes the implementation of optimization algorithms within the Rhinoceros+Grasshopper platform, using the integration of Python libraries through the GHPythonRemote tool (Cuvilliers [8]). As a result of this coupling, an optimization plugin based on Differential Evolution (DE) will be created for use in Grasshopper, and an ANN will be used to predict the fitness value of some solutions. The combination of DE and ANN opens the possibility of a significant reduction in the processing time of complex optimizations, since the ANN can predict simulation results more quickly by having a robust database of previously simulated solutions available. This allows a greater number of solutions to be explored in a reduced time, increasing the chances of finding more energy-efficient design configurations without overloading computational resources. In this way, the proposed methodology aims to contribute directly to sustainability and efficiency in architectural design, aligning with contemporary demands for more intelligent and environmentally responsible buildings.

2 Methodology

The coupling between visual and coded programming languages will be analyzed through the implementation of optimization algorithms and applied to an energy efficiency problem. In general, this work used the potential of both languages to implement an optimization algorithm called Differential Evolution (Price [9]) and applied it to an energy consumption problem of an office building. The results were compared with those produced by a genetic algorithm-based optimizer natively available in Grasshopper, Galápagos (Grasshopper Docs [10]). The analyses obtained from this comparison were able to draw conclusions about the advances produced by the coupling of the languages, both in terms of the quality of the optimized solutions and the savings in computational time. Finally, a new optimization algorithm, now incorporating predictions throughout the DE, was created and more computational advances could be noted.

2.1 Problem Description

The optimization problem adopted is based on the benchmark proposed by Wetter and Wright [11], which aims to minimize the annual consumption of air conditioning systems in an office building. In this problem, an office floor is used (Fig. 1), totaling a thermal zone with 6 meters wide and 16 meters deep, and is analyzed based on meteorological data from the city of Seattle, USA. Four parameters make up a solution to this problem: building orientation (-180° to 180°); width of openings facing east (0.1 to 5.9 meters); width of openings facing west (0.1 to 5.9 meters); and the transmittance of the shading elements (0.2 to 0.8). As described in Silva et al. [12], regarding the materials used in the simulation, the external walls have a thermal transmittance of 0.25 W/(m²K), while the windows of the openings are of the low-e type insulated with Krypton gas, equipped with external shading louvers that are automatically activated during the summer if the radiation in the openings exceeds 200 W/m². The internal walls were modeled as 12 cm thick solid bricks and both the floor and ceiling were treated as concrete slabs with insulating material and carpet. The simulator available in Waibel et al. [13] was used to calculate annual energy

consumption.

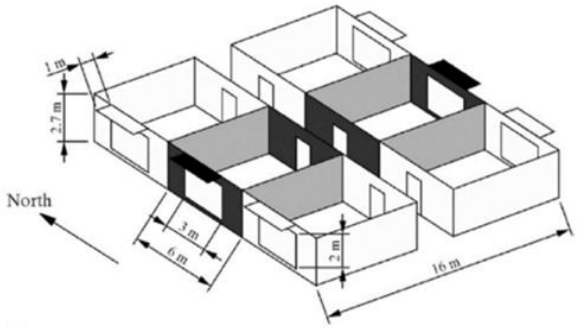


Figure 1. 3D Model of the office building

2.2 Coupling visual and coded languages

Some challenges arise when integrating the Python language into the Grasshopper implementation steps. One of the main ones is ensuring compatibility between existing Python libraries and the visual programming environment. This obstacle is overcome by using the GHPythonRemote plugin (Cuvilliers and Mueller [8]). This tool allows direct communication between Grasshopper and Python and facilitates the execution of scripts and the use of external libraries without compromising the fluidity of the visual optimization process. However, adjusting the data flow between the two environments and ensuring computational efficiency during execution is another challenge that requires careful refinement of the communication between the platforms. The first step in this refinement was to install Anaconda (Rolon-Mérette et al. [15]), a Python distribution focused on scientific computing that includes machine learning and large-scale data processing techniques. Next, we created a virtual environment with Python 2.7 and installed all the necessary libraries, such as scikit-learn and pandas. Finally, we installed GHPythonRemote in the virtual environment. This process can be accomplished by typing the commands "pip install gh-python-remote --upgrade" and "python -m ghpythonremote._configure_ironpython_installation" in the virtual environment's terminal.

To configure the GHPythonRemote plugin in Grasshopper, some input parameters must be provided, including: (i) LOCATION, which is the path to where the virtual environment was created for Rhinoceros; (ii) RUN, where you define whether the plugin will be executed; (iii) MODULES, where you enter the name of the imported Python libraries; (iv) LOG_LEVEL, which specifies the level of log details (such as DEBUG, INFO, WARNING, ERROR, and CRITICAL); and (v) WORKING_DIR, the directory where the Python scripts and related files are located.

To ensure proper functioning of the process, the BEOBenchmark simulator must also be installed and configured. This simulator is available on GitHub at christophwaibel/BEOBenchmark, where instructions for its installation and configuration are also described. GHPythonRemote interacts with the simulator by saving data locally on the user's computer and managing the flow of information between the simulation and the differential evolution algorithm. It controls the "path" of the solutions and facilitates communication between the visual and coded environments. Additionally, it prepares the system for subsequent implementation of the ANN.

2.3 Coupling flowchart between visual and coded language for differential evolution implementation

Inspired by Darwinian evolution, evolutionary optimization algorithms use selection, crossover, and mutation to evolve a population of solutions toward optimality. They simulate natural selection by evaluating solution quality with a fitness function and generating new generations from the best individuals. The fitness function may be based on the objective function, combined with penalties, or, in this work, a simulator. Crossover explores existing traits, while mutation introduces random variations to maintain diversity and avoid stagnation. Galapagos is a genetic optimizer plugin native to Grasshopper which can be fully configurable via visual programming. It has shown effective results in studies by Wetter and Wright [11], and Gerbo and Saliklis [14]. Differential Evolution (Price [9]) is a simple and effective evolutionary algorithm, though not yet widely used on the

Rhinoceros+Grasshopper platform. For each new solution x_p, three random solutions x_1, x_2 and x_3 generate a mutated solution $x_t = x_1 + F(x_2 - x_3)$, with F = 0.7. A crossover with 80% probability combines x_p and x_t to produce x_o. If x_o has better fitness than x_p, it replaces x_p in the next generation.

We used Grasshopper in Rhinoceros as a visual programming implementer and Python as the coding language. It is worth noting that Grasshopper provides natively a plugin that allows the execution of Python scripts, although it has limitations regarding the use of advanced libraries, such as scikit-learn and pandas. The coding language C# was also used, but in a secondary role, to integrate the visual and Python implementations and to create a new visual DE plugin for direct use in Grasshopper.

The parameters required for the execution of the DE plugin, such as population size (pop), scale factor (sf), crossover probability (cp) and number of iterations (n), are defined by the user directly in Grasshopper using visual tools. The flowchart in Fig. 2 illustrates the implementation process of the DE algorithm in the Grasshopper environment and its interaction with the simulator. The first step is the generation of the initial population by the DE plugin which creates a set of solutions based on the parameters of the plugin (coded language). Then, the DE plugin generates new solutions at each iteration cycle, applying evolutionary operators to refine the solutions (coded language). After generating new solutions, the flow is divided into two options: simulate or end. If the number of iterations has not yet been reached, the new solution is passed on to the simulator to evaluate its fitness value (visual language). The simulator performs the necessary calculations and returns the fitness value to the DE plugin. Otherwise, if the number of iterations has been reached, the flow ends, and the data generated during execution is saved for future analysis (coded language). If the path followed is "simulate", the flow returns to the DE plugin with the fitness value of the new solution. The DE plugin uses this information to create the next generation of solutions, and the iteration process continues until the defined number of iterations is reached.

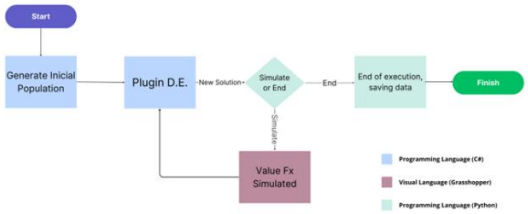


Figure 2. DE flowchart

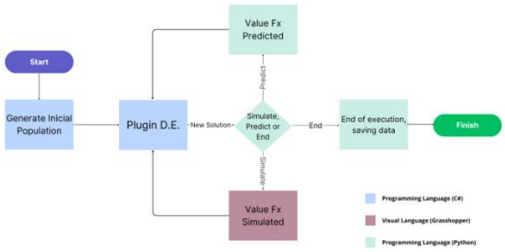


Figure 3. Flowchart of ANN coupled to DE

2.4 Differential evolution supported by an artificial neural network

Population-based evolutionary algorithms inherently involve generating many solutions, each requiring evaluation of their fitness value. As mentioned before, evaluating solutions via simulator is, in most cases, computationally expensive. An alternative to reduce the number of simulator evaluations is to create a database of previously evaluated solutions and use it as a training set for future ones. Therefore, this work also proposes a DE configuration integrated with an ANN, in which 60% of candidate solutions are evaluated by the simulator, while the remaining 40% have their fitness values predicted by the ANN. This approach is a modification of the flowchart presented in Fig. 2, now systematized in Fig. 3. The initial population will be fully evaluated by the simulator to create an initial database (visual language). Then, for each new solution, a probability factor will be applied to decide whether the evaluation of its fitness function will come from the simulator or the ANN (roulette wheel method, controlled by Python code). Each new solution evaluated by the simulator is added to the database, increasing the robustness of the predictions over the generations.

The choice of ANN is supported by findings of Lu et al. [7], whose literature review highlights the ability of ANNs to model complex, non-linear relationships and uncover hidden patterns in large datasets. The implementation of the ANN was performed using the GHPythonRemote plugin, which enables the use of scikit-learn and pandas libraries for training purposes. For this problem, an ANN with 3 hidden layers, each containing 30 neurons, was implemented using the MLPRegressor network. The simulated solutions, along with all variables and fx values, are saved locally by a Python script. This file is then used by the ANN as a training database.

In addition, a new strategy was implemented to improve the control and quality over the predictions made by

ANN. Instead of stopping the simulations after a certain point in the evolution and predicting all subsequent solutions, as done by Silva et al. [12], this work implements a roulette-based selection method (Lipowski and Lipowska [16]) to randomly decide whether a solution will be simulated or predicted. This approach provides flexibility in adjusting the proportion of simulations and predictions throughout the evolutionary process, avoiding overfitting and ensuring greater accuracy in optimization. The call for prediction or simulation is made by another Python script that controls the simulation and prediction rate using the roulette wheel selection method and ensuring that, in the end, 60% of the solutions will be simulated and 40% will be predicted.

3 Results

The first analysis involved comparing the results produced by Galápagos (visual programming) and DE (visual and coded programming) applied to the energy performance optimization problem. A total of 10 independent runs were performed for each optimizer, with 500 iterations (simulator evaluations) used as the stopping criterion for each run. A total of 30 solutions were considered in the populations. For DE, a scaling factor of 70% (sf = 0.7) and a crossover probability of 80% (cp = 0.8) were used. For Galápagos, the default values for its parameters were used. Then, after analyzing these results and the conclusions obtained from them, we moved on to the next comparisons involving the results obtained by DE and the coupling between DE and ANN.

3.1 DE vs. Galápagos

First, we conducted an overall analysis of the best results found by DE and Galápagos. To do this, we captured the best values from each of the 10 runs for each optimizer. The statistics of these results can be found in Tab. 1. For all of them, DE finds better values.

Table 1. Statistics with the best results found in the 10 executions

Optimizer	Best	Average	Standard Deviation	Worst
Galápagos	133.092	134.125	0.811	135.563
DE	**132.993**	**133.194**	**0.100**	**133.339**

To analyze the execution that obtained the best solution for each optimizer, the convergence graph shown in Fig. 4(a) was generated. Although the initial population of DE starts from higher values, it converges quickly, reaching significant values already in the 10th generation. It reaches the optimal value in the 14th generation. On the other hand, Galápagos converges more slowly and produces its best solution only in the 17th generation.

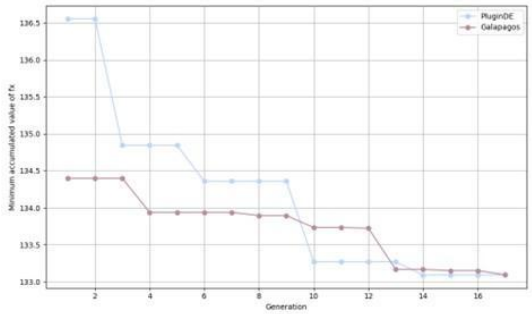


Figure 4. a) Convergence curve of the best execution

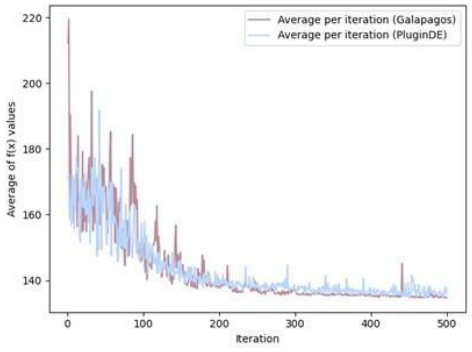


Figure 4. b) Average Convergence over 10 runs

To generalize the convergence analysis of the algorithms' executions, the graph in Fig. 4(b) was produced. It plots the average fitness considering the 10 executions. The convergence of DE exhibited more "peaks" and variations throughout the iterations. In contrast, Galápagos shows signs of stability after 300 iterations. This suggests that DE, even in more advanced stages of evolution, continues to explore the search space more extensively, while Galápagos avoids abrupt changes. This continuous exploration characteristic of DE can be a

significant advantage in problems where the optimal solution is in hard-to-reach regions of the search space. This behavior suggests that DE achieves more promising results when it is given more time for its population to converge.

Overall, the results show promising signs for coupling visual and coded languages, since DE not only found better results when compared to Galápagos, but also achieved faster convergence. Furthermore, we observed that DE tends to explore the search space more effectively, even in advanced stages of evolution. This behavior is one of the motivations for coupling ANN to ED, allowing the algorithm to continue exploring the search space for more generations.

3.2 DE vs. DE+ANN

To evaluate the performance of the ANN when coupled with DE, we randomly selected 30 solutions that were predicted during the evolution process and evaluated them through simulation. From the graph in Fig. 5(a), we see that although the ANN does not achieve extremely high accuracy, it does show signs of learning. The ANN tends to predict higher values for larger fitness function values and lower values for smaller peaks, indicating that it is capturing the underlying trends in the data. On average, the ANN achieves a RMSE (Root Mean Square Error) of 2.484 and a mean error of 1.507.

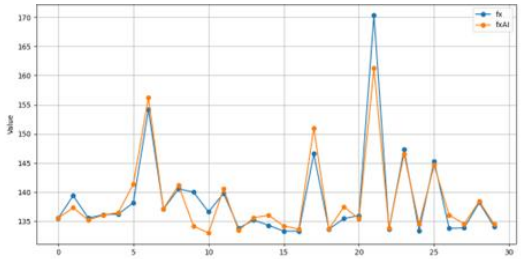


Figure 5. a) Comparison between predicted (fxAI) and real value (fx)

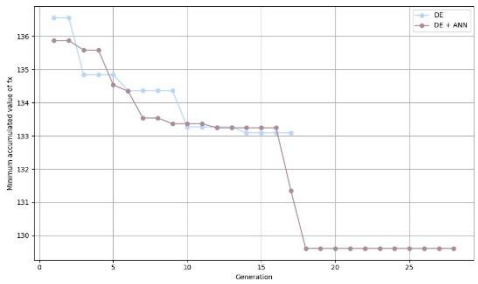


Figure 5. b) Average Convergence of Iterations

Fig. 5(b) compares the convergence of DE and DE coupled to ANN (DE+ANN). It should be noted that the convergence of DE+ANN extends over more generations, saving simulations in previous generations. Despite this advantage, DE+ANN obtains as the best solution an underestimated value, with prediction errors. This phenomenon is a limitation of this coupling, and new approaches should be studied to overcome it. Even so, when comparing the convergence trend between the two algorithms until approximately the 16th generation, DE+ANN converges at a pace compatible with DE. This makes DE+ANN a competitive approach, since its evolution could have been interrupted at the end of DE, which would result in savings in computational time and robust solutions.

4 Conclusions

The results showed that DE produced better solutions and good convergence when compared to the solutions produced by Galápagos. In addition, we showed that the coupling between the languages also allows the incorporation of prediction strategies, by inserting an ANN throughout the DE generations. This strategy shows promising signs of saving computational resources by producing good solutions with fewer simulations.

The strategy of randomly selecting a solution to be predicted, within the rate previously defined by the user, allows the prediction process to operate from the second generation of the evolution to the last. This behavior was intrinsically beneficial for the quality of the predicted solutions since, as the solutions evolve over the generations, the convergence of the population within the search space can change drastically and the database may not keep up with this change. Updating the database throughout the evolution, by constantly receiving new simulated solutions, contributes to the next predicted solutions receiving more accurate values.

However, future studies should investigate ways to mitigate the limitations observed in the DE+ANN strategy, exploring alternatives that can avoid the production of underestimated solutions in advanced generations. Other approaches, such as dynamically adjusting the ANN parameters or integrating cross-validation techniques

throughout the evolutionary process, may be promising to overcome this challenge and further improve the efficiency of the algorithm.

Acknowledgements. The authors would like to thank the Brazilian funding agency CNPq for its support - Project Number 210526 (UFV/PIBIC/CNPq 2023 – 2024).

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] LANDIM, Gabriele do Rosario. Programação para Arquitetura: linguagens visuais e textuais em Projeto Orientado ao Desempenho. Dissertação (Mestrado em Arquitetura e Urbanismo). Universidade de São Paulo, São Paulo, 2019.
[2] DE KESTELIER, X.; PETERS, Brady. Computation works: the building of algorithmic thought. 1.ed. John Wiley & Sons, New Jersey, 2013.
[3] WAIBEL, Christoph; EVINS, Ralph; CARMELIET, Jan. Co-simulation and optimization of building geometry and multi-energy systems: Interdependencies in energy supply, energy demand and solar potentials. Applied Energy, Porto, v. 242, p. 1661-1682, 2019.
[4] TURRIN, Michela; VON BUELOW, Peter; STOUFFS, Rudi. Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. Advanced Engineering Informatics, Singapore, v. 25, n. 4, p. 656-675, 2011.
[5] SILVA, Mario Alves da; GARCIA, Rafael de Paula; CARLO, Joyce Correna. Performance assessment of RBFMOpt, NSGA2, and MHACO on the thermal and energy optimization of an office building. Journal of Building Performance Simulation, London, v. 16, n. 3, p. 366-380, 2023.
[6] MELO, A. P. et al. Development of surrogate models using artificial neural network for building shell energy labelling. Energy Policy, London, v. 69, p. 457-466, 2014.
[7] LU, Chujie; LI, Sihui; LU, Zhengjun. Building energy prediction using artificial neural networks: A literature survey. Energy and Buildings, Amsterdam, v. 262, p. 111718, 2022.
[8] CUVILLIERS, P.; MUELLER, C. ghpythonremote (versão 1.4.6). GitHub, 2022. Software. Disponível em: <https://github.com/pilcru/ghpythonremote>. Acesso em: 20 de julho de 2024.
[9] PRICE, K. V. Handbook of optimization: From classical to modern approach. 1.ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 187-214.
[10] GRASSHOPPER DOCS. Grasshopper documentation. Grasshopper Docs, 2024. Disponível em:<https://grasshopperdocs.com>. Acesso em: 20 de julho de 2024.
[11] WETTER, M.; WRIGHT, J. A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization. Building and Environment, Hong Kong, v. 39, n. 8, p. 989–999, 2004.
[12] SILVA, M.; VERGÍNIO, I.; GARCIA, R.; CARLO, J.. Redução do custo computacional do processo de otimização baseada em simulação utilizando redes neurais aplicadas a um problema de consumo energético com sistemas de condicionamento artificial. In: ENCONTRO NACIONAL DE TECNOLOGIA DO AMBIENTE CONSTRUÍDO, 19., 2022, Canela. ANTAC. 2022.
[13] WAIBEL, Christoph et al. Building energy optimization: An extensive benchmark of global search algorithms. Energy and Buildings, Amsterdam, v. 187, p. 218-240, 2019.
[14] GERBO, E. J.; SALIKLIS, E. P. Optimizing a trussed frame subjected to wind using Rhino, Grasshopper, Karamba and Galapagos. In: Proceedings of IASS Annual Symposia, 13., 2014, Brasilia. IASS. 2014. p. 1-7.
[15] ROLON-MÉRETTE, Damien et al. Introduction to Anaconda and Python: Installation and setup. Quant. Methods Psychol, Ottawa, v. 16, n. 5, p. S3-S11, 2016.
[16] LIPOWSKI, Adam; LIPOWSKA, Dorota. Roulette-wheel selection via stochastic acceptance. Physica A: Statistical Mechanics and its Applications, Amsterdam, v. 391, n. 6, p. 2193-2196, 2012.