WebGE: an open-source tool for symbolic regression using Grammatical Evolution*

J. Manuel Colmenar[1][0000−0001−7490−9450], Raúl Martín-Santamaría[1][0000−0002−9396−5375], and J. Ignacio Hidalgo[2][0000−0002−3046−6368]

[1] Rey Juan Carlos University, Móstoles, 28933, Spain
{josemanuel.colmenar,raul.martin}@urjc.es
[2] Complutense University, Madrid, 28040, Spain
hidalgo@ucm.es

Abstract. Many frameworks and libraries are available for researchers working on optimization. However, the majority of them require programming knowledge, lack of a friendly user interface and cannot be run on different operating systems. *WebGE* is a new optimization tool which provides a web-based graphical user interface allowing any researcher to use Grammatical Evolution and Differential Evolution on symbolic regression problems. In addition, the fact that it can be deployed on any server as a web service also incorporating user authentication, makes it a versatile and portable tool that can be shared by multiple researchers. Finally, the modular software architecture allows to easily extend *WebGE* to other algorithms and types of problems.

Keywords: Grammatical Evolution · Differential Evolution · Symbolic regression · Open-source Software.

1 Introduction

In the field of optimization and, more precisely, in the metaheuristics area, many software frameworks and libraries are available. Researchers may select among different alternatives implemented with different programming languages. For instance, jMetal [12], ECJ [26] or JCLEC-MO [24] are programmed in Java, LEAP [7] or EvoCluster [23] are coded in Python, PlatEMO [28] is coded in MATLAB and predtoolsTS is coded in R [6]. These are several of the most complete and recently available frameworks and libraries with different aims and scopes.

The common features among them are the availability of many algorithms and the possibility of adapting the framework to any target problem. However,

* This work has been partially supported by the Spanish Ministerio de Ciencia, Innovación y Universidades (MCIU/AEI/FEDER, UE) under grants ref. PGC2018-095322-B-C22 and RTI2018-095180-B-I00; and Comunidad de Madrid y Fondos Estructurales de la Unión Europea with grants ref. P2018/TCS-4566, B2017/BMD3773 (GenObIA-CM) and Y2018/NMT-4668 (Micro-Stress - MAP-CM),

there is one mandatory requirement: the researcher must have learnt how to program using the language of the selected framework. Besides, the majority of these libraries and frameworks lack of a friendly and modern graphical user interface (GUI) for an easier use.

Optaplanner is a constraint solver mainly specialized in scheduling and routing problems [10]. It provides different algorithms and metaheuristics like Tabu Search, Genetic Algorithms or Simulated Annealing, among others. The use of Optaplanner requires coding abilities, since any new problem has to be programmed either from scratch or using one of the provided examples as a template. Although Optaplanner provides GUI support for solution visualization for some examples, the experiments configuration and modelling has to be made on the source code.

Perhaps one of the most interesting optimization tools providing a GUI is HeuristicLab [13], which is an integrated environment for heuristic optimization. It includes many different types of algorithms and problem families, allowing the user also to create new problems using a template-based system. However, the main drawback of HeuristicLab is that it cannot be shared among researchers since it lacks of a user authentication system. Besides, it is developed for Windows and its portability to other operating systems like Linux or MacOS depends on third-party elements like Mono.

Therefore, despite that many optimization tools and frameworks are available for a researcher, none of them meet the following requirements: friendly GUI, no need of coding abilities, portability and shareable as a web service.

In this way, a new tool named WebGE, which stands for Web Grammatical Evolution, is proposed in this paper. WebGE provides a web-based GUI for experiments management, where the algorithm tuning can be performed using friendly web forms. Moreover, WebGE is packed using the Docker container technology, which allows WebGE to be run in any operating system. In addition, WebGE can be run on a shared server by multiple researchers, since it implements granular access controls, and it stores all the data and results in a relational database. Finally, WebGE is proposed as an open-source software, already available in `https://github.com/GRAFO-URJC/WebGE`, and specifically designed to be extended to other algorithms and problems.

In its current state of development, WebGE allows the user to work on symbolic regression problems. That is, problems devoted to finding models for a target variable from a given dataset. To this aim, not only does WebGE provide GE as an optimization algorithm, but it also provides the combination of GE with Differential Evolution (GE+DE). Therefore, WebGE can be used in any symbolic regression problem by any researcher with no programming background.

The rest of the paper is organized as follows. First, the integration of GE and DE into WebGE is explained in Section 2. Then, a general software description and the main features of WebGE are detailed in Section 3 and Section 4, respectively. In order to test the proposed application, a well-known benchmark is studied in Section 5. Finally, conclusions are drawn in Section 6.

2 Grammatical Evolution and Differential Evolution

Grammatical Evolution (GE) is a metaheuristic method belonging to the Genetic Programming family [22]. Its main advantage lies in being able to include particular knowledge of a problem into the grammar in order to guide the optimization process. Different GE implementations have been successfully applied to diverse problems like machine learning pipeline optimization [14] [3], feature extraction on accelerometer data [19], glucose forecasting in diabetic patients [16] or energy demand estimation [18].

In particular, many of the works whose aim is to produce models tackle the problem as a symbolic regression process. This process begins with the compilation of a dataset composed by a set of input variables and a target variable to be modelled. This dataset is usually split in two: one part used in the process to obtain the models (training phase) and a second part devoted to assessing the quality of the obtained models (testing phase). The models are generated as mathematical expressions which produce series of data that are compared to the target variable one. The difference between the predicted values and the reference is called error, and there exists multiple metrics to measure it, such as the root mean squared error, the absolute error, the average error or R^2. Since WebGE is focused on this kind of problems, it is designed to ease the process of dataset division into training and test as well as the quality measure process, implementing all the previously mentioned metrics.

The GE implementation included in WebGE comes from the JECO library [1], which has been also used in several works like [16] and [18]. However, due to the modular design of WebGE, any other GE implementation could be integrated in the application.

In addition to GE, WebGE also provides the combination of GE with Differential Evolution (DE) proposed in [8] and called GE+DE. DE [27] is a metaheuristic algorithm very well suited for continuous optimization, which makes it interesting to ensemble with GE. In particular, the approach followed by the GE+DE implementation lets DE take care of the generation of constant values, which is a delicate task in GE [11] [4], and allows GE to focus on the generation of parameterized models.

Fig. 1 shows the optimization cycle performed by GE+DE in WebGE. As it can be seen, all the information related to an experiment is stored in the database. This way, the configuration of the experiment is loaded from the database and the algorithm begins its execution. Here, each generation of individuals in GE consists of a set of parameterized models generated under the guidance of the grammar. This population is sent to DE, which searches for the best parameter values of the models according to the selected objective function calculated on the training data. Once DE finishes executing, the models with parameter values are returned to GE and the evolutionary process is repeated until the maximum number of generations of GE is reached. At the end of the execution, the best model and its corresponding optimized parameters are stored in the database.

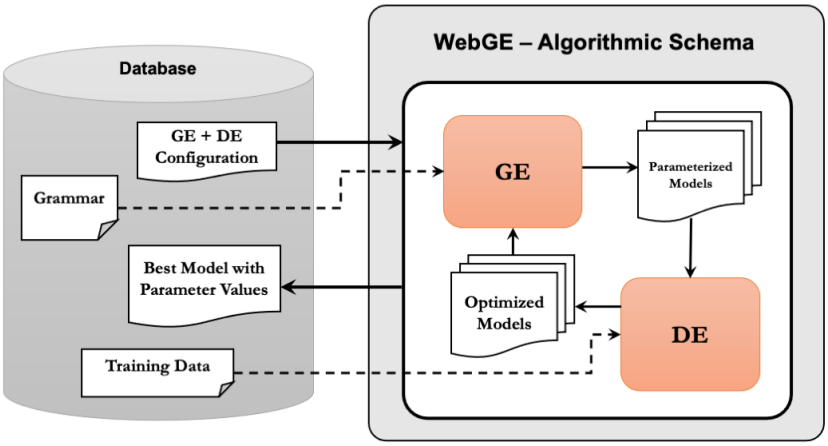

Fig. 1. GE+DE algorithm schema implemented in WebGE.

3 Software description

WebGE has been developed as an open-source project under the GNU General Public License v3.0, and its source code is available at GitHub `https://github.com/GRAFO-URJC/WebGE`. Several students have contributed to this project from the very beginning, some of them working on WebGE for their final degree projects. We would like to acknowledge here their work and note that their contribution is credited at `https://grafo-urjc.github.io/WebGE/`.

Since the project follows the SOLID principles, in particular, *the Single responsibility principle*, the number of classes developed in this project is high, so no class diagram is provided. However, key design aspects and principal components are next described.

3.1 Modular design

WebGE has been developed following a modular design whose aim is two-fold. On the one hand, in terms of code development, it allows the future extension of the functionalities of the application. On the other hand, the division into components allows a more efficient execution, identifying performance bottlenecks.

Fig. 2 shows the modular design of WebGE in terms of component features. In particular, four main components, identified with white background in the figure, were developed. The first one, the *GUI + Endpoints* element, is the set of web-based interfaces which encompasses the communication with the user. These elements required the development of a whole set of endpoints which decouple the implementation of the GUI with the rest of the application, allowing a future extension to different interfaces. The second one is the *Persistence Wrapper*, which includes all the operations related to database storage and retrieval of information. The third one is the *Launcher*, which is the component that takes

the information of the experiments (retrieved from the database by the *Persistence Wrapper*) and creates the experiment tasks to be run upon resources availability. As seen in the figure, the *Launcher* receives the command from the user through the corresponding endpoint. Then, it stores the generated experiment into a waiting queue, where the experiments are kept while no processors are available. Finally, the fourth component is the *Execution Engine*, which is in charge of monitoring the resource availability and to actually run the pending experiments.

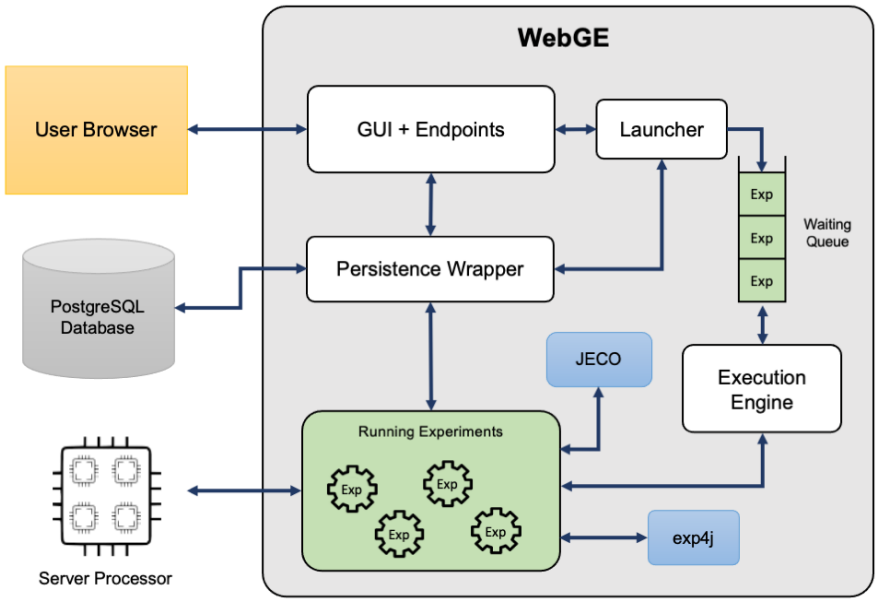

Fig. 2. WebGE components schema.

Notice also that the running experiments make use of two external libraries, identified with blue background in the figure: JECO and exp4j. JECO is the library which provides the evolutionary algorithms, as previously explained. The evaluation of model expressions is performed by exp4j [2]. This library provides a standard set of built-in functions and operators, allowing the developer to create new functions.

3.2 Parallel Execution

One of the most important features of WebGE is how the experiment execution is speed up by an automatic parallel executor. In particular, each experiment run is sliced into a set of execution tasks, which are actually run in parallel taking into account the resources available in the server where it is installed.

As seen in Fig. 2, this process is handled by the *Execution Engine* component, which takes the pending experiment runs from the waiting queue and generates the corresponding executable task.

The queue is implemented using RabbitMQ [17], which provides an asynchronous interface to reliable process all execution tasks. The queue is persistent, which means that in case of application failure or restart, the experiment can continue executing with a minimal loss of work (execution tasks dispatched but not committed are restarted). Moreover, using a neutral message broker such as RabbitMQ allows the application to completely decouple the experiment launch from the actual experiment execution. This is intentional, since a future extension will allow distributed computing to further accelerate experiment execution.

3.3 Persistence layer

WebGE stores all the information related to users, experiments and datasets in a PostgreSQL relational database. Fig. 3 shows the relational diagram of its current state of development. As seen in the figure, experiments and runs are separated into different entities since an experiment with a given set of parameters and input data, may be executed several times. Hence, each execution is considered a *run*, and all its related information is associated in the database. In addition, datasets and grammars are also separately stored, in order to allow the users to perform the typical CRUD (create, read, update, delete) operations. The information related to the user and the session are also stored in the database.

3.4 Implementation technologies

Regarding the implementation technologies, WebGE is a Spring Boot application whose persistence layer relies on Spring Data [9], using Flyway to control the evolution of the database design [25] (Fig. 3 shows the corresponding history table). Besides, WebGE is packed as a Docker container, providing a *docker-compose* template to ease the deployment process. Notice that the use of Docker is currently considered one of the best practices for reproducible experimentation [5]. The latest Docker images of WebGE are also available at Docker Hub (`https://hub.docker.com/r/jmcolmenar/webge`) where releases are automatically generated by the continuous integration server, which allows a more efficient and less error-prone development process [20].

4 WebGE most relevant features

Probably the most important feature of WebGE is that it provides a friendly web-based GUI supported by database storage. In addition, some other important features such as the integrated cross-fold validation and the detailed statistics are described in this section.

Fig. 3. WebGE relational database design.

4.1 GUI for experiments management

The design of WebGE pivots around the concept of experiment. An experiment includes four types of elements: algorithmic parameters, input data, complementary attributes and list of runs.

Fig. 4 shows the experiment configuration interface for the example experiment described in Section 5. As it can be seen, the algorithmic parameters are shown under the *Properties of the experiment* label. The user is able to select the typical parameters for GE such as number of generations, population size, crossover and mutation probabilities, etc., and a specific field for the grammar, which is stored within the experiment. Besides, there is an additional interface for grammar management which allows creating, deleting and listing stored grammars. Notice that the *Copy* button and the combo box on its left allows the user to select and copy a stored grammar. In addition to these parameters, four different objective functions are available: root mean squared error (RMSE), mean squared error, absolute error and R^2. Finally, the hybrid GE+DE algorithm is also available, and the GUI allows tuning its own parameters.

The input data elements correspond to the training and test datasets. As shown in the snapshot, these datasets can be selected in the lower part of the form. As in the case of the grammars, a special interface to upload, delete and list datasets is also available. If a k-fold cross-validation is required, the user may *fold* a dataset when uploading it, indicating the number of folds to divide the data. If a folded dataset is selected for training and cross-validation run is marked, the number of runs is automatically set to the number of folds and no test dataset can be selected. On the contrary, if no cross-validation is indicated or the training dataset is not folded, it is possible to select a test dataset, as shown in the figure.

Complementary attributes can also be incorporated to the experiment configuration area. These attributes are the name and description of the experiment, which are mandatory elements shown in the upper part of Fig. 4, and also the tags. Adding tags to the experiments allows an easier search in the list of experiments.

Once an experiment is configured, it can be run by clicking on the *Save and run experiment* button. At that point, a list of runs is displayed as shown in Fig. 5. The buttons on the list allow the user to check the evolution and the results of a run, stop any (or all) runs, or delete finished runs.

Fig. 6 shows an example of the interface that displays the evolution of a run. In the upper part of the figure, a plot with the best individual cost function is displayed. In the lower part, the execution report generated by the optimization algorithm is also shown.

4.2 Cross-fold validation

In symbolic regression problems it is usual to perform cross-validation [15]. Therefore, WebGE incorporates the leave-one-out cross-validation. This procedure is automatically implemented in two ways: providing the ability to fold the

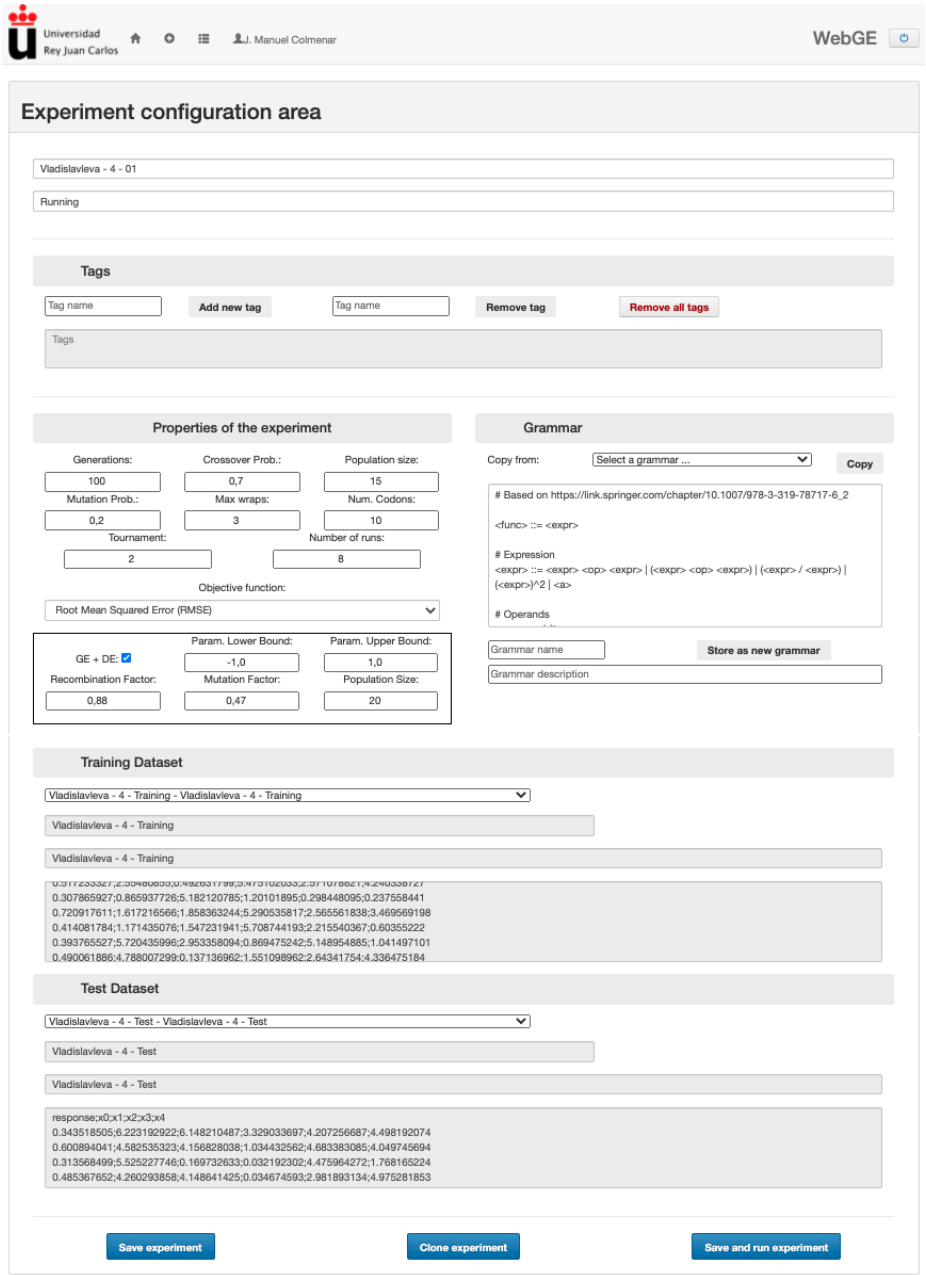

Fig. 4. Experiment configuration interface.

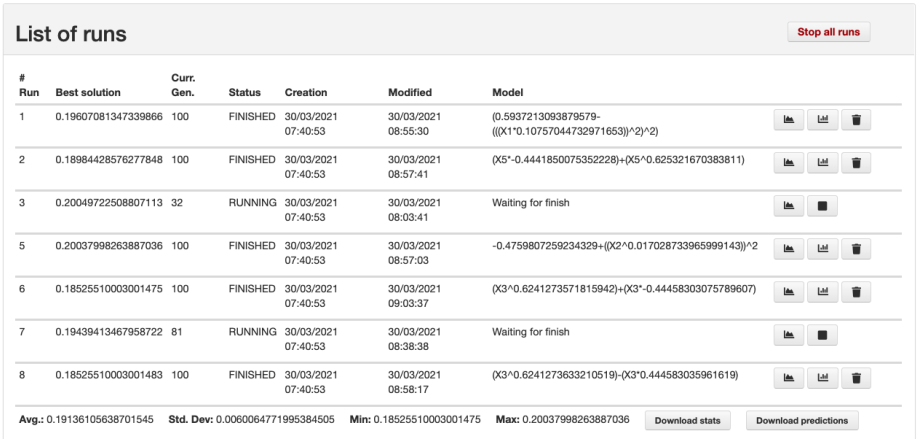

Fig. 5. List of runs for an experiment currently executing.

Fig. 6. Run progression interface.

dataset and automatically configuring the runs of the experiment to use the folds accordingly.

Regarding the folding of the dataset, this operation can be performed either when uploading the data file or once the data are stored. The user may select the number of folds (k) and, as recommended in the literature, WebGE randomly distributes the data in k folds of similar size.

If a folded training set is selected, a researcher may choose the cross-validation run. In this case, WebGE automatically configures the algorithm to execute k runs where, in run i, fold i is used for test while the rest of folds are used for training.

4.3 Detailed statistics

Once a run is finished, the researcher may access to the detailed statistics of the model obtained in the run. Fig. 7 shows a snapshot of this feature. As it can be seen, the average error, root mean squared error, absolute error, relative error and R^2 metrics are calculated for both the training and test datasets. A plot of the data generated by the model is also available for training and test, which is zoomed in the figure.

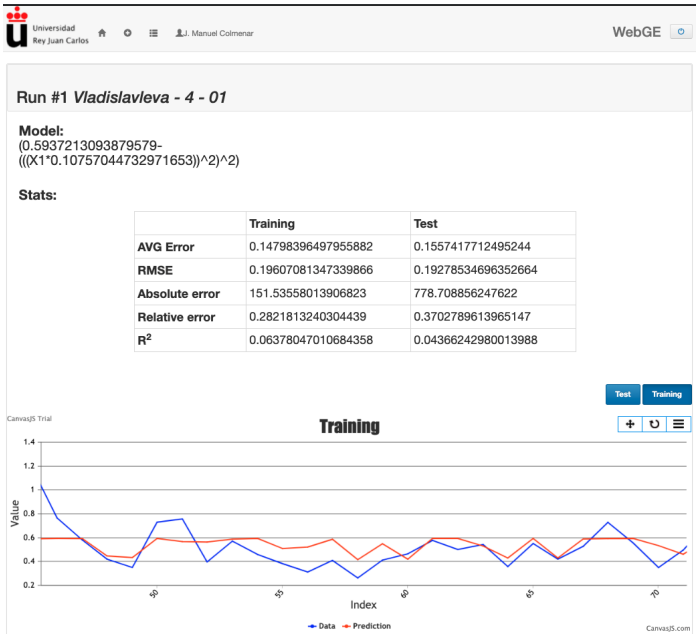

Fig. 7. Run statistics interface.

Moreover, in order to ease the work with the results, the user may download a CSV file with the detailed statistics obtained in all runs. This file is retrieved

when clicking on the *Download stats* button shown in the lower part of Fig. 5. In addition, a CSV file with the predictions given by the models can be downloaded with the *Download predictions* button.

5 Use case: Vladislavleva-4

In order to illustrate the use of WebGE, the well-known symbolic regression benchmark Vladislavleva-4 [29], also known as UBall5D, will be used. It is a synthetic benchmark with five input variables, where the target value is obtained with Equation (1). The training dataset has 1024 entries where the five input variables values belong to the range $(0.05, 6.05)$. The validation dataset has 5000 entries and their input variables values belong to the range $(-0.25, 6.35)$.

$$\frac{10}{5 + \sum_i^5 (x_i - 3)^2} \tag{1}$$

In this example, the GE+DE algorithm is run using the parameter values shown in Fig. 4. The grammar used in this experiment is an adaption of the grammar for Vladislavleva-4 proposed in [21] where the constant value generation of GE has been replaced with parameters whose values will be explored by DE. Fig. 8 shows the complete grammar where `w1` to `w4` are the parameters to be explored by DE and `X1` to `X5` are the input variables from the dataset.

```
<func> ::= <expr>

<expr> ::= <expr> <op> <expr> | (<expr> <op> <expr>) |
           (<expr> / <expr>)  | (<expr>)^2           | <a>

<op> ::= +|-|*

<a> ::= (<var>^w1) | (<var> + w2) | (<var> * w3) | <var> | w4

<var> ::= X1|X2|X3|X4|X5
```

Fig. 8. Grammar for the Vladislavleva-4 benchmark.

After the experiment execution, WebGE was not able to find the optimal solution. However, it obtained an average RMSE value of 1.4643 for validation. This result is slightly worse than the one presented in [29], but using much less computational effort. However, the aim of this use case is to illustrate the ease of use of WebGE (note that all figures shown in this paper come from this use case) and not benchmarking the underlying JECO library.

6 Conclusions

In this paper, WebGE, an open-source tool for symbolic regression problem optimization based on Grammatical Evolution (GE), is presented. WebGE provides

a friendly web-based user interface which allows researchers with no programming background to deal with this kind of problems. WebGE also includes as optimization algorithm the combination of GE with Differential Evolution, which allows a greater intensification of the search process.

WebGE can be easily deployed on any operating system since it has been packaged using the Docker container technology. In addition, all the information about experiments is synchronized to the persistence layer in real time, which allows WebGE to be used on a shared server by several concurrent users. The design of WebGE is modular and extensible, which allows the future integration of different algorithms to tackle new families of problems.

References

1. Adaptive and Bioinspired Systems Group: ABSys JECO (Java Evolutionary COmputation) library. Available at: https://github.com/ABSysGroup/jeco (accessed 2021)
2. Asseg, F., Chatterjee, S.: exp4j: a library for expression evaluation in Java. Available at: https://www.objecthunter.net/exp4j/index.html (accessed 2021)
3. Assunção, F., Lourenço, N., Ribeiro, B., Machado, P.: Evolution of Scikit-Learn pipelines with Dynamic Structured Grammatical Evolution. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). pp. 530–545. Springer (2020)
4. Augusto, D.A., Barbosa, H.J., Barreto, A.M., Bernardino, H.S.: A new approach for generating numerical constants in grammatical evolution. In: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. pp. 193–194 (2011)
5. Boettiger, C.: An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review **49**(1), 71–79 (2015)
6. Charte, F., Vico, A., Pérez-Godoy, M.D., Rivera, A.J.: predtoolsts: R package for streamlining time series forecasting. Progress in Artificial Intelligence **8**(4), 505–510 (2019)
7. Coletti, M.A., Scott, E.O., Bassett, J.K.: Library for evolutionary algorithms in Python (LEAP). In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. pp. 1571–1579 (2020)
8. Colmenar, J., Hidalgo, J., Salcedo-Sanz, S.: Automatic generation of models for energy demand estimation using Grammatical Evolution. Energy **164**, 183–193 (2018)
9. Davis, A.L.: Spring data. In: Spring Quick Reference Guide, pp. 43–59. Springer (2020)
10. De Smet, G., open source contributors: OptaPlanner User Guide. Red Hat, Inc. or third-party contributors (2006), `https://www.optaplanner.org(accessed2021)`
11. Dempsey, I., O'Neill, M., Brabazon, A.: Constant creation and adaptation in grammatical evolution. In: Foundations in Grammatical Evolution for Dynamic Environments, pp. 69–104. Springer (2009)
12. Durillo, J.J., Nebro, A.J.: jMetal: A Java framework for multi-objective optimization. Advances in Engineering Software **42**(10), 760–771 (2011)
13. Elyasaf, A., Sipper, M.: Software review: the HeuristicLab framework. Genetic Programming and Evolvable Machines **15**(2), 215–218 (2014)

14. Estévez-Velarde, S., Gutiérrez, Y., Almeida-Cruz, Y., Montoyo, A.: General-purpose hierarchical optimisation of machine learning pipelines with grammatical evolution. Information Sciences **543**, 58–71 (2021)
15. Fushiki, T.: Estimation of prediction error by using k-fold cross-validation. Statistics and Computing **21**(2), 137–146 (2011)
16. Hidalgo, J.I., Botella, M., Velasco, J.M., Garnica, O., Cervigón, C., Martínez, R., Aramendi, A., Maqueda, E., Lanchares, J.: Glucose forecasting combining markov chain based enrichment of data, random grammatical evolution and bagging. Applied Soft Computing **88**, 105923 (2020)
17. Johansson, L., Dossot, D.: RabbitMQ Essentials: Build distributed and scalable applications with message queuing using RabbitMQ. Packt Publishing Ltd (2020)
18. Martínez-Rodríguez, D., Colmenar, J.M., Hidalgo, J.I., Villanueva Micó, R.J., Salcedo-Sanz, S.: Particle swarm grammatical evolution for energy demand estimation. Energy Science & Engineering **8**(4), 1068–1079 (2020)
19. Mauceri, S., Sweeney, J., McDermott, J.: One-class subject authentication using feature extraction by grammatical evolution on accelerometer data. In: Heuristics for Optimization and Learning, pp. 393–407. Springer (2021)
20. Meyer, M.: Continuous integration and its tools. IEEE software **31**(3), 14–16 (2014)
21. Nicolau, M., Agapitos, A.: Understanding grammatical evolution: Grammar design. In: Ryan, C., O'Neill, M., Collins, J. (eds.) Handbook of Grammatical Evolution. pp. 23–53. Springer International Publishing, Cham (2018)
22. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Norwell, MA, USA (2003)
23. Qaddoura, R., Faris, H., Aljarah, I., Castillo, P.A.: EvoCluster: An open-source nature-inspired optimization clustering framework. SN Computer Science **2**(3), 1–12 (2021)
24. Ramírez, A., Romero, J.R., García-Martínez, C., Ventura, S.: JCLEC-MO: A Java suite for solving many-objective optimization engineering problems. Engineering Applications of Artificial Intelligence **81**, 14–28 (2019)
25. Red Gate Software Ltd: Flyway open-source database migration tool. https://flywaydb.org/ (accessed 2021)
26. Scott, E.O., Luke, S.: ECJ at 20: toward a general metaheuristics toolkit. In: Proceedings of the genetic and evolutionary computation conference companion. pp. 1391–1398 (2019)
27. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization **11**(4), 341–359 (1997)
28. Tian, Y., Cheng, R., Zhang, X., Jin, Y.: PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. IEEE Computational Intelligence Magazine **12**(4), 73–87 (2017)
29. Vladislavleva, E.J., Smits, G.F., Den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Transactions on Evolutionary Computation **13**(2), 333–349 (2008)