International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8, Issue-2S11, September 2019

A New Domain Specific Scripting Language for Automated Machine Learning Pipeline

Suraya Masrom, Abdullah Sani Abd Rahman, Nasiroh Omar, Norhayati Baharun

Abstract— This paper concerns on two difficulties faced by non-experts' users in the utilization of machine learning; design of the model and the programming task for implementation. From the varieties of the machine learning algorithms, selecting the best model with the best configurations is a critical and complex design issue. In light of this situation, automated machine learning pipeline is highly beneficial. Research has proved that Genetic Programming is highly useful to find the best pipeline of an automated machine learning model. However, in respond to the implementation difficulty, there exists a limited software tool that support easy implementation for automated machine learning based on Genetic Programming. This paper presents the specifications of a domain specific scripting language for the easy development of an automated machine learning with the underlying Genetic Programming. The scripting language has a very minimal characters of codes, hence easier to understand and more concise than the Python programming language.

Index Terms: Domain specific language, genetic programming, machine learning, scripting language..

I. INTRODUCTION

Today, in the twenty-first century, rapid software development (RAD) or rapid prototyping has emerged as a preferable approach in software development, especially to computer applications with complex computations such as machine learning modelling. Machine learning model are prevalent in real-life applications mainly in classification and prediction problems [1]–[8]. Time is a critical factor for solving such kind of problems and difficulty of programming implementation delay the process to complete. As a result, rapid software development, in many aspects, is able to provide easier, effective and more productive approach for the development of such complex systems. In order to facilitate rapid software development, innovations and improvements from software perspectives have been studied and invented. The ideas are varied from assorted perspectives that include software design [9], software modeling [10] and to programming paradigms [11]-[13]. This paper focuses on presenting the ideas of using an easy programming paradigm with a new domain specific scripting language.

The contribution of this paper is two-folds. This paper introduces the syntactical design of scripting language constructs and the general software architecture of the scripting language software framework as well as the compiler software structure. Second, the scripting language designed introduced in this paper is a new domain specific programming language on Automated Machine Learning (AML) with the underlying Genetic Programming (GP) approach. The GP is useful in optimizing the best pipeline of machine learning models. All the tasks in the machine learning pipelines such as selecting suitable machine learning algorithms, hyperparameter tunings, and feature selections are considerably difficult and sometimes tedious. As all the processed are merely dependent on the pattern of datasets, selecting the best pipeline may require a very long time to complete. Here is where the GP plays its role in automating the processes.

This paper is organized as follow. Section II provides the literatures of research background on domain specific scripting language and existing software tool of AML. Section III describes the new domain specific scripting language syntactical design and the software framework architecture. This paper also provides the easiness and conciseness measurements of the scripting language in Section IV followed with the conclusion remarks in the last section.

Revised Version Manuscript Received on September 16, 2019.

Suraya Masrom, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Perak Branch, Tapah Campus, Malaysia.
Abdullah Sani Abd Rahman, Faculty of Science and Information Technology, Universiti Teknologi Petronas, Perak, Malaysia
Nasiroh Omar, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam, Selangor, Malaysia.
Norhayati Baharun, Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Perak Branch, Tapah Campus, Malaysia.

II. BACKGROUND OF STUDY

A. Domain specific scripting language

Different with General-purpose programming language (GPL), domain-specific language (DSL) is a programming language that is tailored to a specific functionality of application. To enhance this further, the following describes more details of the four key elements of DSL as suggested by [14]:

- *Computer programming language*

The use of DSL is just another kind of programming language used by humans to interact with a computer; the purpose to achieve something. While the structure of DSL is designed with humans in mind, it should be readable and must also be executable by a computer. Pseudo-code is a code that can be read easily by humans, but it is not a computer programming language as pseudo-code cannot directly a computer executable.

- *Language nature*

DSL is a programming language, but it should have a sense of fluency like human language. In simple chronology, it is always true to say that better fluency fosters better comprehension among people. With fluency, they can speak and convey information easily, smoothly and effectively. Similarly, DSL fluency is a good principle that can improve a programmer productivity. Fluency is highly achievable if the language has small amounts of elements, keywords characters and expressions.

- *Limited expressiveness for conciseness*

Unlike GPL, a DSL supports a bare minimum of features good enough to support its domain functionalities. For DSL, expressiveness can be derived not just from individual expressions, but also from the way they can be composed together. Instead of expressiveness, conciseness is more relevant to DSL.

- *Domain focus*

Domain focus is the fundamental aspect that made a DSL worthwhile. A DSL can feature a small scope of domain and can be designed with lesser program elements. The tendency of programmers to be more understandable and competent in the programming language would be greatly attained with domain focus DSL. Due to domain focus, DSL tends to have more concern on knowledge abstraction. One of the abstracts features in a DSL is the keywords utilization. Most GPL has plural keywords for a particular operation thus a larger number of words are required to convey the equivalent information. For example, *while*, *do-while* and *for* are representing control keywords for repetitive expressions with each one having a different syntactical presentation. DSL in contrast, reduces the generality of keywords or jargon elements within the scope and the characteristics of a specific domain.

DSL gains more advantages with the four key elements. Other than improving programmer productivity, DSL allows non-programmers to easily read the real codes.

Exposing real implementation codes, which also act as a description of the domain, will enable a much richer communication between the developers and other people working within that domain.

Scripting language can be GPL or DSL, but modern scripting language usually tends to be more DSL. Older scripting language such as *Perl*, *TCL*, *Phyton* and *CGI* is developed to support the more common types of traditional scripting including system administration, controlling remote applications, command line interface and server-side programming on the Web [15]. At the beginning, these scripting languages are developed with DSL in mind. The *Perl* language for example, was developed to support traditional activities such as navigating large file systems and manipulating large amounts of text, but due to the common requirement to networking, the language is supported with network and socket programming. While this language is usable for traditional scripting, major advancements have been done with the language so that now the language is capable of developing many kinds of applications with different technologies, for examples database, GUI, networking and distribution processing. Another example is *R* scripting language that was originally developed as a language and platform that very much targeted for statistical work, but it has all the expressiveness of a GPL. Hence, despite its domain focus, it is not categorized as DSL [14].

Complimentary to the traditional scripting, the application that comprises most of modern scripting includes components scripting, agent-based scripting and client-side network scripting. In the modern world, the idea of gluing is to use scripting language to manipulate a collection of components, which comprise of objects, components, functions, code libraries and frameworks. To date, the number of modern scripting languages has been rapidly increasing. These languages still have their limitations in respect to the functional within a specific type of application domain; yet the language is more applicable to DSL. In this research, the aim is to develop a set of scripting languages specific for machine learning prediction based on GP optimization. Therefore, DSL and scripting languages will be used interchangeably throughout the rest of this paper.

B. Software tools for AML

Auto-Weka [16], [17] and Auto-SkLearn [18] are among the initial works of pipeline optimization to discover the best combination of data pre-processing, hyper-parameter tuning and model selection for AML. Bayesian optimization [19] has been adapted to avoid exhaustive grid-search parameter enumeration in both Auto-Weka and Auto-SkLearn.

Cognitive Automation of Data Science (CADS) [20], [21] is another software tool built on top of Weka, SPSS and R to automate the selection of model and hyper-parameter tuning. Three basic components of CADS are: a collection of analytics algorithms, a learning control mechanism and an interactive user interface. Utilizing stacking and metadata to automate the model selection and hyper tuning, Automatic Ensemble [22] is the most recent work found in the literature. Additionally, Tree-based Pipeline Optimization Tool (TPOT) [23], [9], [24] is another recent AML that can optimize machine learning pipeline by finding the best model configuration and pre-processing work-flow.

In this research, the interest is directed to TPOT, an AML library in Python programming. The algorithm used Genetic Programming (GP) [25]-[27] to optimize the best pipelines suitable for the dataset.

III. THE SCRIPTING LANGUAGE

A. General software architecture

Fig. 1 presents the general architecture of the software framework.

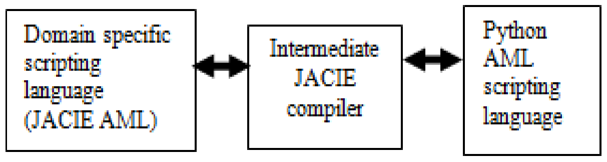
Domain specific scripting language (JACIE AML)
Intermediate JACIE compiler
Python AML scripting language

Fig. 1: General software architecture

Domain specific scripting language is the front-end component that allows users to write the easy codes. It was named as JACIE AML because the language was an extendable of JACIE scripting language [28], [29] that used JACIE compiler to convert the source codes into the Python codes for implementing the AML TPOT. The JACIE AML codes can be written by using any kind of word editor such as Notepad or WordPad.

B. The syntactic specifications of JACIE AML

In this part, the syntactic specification of the domain specific scripting language called JACIE AML-TPOT is described. Described in the next section, JACIE is the name of compiler software used to construct the AML scripting language. In the design, symbol ∂ is used to represent an input variable while all the keywords and symbols are exactly use in the specifications. It is classified into six elements according to the language functions. These elements are:

- *Begin program*

General codes are to initial the program, which give an indicator to the compiler the start line of the program. The code is a statement with keyword *begin AML TPOT* as in Fig. 2.

```
begin AML TPOT (∂ pathdirectoryexperimentname)
```

Fig. 2: The syntax specification to begin the program

To give specific name for the program file, *pathdirectoryexperimentname* variable is used to save and locate the autogenerated Python codes in the computer. This file can be accessed, viewed or edited (if required) by the users, which stored in the set location.

- *Dataset assignment*

The most important element in machine learning is assignment of training and testing datasets. The syntax specification is given in Fig. 3.

```
trainingset (∂ filename )
testingset ((∂ filename )
droptraining( ∂ colname | ∂ colname2…| ∂ colnameN)
droptesting( ∂ colname | ∂ colname2…| ∂ colnameN)
target (∂ targetcolumn)
```

Fig. 3: The syntax specification for dataset assignment

The keyword for assigning training and testing datasets is *trainingset* and *testingset* respectively. If the users need to drop any features from the training or testing dataset, statement *droptraining* or *droptesting* can be used by passing the column name. In the one statement of each, more than one column can be dropped. The *target* is to set the dependent variable (DV) or the target prediction column. The set of statements for these tree statements must be written in the sequence of *trainingset, drop* and *target*. However, *drop* is an optional code.

- *Validate configuration*

There exist three common approaches of validating the machine learning namely split, split with cross validation and Hold out. Hold out approach is a non-split dataset for training and validating the machine learning model, which the dataset for testing is not used in training. In this paper, we only focus on split approach first. This approach split the training datasets into training and testing portions, depending on the test size on the configuration. Refer to Fig. 4, the test size value can be assigned into the *test_size* assignment. The *random_state* is an optional, which set to 0 by default or can be assigned by users to any value. It is not expected that the non-expert users can provide a value to the random state.

```
validate (random_state=0 | 0 randstate, test_size=∂ size)
```

Fig. 4: The syntax specification for dataset assignment

- *Regression*

This element is to call fitness method in Python for training the regressor model with the training dataset. As AML TPOT used GP, it involves genetic parameters such as generation, population size, mutation rate and crossover rate. By default, the mutation and crossover rates are 0.1 and 0.9 respectively, but optional for changes with the new values. Generation and population size are also designed to be flexible with default values or users input. Default value for generation is 5 and population size is 20. Fig. 5 presents the syntax specification for AML fitness regressor. To simplified, AML fitness classifier is not described in this paper.

```
fitnessRegressor (gen=5 | ∂ gen, pop_size= 20 | ∂ size, mutation=0.1 | ∂ mrate, crossover=0.9 | ∂ crate)
```

Fig. 5: The syntax specification for AML fitness regressor

- *Prediction*

It is just enough to use one keyword to call the predict action. As in Fig. 6, the keyword is *predict* and really representing the process. The non-expert users should not be hassled with the underlying processes.

```
predict
```

Fig. 6: The syntax specification for dataset assignment

- *Measurement*

Two basic measurements of regression is R squared and Root Mean Square Error (RMSE). R squared present the accuracy of the model while RMSE present the prediction model. To ensure that the non-expert users aware about these two measurements, two statements are used as presented in the following Fig. 7.

```
Rsquared
RMSE
```

Fig. 7: The syntax specification for performance measurement

- *End Program*

To end the program, simple statement of *end AML TPOT* is used.

C. Intermediate JACIE compiler

Intermediate JACIE compiler translates the new scripting language into the relevant Python codes for execution. Since JACIE is a language based platform, the inclusions of new components as well as new algorithm is always permitted [29]. Therefore, the compiler can be extended to translate new scripting language constructions for different domain applications other than the current applications supported by JACIE. Two important elements in JACIE is *jflex* and *jCup* as presented in Fig. 8.

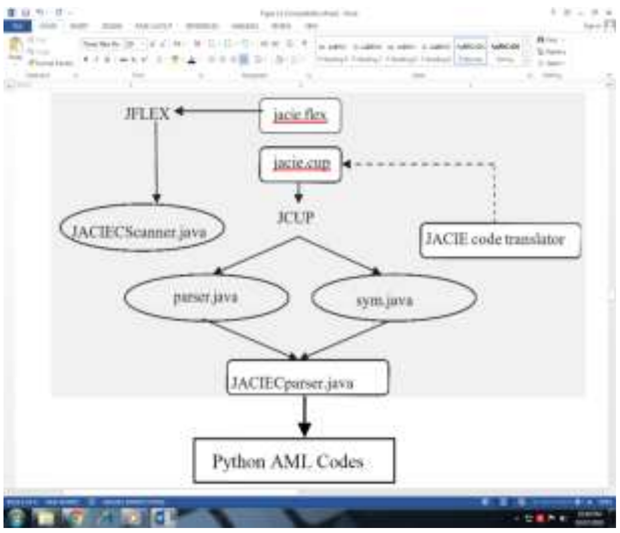

Fig. 8: The syntax specification for dataset assignment

JFlex accomplishes lexical analyzer of the source scripts and generates a token stream for the next parsing stage undertakes by *JCup*. All the JFlex specifications for the JACIE AML-TPOT can be added into the *jacie.flex* and the identified tokens are located in a file called *JACIEScanner.java*. Then, this *JACIEScanner.java* together with *jacie.cup*, will generate intermediate codes (*parser.java*) and compiling states (*sym.java*). The *jacie.cup* contains grammar specifications for the original *JACIE*, the new specifications for the AML-GP and several objects used by the *JACIE code translator*. The *JACIE code translator* that is written in *JAVA* must be compiled prior to executing *JCup*. This file consists a set of *System.out.print* or *System.out.println* from JAVA *java.io.PrintStream* written to translate the front-end JACIE AML-TPOT into Python codes.

D. Auto-generated AML codes

The auto-generated codes that are results of compiler is a Python codes for AML TPOT. With the generated codes, users can directly copy the codes into their Python editor or IDE (Integrated Development Environment).

IV. MEASURMENT OF EASINESS AND CONCISE & RESULTS

The scripting language constructs that have been completed with the desired features are required to be evaluated. In this paper, the codes will be measured with easiness and conciseness aspects. Source Lines of Codes (SLOC) metric may not be adequate in predicting the factors such as differing programmer efficiencies, it is nevertheless providing a good estimation of the overall easiness features. SLOC has been extensively used in the software development industry such as in COnstructive COst MOdel (COCOMO) [30] and IEEE Standard for Software Productivity Metrics [31]. To be fair in the measurement, only the physical lines of codes are measured, but excluded all comments codes and empty lines. In addition to SLOC, ESDL[11] evaluation has accounted the word of codes (WOC) and characters of codes (COC) to present the language simplicity. Responding to this, this research has considered all the LOC, WOC and COC to measure the easiness level of the proposed scripting language constructs.

The second evaluation is conciseness that presents the ratio of a program task over its relevant codes. This paper presents a new formal approach to qualitatively indicate the concise of such scripting language constructs in comparison to Python programming language. In (1) defined the calculation of conciseness introduced in this research.

$$C = \sum_{i=1}^{n} \frac{1}{h_i} \tag{1}$$

where C is the total conciseness from each tasks $i=\{1..n\}$, where in this work n=6 and h is the number of non-space characters exist in the codes. Results of measurements are presented after the following codes comparisons.

Comparisons of codes

Following is the comparisons of JACIE AML-TPOT codes with Python AML-TPOT codes.

- Begin program with JACIE AML-TPOT

1.begin AML TPOT (C:\experiment)

- Begin program with Python AML-TPOT

1. import NumPy as np

2. import pandas as pd

- Dataset assignment with JACIE AML-TPOT

1. trainingset(train.csv)
2. testingset(test.csv)
3. droptraining(SellPrice)
4. target(SellPrice)

- Dataset assignment with Python AML-TPOT

1. training = pd.read_csv("train.csv")
2. testing= pd.read_csv("test.csv")
3. x_train = training.drop(["SellPrice"], axis=1).values
4. y_train = training["Selling Price"].values

- Validate configuration with JACIE AML-TPOT

1. validate(random_state=0, test_size=0.2)

- Validate configuration with Python AML-TPOT

1. from sklearn.model_selection import train_test_split
2. x_training, x_valid, y_training, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=0)

- Regression with JACIE AML-TPOT

1.fitnessRegressor (generations=5, pop_size=20, mutation=0.2, crossover=0.8)

- Regression with Python AML-TPOT

1.from tpot import TPOT Regressor
2.tpot = TPOTRegressor(verbosity=3, random_state=25, n_jobs=-1, generations=5, population_size=20,mutation=0.2, crossover=0.8 , early_stop = 5)
3.tpot.fit(X_train, y_train)
4.tpot.export('tpotpipeline.py')

- Prediction with JACIE AML-TPOT

1. predict

- Prediction with Python AML-TPOT

1. rf_tpot = tpot.predict(X_valid)

- Measurement with JACIE AML-TPOT

1. Rsquared
2. RMSE

- Measurement with JACIE AML-TPOT

```
r2_tpot=r2_score(y_valid, rf_pred)
rmse_tpot= np.sqrt(mean_squared_error(y_valid, rf_pred))
print("R^2=" + str(r2_tpot))
print("RMSE="+ str(rmse_tpot))
```

Results of measurement

Table 1 lists the LOC, WOC and COC of the programming languages for AML-TPOT.

Table 1: The programming language easiness

Measurement	JACIE AML-TPOT	Python AML-TPOT
LOC	10	16
WOC	21	69
COC	219	678

Across all measures, the scripting language programs consistently have shown less code than the Python codes. This implies that to use the scripting language requires less effort to comply. Table 2 shows the conciseness result.

Table 2: The programming language conciseness

Task	JACIE AML-TPOT	Python AML-TPOT
Begin	0.370	0.030
Dataset assignment	0.010	0.006
Validate configuration	0.026	0.007
Regression	0.014	0.005
Prediction	0.142	0.034
Measurement	0.031	0.007

Table 2 shows that the total conciseness of JACIE AML-TPOT is much bigger than Python AML-TPOT, which is 0.593 and 0.089 respectively.

V. CONCLUSION

This paper describes the design of a new domain specific scripting language to implement AML with GP and demonstrate comparisons of codes, easiness and conciseness of two programming languages with scripting paradigm. Python programming language is a kind of scripting-based language, but the codes still consists of hassle characters and jargon to be learnt and used by non-expert users. The JACIE AML-TPOT introduced in this paper is a higher-level scripting language for Python. Hence, the JACIE AML-TPOT codes should be designed with better easiness and conciseness than the Python. This paper has given fundamental ideas on evaluating programming languages quantitatively and the results show that the new scripting language codes is easier and more concise than Python codes.

VI. ACKNOWLEDGMENT

The authors would like to thank Universiti Teknologi MARA and Ministry of Education Malaysia for the financial support of this project under FRGS grant no. FRGS/1/2018/ICT02/UITM/02/2.

REFERENCES

1. A. K. Mishra and P. Vijaykumar, “Analysis of spectrum occupancy using Naïve Bayesian classifier,” Int. J. Recent Technol. Eng., 7(4), 2018, pp. 115–117.
2. S. Singh, M. Kaushik, A. Gupta, and A. K. Malviya, “Weather forecasting using machine learning techniques,” SSRN Electron. J., 6, 2019, pp. 38–41.
3. C. Rajinikanth and S. A. Lincon, “A semi supervised based Hyper Spectral Image (HSI) classification using machine learning approach,” International Journal of Recent Technology and Engineering, 7(5S2), 2019, pp. 13–16.
4. M. Bharathi, “A review of recent advancements in Flux Reversal Permanent Magnet Machine (FRPMM),” Majlesi Journal of Mechatronic Systems, 8(1), 2019, pp. 33–45.
5. S. Kavipriya and T. Deepa, “Dual Edge Classifier Based Support Vector Machine (DESVM) classifier for clinical dataset,” International Journal of Recent Technology and Engineering, 7(6), 2019, pp. 331–338.
6. A. K. Rai, S. Agarwal, M. Khaliq, and A. Kumar, “Quantitative analysis of development environment risk for agile software through machine learning,” International Journal of Recent Technology and Engineering, 7(6), 2019, pp. 83–89.
7. G. Magesh and P. Swarnalatha, “Attribute reduction and cost optimization using machine learning methods to predict breast cancer,” International Journal of Recent Technology and Engineering, 7(6), 2019, pp. 306–308.
8. R. Balakrishna and R. Anandan, “Early diagnosis of chronic and acute pancreatitis using modern soft computing techniques,” International Journal of Recent Technology and Engineering, 7(4S), 2018, pp. 65–69.
9. C. Angelov, K. Sierszecki, and N. Marian, “Component-based design of embedded software: An analysis of design issues,” International Workshop on Scientific Engineering of Distributed Java Applications, 2005, pp. 1–11.
10. F. Stonedahl and U. Wilensky, NetLogo Particle Swarm Optimization model. Available: http://ccl.northwestern.edu/netlogo/models/.
11. S. Dower and C. J. Woodward, “ESDL: A simple description language for population-based evolutionary computation,” 13th Annual Conference on Genetic and Evolutionary Computation, 2011, pp. 1045–1052.
12. T. C. Oliveira, P. S. C. Alencar, C. J. P. de Lucena, and D. D. Cowan, “RDL: A language for framework instantiation representation,” J. Syst. Softw., 80(11), 2007, pp. 1902–1929.
13. J. J. Durillo and A. J. Nebro, “jMetal: A Java framework for multi-objective optimization,” Adv. Eng. Softw., 42, 2011, pp. 760–771.
14. M. Fowler, Domain Specific Languages. England: Pearson Education, 2010.
15. D. Barron, The World of Scripting Language. New Jersey: John Wiley and Sons, 2010.
16. L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, “Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA,” J. Mach. Learn. Res., 18(1), 2017, pp. 826–830.
17. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms,” 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 847–855.
18. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, “Efficient and robust automated machine learning,” Advances in Neural Information Processing Systems, 2015, pp. 2962–2970.
19. E. Brochu, V. M. Cora, and N. De Freitas, A tutorial on

Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Available: https://arxiv.org/pdf/1012.2599.pdf.

20. A. Biem, M. Butrico, M. Feblowitz, T. Klinger, Y. Malitsky, K. Ng, A. Perer, C. Reddy, A. Riabov, H. Samulowitz, and D. Sow, “Towards cognitive automation of data science,” Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 4268-4269.
21. U. Khurana, S. Parthasarathy, and D. S. Turaga, “READ: Rapid data Exploration, Analysis and Discovery,” EDBT, 2014, pp. 612–615.
22. M. Wistuba, N. Schilling, and L. Schmidt-Thieme, “Automatic Frankensteining: Creating complex ensembles autonomously,” SIAM International Conference on Data Mining, 2017, pp. 741–749.
23. P. Gijsbers, J. Vanschoren, and R. S. Olson, “Layered TPOT: Speeding up tree-based pipeline optimization,” CEUR Workshop Proc., 2017. pp. 1-24.
24. R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, J. H. Moore, and others, “Automating biomedical data science through tree-based pipeline optimization,” European Conference on the Applications of Evolutionary Computation, 2016, pp. 123–137.
25. D. Andre and J. R. Koza, “Parallel genetic programming: A scalable implementation using the transputer network architecture,” in Advances in Genetic Programming 2, P. J. Angeline and K. E. Kinnear Jr., Eds. Cambridge: MIT Press, 1996, pp. 317–337.
26. M. Affenzeller, S. Winkler, S. Wagner, and A. Beham, Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications. Florida: CRC Press, 2009.
27. W. B. Langdon and M. Harman, “Optimizing existing software with genetic programming,” IEEE Trans. Evol. Comput., 19(1), 2015, pp. 118–135.
28. A. S. Ismail, M. Chen, P. W. Grant, and M. Kiddell, “JACIE-An authoring language for rapid prototyping net-centric, multimedia and collaborative applications,” Ann. Softw. Eng., 12(1), 2001, pp. 47–75.
29. S. Z. Z. Abidin, Interaction and interest management in a scripting language. PhD thesis, Swansea: University of Wales, 2006.
30. B. W. Boehm, Software Engineering Economics. New Jersey: Prentice Hall, 1981.
31. Institute of Electrical and Electronics Engineers (IEEE), “Standard for Software Productivity Metrics,” IEEE Std 1045 1992, 1993, pp. 1–38.