Symbolic Regression Using Genetic Programming with Chaotic Method-based Probability Mappings

Pu Cao[1], Yan Pei[1⋆], and Jianqiang Li[2]

[1] Graduate School of Computer Science and Engineering, University of Aizu
Aizuwakamatsu, Fukushima, 965-8580, Japan
m5252101@u-aizu.ac.jp, peiyan@u-aizu.ac.jp
[2] Faculty of Information Technology, Beijing University of Technology
Beijing, 100124, China
lijianqiang@bjut.edu.cn

Abstract. In this study, we propose a novel pre-learning approach for genetic programming (GP) that aims to investigate the effect of the probability of being selected for each operator. Furthermore, we present a technique that combines chaos theory and searches for a relatively good possibility mapping for each operator using one-dimensional chaotic mapping. We conducted several sets of comparative experiments on real-world data to test the viability of the proposal. These experiments included comparisons with conventional GP, examination of the impact of various chaotic mappings on the proposed algorithm, and implementation of different optimization strategies to find the relative optimal probability mapping. The experimental results demonstrate that the proposed method can achieve better results than conventional GP in the tested dataset, without considering the total quantitative calculation amount. Through statistical tests, it has been proven that the proposed method is significantly different from the conventional method. However, the discussion regarding the circumstances under which the proposed method can obtain better results when the total calculation amount is limited is not yet fully explored due to the small-scale nature of the experiments. Our future studies will focus on improving and fully discussing this idea.

Keywords: Genetic programming, Chaos theory, Symbolic regression, Evolutionary computation, Pre-learning

1 Introduction

Symbolic regression (SR) is a relatively specialized regression problem that requires consideration not only of the results but also of the structure. The goal of symbolic regression is to generate a set of formulas that can be used to fit the target data. The result is visible and understandable, such as generating an equation equivalent to $\sqrt{x_0^5 + x_1^2 - x_2^3}$. Genetic programming (GP) is a method that can generate explicit solutions without prior assumptions and only requires

⋆ Dr. Yan Pei is the corresponding author, peiyan@u-aizu.ac.jp

knowledge of the basic components of the problem. Therefore, it is well-suited for solving symbolic regression problems [2].

In conventional tree-based GP, there are two categories of nodes: operator nodes(e.g. addition, subtraction, multiplication, division, etc.) and termination nodes, each with several types of operators and terminal nodes, respectively. During initialization and mutation, a random selection is made to decide what type of node the new node should be. At this point, the probability of each operator node being selected is equally likely.

However, there has not been extensive discussion on the selection of GP operators. When exploring non-prior knowledge using Symbolic Regression (SR), we do not know what basic components should be included in the target system. Therefore, we should provide enough choices for the initial operator set. But if the operator set is too large, the solution space will also be too large, which affects the algorithm's convergence performance. On the other hand, if the operator set is too small, the fitting of the target problem will not be accurate [8]. Therefore, this paper further discusses the selection of GP operators.

The motivation for this study is that certain operators should not be employed for specific problems as they can make the solution more complex and difficult to converge to. Therefore, it would be effective to propose a scheme that provides different probabilities of the operator being selected for different problems and data.

In response to this view, we propose a pre-learning method based on chaos theory. The aim is to find one or several sets of probability mappings that make it easier to obtain better answers from the searching space of different operator node possibilities. To provide more realistic and applicable results, this paper uses real-world data for experimentation and research.

Furthermore, we explore the use of chaotic systems to optimize our proposed algorithm. We conduct three sets of comparative experiments, including comparing the performance of our proposed algorithm with the original algorithm, exploring how different chaotic systems affect the algorithm's performance, and analyzing the algorithm's learning performance under controlled computational conditions. These experiments provide insights into the optimal conditions for the proposed algorithm and its sensitivity to the chaotic system used.

After the introduction, we present related studies in Section 2. In Section 3, we provide a detailed description of our proposed algorithm and two different implementation methods. Section 4 describes three sets of experiments conducted from different perspectives to analyze the proposed algorithm, and we discuss the experimental results. Finally, we summarize the contributions of this study and highlight current issues and future directions.

2 Related Works

2.1 Genetic Programming

Genetic Programming (GP) [2] is an evolutionary algorithm that utilizes biological phenomena such as heredity, mutation, selection, and crossover to generate solutions to problems. Unlike other optimization algorithms, GP resembles more of a machine learning method as it uses hierarchical data structures to generate

solutions. In recent years, some studies have focused on optimizing algorithms by leveraging external information.

For instance, Ying Bi, Bing Xue, and Mengjie Zhang decomposed the genetic programming problem into multiple sub-problems, which were solved separately, and their final results were combined [3]. The advantage of this approach is that it can reduce the search space's size and speed up the algorithm's running time, leading to improved performance. While various means exist to improve performance outside the algorithm, we focus on optimizing the algorithm itself by exploring the impact of operators on its overall performance.

Hengrui Xing, Ansaf Salleb-Aouissi, and Nakul Verma explored the use of convolutional neural networks to convert data into visual representations and then discussed the probability of operators being selected too [8]. However, they did not utilize real data for experiments. Moreover, we believe that converting data into images and pre-training neural networks for images is not an efficient approach. Thus, we propose a different implementation idea.

Overall, our research focuses on optimizing the genetic programming algorithm's performance by analyzing the impact of operators on its overall performance. We believe that our approach can lead to significant improvements in the algorithm's performance, thereby enhancing its effectiveness and efficiency in solving problems.

2.2 Chaos Theory for Optimization Algorithm

The chaotic system refers to a class of dynamical systems that demonstrate intricate, unpredictable behavior in both time and space. Such systems exhibit behavioral characteristics, including high sensitivity to initial conditions, deterministic chaos, and adaptability, which render them fascinating objects of study in various fields of science and engineering [9].

Numerous studies have investigated the utilization of the ergodicity and non-repetition of chaotic systems to enhance the performance of optimization algorithms. For instance, recent research has proposed an approach that enhances the performance of JADE by utilizing chaotic systems to generate more diverse initial populations [4]. In addition, CE is a new algorithm based on differential evolution that leverages the ergodic motion in the search space to improve performance by exploiting the ergodicity of chaos [5]. These studies suggest that the characteristics of chaos can enhance search efficiency to a certain extent for random-based optimization algorithms. By introducing chaos into the optimization process, the search space is explored more efficiently, leading to better solutions. Based on past research, we have reason to believe that the utilization of the characteristics of chaos can enhance the performance of optimization algorithms, especially for random-based optimization algorithms.

3 Different Probability Mapping for Operators Are Used in Genetic Programming

As mentioned previously, a novel approach called "Different Probability Maps for Operators in Genetic Programming" (DPMOGP) has been proposed to investigate the impact of different operator selection probabilities on GP performance.

The key idea behind DPMOGP is to use an appropriate optimization strategy to perform pre-learning and generate one or more probabilistic mappings for different operators, which are then saved to initiate the formal GP process. During the initialization and mutation phases, when a new operator node is generated, one of the stored probability mappings is selected using a weighted random search via the roulette algorithm to determine which operator the new node should be assigned to. The algorithmic implementation of DPMOGP is shown in Algorithm 1.

It can be seen that this will be an optimization problem to find some probability mappings that can relatively easier to generate the higher fitness individual from the searching space. Thus, DPMOGP is a versatile concept that can be implemented in various ways. At present, we have implemented two versions of DPMOGP: random search-based DPMOGP (RS-DPMOGP) and genetic algorithm-based DPMOGP (GA-DPMOGP).

Algorithm 1 DPMOGP

PS: population; **PM**: probability mapping; **CP**: cumulative probability

```
/* pre-learning for search the relatively optimal PM */
initialize PM = [], CP = []
PM = preLearning() {discuss in detail later}
CP = rouletteAlgorithm(PM)
/* start conventional GP */
/* Once the new node of tree is generated */
num = random.double() { random number between 0-1}
count = 0
for  count = 0 to num.size() do
   if num <CP[i] then
      break
   end if
end for
newNode = PM[count]
```

3.1 Random Search-based DPMOGP

In Random Search-based Different Probability Mapping for Operators used in Genetic Programming (RS-DPMOGP), the primary objective is to generate a set of probability mappings using a random number generator, which is explained in detail later. Each probability mapping is used to create a fixed number of solutions (trees) of a specified size, and the average fitness of all solutions is calculated as the final fitness. Finally, several relatively good mappings are selected among all the generated probability mappings, and it is saved as the selection basis when a new operator node is generated in the conventional GP. The algorithm for RS-DPMOGP is shown in Algorithm 2.

Algorithm 2 RS-DPMOGP

PM: probability mapping; **AN**: amount of random mapping; **AT**: amount of trees for each random mapping; **AP**: amount of PM

```
/* random searching for relatively optimal PM */
randomMappings =[]
for  int i = 0 to AN do
   randomMappings.add(randomNumGenerator())
end for
sortingByFitness(randomMappings){each random mapping need to generate AT tress then take the average fitness}
PM = randomMappings[0:AP]{pick top AP relatively better probability mapping as PM}
/* The rest follow Algorithm 1*/
```

3.2 Genetic Algorithm-based DPMOGP

Given that the solution space of the optimization problem presented above involves permutations and combinations of N arbitrary numbers ranging from 0 to 1, it is evident that it belongs to the class of NP problems. To solve this type of problem, metaheuristic algorithms are commonly employed [6], with the genetic algorithm (GA) being a popular choice due to its simplicity and efficiency.

Algorithm 3 GA-DPMOGP

PM: probability mapping;
AP: amount of PM

```
/*genetic algorithm for relatively optimal PM */
randomMappings =[]
for  int i = 0 to AP do
   PM.add(genetic algorithm())
end for
/* The rest follow Algorithm 1*/
```

In genetic algorithm-based Different Probability Mapping for Operators is used in Genetic Programming(GA-DPMOGP) Similar to the RS-DPMOGP approach, the genetic algorithm utilizes the probability mapping generated to produce one or more trees. The trees are then utilized to assess the fitting performance of the generated formula using the least square method. The iteration process is stopped after a specified generation, and the best individuals are selected. Additionally, chaotic mapping is used as a random number generator when generating the initial population and mutations.

4 Experiments and Discussion

4.1 Dataset

Real-world data is utilized as a sample problem in this study. Specifically, the Yacht Hydrodynamics Data Set, which is a relevant dataset in fluid mechanics for exploring various ship hulls, is used for conducting experiments. The dataset is well-suited for applying symbolic regression to discover physical formulas. Refer to TABLE. 1 for further details.

Table 1. Yacht Hydrodynamics Data Set

Date Donated	***Instances:***	***features:***	***Area:***
2013-01-03	308	6	Physical

4.2 Chaotic Mapping

In the experiments, one-dimensional chaotic maps are employed as the random number generator. If not explicitly stated, the logistic mapping is assumed to be the default choice for the random number generator.

A logistic mapping [7] is a classical and straightforward chaos model whose modeling expression is presented in (1). The distribution of the model is largely determined by the variable K. As shown in Fig.1, when K equals four, the system is in a state of complete chaos, and the final long-term behavior is uniformly distributed in the interval [0,1]. In this study, all chaotic maps utilized are based on the scenario when the system is in a state of complete chaos.

$$X_{n+1} = X_n \cdot K(1 - X_n), \;\; K \in [0, 4], X \in [0, 1]. \tag{1}$$

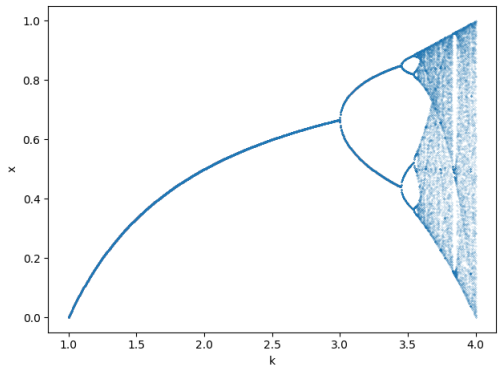


Fig. 1. Bifurcation diagram of the logistic mapping. When K is equal to 4, we consider the system to be in a chaotic state.

4.3 Results and Discussion

We conducted sets of comparative experiments to investigate the performance of DPMOGP in various aspects and the impact of using chaotic mapping for each experiment. Specifically, we performed 50 runs for each experimental group and then computed the R-squared as the result. Given that the division may be zero during calculations, we used protected division to replace the division with $\frac{y}{\sqrt{1+x^2}}$.

Initially, we compared the performance of DPMOGP and conventional GP with identical parameter settings. The parameter settings used for GP are presented in TABLE. 2.

Table 2. parameter for GP

Parameter	***Description***
Population size	100
iterations	100
initialize tree size	3
mutation rate	0.1
Operator set	+,-,*, / protected, tan,sin,cos, reciprocal, remainder
Terminal set	x_0,x_1,x_2,x_3,x_4,x_5
Selection	Tournament selection of size 4

(1) initially, we compared the performance of conventional GP and RS-DPMOGP, this group of experiments also evaluated the algorithm's performance in scenarios involving different numbers of chaotic mapping iterations (i.e., the total number of random searches). The parameter settings for this experiment are presented in TABLE. 3.

Table 3. parameter for RS-DPMOGP

Parameter	***RS-DPMOGP A***	***RS-DPMOGP B***
number of chaos mapping	10000	15000
trees generated per mapping	1	1
number of saved mapping	5	5
depth of tree	7	7

From the results depicted in Fig.2, we observe that RS-DPMOGP B exhibits the best performance, while conventional GP yields relatively poor results. These results suggest that, at least for the dataset used in the experiment, DPMOGP can enhance the algorithm's performance. By comparing RS-DPMOGP A and RS-DPMOGP B, we note that, in general, increasing the amount of computation in DPMOGP pre-learning leads to better performance before reaching the threshold. Prior to reaching the global optimum, enhancing the optimization algorithm's amount of calculation in pre-learning can enhance the final GP's performance.

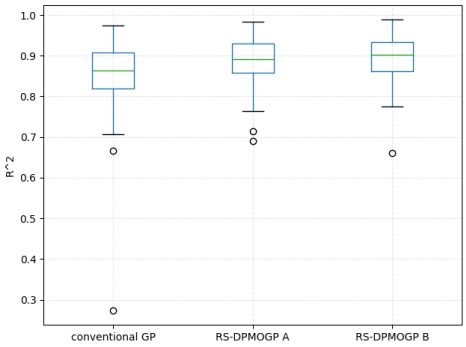


Fig. 2. The R^2 of results for conventional GP and DPMOGP. R^2 ranges from 0 to 1, and a value closer to 1 indicates a better fit of the model to the data. Typically, an R^2 value greater than 0.7 is considered a good fit.

In this experiment, we adopt Wilcoxon signed-rank test to compare the performance of different algorithms. The use of the Wilcoxon signed-rank test in algorithm comparison has been well established in the literature [10]. Specifically, we use this non-parametric test to assess whether there is a statistically significant difference in the median performance of the two algorithms. By comparing the p-value of the Wilcoxon signed-rank test with a predetermined significance level (usually set at 0.05), we can determine whether the observed difference in performance is statistically significant or not. In this experiment, a p-value less than 0.05 indicates a statistically significant difference between the two algorithms with a high probability. The detailed results are presented in TABLE.4. Through the results of the statistical test, we have observed a significant difference between the conventional GP and the proposed method, which greatly strengthens the persuasiveness of our research.

Table 4. Results of Wilcoxon signed-rank test for experiment (1). The results of the conventional GP and those obtained with different parameters of the proposed algorithm exhibit significant differences.

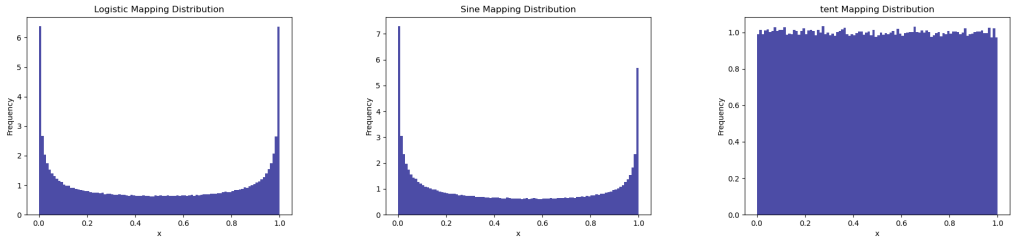
Comparison	GP	RS-DPMOGP A
GP	-	0.023592951063958378
RS-DPMOGP B	0.01688572946162479	0.4961523833965189

(2) After confirming the effectiveness of DPMOGP, we conducted comparative experiments to investigate the influence of different chaotic mappings on the algorithm. One-dimensional chaotic mappings such as tent mapping and sine mapping were added to the experiment, and the distribution of mappings is shown in Fig.3.

Upon observing the experimental results, we found that when logistic mapping is used as a random number generator, the probability of retention results falling within the range of 0-0.1 and 0.9-1 is higher than when random numbers

are used directly. This phenomenon has a significant impact on the probability that the corresponding operator is selected, as we use these 0-1 numbers as the weight of each operator and then use the roulette algorithm to select. In other words, the larger the weight of the operator, the easier it is to be selected. If the weight is a tiny value, the probability of its corresponding operator being selected will obey exponential decay.

Therefore, we hypothesize that the special distribution of logistic mapping may lead to better results. To test this hypothesis, we constructed a set of pseudo-random models with similar distributions to logistic mapping and conducted experiments. The results are shown in Fig.4. We can observe that logistic mapping and sine mapping follow the distribution as shown in Fig.3 with more points located at the earlier and end stages, while the distribution of pseudo-random looks similar to logistic mapping and sine mapping. For tent mapping, the points are evenly distributed in the coordinate system.



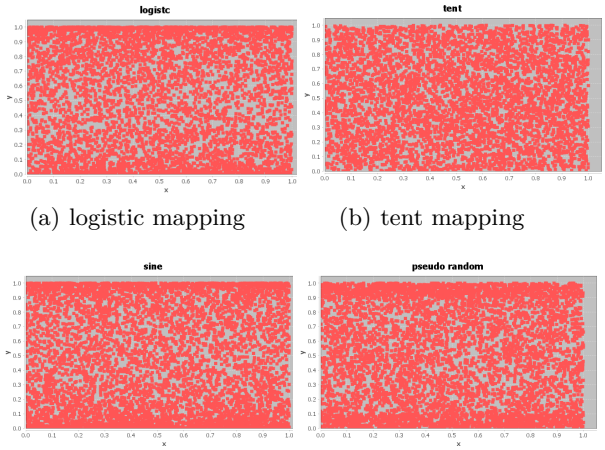
(a) logistic mapping (b) Sine mapping (c) Tent mapping

Fig. 3. Distribution histogram for chaotic mappings. The horizontal axis represents the generated results, and the vertical axis represents the frequency of occurrence of the generated numbers within that range.

The results of the comparative experiments are presented in Fig.5. It is observed that logistic mapping, sine mapping, and the pseudo-random model outperformed random and tent mapping. As mentioned earlier, logistic mapping, sine mapping, and the pseudo-random model have the characteristic of generating more tiny and huge numbers, whereas the distribution of tent mapping closely resembles a random distribution. This suggests that our hypothesis that the distribution of the random number generator influences the performance of the algorithm is validated.

Additionally, analyzing the data from Fig.5, it is observed that the pseudo-random model generates more outlier data than other models. We hypothesize that this could be due to the ergodicity of chaotic mappings, which we plan to investigate in our future study.

(3) In the final stage of our research, we examined the differences between conventional GP, RS-DPMOGP, and GA-DPMOGP when the amount of computation is limited. To ensure that we control the amount of computation for each algorithm, we redesigned the parameters as shown in TABLE.5. The total computation was set to 10,000 fitness evaluations, with 9,000 divided by for-



(a) logistic mapping (b) tent mapping

(c) sine mapping (d) pseudo random

Fig. 4. Distribution Scatter Plot for each model. Through the distribution of the lattice, we can see that (a), (b), and (d) are similar: more points are located near the vertical axis of 0 and 1.

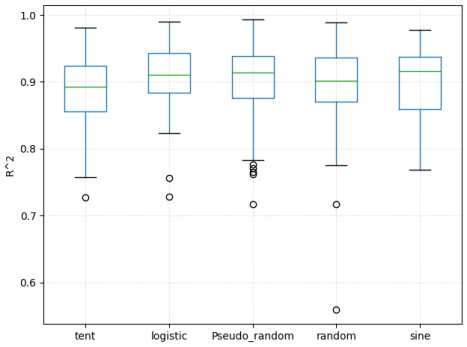


Fig. 5. The R^2 result for comparing the different random number generator.

mal GP for both RS-DPMOGP and GA-DPMOGP and only 1,000 reserved for pre-learning.

The experimental results are presented in Figure.6. When the computation amount was limited to 10,000 times, our proposed method did not achieve significant advantages over conventional GA. This was due to the pre-learning process not being fully executed. However, we believe that when larger amounts of data are used in the experiments and sufficient runs are given for pre-learning, DPMOGP will achieve better results, even if the amount of computation is controlled to be the same as conventional GP. We will discuss this in more detail in future work.

Table 5. parameter for experiment 3. Due to different algorithms, the parameters are not the same, but they have the same computational cost.

Parameter	*GP*	*RS-DPMOGP*	*GA-DPMOGP*
population size	100	100	100
iteration	100	90	90
number of chaos mapping(RS)	-	1000	-
trees generated per map(RS)	-	1	-
number of saved mapping(RS)	-	20	-
depth of tree(RS)	-	11	-
population size (GA)	-	-	10
depth of tree(GA)	-	-	11
mutation(GA)	-	-	0.1
iteration(GA)	-	-	10
number of saved mapping(GA)	-	-	20
total computation	10000	10000	10000

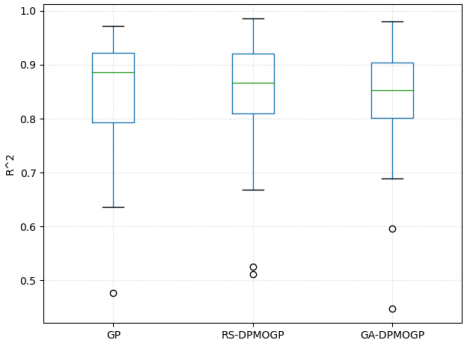


Fig. 6. The R^2 result for comparing with limited computation.

From the figure, we can see that the results of the three methods are similar, but the results of GA-DPMOGP are the most constricted because all the results will be optimized in one direction due to the characteristics of the genetic algorithm. However, it is unclear whether fitness can fully account for the performance of the generated probability mapping, so we cannot determine whether the converging direction of the algorithm's result is correct or incorrect.

We found in our experiments that, although GA-DPMOGP has a faster convergence speed for pre-learning than RS-DPMOGP, the final results are generally better for RS-DPMOGP, especially when there is a large amount of computation. This is because GA-DPMOGP is more likely to fall into a local optimum, whereas RS-DPMOGP will not have such a problem.

Therefore, we can conclude that DPMOGP is not suitable for scenarios where computing resources are scarce. When using DPMOGP, GA-DPMOGP should be selected if the amount of computation given to pre-learning is insufficient. If a large amount of computation is allocated to pre-learning, it is recommended to choose RS-DPMOGP.

5 Conclusion and Future Works

In this study, we propose DPMOGP to discuss the impact of the probability of being selected for each operator on genetic programming. We implemented two types of implementations using random search and genetic algorithms.

Through multiple experiments, we demonstrate that different probability mappings of operators significantly affect the performance of GP. Thus, optimizing the algorithm by finding an excellent probability mapping is effective.

However, we identified several limitations of the current method. Firstly, the pre-learning method consumes a significant amount of computation, which may not be a viable approach to finding the probability mapping. Therefore, we propose to integrate the pre-learning of DPMOGP into the GP process to increase its efficiency in the future study. Secondly, RS-DPMOGP and GA-DPMOGP have an important hyperparameter, i.e., the depth of the generated tree, which we currently set based on experience rather than mathematical analysis. If this parameter is too small, it may not fully explore the mapping, while setting it too large will generate numerous invalid nodes. Moreover, the suitable depth varies across different problems. We plan to address this issue in future studies.

References

1. Vapnik, V.: Principles of risk minimization for learning theory. In: Proceedings of Advances in Neural Information Processing Systems, pp. 831-838 (1991)
2. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. In: Proceedings of Statistics and Computing, vol. 4, no. 2, pp. 87-112 (1994)
3. Ying, B., Xue, B., Zhang, M.: A divide-and-conquer genetic programming algorithm with ensembles for image classification. In: Proceedings of IEEE Transactions on Evolutionary Computation, vol. 25, no. 6, pp. 1148-1162 (2021)
4. Gao, S., Yu, Y., Wang, Y., et al.: Chaotic local search-based differential evolution algorithms for optimization. In: Proceedings of IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 51, no. 6, pp. 3954-3967 (2021)
5. Pei, Y.: Chaotic evolution: fusion of chaotic ergodicity and evolutionary iteration for optimization. In: Proceedings of Natural Computing, vol. 13, no. 1, pp. 79-96 (2014)
6. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. In: Proceedings of ACM Computing Surveys (CSUR), vol. 35, no. 3, pp. 268-308 (2003)
7. May, R.M.: Simple mathematical models with very complicated dynamics. In: Proceedings of The Theory of Chaotic Attractors, Springer, New York, NY, pp. 85-93 (2004)
8. Xing, H., Salleb-Aouissi, A., Verma, N.: Automated symbolic law discovery: A computer vision approach. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 1, pp. 508-515 (2021)
9. Lathrop, D.: Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering. Physics Today, vol. 68, no. 4, pp. 54 (2015)
10. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. In: Proceedings of Information Sciences, vol. 180, no. 10, pp. 2044-2064 (2010)