*Journal of **AI** and **D**ata **M**ining*
Vol 8, No 3, 2020, 331-341.

DOI: 10.22044/JADM.2020.7847.1924

Shuffled Frog-Leaping Programming for Solving Regression Problems

M. Abdollahi[1*] and M. Aliyari Shoorehdeli[2]

1. Department of Computer Engineering, K.N. Toosi University of Technology, Tehran, Iran.
2. Department of Electrical Engineering, K.N. Toosi University of Technology, Tehran, Iran.

Received 19 December 2018; Revised 08 January 2020; Accepted 05 February 2020
**Corresponding author M.abdollahi@ee.kntu.ac.ir (M. Abdollahi).*

Abstract

There are various automatic programming models inspired by evolutionary computation techniques. Due to the importance of devising an automatic mechanism to explore the complicated search space of mathematical problems where numerical methods fail, the evolutionary computations are widely studied and applied to solve real-world problems. One of the famous algorithms in an optimization problem is the shuffled frog leaping algorithm (SFLA), which is inspired by the behavior of frogs to find the highest quantity of the available food by searching their environment both locally and globally. The results of SFLA prove that it is competitively effective to solve problems. In this paper, Shuffled Frog Leaping Programming (SFLP) inspired by SFLA is proposed as a novel type of automatic programming model to solve the symbolic regression problems based on tree representation. Also, in SFLP, a new mechanism is proposed for improving constant numbers in the tree structure. In this way, different domains of mathematical problems can be addressed with the use of the proposed method. To find out about the performance of the generated solutions by SFLP, various experiments are conducted using several benchmark functions. The results obtained are also compared with other evolutionary programming algorithms like BBP, GSP, GP, and many variants of GP.

Keywords: *Genetic Programming, Shuffled Frog Leaping Algorithm, Shuffled Frog Leaping Programming, Regression Problems.*

1. Introduction

The evolutionary computation approach is a novel paradigm in the machine learning area. As a part of these approaches, there are a number of evolutionary iterations for growth and development in the population. Through a random search, the population is created to acquire the expected conclusion. These operations are inspired by evolution inside biological mechanisms [6]. Genetic programming (GP), which stems from the genetic algorithm (GA), shows the solutions as computer programs instead of binary strings [1]. GP offers solutions to complex problems by searching in the problem space automatically. The population of computer programs in GP is evolved during a number of generations. GP randomly changes and places populations of programs into new populations of programs with the hope of better solutions while the search is done using GA [21]. There are many types of GP with different cross-over operators including standard cross-over (SC) [21] operator in basic GP, semantic similarity-based cross-over (SSC) [29], semantics-aware cross-over (SAC) [27], soft brood selection (SBS) [22], context-aware crossover (CAC) [25] and no same mate (NSM) [13]. In addition, modified versions of GP have been proposed with some improvement, for instance, linear GP (LGP) [4], cartesian GP (CGP) [26], and gene expression programming (GEP) [9]. Another evolutionary computation technique inspired from the biological immune system concepts is clone selection programming (CSP) [10].

Dynamic ant programming (DAP) is another technique for intelligent programming derived from ant colony optimization. This algorithm uses a pheromone table that changes dynamically. It is a pheromone value upon which the nodes (terminal and nonterminal) are chosen. The selection of nodes with high pheromone rates has more feasibility than the other ones. In this method, the search space changes dynamically and the ants discover better solutions using portions of solutions based on the pheromone value [34].

Artificial bee colony programming (ABCP) is another method for automatic programming derived from the artificial bee colony (ABC) algorithm. Similar to the relation between GP and GA, ABCP is an extension of ABC algorithm that deals with the representation of the problem using substantially complex structures [18].

Biogeography-based optimization (BBO), proposed by Simon, is an inspiration of the geographical distribution and migration of species in an ecosystem [35]. Biogeography-Based Programming (BBP), inspired by BBO, has been offered as a paradigm combining the program-like representation of solutions to symbolic regression problems regarding the biogeographical system. BBP can be applied to different types of problem domains and its performance indicates that it can find solutions to problems effectively [11].

Gravitational search algorithm (GSA) is an optimization technique that is inspired by the Newton's law of universal gravitation and how the law of motion happens. Gravitational search programming (GSP) is one of the recent methods for automatic programming proposed by Mahanipour et.al. [12]. This method focuses on a new gravitational search algorithm (GSA)-based algorithm to generate computer programs, automatically. GSP employs the GSA approach to generate the tree structure and insertion of internal nodes in a discrete space of tree-based solution representation.

SFLA is one of the successful meta-heuristic methods in the evolutionary computation [8]. SFLA simultaneously takes benefit of the genetic-based memetic algorithms and the social-based PSO algorithms [20]. Different from the PSO algorithm, SFLA considers the evolution of both genes and memes. Elbeltagia et al. compared among five meta-heuristic methods including PSO and SFLA [7]. So far, SFLA has been used in many optimization problems such as cloud resource scheduling [28, 30, 38], identification of astrocytoma [36], planning and scheduling [3, 33], acoustic emissions waveform analysis [23], fuel management optimization [2], vehicle routing problem [5, 24], grade identification of astrocytoma [36], and photovoltaic model identification [14]. In this paper, Shuffled Frog Leaping Programming (SFLP) inspired by SFLA, is proposed as a novel model that aims at finding the mathematical solution for symbolic regression problems with the principles and theories of the memetic-based algorithm. The rest of this paper is formed as follows. In Section 2, a brief overview of SFLA is described. Section 3 presents the proposed SFLP. In Section 4, the performance of SFLP is tested on ten well-known benchmark datasets and the results obtained are compared with some existing algorithms. Also, the analysis of the performance sensitivity of the parameter settings for the proposed SFLP is presented in this section. Finally, the conclusion and further work are provided in Section 5.

2. Shuffled Frog Leaping Algorithm (SFLA)

SFLA is originally inspired by imitating, observing, and modeling of frogs' behavior when they are seeking for a location containing the highest quantity of food [8]. Eusuff and Lansey have first introduced SFLA in 2003. This method is capable of dealing with many complex optimization problems that are nonlinear, non-differentiable, and multi-modal [31]. The most prominent benefit of SFLA is its fast convergence speed [7]. SFLA takes advantage of both a memetic algorithm based on genetic and PSO algorithm depending on the social behavior [20]. The flowchart in figure 1 illustrates the SFLA approach. This algorithm is affected by the natural memetic benefits from the population with a random search. In this algorithm, several communities that contain groups of potential solutions are formed. Each solution represents a frog in the algorithm. Also, these communities are called memeplexes, and inside each one, the frogs perform a local search. Inside each memeplex, the behavior of a frog can be affected by other frogs' behaviors, and through a process of memetic evolution, its behavior improves. With the lapse of some evolutionary steps, the memeplexes are mixed and then new memeplexes are developed using the shuffling process. Using the shuffling, the exchange of information is done among local searches, leading to a move toward a global optimum. The two processes of shuffling and local search continue until the convergence target is fulfilled (1). SFLA can be delineated in the following:

- SFLA holds a population of possible solutions F, defined by a group of virtual frogs (n).
- According to their fitness, frogs are sorted descendingly, and then divided into subsets named as memeplexes (m).
- Frog i is shown as $X_i = (X_{i1}, X_{i2}, \dots, X_{iS})$ where S represents the number of variables.

- Inside each memeplex, the worst X_w and the best X_b frogs are identified based on fitness.

- Frog with the global best fitness X_g is identified.

- The fitness of the worst frog is improved using the following Equation:

$$D_i = rand()(X_b - X_w) \tag{1}$$

$$X_{wnew} = X_{wold} + D_i(-D_{max} \leq D_i \leq D_{max}) \tag{2}$$

where rand stands for a random number in [0,1]; D_i is the frog leaping step size of the $\mathrm{i-th}$ frog. D_{max} is the maximum step allowed to make changes in a frog position. If the fitness of the new X_w is higher than the current one, X_w will be accepted. If it is not improved, then Equations (1) and (2) are repeated by replacing X_b with X_g. If no improvement becomes possible in the solution, a new X_w will be generated randomly. Repeat the updating operation for a number of iterations. The local search flowchart of SFLA is demonstrated in figure 2.

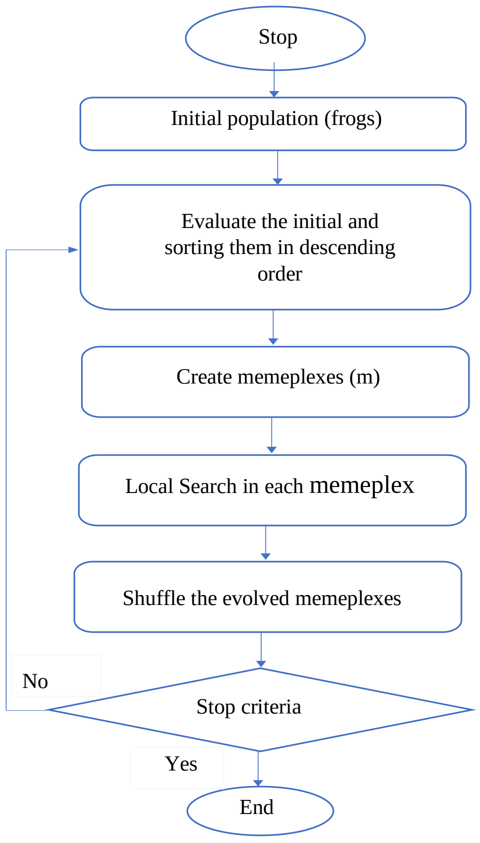
Stop
Initial population (frogs)
Evaluate the initial and sorting them in descending order
Create memeplexes (m)
Local Search in each memeplex
Shuffle the evolved memeplexes
No
Stop criteria
Yes
End

Figure 1. Flowchart of SFLA.

After some memetic evolutionary steps inside each memeplex, the obtained solutions of evolved memeplexes are replaced into the new population. This process is called shuffling. The shuffling process encourages a global information exchange among the frogs. Afterward, the population is sorted in a descending manner based on the performance value, and the position of the best frog of the population is updated. Next, frogs are repartitioned into memeplexes, and the algorithm carries on the evolution within each memeplex until the convergence criteria are met.

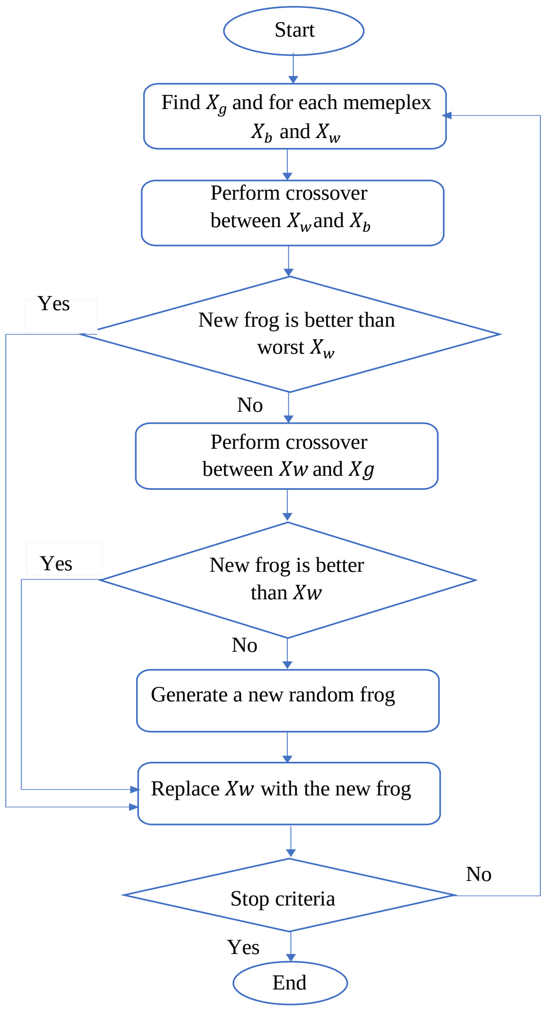
Start
Find X_g and for each memeplex X_b and X_w
Perform crossover between X_w and X_b
Yes
New frog is better than worst X_w
No
Perform crossover between Xw and Xg
Yes
New frog is better than Xw
No
Generate a new random frog
Replace Xw with the new frog
No
Stop criteria
Yes
End

Figure 2. Flowchart of Local Search.

3. Shuffled Frog Leaping Programming (SFLP)

SFLP is an extension of SFLA, which aims at finding an explicit mathematical expression for a given set of input and one output by applying the terminals and non-terminals defined in the algorithm. This type of programming intends to focus on the representation of the problem using a more complicated structure as the case between GP and GA. Despite the conventional regression, where the coefficients of functions are not calculated, the generated program carries out an extensively structured search in an evolving search space of symbolic functions. Solutions use the tree representation, which contains a group of terminals and non-terminals. Terminals are constant or variable that usually are taken from mathematical or arithmetic operators and domain-specific functions that are selected inconsistent with the problem.

In this work, SFLP is applied to symbolic regression problems to test its capability of introducing solutions. The global exchange of information among frogs and their interactions are the key elements of evolving memes in the proposed method. By employing tree representation for the structure of frogs, the storage of expressions in the computer memory is facilitated. Besides, the operations of the swap, add and remove of subtrees are done so quickly.

One of the main features of SFLP is the ability to do the local search effectively by dividing the initial population in to a number of sub-populations. In this way, in each sub-population we can find the local best much more effectively, and by means of an information exchange mechanism we can find the global best found among the sub populations.

Thus, such structure should literally outperform algorithms such as Genetic Programing in which in GP there is no such population segmentation. It is also shown in the experimental results that population segmentation structure helps SFLP outperform GP in practice. In the algorithm, the performance of each computer program is evaluated to measure the quality of each frog position, which is named the fitness measurement. This measurement shows how much the result of the obtained solution fits with the desired function. In the fitness measurement procedure, there is a tendency for raw fitness towards zero, as in [32]. The pseudo-code of SFLP is defined in Algorithm 1.

In order to distribute the initial population among memeplexes, first we sort the population in a descending order. Then, the best individual in the initial population is placed in the first memeplex, and the second-best individual in the second memeplex, and so on and so forth.

For the last memeplex ($n-th$), we let this memeplex selects two best individuals of the initial population i.e. n and $n+1$. In this way, we let each memeplex has a fair quality in terms of solution accuracy.

The raw fitness is calculated by cumulating the absolute errors between the obtained and the desired functions derived from a number of fitness cases. Hits measurement is a criterion usually calculated in symbolic regression studies.

Algorithm 1: Pseudo-code of SFLP

```
Generate random frogs of F with Ramped half-and-half method
Evaluate the frogs' fitness
Divide the whole frogs into m memeplexes, and each memeplex
   contains $n$ frogs
**for** max iteration **do**
    **for** each memeplex **do**
        Construct a sub-memeplex Set $i = 0$;
        **for** $i < Nstep$ **do**
            Performing cross-over between worst frog and local
               best frog in every sub-memeplex
            Improving constant numbers of resultant new tree
            **if** $X_w \leq X_{wnew}$ **then**
                replace $X_w$ with $X_{wnew}$
            **else**
                Performing cross-over between worst frog in
                   sub-memeplex and global best frog
                Improving constant numbers of resultant new tree
                **if** $X_w \leq X_{wnew}$ **then**
                    replace $X_w$ with $X_{wnew}$
                **else**
                    Perform mutation on local best frog and replace
                       replace $X_w$ with it
                **end if**
            **end if**
            $i = i + 1$
        **end for**
    **end for**
    Shuffle the memeplexes and update global best frog
    Check termination criteria
**end for**
```

The quantity of the absolute error of fitness cases with the values lower than the hits criterion will form the hit measurement. Some modifications are exerted on the GP basic operators to introduce the operators best fitted with the nature of SFLP, which are explained in the followings:

3.1. Cross-over

In SFLP, a cross-over operator is used to transfer the behavior of the local or global best frog to the worst frog of the current sub-memeplex. To perform cross-over, trees of the worst frog and the best frog in the sub-memeplex are considered as inputs. By swapping the random subtrees of the inputs the cross-over is performed. The proposed operator used in SFLP proceeds with the following steps:

- Select the worst and the best frogs based on their fitness in the sub-memeplex. The frog with the best fitness is selected and named as mimicked frog and the frog with the worst fitness as the memetic frog.
- Select a random sub-tree in each mimicked frog and the memetic frog.
- Replace the selected subtree of the memetic frog by the selected sub-tree of the mimicked frog.

The resultant tree is a new frog whose depth should be checked. The transition process is shown in figure 3. After the changed memetic frog was created, its fitness is evaluated and compared with the fitness of the first memetic frog. If the fitness is better than the initial memetic frog, the worst frog is replaced with it; otherwise, this process is repeated using the best frog of the whole population along with the previous memetic frog.

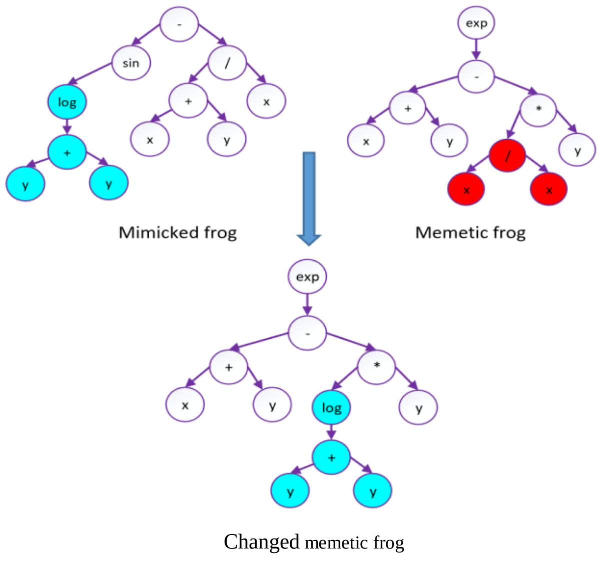

Figure 3. Proposed Cross-over.

3.2. Mutation

The mutation operator in SFLP causes random changes in the structure of the local best frog of the current sub-memeplex, while two previous performed cross-overs on the local best and global best frogs have not been successful to suggest a new candidate with better fitness comparing to the memetic frog. At this stage, SFLA generates a new individual to replace the memetic frog, while in SFLP, the local best frog is mutated for replacement. More specifically, the mutation operator selects a random point in the structure of the local best frog and then removes the selected point and its sub-tree at the current point and then inserts a subtree generated randomly in its place.

To control an appropriate replacement during the mutation operator, the depth of the generated sub-tree is checked to be in the valid range using the parameter of maximum tree depth. The relevant process is illustrated in figure 4 in which an optional candidate frog is mutated using the described process of replacing a random sub-tree with a generated one. As the next step, the performance of the mutated frog is evaluated and then inserted into the population.

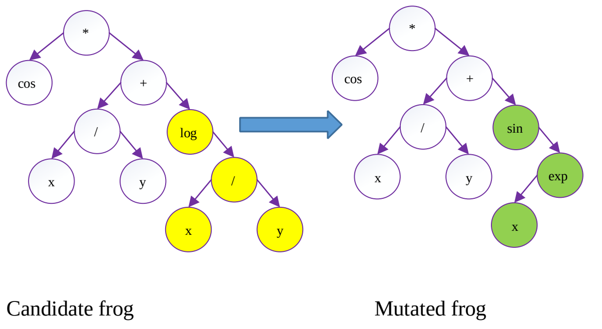

Figure 4. Basic mutation.

The frogs' target is to move towards the best solution by improving their memes. Others can use either the ideas from the best frog from the memeplex as a local search or from the currently found global best. Considering the selection of the memeplex best, it is not always suitable to use the best frog because the frogs would tend to focus about that specific frog that may be a local optimum. Therefore, a subset of the memeplex called a sub-memeplex is picked up [8]. To start the local search, each memeplex acts as an independent culture. A sub-memeplex including q frogs in the memeplex is selected randomly, where $q < n$. Conceptually, since the local meme of each sub-memeplex is employed to guide its frogs toward the local best frog, it is rational to use the local best frog of each sub-memeplex for mutation. If the global best frog is mutated as the last operation on each sub-memeplex, the convergence of the population stops at a certain value, because the whole population mimics the behavior of the first best frog. As a result, the information of each sub-memeplex is lost, and the whole structure of the algorithm becomes meaningless.

3.3. Constant creation method

One of the challenging fields of automatic programming is the improvement of constant numbers during the evolution. For this reason, methodologies such as ABCP use the value of one as the constant number for the terminal with no changes during the evolution. In this way, no attempt will be made to estimate the constant numbers in the target solution.

In SFLP, we proposed a novel method to improve the constant numbers. The inputs of this method are two trees named as W and B, which stand for the worst tree and the best tree, respectively. The number of constants in W is q and $W(i)$ is the $i-th$ constant number of W, and i occurs between 0 and q. Also, the number of constants in B is p and $B(j)$ is the $j-th$ constant number of B and j occurs between 0 and p. Each constant number in W is randomly changed to one of the constant numbers of B by the formula inspired by the PSO algorithm. The equation is displayed below:

$$W(i) = W(i) + 2 * rand((B(j) - W(i))) \tag{3}$$

In the equation, rand is a random number in the interval of [0,1]. This method will help to use an interval of constant numbers in GP systems when we define constant numbers. Changes of the constant numbers will be towards better ones, which increase the possibility of using better constant numbers in W.

Algorithm 2 Constant number creation

```
Inputs: $W, B$ Outputs: $W$ $q = nconst(W)$; $p = nconst(B)$;
for $i = 1 : q$ do
    $j = randi(1, p)$
    $j = randi(1, p)$
    $W(i) = W(i) + 2 * rand((B(j) - W(i)))$
endfor
```

where $nconst()$ is a function that counts constant numbers of a tree. $randi$ is also a function that generates uniformly distributed pseudo-random integers within a given interval.

4. Experiments and results

4.1. Experimental results

In order to find out about the performance of SFLP, this method was compared with algorithms of this class such as BBP [11], and GSP [12]. Some techniques based on GP, namely, semantic similarity-based cross-over (SSC) [29] plus standard cross-over (SC) [21], soft brood selection (SBS) [22], no same mate(NSM) [13], context aware cross-over (CAC) [25], and semantics-aware cross-over (SAC) [27]. The performance criteria for comparison among them are a set of problems of symbolic regression employed in references [15, 17]. These ten real-valued problems are classified into three categories:

1. polynomial functions
2. logarithm, trigonometric, and square root functions
3. bivariate functions

Their problem definitions are given in table 1. The number of function node evaluations was considered during the experiment to control the computational cost for each run. This measurement has been utilized in some recent automatic programming studies [16, 29, 37]. In order to remove random correlations in the experiment, each solution was independently executed 100 times. The number of node evaluations was set to 15×106 in the experiments. To examine and compare the performance of solutions, two classic performance metrics were calculated. One was the matrix of mean best fitness and the other, matrix of the percentage of successful runs.

A successful run occurs when an individual's score on all fitness cases reaches a value < 0.01. This fitness value is called the score hit. The main and internal parameters of the SFLP, BBP, GSP and GP-based techniques are given in tables 2 and 3, respectively. The results of other considered algorithms were reported from references [18, 29], which were compared with the proposed method in tables 4 and 5. The number of successful runs and the mean best fitness values of SFLP and other techniques are presented in tables 4 and 5, respectively. In each setting, the best obtained result is written in bold face. The number of local evolutions is $Nstep$, sub-memeplex size is q, the number of memeplexes m, the memeplex size is n, and the total number of frogs is $m \times n$. The parameter values employed in the method are $Nstep = 50$, $q = 2$, $m = 100$, $n = 5$. The relevant parameter analysis for each of them has been presented in the following sections. Can be seen in tables 4 and 5, SFLP considerably outperforms other algorithms and on all benchmark functions in both aspects of the number of successful runs and the mean best fitness values.

Also, the results of table 4 are consistent with those of table 5. In evolutionary computation, the optimal search strategy is regarded as a vital aspect affecting the performance of the algorithms. Indeed, the ability to perform global and local search simultaneously is because of its hybrid nature of GA and PSO, which helps the algorithm to have an appropriate search strategy.

Table 1. Symbolic regression functions.

Functions	Fit cases
$F_1 = x^3 + x^2 + x$	$20\ random\ points \subseteq [-1,1]$
$F_2 = x^4 + x^3 + x^2 + x$	$20\ random\ points \subseteq [-1,1]$
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	$20\ random\ points \subseteq [-1,1]$
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	$20\ random\ points \subseteq [-1,1]$
$F_5 = sin(x^2)\ cos(x) - 1$	$20\ random\ points \subseteq [-1,1]$
$F_6 = sin(x) + sin(x + x^2)$	$20\ random\ points \subseteq [-1,1]$
$F_7 = log\sqrt{(x+1)\ + log\ (x2\ + 1)}$	$20\ random\ points \subseteq [0,2]$
$F_8 = x$	$20\ random\ points \subseteq [0,4]$
$F_9 = sin(x) + sin(y^2)$	$100\ random\ points \subseteq [-1,1] \times [-1,1]$
$F_{10} = 2sin(x)cos(y)$	$100\ random\ points \subseteq [-1,1] \times [-1,1]$

Table 2. Main parameters of models.

Parameter	Value
Selection	Tournament
Tournament size	3
Initial max depth	6
Max depth	15
Max depth of mutation tree	5
Non-terminals	$+, -, \times, /, sin, cos, exp, rlog$
Terminals	X, 1 for single variable problems and X, Y *and* 1 for bi-variable problems
Row fitness	Sum of absolute error on all fitness cases
Hit	when an individual's absolute error is < 0.01 on a fitness case
Successful run	When an individual's score hits on all fitness cases
Trials per treatment	100 independent runs for each value

The results obtained demonstrate that SFLP is a robust algorithm and its search strategy has a positive impact on the performance of the proposed model.

Simply, the superior performance of the SFLP should be attributed to its ability to take advantage of local search while it still searches globally.

Also, the top three methods are statistically analyzed using 1-tailed and 2-tailed multiple-problem Wilcoxon's signed rank tests in table 6. According to the table and considering $p-value < 0.05$ at $\alpha = 0.05$, SFLP significantly outperforms BBP in the case of total performance, which proves that SFLP can generate competitive solutions for the problems.

Also, it exhibits better average error than BBP based on table 5. SFLP also significantly out-performs GSP in the case of total performance, which proves that SFLP can generate competitive solutions for the problems.

Besides, it exhibits better average error than GSP based on table 5. It should be noted that SFLP introduces a new methodology of automatic programming and opens a new field for investigation toward generating programs using the genetic programming concepts. Finally, by comparing SFLP with GP, it is clear that SFLP is significantly better than GP.

4.2. Effects of number of memeplexes and size

The main parameters of SFLP were aimed to study their effect on the results of the runs of the benchmark problems. At first, the number of memeplexes, as well as the memeplex size, are variable and other parameters, namely, the number of local evolutions is $Nstep = 30$ and, sub-memeplex size is $q = 5$. It should be noted that the total number of frogs is the multiplication of the number of memeplexes m and the memeplex size n that is $m \times n$. Therefore, by studying the number of memeplexes for the fixed value of the total number of frogs, the memeplex size is implicitly investigated. By comparing SFLA with SFLP in the case of search space, SFLP performs the search in a more complex search space; therefore, larger values for parameters should be set, and on the other hand, to help the algorithm do the local search more efficiently, lower values for local search is required. In order to compare the outputs, the parameters are set in a way that the local search is performed more precisely and effectively while their values are bounded to certain values. Table 7 shows the results of different values of memeplexes and their size. According to the results obtained, the performance of SFLP improves as the number of memeplexes becomes larger.

4.3. Effects of sub-memeplex size

In order to understand the impact of sub-memeplex size on the algorithm, the parameter q showing the frogs in a sub-memeplex was varied from 2 to 10. The other parameters were set to $m = 100$, $n = 5$, and $Nstep = 30$. As it can be observed in table 8, for lower values of q the number of successful runs is increased.

Table 3. Algorithms internal parameter.

SFLP		BBP		GSP		GP	
Parameters	Values	Parameters	Values	Parameters	Values	Parameters	Values
$Number of frogs$	500	$Habitat\ size$	500	$Number\ of\ agents$	500	$Population\ size$	500
$\#\ of\ node\ evaluations$	15 × 106	$\#\ of\ node\ evaluations$	15 × 106	$\#\ of\ node\ evaluations$	15 × 106	$\#\ of\ node\ evaluations$	15 × 106
$Memeplex\ size$	5	$Pmigration$	0.9	GC_0	5	$Pcrossover$	0.9
$Number\ of\ memeplexes$	100	$Pmutation$	0.1	α	20	$Pmutation$	0.05
$Submemeplex\ size$	2			K_0	50		
$Local\ iteration$	50			$Internal\ connec-tion's\ operator$	$No\ change$, ^2,^3, $square\ root$		

Table 4. Number of successful runs of SFLP and other considered models with parameter setting of $Nstep = 50, q = 2, m = 100, and\ n = 5$.

	Functions									
Models	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SFLP	**100**	**97**	82	75	**94**	94	93	**94**	**100**	78
GSP	92	85	**90**	**86**	84	80	75	79	76	65
BBP	**100**	96	80	69	90	**98**	**97**	90	77	**86**
ABCP	89	50	22	12	57	87	58	37	33	21
SC	48	22	7	4	20	35	35	16	7	18
NSM	48	16	4	4	19	36	40	28	4	17
SAC2	53	25	7	4	17	32	25	13	4	4
SAC3	56	19	6	2	21	23	25	12	3	8
SAC4	53	17	11	1	20	23	29	14	3	8
SAC5	53	17	11	1	19	27	30	12	3	8
CAC1	34	19	7	7	12	22	25	9	1	15
CAC2	34	20	7	7	13	23	25	9	2	16
CAC4	35	22	7	8	12	22	26	10	3	16
SBS31	43	15	9	6	31	28	31	17	13	33
SBS32	42	26	7	8	36	27	44	30	17	27
SBS34	51	21	10	9	34	33	46	25	26	33
SBS41	41	22	9	5	31	34	38	25	19	33
SBS42	50	22	17	10	41	32	51	24	24	33
SBS44	40	25	16	9	35	43	42	28	33	34
SSC8	66	28	22	10	48	56	59	21	25	47
SSC12	67	33	14	12	47	47	66	38	37	51
SSC16	55	39	20	11	46	44	67	29	30	59
SSC20	58	27	10	9	52	48	63	26	39	51

The frogs of each memeplex have more chance to exchange the existing information among them, and therefore, it helps them to move towards the local best frog in each memeplex.

It is better to select a smaller size of sub-memeplex for more complex problems to introduce better global solutions and to let the local search exchange the information within each sub-memeplex in depth.

Table 6. Results of the multiple-problem Wilcoxon's signed rank test for SFLP, BBP, and GSP.

Methods	SFLP vs BBP	SFLP vs GSP	SFLP vs GP
p-Value(1-tailed)	0.8499	0.0008	0.0001
p-Value(2-tailed)	0.3387	0.0036	0.0001

4.4. Effects of Nstep size

The number of evaluation steps for different Nstep size is shown in table 9, in which the values are 10, 20, 30, 50, and 100.

Table 5. Mean best fitness values of SFLP and other mentioned methods with parameter setting of $Nstep = 50, q = 2, m = 100, and\ n = 5$.

	Functions									
Models	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SFLP	**<0.01**	**<0.01**	0.01	**0.02**	**0.01**	**<0.01**	0.01	**<0.01**	**<0.01**	**0.05**
GSP	0.01	0.05	**<0.01**	0.1	0.05	0.02	0.06	0.1	0.47	1.06
BBP	**<0.01**	**<0.01**	0.02	0.03	**0.01**	**<0.01**	**<0.01**	0.02	0.20	0.09
ABCP	0.01	0.05	0.07	0.1	0.05	0.02	0.06	0.1	0.47	1.06
SC	0.18	0.26	0.39	0.41	0.21	0.22	0.13	0.26	5.54	2.26
NSM	0.16	0.29	0.34	0.4	0.19	0.17	0.11	0.19	5.44	2.16
SAC2	0.16	0.27	0.42	0.5	0.22	0.23	0.15	0.27	5.99	3.19
SAC3	0.13	0.27	0.42	0.48	0.18	0.23	0.15	0.27	5.77	3.13
SAC4	0.15	0.29	0.41	0.46	0.17	0.22	0.15	0.26	5.77	3.03
SAC5	0.15	0.29	0.4	0.46	0.17	0.21	0.15	0.26	5.77	83.03
CAC1	0.33	0.41	0.51	0.53	0.31	0.42	0.17	0.35	7.83	4.4
CAC2	0.32	0.41	0.52	0.53	0.31	0.42	0.17	0.35	7.38	4.3
CAC4	0.33	0.41	0.53	0.53	0.3	0.42	0.17	0.35	7.8	4.32
SBS31	0.18	0.29	0.3	0.36	0.17	0.3	0.15	0.19	4.78	2.75
SBS32	0.18	0.23	0.28	0.36	0.13	0.28	0.1	0.18	4.47	2.77
SBS34	0.16	0.23	0.31	0.33	0.13	0.21	0.11	0.19	4.17	2.9
SBS41	0.18	0.26	0.27	0.38	0.12	0.2	0.13	0.2	4.4	2.75
SBS42	0.12	0.24	0.29	0.3	0.12	0.18	0.1	0.16	3.95	2.76
SBS44	0.18	0.24	0.33	0.35	0.15	0.16	0.11	0.19	2.85	1.75
SSC8	0.09	0.15	0.19	0.29	0.1	0.09	0.07	0.15	3.91	1.53
SSC12	0.07	0.17	0.18	0.28	0.1	0.12	0.07	0.13	3.54	1.45
SSC16	0.1	0.15	0.23	0.26	0.1	0.1	0.06	0.14	3.11	1.22
SSC20	0.08	0.18	0.23	0.3	0.09	0.1	0.06	0.14	2.64	1.23

The other parameters are $m = 100$, $n = 5$, and $q = 2$. It can be realized from table 9 that a higher value for Nstep leads to a greater number of successful runs. As a result, this parameter helps the algorithm converge by generating better solutions, whereas it lengthens the evaluation process. For a reasonable evaluation time with considering the benchmark functions, it is recommended to set $Nstep$ to 50 because for values higher than 50, despite a massive increase in the evaluation time, the number of successful runs does not have a bilateral growth. Tables 4 and 5 in the above section have considered $Nstep = 50$. It seems rational to boost the algorithm by selecting higher values for either of the parameters while the evaluation process of the algorithm becomes longer. Parameters such as Nstep and the population of frogs have direct impacts on the algorithm performance.

Table 7. Number of successful runs for different memeplex size. The total number of frogs is fixed and is equal to 500.

Number of memeplexes (memeplex size)	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
5(100)	65	52	25	22	49	55	36	44	54	42
10(50)	74	56	28	26	58	59	46	51	60	48
25(20)	88	65	33	31	61	68	56	58	64	57
50(10)	91	71	39	37	70	75	62	65	77	61
100(5)	95	80	60	58	79	83	73	78	84	64

Table 8. Number of successful runs for different sub-memeplex size(q).

Sub-memeplex	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
2	89	81	51	52	64	69	47	68	100	71
3	92	72	30	37	61	83	48	69	99	68
4	90	71	42	31	50	89	50	19	89	67
5	94	82	37	34	64	73	41	73	92	65
6	86	62	36	32	48	76	35	69	79	63
7	91	59	41	36	62	75	32	68	78	55
8	84	65	28	32	54	66	26	64	86	53
9	87	60	26	27	45	64	17	60	78	51
10	81	44	18	14	27	42	18	56	66	45

Table 9. Number of evaluation steps for different Nstep size.

Number of evaluation steps (Nstep)	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
10	67	61	41	49	53	67	61	69	76	55
20	85	71	46	55	69	74	73	77	83	61
30	96	80	60	60	75	79	81	84	91	64
50	100	91	64	74	90	90	89	85	100	69
100	100	93	74	75	92	90	92	91	100	70

5. Conclusion

In this paper, a novel method of evolutionary computation was proposed, which is an extension of SFLA to study the symbolic regression problems. The new approach named as shuffled frog leaping programming is able to evolve expressions and constants in the tree representation

and build mathematical functions automatically. Also, a new method for creating constant numbers is presented, which helps to change the constant numbers towards the target solution. The proposed approach was tested on a set of symbolic regression benchmark problems which exhibited to be competitive with other popular automatic programming algorithms such as BBP, GSP, canonical GP, and different approaches of GP. Based on the simulations results, the developed model outputs are superior to all of the considered well-known methods in the symbolic regression problems. Also, the statistical analysis, Wilcoxon's signed rank test, proves that the overall performance of the proposed method is remarkably better than the other competitors. The superior performance of SFLP is due to its ability to simultaneously do a local search while still searching globally.
Overall, the success rate of the algorithm is acceptable and promising. In general, this number upsurges while the number of frogs in the population or each sub-memeplex increases but it increases the function evaluation required to find the solution. Also, it was noted that the success rate had a higher sensitivity to the memeplexes number comparing to the frogs' number in a memeplex. Therefore, it reinforces the idea of exploring more regions in the domain. Also, the success rate decreased when the number of frogs in a memeplex reached a particular value. This matter may refer to the nature of this type of problem. It is recommended to research on this phenomenon of SFLP. In conclusion, SFLP is capable of solving modeling and automatic programming problems effectively. As a future work, it is planned to apply SFLP to solve complex real-world regression problems and study on some modifications that lead to increase speed and accuracy.

References

[1] Alavi, A. H., & Gandomi, A. H. (2011). A robust data mining approach for formulation of geotechnical engineering systems. Engineering Computations, vol. 28, no. 3, pp. 242-274.

[2] Arshi, S. S., Zolfaghari, A., & Mirvakili, S. (2014). A multi-objective shuffled frog leaping algorithm for in-core fuel management optimization. Computer Physics Communications, vol. 185, no. 10, pp. 2622-2628.

[3] Bala, S. M., & Meenakumari, R. (2015). Optimum generation scheduling using an improved adaptive shuffled frog leaping algorithm. In Cognitive computing and information, international conference on, pp. 1-6.

[4] Brameier, M. F., & Banzhaf, W. (2010). Linear genetic programming (1st ed.): Springer Publishing Company, Incorporated.

[5] de Oliveira da Costa, P. R., Mauceri, S., Carroll, P., & Pallonetto, F. (2018). A genetic algorithm for a green vehicle routing problem. Electronic Notes in Discrete Mathematics, vol. 64, pp. 65-74. 8th International Network Optimization Conference - INOC 2017.

[6] Eiben, A. E., & Smith, J. E. (2003). Introduction to evolutionary computing: SpringerVerlag.

[7] Elbeltagi, E., Hegazy, T., & Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. Advanced Engineering Informatics, vol. 19, pp. 43-53.

[8] Eusuff, M., Lansey, K., & Pasha, F. (2006). Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. Engineering Optimization, vol. 38, pp. 129-154.

[9] Ferreira, C. (2001). Gene expression programming: a new adaptive algorithm for solving problems. CoRR, cs.AI/0102027, pp. 87-129.

[10] Gan, Z., Chow, T. W., & Chau, W. (2009). Clone selection programming and its application to symbolic regression. Expert Systems with Applications, vol. 36, pp. 3996-4005.

[11] Golafshani, E. M. (2015). Introduction of biogeography-based programming as a new algorithm for solving problems. Applied Mathematics and Computation, vol. 270, pp. 1-12.

[12] Gustafson, S., Burke, E. K., & Krasnogor, N. (2005). On improving genetic programming for symbolic regression. In Ieee congress on evolutionary computation, vol. 1, pp. 912-919.

[13] Hasanien, H. M. (2015). Shuffled frog leaping algorithm for photovoltaic model identification. IEEE Transactions on Sustainable Energy, vol. 6, pp. 509-515.

[14] Hoai, N. X., McKay, R. I., Essam, D., & Chau, R. (2002). Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. In Evolutionary computation. cec '02. proceedings of the 2002 congress, Vol. 2, pp. 1326-1331.

[15] Hoang, T.-H., Essam, D., McKay, B., & Hoai, N.-X. (2007). Advances in computation and intelligence: Second international symposium, isica 2007 Wuhan, china, 2007 proceedings, pp. 137–146. Berlin, Heidelberg: Springer Berlin Heidelberg.

[16] Johnson, C. G. (2009). Genetic programming: 12th European conference, eurogp 2009 Tübingen, Germany, April 15-17, 2009 proceedings, pp. 97-108. Berlin, Heidelberg: Springer Berlin Heidelberg.

[17] Karaboga, D., Ozturk, C., Karaboga, N., & Gorkemli, B. (2012). Artificial bee colony programming for symbolic regression. Inf. Sci., vol. 209, pp. 1-15.

[18] Keijzer, M. (2003). Genetic programming: 6th European conference, eurogp 2003 Essex, uk, April 14–16, 2003 proceedings. pp. 70-82. Berlin, Heidelberg: Springer Berlin Heidelberg.

[19] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Neural networks, proceedings., ieee international conference, vol. 4, pp. 1942-1948.

[20] Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection. Cambridge, MA, USA: MIT Press.

[21] L.Altenberg. (1996). Advances in genetic programming (Vol. 2). Cambridge, USA: MIT Press.

[22] Li, J., Zhao, L., Yue, J., & Yang, Y. (2015). Acoustic emissions waveform analysis for the recognition of coal rock stability. In International conference on information technology systems and innovation (icitsi), pp. 1-6.

[23] Luo, J., Li, X., Chen, M.-R., & Liu, H. (2015). A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows. Information Sciences, vol. 316, pp. 266-292.

[24] Mahanipour, A., & Nezamabadi-pour, H. (2018). GSP: an automatic programming technique with gravitational search algorithm. Applied Intelligence, vol. 49, pp. 1502-1516.

[25] Majeed, H., & Ryan, C. (2006). Genetic programming: 9th European conference, eurogp 2006, Budapest, Hungary, April 10-12, 2006. proceedings. pp. 36-48. Berlin, Heidelberg: Springer Berlin Heidelberg.

[26] Miller, J. F., & Harding, S. L. (2008). Cartesian genetic programming. In Proceedings of the 10th annual conference companion on genetic and evolutionary computation, pp. 2701-2726. New York, USA: ACM.

[27] Nguyen, Q. U., Nguyen, X. H., & O'Neill, M. (2009). Semantic aware crossover for genetic programming: The case for real-valued function regression. In Genetic programming: 12th European conference, eurogp 2009 Tübingen, Germany, 2009 proceedings, pp. 292–302. Berlin, Heidelberg: Springer Berlin Heidelberg.

[28] Niu, K., Wang, J. D., Zhang, H. W., & Na, W. (2015). Cloud resource scheduling method based on estimation of distribution shuffled frog leaping algorithm. In Third international conference on cyberspace technology, pp. 1-6.

[29] N.Q.Uy, M. N. R. E.-L., N.X.Hoai. (2010). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines, vol. 12(2), pp. 91-119.

[30] ping Luo, J., Li, X., & rong Chen, M. (2014). Hybrid shuffled frog leaping algorithm for energy-efficient dynamic consolidation of virtual machines in cloud data centers. Expert Systems with Applications, vol. 41(13), pp. 5804-5816.

[31] Rahimi-Vahed, A., & Mirzaei, A. H. (2007). A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem. Computers & Industrial Engineering, vol. 53(4), pp. 642-666.

[32] R.Whittaker. (1998). Island biogeography: ecology, evolution and conservation. Oxford, UK: Oxford University Press.

[33] Samuel, G. G., & Rajan, C. C. A. (2015). Hybrid: Particle swarm optimization–genetic algorithm and particle swarm optimization–shuffled frog leaping algorithm for long-term generator maintenance scheduling. International Journal of Electrical Power & Energy Systems, vol. 65, pp. 432-442.

[34] Shirakawa, S., Ogino, S., & Nagao, T. (2008). Dynamic ant programming for automatic construction of programs. IEEJ Transactions on Electrical and Electronic Engineering, vol. 3(5), pp.540-548.

[35] Simon, D. (2008, Dec). Biogeography-based optimization. IEEE Transactions on Evolutionary Computation, vol. 12(6), pp. 702-713.

[36] Subashini, M. M., Sahoo, S. K., Sunil, V., & Easwaran, S. (2016). A non-invasive methodology for the grade identification of astrocytoma using image processing and artificial intelligence techniques. Expert Systems with Applications, vol. 43, pp. 186-196.

[37] Wong, P., & Zhang, M. (2008). Scheme: Caching subtrees in genetic programming. In Ieee congress on evolutionary computation (ieee world congress on computational intelligence), pp. 2678-2685.

[38] Zhan, Z.-H., Li, Y., & Zhang, J. (2016). Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version. IEEE Transactions on Parallel and Distributed Systems, vol. 28, pp. 1-1.

نشریه هوش مصنوعی و داده کاوی

برنامه نویسی جهش قورباغه برای حل مسائل رگرسیون سمبلیک

محمد عبدالهی[1]* و مهدی علیباری شوره دلی[2]

[1] گروه مهندسی کامپیوتر ، دانشکده کامپیوتر، دانشگاه صنعتی خواجه نصیرالدین طوسی، تهران، ایران.

[2] گروه مهندسی مکاترونیک ، دانشکده الکترونیک، دانشگاه صنعتی خواجه نصیرالدین طوسی، تهران، ایران.

ارسال ۲۰۱۸/۱۲/۱۹؛ بازنگری ۲۰۲۰/۰۱/۰۸؛ پذیرش ۲۰۲۰/۰۲/۰۵

چکیده:

مدل‌های مختلف از برنامه نویسی خودکار الهام گرفته از تکنیک‌های محاسباتی تکاملی وجود دارد. با توجه به اهمیت طراحی مکا نیزمی خودکار برای کشف فضای جستجو پیچیده‌ای از مسائل ریاضی که در آن روش‌های عددی ناموفق هستند، محاسبات تکاملی به طور گسترده مورد مطالعه و کاربرد برای حل مسائل دنیای واقعی قرار گرفته‌اند. یکی از الگوریتم‌های معروف در مسئله بهینه‌سازی، الگوریتم جهش قورباغه (SFLA) است که از رفتار قورباغه‌ها برای پیدا کردن بیشترین مقدار غذای موجود با جستجوی در محیط اطراف خود هم به صورت محلی و هم سراسری می‌باشد. نتایج ثابت می‌کند که SFLA برای حل مسائل موثر است. در این مقاله، برنامه نویسی جهش قورباغه (SFLP) با الهام از الگوریتم جهش قورباغه SFLA به عنوان یک نوع جدید از مدل برنامه نویسی خودکار برای حل مسائل رگرسیون سمبلیک بر اساس نمایش درخت پیشنهاد شده است. همچنین در SFLP یک مکانیزم جدید برای بهبود عدد ثابت در ساختار درخت پیشنهاد شده است. به این ترتیب، با استفاده از روش پیشنهادی، زمینه‌های مختلفی از مسائل ریاضی را می‌توان حل کرد. برای آگاهی یافتن در مورد عملکرد راه حل‌های تولید شده توسط SFLP، آزمایش‌های مختلف با استفاده از تعدادی از توابع معیار انجام شد. نتایج نیز با دیگر الگوریتم‌های برنامه نویسی تکاملی مانند برنامه نویسی مبتنی بر جغرافیای زیستی (BBP) برنامه نویسی جستجوی گرانشی (GSP)، برنامه نویسی ژنتیک (GP) و بسیاری از انواع GP ها مقایسه شده است.

کلمات کلیدی: برنامه نویسی ژنتیک، الگوریتم جهش قورباغه، برنامه نویسی جهش قورباغه، مسائل رگرسیون.