TRANSFER LEARNING BASED ON SEMANTIC APPROXIMATION TECHNIQUES IN GENETIC PROGRAMMING

Thi Huong Chu[1]

https://doi.org/10.56651/lqdtu.jst.v11.n01.361.ict

Abstract

Transfer learning aims to reuse the knowledge taken from different but related source domains to enhance learning performance on target domains. In Genetic Programming (GP), transfer learning methods have helped GP systems expand their applicability in real applications. In this work, two transfer learning methods based on semantic approximation techniques (SAT) for GP are proposed. The proposed methods use the best individuals in the final generation of the source task as transfer material to the target task. Before transferring, SAT is applied on these individuals in order to simultaneously satisfy two objectives of reducing the size of the transferred individuals and preserving the knowledge of them. The methods are tested on ten symbolic regression datasets and compared with GP without transfer learning and two GP transfer learning methods. Experimental results showed that the proposed methods could effectively extract more useful knowledge from source domains to help to improve the performance of GP systems on the target domains.

Index terms

Genetic Programming, Semantic Approximation, Transfer learning

1. Introduction

Transfer learning aims to improve the learning performance on target domains by transferring the knowledge taking from different but related source domains. Along with the above benefit, transfer learning has become a popular and promising area in machine learning [1]. Generally, a formal definition of transfer learning has been characterized as follows [1], [2]: Given some/an observation(s) corresponding to $m^S \in \mathbb{N}^+$ source domain(s) and task(s) (i.e., $\left\{(\mathcal{D}_{Si}, \mathcal{T}_{Si}) \,|i = 1, ..., m^S\right\}$) and some/an observation(s) about $m^T \in \mathbb{N}^+$ target domain(s) and task(s) (i.e., $\left\{(\mathcal{D}_{Tj}, \mathcal{T}_{Tj}) \,|j = 1, ..., m^T\right\}$), transfer learning uses the knowledge implied in the source domain(s) to improve the performance of the learned decision functions $f^{Tj}\left(j = 1, ..., m^T\right)$ on the target domain(s) where a domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ consists of the feature space $\mathcal{X}$ and the marginal probability distribution $P(X)$, $X = \{x_1, x_2, ..., x_n\} \in \mathcal{X}$, and a task $\mathcal{T} = \{\mathcal{Y}, f(X)\}$ consists of a label space $\mathcal{Y}$ and a decision function $f(X)$.

[1]Faculty of Information Technology, Le Quy Don Technical University

GP is considered as a machine learning method that allows computer programs encoded as a set of tree structures to be evolved using an evolutionary algorithm [3]. A GP system is started by initializing a population of individuals. The population is then evolved for a number of generations using genetic operators such as crossover and mutation. At each generation, the individuals are evaluated using a fitness function, and a selection schema is used to choose better individuals to create the next population. The evolutionary process is continued until a desired solution is found or when the maximum number of generations is reached. GP has been successfully applied in many real applications. As with other machine learning techniques, the research on transfer learning in GP has rapidly increased in recent years to extend the applicability of GP. There are some common approaches in these studies such as transferring knowledge from generations evolved using a source domain to a target domain [4], [5], [6], [7], [8], [9] using the useful source domain instances taken part in a target task [2], [10], [11] or transferring data from a complete source domain to a different, incomplete target domain [12], [13]. This work is one of the few attempts to improve the learning performance of target domains inspired by two hypotheses:

- Firstly, the programs of the last generation in the source problem, going through an evolutionary process, will contain the most knowledge about this domain. Thus, these programs will have a potential to extract reusable knowledge of the source domain to improve learning in the target domain.
- Secondly, replacing a subtree in a program with another subtree of approximate knowledge may not significantly affect the ability of this program to transfer knowledge from the source problem to the target problem.

The first hypothesis has been proven in several recent studies [4], [5], [7], [8], [9]. These studies have showed the effectiveness of extracting useful knowledge of the programs of the last generation in the source problem to the target problem. For the second hypothesis, SAT proposed in our previous research [14], [15] helps to produce a (sub)program whose semantics approximates a given semantics. Based on the hypotheses, we propose two transfer learning methods in GP. In the proposed methods, the best programs in the final generation on the source problem are used as the transfer material to the target problem. Before these programs are transferred, SAT is applied on them to simultaneously satisfy the two goals of reducing the size of the transfer programs and preserving the useful knowledge of them. The main contribution of this paper is to illustrate how to use semantic approximation to preserve knowledge extracted from source domain stored in the evolved programs while reducing the size of these program. The proposed methods are thoroughly examined and compared with traditional GP and two GP transfer learning methods with the same approach on large regression problems.

2. Related work

This section briefly introduces previous work related to this work, including transfer learning methods in GP and semantic approximation techniques.

2.1. Transfer learning in GP

Research on transfer learning in GP has perhaps only received much attention in recent years. With the approach of transferring knowledge from generations evolved using a source problem to a target problem, Dinh et al. [4] proposed three methods, consisting of *FullTree*, *SubTree* and *BestTree*. In the *FullTree* algorithm, k percent of the best individuals in the final population of a source domain are chosen for initialing GP individuals on a target domain. The algorithm of the other methods is similar to that of *FullTree*. However, instead of taking the best individuals in the final generation as in *FullTree*, random subtrees from the final generation (in *SubTree*), or the best individual of all generations (in *BestTree*) are selected to initial the population in target problem. Haslam et al. [5] next extended the research in [4] with the variety of source and target problems and proposed a strategy for dynamically choosing the parameter k, namely $k\ Throttle$. In the similar approach, O'Neill et al. [6] used common subtrees in the best individuals produced in two source domains for initialization and mutation in the target domain. The experimental results showed that the methods in these studies improved the GP performance.

Recently, as the same above approach, Ardeh et al. [7], [8], [9] focused to solve the Uncertain Capacitated Arc Routing Problem by using GP transfer learning techniques. The work in [7] proposed a method called *FreqSub*. *FreqSub* extracts all subtrees from the best individuals in the final generation of source domain, and all the most frequent subtrees then are transferred as individuals of the initial population of the target problem. The work in [8], [9] introduced the framework that extracts and stores the transferred knowledge of the source domain to Probabilistic Prototype Tree (PPT). The best individuals or the winner individuals of the tournament selections in the final generation of the source problem are chosen. After that, PPT is constructed from the these individuals by calculating the probability of each node. Final, PPT is used to generate individuals in the initial population of target problem. The experimental results on solving the Uncertain Capacitated Arc Routing Problem showed that the potential for transfer learning of these methods is similar to *FullTree*, *SubTree* and *BestTree* in [4]. Wenlong Fu et al. [16] proposed a transferring schema for document classification called Output-based transfer learning. In Output-based transfer learning, GP system is run on the source domain with N independent trials to product N programs/classifiers. These programs and their mutations are then combined by a linear model for solving the target domain problem. The weights of the linear model are optimized by Particle Swarm Optimisation (PSO) with Differential Evolution (DE) technique.

Another approach is to utilize the useful source domain instances taken part in a target task. Chen et al. [2], [10], [11] proposed instance-based transfer learning methods to improve the symbol regression performance in target domains. In [2], a local weighting scheme is utilized to weight the source domain instances when used in solving a target task. The method in [10] uses differential evolution to search for optimal weights for source-domain instances during the GP evolutionary process. This method is time-consuming when the source domain has a large number of instances.

The work in [10] was then extended in [11], where a more effective and efficient instance weighting framework which attempts to alleviate the dominance of the source-domain data in the learning process was proposed. Meanwhile, a distribution estimation method is employed for providing better starting points for the search process while discarding some irrelevant or less important source-domain instances before learning regression models. Alexander Wild et al. [17], [18] used deep neural networks to identify donor programs from already-solved problems in program synthesis field. In [18], code fragments in the prior solved problem are stored and used to transfer in to new problems. In the target problem, these code fragments are estimated by a neuron network, and the code fragments with the highest ranking are selected. Then, GP run on the problem with these highest ranking fragments to generate a training corpus of programs in a step of the process of the program synthesis.

More recently, Al-Helali et al. [12] proposed a multi-tree genetic programming algorithm based feature-based transformation for transferring data from a complete source domain to a different, incomplete target domain. In this algorithm, each individual is represented by a number of trees which is equal to the number of the target features. The algorithm is used to evolve the individual that makes the transformed instances from the source domain help improving the treatment of missing values in the target domain. The limitation of the algorithm in [12] is that the transformation treats all source features and instances equally. To address this limitation, the work in [13] proposed another method that constructs an asymmetric mapping from a complete source domain to an incomplete target domain. During the construction process, feature and instance knowledge extracted from the learned models in the source domain are employed to estimate the target missing values and reduce the distribution difference between the two domains. The results showed that these methods improved the GP performance.

In this paper, two transfer learning methods based on semantic approximation techniques for GP are proposed. The most similar approach in this work is *FullTree* method [4]. Instead of transferring the best complete programs of the last generation in the source problem to the target problem as in *FullTree*, we use them as transferred material. These programs before transferring to the target problem are pruned to reduce the size but still preserve the knowledge stored in them. A detailed description of the proposed methods will be presented in the next section.

2.2. Semantic Approximation Techniques

SAT was proposed in [14], [15] allows to generate a program that approximates a given semantics. In GP, the semantics of a program is often defined as the vector of output values obtained by running that program on all samples [15], [19], [20]. SAT is described as follows. Let $S = (s_1, s_2, ..., s_n)$ be the target semantics and $sTree$ is a small randomly generated tree. The objective of SAT is to grow a tree in the form: $newTree = \theta \cdot sTree$ so that the semantics of $newTree$ is most similar to S.

Let $Q = (q_1, q_2, ...q_n)$ be the semantics of $sTree$, then the semantics of $newTree$ is $P = (\theta \cdot q_1, \theta \cdot q_2, ..., \theta \cdot q_n)$. To approximate S, θ is found so that the squared

Euclidean distance between two vectors S and P is minimal. In other words, the function $f(\theta) = \sum_{i=1}^{N}(\theta \cdot q_i - s_i)^2$ with respect to θ is minimized. The quadratic function $f(\theta)$ achieves the minimal value at θ^* calculated in Equation 1:

$$\theta^* = \frac{\sum_{i=1}^{N} q_i s_i}{\sum_{i=1}^{N} q_i^2} \tag{1}$$

The (sub)tree $newTree = \theta^* \cdot sTree$ is grown and called the approximate tree of the semantic vector S. The advantage of SAT is that the size of $newTree$ can be constrained by limiting the size of $sTree$. $sTree$ can be randomly generated by using GP's popular population initialization methods or taken from a terminal set. Figure 1 illustrates an example of SAT. In the figure, the set of $\{(x_1, x_2)\}$=$\{(0.2, 0.4); (0.2, 0.5); (0.1, 0.6)\}$ is the samples of the problem, $sTree = x_1 + x_2$, and $s = (0.9, 1.1, 1.0)$ is a given semantics. So, the semantics of $sTree$ is computed as $q = (0.6, 0.7, 0.7)$ and θ^* is caculated as 1.5 by using Equation 1.

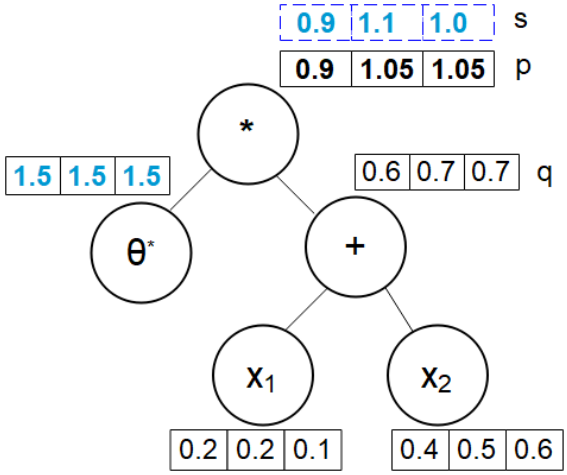


Fig. 1. An example of Semantic Approximation Techniques.

3. Proposed methods

This section introduces in detail two proposed methods implemented for transfer learning in GP, namely *Transfer Learning based on Semantic Approximation Techniques* and *Dynamic Transfer Learning based on Semantic Approximation Techniques*.

3.1. Transfer Learning based on Semantic Approximation Techniques

The first proposed method is called *Transfer Learning based on Semantic Approximation Techniques* and shortened as TLSA. Figure 2 presents the algorithm of TLSA.

In the algorithm of TLSA, $k\%$ the best individuals in the last generation of the source problem are used for transferred material to the target problem. These individuals are

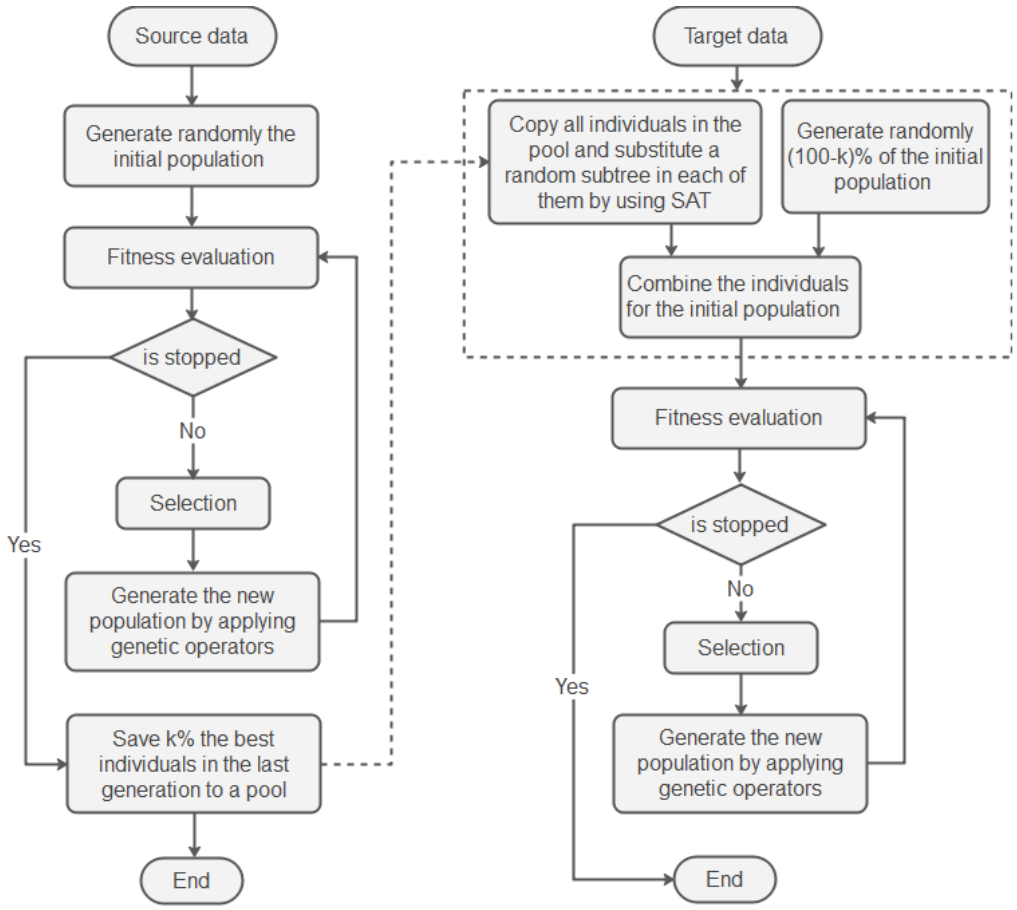
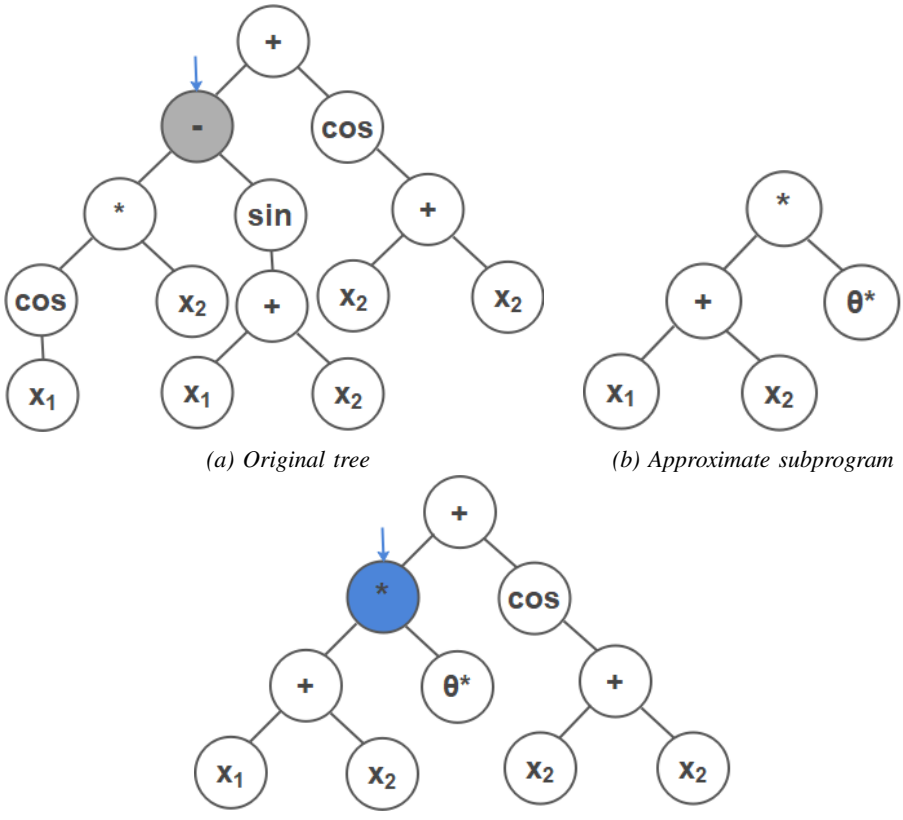


Fig. 2. Transfer Learning based on Semantic Approximation Techniques.

copied and stored in a pool. Then, a random subtree in each of them is replaced with a small and semantically approximate subprogram by using SAT, hopefully preserving the semantics of this subtree. After that, these substituted individuals are fed to the initial population of the target problem.

Figure 3 illustrates an example of pruning a tree using SAT before transferring this tree to the initial population of the target problem. Figure 3 (a) is an original tree taken from the pool which is one of the best individuals of the final generation in the source problem. A random subtree (indicated by a down arrow) of the tree is selected, and semantics of this subtree is calculated. SAT is then used to generate a small subprogram (Figure 3 (b)) in the form of $newTree = \theta \cdot sTree$ so that the semantics of this subprogram ($newTree$) approximates the semantics of this subtree. Finally, this subtree in the original tree is replaced by this approximate subprogram to obtain an initialized tree for the target problem as Figure 3 (c). It notes that the size of $sTree$ needs to be smaller than the size of the subtree -2 to reduce the size of this transferred individual. If this condition is not satisfied, the process of selecting subtree and generating $sTree$ is repeated. If no pair of subtree and $sTree$ is found after the maximum number of

trials, no subtrees in this individual are replaced.



(a) Original tree *(b) Approximate subprogram*

(c) Transferred tree

Fig. 3. An example of pruning a tree using SAT before transferring.

The rest of the initial population in the target problem are randomly generated using GP's common population initialization methods. It means that $k\%$ the substituted individuals and $(1-k)\%$ the randomly generated individuals are combined to the initial population for the target problem. The process of TLSA is then done like a traditional GP.

3.2. Dynamic Transfer Learning based on Semantic Approximation Techniques

The second method is called *Dynamic Transfer Learning based on Semantic Approximation Techniques* and shortened as DTLSA. DTLSA aims at the average size of the individuals transferred to the initial population of target problem similar to the average size of individuals generating by popular population initialization methods as in traditional GP. The structure of DTLSA is similar to that of TLSA. However, the transferred programs in DTLSA are pruned repeatedly until the size of them less than or equal to the average size of the individuals generated by random methods. Algorithm of DTLSA is illustrated in Figure 4.

In the algorithm of DTLSA, $k\%$ good individuals in the last generation of the source domain are copied to a pool as material to breed the initial population in the target

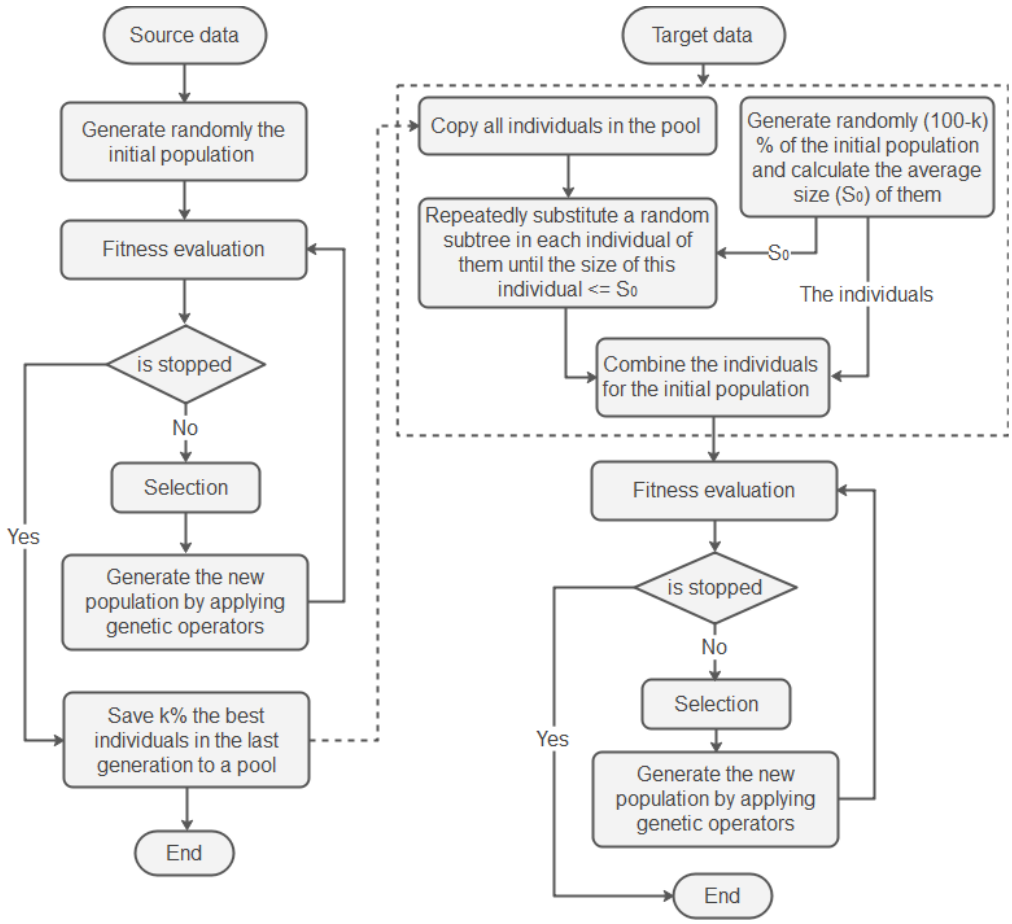


Fig. 4. Dynamic Transfer Learning based on Semantic Approximation Techniques.

domain. Simultaneously, $(100-k)\%$ remaining individuals in the initial population are randomly generated, and their average size was calculated as the pruning threshold, S_0. After that, the process of replacing a random subtree in each individual of the pool with a approximate subprogram generated by SAT is repeated until the size of them less than or equal to threshold S_0. Finally, these individuals and the randomly generated individuals are included in the initial population in the target problem.

4. Experimental settings

In order to evaluate the performance of the proposed methods, we compared TLSA and DTLSA with traditional GP (shortened as GP) and two transferred learning methods in GP, including FullTree [4] and k Throttle [5]. The number of valuable individuals in the final population of the source problem transferred to partially initialize the population in the target problem is set for FullTree and k Throttle as in [5], [9], $k\% = 50\%$ and referred to FullTree50 and Throttle50. For TLSA and DTLSA, $k\%$ is tested with $10\%, 30\%$ and 50% and shortened as TLSA10, TLSA30, TLSA50, DTLSA10, DTLSA30

and DTLSA50, respectively. In SAT, the max depth of $sTree$ is 2, and the maximum number of trials to find a pair of subtree and $sTree$ is set 100.

For datasets, due to the lack of benchmark datasets for transfer learning, we use the symbolic regression datasets that the previous researches utilized for testing GP transfer learning methods [5], [11], including two relational-knowledge transfer datasets, two feature transfer datasets, two model transfer datasets and four real-word datasets taken from UCI machine learning dataset [21]. In total, the tested system are evaluated in ten datasets. The detailed descriptions of these datasets are shown in Table 1. In this table, the data points are randomly taken with x in range of $[-1, 1]$ for the problems $F1, F2, F5, F6$, and x_i in range of $[2i - 1, 2i + 1]$ for the problems $F3, F4$.

Table 1. Problems for testing the proposed methods

Shorthanded	Name	Features	Number of samples	
			Training	Testing
F1	**Relational-knowledge Transfer**			
	Source domain: $x^4 + x^3 + x^2 + x$	1	100	-
	Target domain: $x^4 + x^3 + 2x^2 + x + 2$	1	100	100
F2	Source domain: $x^4 + x^3 + x^2 + x$	1	100	-
	Target domain: $x^6 + x^5 + 3x^4 + 3x^3 + 2x^2 + 2x$	1	100	100
F3	**Feature Transfer**			
	Source domain: $\sum_{i=1}^{3} (x_i - x_{i-1})^2$	4	100	-
	Target domain: $\sum_{i=2}^{4} (x_i - x_{i-1})^2$	4	100	100
F4	Source domain: $x_0 + x_1^2$	2	100	-
	Target domain: $x_0 + (x_1 + x_2)^2$	3	100	100
F5	**Model Transfer**			
	Source domain: $x^4 + x^3 + x^2 + x$	1	100	-
	Target domain: $x^4 + 2x^3 + 3x^2 + 4x$	1	100	100
F6	Source domain: $x^4 + x^3 + x^2 + x$	1	100	-
	Target domain: $-x^4 - x^3 - x^2 - x$	1	100	100
F7	**Abalone dataset**			
	Source domain: The male data	7	1528	-
	Target domain: The female data	7	655	652
F8	**Housing dataset**			
	Source domain: The data with $TAX \leq 600$	13	172	-
	Target domain: The data with $TAX > 600$	13	167	167
F9	**Wine quality dataset**			
	Source domain: The white wine dataset	11	4898	-
	Target domain: The red wine dataset	11	800	799
F10	**Appliances energy prediction**			
	Source domain: The data with time from 5/7/2016 to 5/13/2016	26	721	-
	Target domain: - Training: The data with time from 5/14/2016 to 5/20/2016 - Testing: The data with time from 5/21/2016 to 5/27/2016	26	721	686

The experimental GP parameters are shown in Table 2. These are the typical settings often used by GP researchers. The raw fitness is the root mean squared error on all fitness cases (samples). Therefore, smaller values are better. For each problem and each parameter setting, 30 runs were performed.

Table 2. Evolutionary parameter values

Parameter	Value
Population size	500
Generations	100
Selection	Tournament
Tournament size	3
Crossover, mutation probability	0.9; 0.1
Function set	$+, -, *, /, sin, cos$
Terminal set	$X_1, X_2, ..., X_n$
Initial Max depth	6
Max depth	17
Max depth of mutation tree	15
Raw fitness	root mean squared error on all fitness cases
Trials per treatment	30 independent runs for each value
Elitism	Copy the best individual to the next generation.

For statistical analysis, Kruskal-Wallis test with a confident level of 95% is used on the results in all result tables. If the result of Kruskal-Wallis test shows that at least one method is significantly different from the others, a post hoc analysis with Dunn's Test is conducted. p-values are adjusted with the Benjamini-Hochberg method. The ST column of the result tables refers to the significance of the difference between the test method compared to GP without transfer learning. The symbol "+" ("-") means that the corresponding method outperforms (is outperformed by) GP, whereas "=" refers to no significant difference. In addition, if the result is the best (the lowest), it is printed underline.

5. Results and discussion

The effectiveness of the tested systems is compared using their performance on the target domain, which is the focal point of transfer learning approaches. Four popular metrics in GP research, including training error, testing error, solution size and running time are analyzed in this section.

The first metric analyzed in this section is the learning performance on target data. The mean of the best fitness values across 30 runs is presented in Table 3. It can be seen from this table that the proposed methods, TLSA and DTLSA, outperform GP. TLSA is probably the best method among the tested methods in learning performance. The training error of TLSA10, TLSA30 and TLSA50 is respectively smaller than that of GP on 8, 7 and 9 problems out of 10 tested problems. DTLSA also achieves good learning performance. Compared to GP, DTLSA is better than GP on 8 problems with $k\% = 10\%$ and 30%, and on 7 problems with $k\% = 50\%$. For FullTree and $k\ Throttle$, the training error of FullTree50 and Throttle50 is also smaller than that of GP on 6 and 8 problems, respectively.

In terms of statistical comparison, the results of Kruskal-Wallis test again confirm the good learning performance of all tested transfer learning methods. The training error of

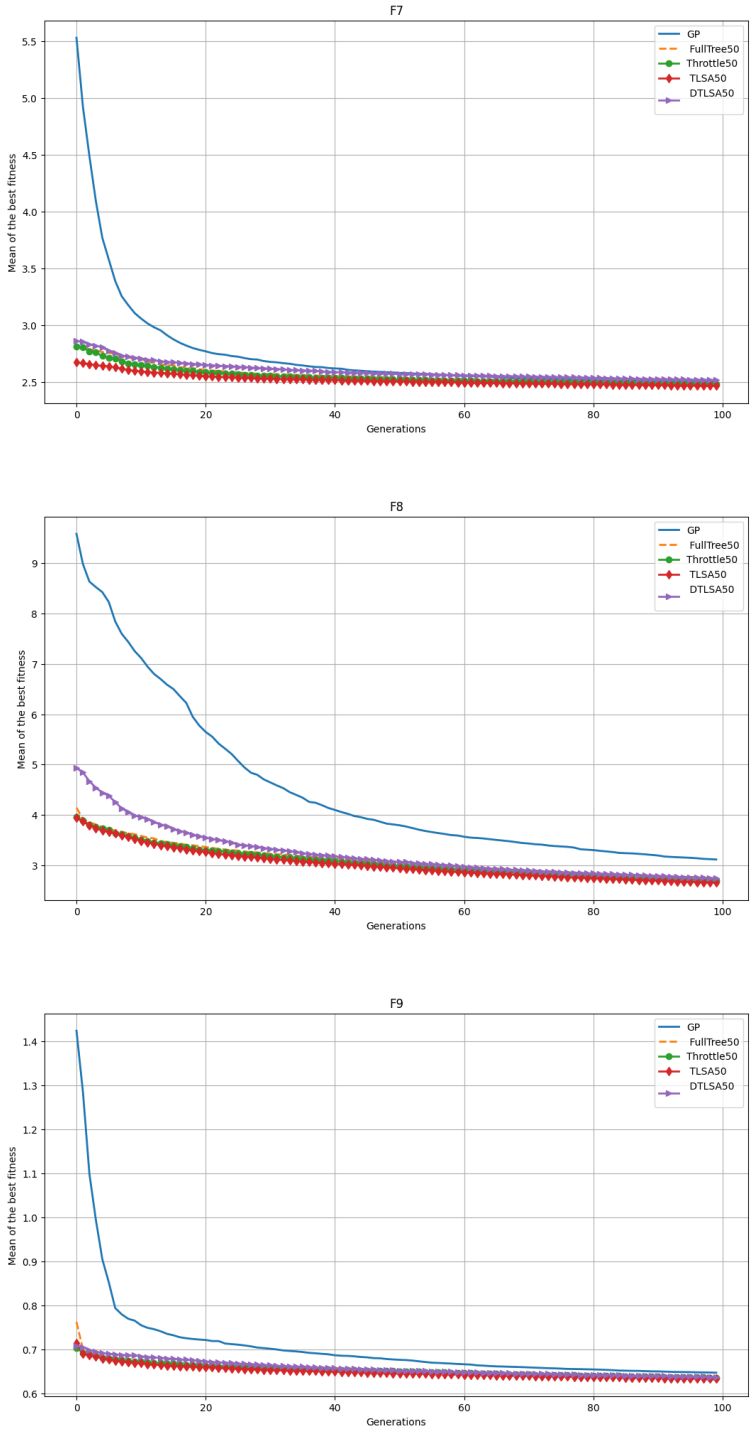


Fig. 5. Mean of the best fitness over generations.

Table 3. The mean of the best fitness on all training data

Pro	GP	FullTree50		Throttle50		TLSA10		TLSA30		TLSA50		DTLSA10		DTLSA30		DTLSA50	
	Mean	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**
F1	0.024	**0.024**	=	0.027	=	**0.019**	=	**0.024**	=	**0.024**	=	**0.014**	=	**0.022**	=	**0.015**	=
F2	0.046	0.048	=	**0.040**	=	**0.035**	=	0.048	=	**0.033**	=	**0.045**	=	**0.043**	=	**0.045**	=
F3	0.984	**0.846**	+	**0.842**	+	**0.878**	+	**0.821**	+	**0.849**	+	**0.981**	=	**0.936**	=	0.992	=
F4	0.548	0.557	=	**0.544**	=	**0.548**	=	**0.546**	=	**0.547**	=	**0.542**	=	**0.547**	=	**0.544**	=
F5	0.024	0.031	=	**0.022**	=	0.030	=	0.025	=	0.033	=	**0.021**	=	**0.021**	=	**0.019**	=
F6	0.012	0.013	=	0.012	=	0.013	=	0.013	=	**0.010**	=	0.017	=	0.014	=	0.014	=
F7	2.500	**2.486**	=	**2.484**	=	**2.481**	=	**2.476**	=	**2.468**	+	2.540	=	2.509	=	2.520	=
F8	3.116	**2.745**	+	**2.704**	+	**2.705**	+	**2.608**	+	**2.654**	+	**2.905**	=	**2.744**	+	**2.743**	+
F9	0.648	**0.635**	+	**0.637**	+	**0.640**	=	**0.635**	+	**0.634**	+	**0.644**	=	**0.637**	=	**0.638**	=
F10	54.34	**54.08**	=	**54.10**	=	**53.28**	=	**53.88**	=	**53.77**	=	**53.65**	=	**53.40**	=	**53.56**	=

FullTree50 and Throttle50 is significantly better than that of GP on 3 problems. TLSA10 and TLSA30 are significantly better than GP on 2 and 3 problems, respectively. TLSA50 probably achieves the best performance among the tested systems. The training error of TLSA50 is significantly better than that of GP on 4 problems and the smallest on 4 problems. For DTLSA, it is also significantly better than GP on one problem with $k\% = 30\%$ and 50%. Conversely, the training error of GP is not significantly better than that of the transfer learning methods any problems.

Figure 5 shows the mean training fitness per generation of each method for three real world problems, including F7, F8 and F9 for $k\% = 50\%$. The figure clearly showed that the transfer learning methods reliably reach a much lower training error than GP, especially at the first generations. This indicates that the source domain has useful knowledge that can be transferred to help improve the learning performance over the target domain. Considering the transfer learning methods, the training error TLSA50 is slightly smaller than that of others during the evolutionary process. That once again confirms the good improvement performance of TLSA on the learning process.

The second metric is the generalisation performance of the learnt models on the target test data. In each run, the best solution is selected and evaluated on the testing data (an unseen data set). The mean of these values across 30 runs is calculated, and the results are shown in Table 4.

The table showed that the testing error of TLSA and DTLSA is often smaller than that of GP. Especially, TLSA and DTLSA are better than GP on most tested problems with $k\% = 50\%$. TLSA50 and DTLSA50 are better than GP on 7 and 9 problems, respectively. The testing error of FullTree and $k\ Throttle$ is also smaller than that of GP corresponding on 3 and 5 problems. In terms of statistical tests, there is no difference between all learning transfer methods and GP.

The third metric to be analyzed is the size of solutions. To do this, we record the size of the selected solution (the number of nodes of this solution) in each run. These values are then averaged over 30 runs and presented in Table 5.

Table 4. The mean of testing error

Pro	GP	FullTree50		Throttle50		TLSA10		TLSA30		TLSA50		DTLSA10		DTLSA30		DTLSA50	
	Mean	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**
F1	0.024	0.025	=	0.0319	=	**0.020**	=	0.029	=	**0.023**	=	0.028	=	**0.022**	=	**0.016**	=
F2	0.047	0.055	=	**0.043**	=	**0.038**	=	0.048	=	**0.033**	=	0.050	=	**0.047**	=	**0.047**	=
F3	1.138	**1.042**	=	**0.995**	=	1.215	=	**1.041**	+	**0.994**	=	2.009	=	**1.022**	=	**1.094**	=
F4	0.590	1.013	=	0.618	=	0.591	=	0.604	=	0.590	=	0.598	=	0.673	=	**0.590**	=
F5	0.027	0.031	=	**0.022**	=	0.033	=	**0.026**	=	0.048	=	**0.021**	=	**0.021**	=	**0.020**	=
F6	0.014	**0.013**	=	**0.013**	=	**0.013**	=	**0.013**	=	**0.010**	=	0.017	=	0.014	=	0.014	=
F7	2.415	2.537	=	**2.410**	=	**2.402**	=	2.451	=	**2.406**	=	2.550	=	**2.397**	=	**2.414**	=
F8	20.32	39.60	=	26.81	=	22.59	=	**19.74**	=	**9.18**	=	60.82	=	**8.66**	=	**16.39**	=
F9	0.706	0.706	=	0.976	=	**0.705**	=	**0.700**	=	0.780	=	0.711	=	**0.694**	=	**0.701**	=
F10	142.7	**138.0**	=	159.7	=	149.3	=	**135.1**	=	**139.8**	=	**142.4**	=	**136.8**	=	**135.5**	=

Table 5. The average of solution size

Pro	GP	FullTree50		Throttle50		TLSA10		TLSA30		TLSA50		DTLSA10		DTLSA30		DTLSA50	
	Mean	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**
F1	151.8	**148.2**	=	179.2	=	**150.4**	=	184.7	=	169.5	=	164.0	=	166.0	=	152.3	=
F2	189.0	**185.2**	=	204.2	=	**168.5**	=	**167.0**	=	**181.6**	=	**161.0**	=	**146.0**	=	**150.9**	=
F3	171.8	219.6	-	221.4	-	189.1	=	210.3	-	202.7	=	**160.4**	=	**155.7**	=	**155.1**	=
F4	175.0	**167.8**	=	184.8	=	**169.4**	=	**170.2**	=	**168.9**	=	**154.3**	=	175.4	=	177.2	=
F5	149.3	161.2	=	166.6	=	165.1	=	178.9	=	181.7	-	**142.5**	=	163.9	=	**135.2**	=
F6	133.6	162.1	=	150.6	=	158.2	=	152.9	=	163.7	=	171.1	=	135.9	=	134.9	=
F7	173.3	264.6	-	267.5	-	268.1	-	245.4	-	259.4	-	**173.1**	=	177.8	=	**156.2**	=
F8	165.4	255.6	-	293.8	-	268.8	-	242.3	-	270.5	-	174.7	=	189.6	=	177.5	=
F9	124.6	189.4	-	171.8	-	163.4	-	169.3	-	169.8	-	132.5	=	144.9	=	145.0	=
F10	161.3	214.6	-	220.6	-	179.3	=	216.3	-	201.5	-	181.6	=	174.0	=	163.6	=

It can be observed on Table 5 that the solutions found by FullTree, $k\ Throttle$ and TLSA are slightly more complex than those of GP. The sizes of the solutions obtained by FullTree50, Throttle50, TLSA10, TLSA30 and TLSA50 are on average 124%, 129%, 118%, 122% and 124% respectively of the size of the solution obtained by GP. For DTLSA, apparently, the solutions evolved by it are as simple as those evolved by GP. The size of solution created by DTLSA achieves the smallest on 3 problems. Considering the transfer learning methods, the solutions obtained by TLSA and DTLSA are simple than those of FullTree and $k\ Throttle$.

Figure 6 presents the average of population size during the evolution of the tested systems on three real world problems F7, F8 and F9 with $k\% = 50\%$. It can be observed in this figure that the average population size of FullTree50, Throttle50 and TLSA50 is much higher than that for GP and DTLSA50 throughout the evolutionary process. This can be caused by copying individuals from the last generation in the source problem to initialize the population for the target problem in FullTree and $k\ Throttle$, and substituted only one subtree on the copied individuals from the final generation in the source problem to the first generation in the target problem in TLSA. Comparing among FullTree, $k\ Throttle$ and TLSA, TLSA is started (at the first generations) with lower than FullTree and $k\ Throttle$.

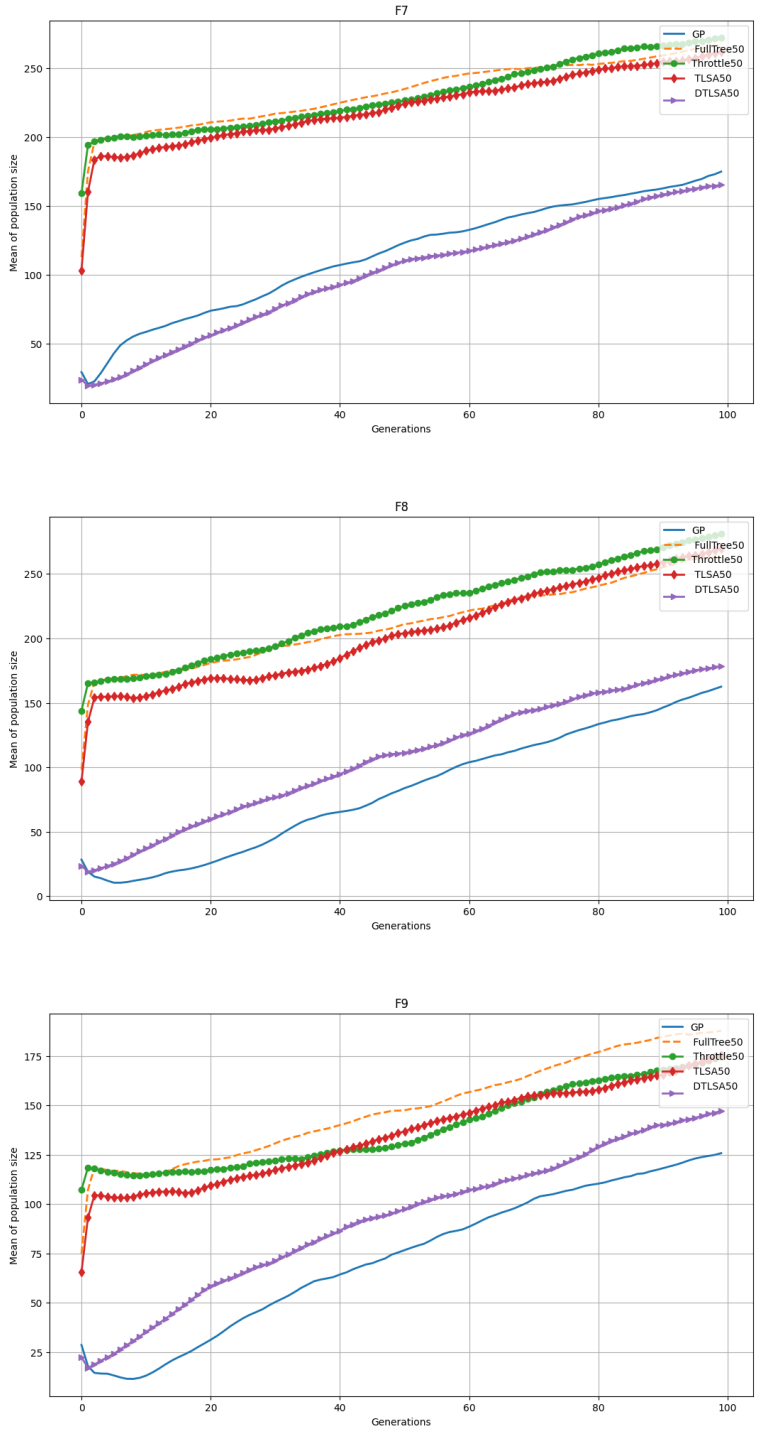


Fig. 6. Mean of the population size over generations.

For DTLSA, the average size of its population is no much difference to that of GP. It can be seen from Figure 6 that the mean of population size of DTLSA is slightly lower than that of GP in the problem F7, and is slightly higher than that of GP in the problems F8 and F9. It partially explains why the solutions found by DTLSA is simple as that of GP.

The last metric we examine is the computational cost of all tested GP systems. The total time needed to complete a GP run is recorded, and these values are then averaged over 30 runs. The results are shown in Table 6.

Table 6. The average of running time in seconds

Pro	GP	FullTree50		Throttle50		TLSA10		TLSA30		TLSA50		DTLSA10		DTLSA30		DTLSA50	
	Mean	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**	**Mean**	**ST**
F1	30.0	70.4	-	51.7	-	**6.4**	+	**12.2**	+	**8.9**	+	**7.7**	+	**9.7**	+	**11.9**	+
F2	32.4	50.9	-	58.1	-	**6.9**	+	**11.2**	+	**10.4**	+	**6.7**	+	**8.6**	+	**7.8**	+
F3	34.1	**32.6**	=	71.3	-	**6.9**	+	**14.8**	+	**7.3**	+	**4.3**	+	**10.2**	+	**4.4**	+
F4	26.7	51.3	-	30.5	=	**6.1**	+	**8.9**	+	**6.2**	+	**4.9**	+	**12.8**	+	**6.2**	+
F5	25.7	61.9	-	31.3	=	**6.6**	+	**11.5**	+	**11.2**	+	**5.6**	+	**7.4**	+	**7.1**	+
F6	27.4	67.2	-	32.7	=	**10.7**	+	**16.9**	+	**7.7**	+	**8.0**	+	**14.8**	+	**6.8**	+
F7	53.5	96.7	-	133.9	-	56.4	=	82.7	-	98.8	-	**22.8**	+	**32.6**	+	55.6	=
F8	20.3	51.6	-	48.4	-	**18.0**	=	**16.5**	=	23.9	=	**6.7**	+	**12.8**	+	**12.7**	+
F9	40.2	110.1	-	74.6	-	**39.9**	=	77.9	-	59.3	-	**22.1**	+	**36.8**	=	46.4	=
F10	30.7	63.0	-	63.4	-	30.8	=	36.1	=	43.2	-	**23.7**	=	30.8	=	**23.4**	=

This table highlights that TLSA and DTLSA run significantly faster than GP in the Relational-knowledge transfer problems, the Feature transfer problems and the Model transfer problems. In four real word problems, the running time of DTLSA10, DTLSA30 and DTLSA50 is significantly smaller than that of GP on 3, 2 and 1 problems, respectively. Conversely, the running time of GP is not significantly smaller than that of DTLSA on any problems. TLSA takes longer than GP in 2 problems with $k\% = 30\%$ and 3 problems with $k\% = 50\%$ in terms of statistical testing. Conversely, the computational cost of FullTree and $k\ Throttle$ is expensive than that of GP. The running time of FullTree50 and Throttle50 is significantly longer than that of GP on 9 and 7 problems out of 10 problems, respectively.

Overall, the results in this section show that TLSA and DTLSA improve the training error and the testing error compared to GP and two transfer learning methods in GP (FullTree and $k\ Throttle$). Moreover, the running time of TLSA and DTLSA is significantly faster than GP, FullTree and $k\ Throttle$. Additionally, the solutions evolved by DTLSA are simple as that of GP. These results partly confirm the correctness of two hypotheses in Section 1. The individuals in the final generation of the source problem apparently contain useful knowledge for the target problem. Furthermore, the substitution of subtrees by other approximate subprograms not only preserves probably this knowledge but also reduces the size of the initial population in the target problem.

6. Conclusions

In this work, two semantic approximation techniques based transfer learning methods have been proposed for GP. The first method is called TLSA. TLSA copies a number good individuals in the final generation of the source problem and substitutes a random subtree of them by using SAT [15]. These individuals are then utilized to initialize the population in the target problem. The second method is called DTLSA. DTLSA is slightly similar to TLSA. The difference between them is that in DTLSA, the process of replacing a random subtree in each cloned individual with a approximate subprogram grown by SAT is repeated until this individual reaches the size as that of randomly generating.

TLSA and DTLSA are evaluated on ten symbolic regression datasets, including two relational-knowledge transfer datasets, two feature transfer datasets, two model transfer datasets and four real-word datasets, and compared with GP without transfer learning and two GP transfer learning methods, FullTree and $k\ Throttle$. The analysis on the experimental results showed that the training error and testing error of TLSA and DTLSA are enhanced on the target problem. Furthermore, TLSA and DTLSA run much faster comparing with traditional GP and two compared transfer learning methods. DTLSA also obtained solutions that are simple as that of tradition GP and much more simple than that of FullTree and $k\ Throttle$.

Overall, the proposed transfer learning methods, TLSA and DTLSA could effectively extract more useful knowledge from the source domain to help to improve the performance of GP systems on the target domain.

References

[1] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[2] Q. Chen, B. Xue, and M. Zhang, "Instance based transfer learning for genetic programming for symbolic regression," in *2019 IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 3006–3013.

[3] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[4] T. T. H. Dinh, T. H. Chu, and Q. U. Nguyen, "Transfer learning in genetic programming," in *2015 IEEE Congress on Evolutionary Computation*. IEEE, 2015, pp. 1145–1151.

[5] E. Haslam, B. Xue, and M. Zhang, "Further investigation on genetic programming with transfer learning for symbolic regression," in *2016 IEEE Congress on Evolutionary Computation*. IEEE, 2016, pp. 3598–3605.

[6] D. O'Neill, H. Al-Sahaf, B. Xue, and M. Zhang, "Common subtrees in related problems: A novel transfer learning approach for genetic programming," in *2017 IEEE Congress on Evolutionary Computation*. IEEE, 2017, pp. 1287–1294.

[7] M. A. Ardeh, Y. Mei, and M. Zhang, "Transfer learning in genetic programming hyper-heuristic for solving uncertain capacitated arc routing problem," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 49–56.

[8] M. A. Ardeh, Y. Mei, and M. Zhang, "A parametric framework for genetic programming with transfer learning for uncertain capacitated arc routing problem," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2020, pp. 150–162.

[9] M. A. Ardeh, Y. Mei, and M. Zhang, "Genetic programming hyper-heuristics with probabilistic prototype tree knowledge transfer for uncertain capacitated arc routing problems," in *2020 IEEE Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.

[10] Q. Chen, B. Xue, and M. Zhang, "Differential evolution for instance based transfer learning in genetic programming for symbolic regression," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 161–162.
[11] Q. Chen, B. Xue, and M. Zhang, "Genetic programming for instance transfer learning in symbolic regression," *IEEE Transactions on Cybernetics*, 2020.
[12] B. Al-Helali, Q. Chen, B. Xue, and M. Zhang, "Multi-tree genetic programming-based transformation for transfer learning in symbolic regression with highly incomplete data," in *2020 IEEE Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.
[13] B. Al Helali, Q. Chen, B. Xue, and M. Zhang, "Multi-tree genetic programming with new operators for transfer learning in symbolic regression with incomplete data," *IEEE Transactions on Evolutionary Computation*, 2021.
[14] T. H. Chu, Q. U. Nguyen, and V. L. Cao, "Semantics based substituting technique for reducing code bloat in genetic programming," in *Proceedings of the Ninth International Symposium on Information and Communication Technology*. ACM, 2018, pp. 77–83.
[15] Q. U. Nguyen and T. H. Chu, "Semantic approximation for reducing code bloat in genetic programming," *Swarm and Evolutionary Computation*, vol. 58, p. 100729, 2020.
[16] W. Fu, B. Xue, X. Gao, and M. Zhang, "Output-based transfer learning in genetic programming for document classification," *Knowledge-Based Systems*, vol. 212, p. 106597, 2021.
[17] A. Wild and B. Porter, "General program synthesis using guided corpus generation and automatic refactoring," in *International Symposium on Search Based Software Engineering*. Springer, 2019, pp. 89–104.
[18] A. Wild and B. Porter, "Neurally guided transfer learning for genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 267–268.
[19] V. Leonardo, C. Mauro, and S. Sara, "A survey of semantic methods in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 15, no. 2, pp. 195–214, 2014.
[20] N. F. McPhee, B. Ohs, and T. Hutchison, "Semantic building blocks in genetic programming," in *European Conference on Genetic Programming*. Springer, 2008, pp. 134–145.
[21] K. Bache and M. Lichman, "UCI machine learning repository," 2013, http://archive.ics.uci.edu/ml.

Manuscript received 27-11-2021; Accepted 14-5-2022.

■

Thi Huong Chu received her B. Eng. degree in Applied Mathematics and Informatics from Hanoi University of Science and Technology and MSc Degree in Computer Science from Le Quy Don Technical University. She received the PhD degree in Mathematical Foundations for Informatics from Le Quy Don Technical University, in 2020. Since 2002, she has been teaching at Faculty of Information Technology, Le Quy Don Technical University. Her research interests are in the domain of Evolutionary Algorithms, Genetic Programming and Machine Learning. E-mail: huongktqs@gmail.com.

HỌC CHUYỂN GIAO DỰA TRÊN KỸ THUẬT XẤP XỈ NGỮ NGHĨA TRONG LẬP TRÌNH DI TRUYỀN

Chu Thị Hường

Tóm tắt

Học chuyển giao nhằm mục đích sử dụng lại các tri thức từ các vấn đề nguồn khác nhau nhưng có liên quan để nâng cao hiệu suất học trong vấn đề đích. Trong lập trình di truyền (GP), phương pháp học chuyển giao giúp các hệ thống GP mở rộng khả năng ứng dụng trong các bài toán thực tiễn. Với nghiên cứu này, hai phương pháp học chuyển giao dựa trên kỹ thuật xấp xỉ ngữ nghĩa cho lập trình di truyền được đề xuất. Phương pháp đề xuất sử dụng các cá thể tốt nhất của thế hệ cuối cùng trong bài toán nguồn làm nguyên liệu chuyển giao sang bài toán đích. Trước khi chuyển giao, kỹ thuật xấp xỉ ngữ nghĩa được áp dụng trên các cá thể này nhằm thỏa mãn đồng thời hai mục tiêu: giảm kích thước của cá thể chuyển giao và bảo toàn tri thức của các cá thể này. Phương thức đề xuất được thử nghiệm trên mười bài toán hồi quy và được so sánh với GP truyền thống và hai phương pháp học chuyển giao trong GP. Kết quả thực nghiệm cho thấy các phương pháp được đề xuất có thể trích xuất hiệu quả tri thức hữu ích từ các vấn đề nguồn để giúp cải thiện hiệu suất của hệ thống GP trên vấn đề đích.