A New Classifier Using Layered Genetic Programming

Jung-Yi Lin  
Computer Science,  
National Chiao Tung University  
jylin@ieee.org

Hao-Ren Ke  
Library and Institute of Information Management,  
National Chiao Tung University  
claven@lib.nctu.edu.tw

Been-Chian Chien  
Computer Science and Information Engineering, National University of Tainan  
bcchien@mail.nutn.edu.tw,

Wei-Pang Yang  
Information Management, National Dong Hwa University  
wpyang@cis.nctu.edu.tw

Abstract

This paper proposes a novel method called layered genetic programming (LAGEP), which is a new architecture for multipopulation GP. LAGEP consists of a number of layers. Each layer is composed of a number of populations and can be viewed as an implementation of isolated multipopulation GP. The solution generated by LAGEP is a classification function composed by functions generated from layers. Many experiments based on medical classification problems with different parameters are conducted to illustrate the behavior of LAGEP. Experimental results prove that accuracy of LAGEP is higher than that of isolated MGP and single population GP.

Keywords: Pattern classification, multipopulation genetic programming

1 INTRODUCTION

Genetic programming (GP) [1], an important evolutionary computation (EC) technique, has developed rapidly in recent years. Researchers have proposed creative ideas to improve the effectiveness and efficiency of GP, such as new fitness functions, new architectures, and new individual expressions. Traditionally, GP works with a single population. Multipopulation GP [3][14], which employs several populations to discover optimal solutions, has been proposed and developed. Many different topologies of MGP have been proposed, such as the circle topology and the random topology. Fernández et al. [14] performed several experiments with parallel and distributed GP (PADGP), isolated multipopulation GP (“isolated” means that there is no migration between populations), and traditional single population GP. Their experiments show that PADGP and isolated multipopulation GP usually obtain better performance than traditional single population GP.

Many classifiers have been developed based on GP in recent years [2]-[13]. To generate classification rules, Freitas [6] proposed the Tuple-Set-Descriptor (TSD), a logical formula to represent an individual. Kotani and Sherrah [9] [13] used GP to perform feature selection before using other classification methods. Multi-category classification problems are more difficult than two-class classification problems. Kishore et al. [7] and the present authors [4] have considered such a problem as multiple two-class classification problems and then generated corresponding expressions or discriminant functions. These methods need $k$ runs for a $k$-class classification problem. Muni et al. [12] proposed a novel method to solve $k$-class classification problems in a single run. Each individual in their work is represented by a multitree. Evolving one individual is equivalent to evolving $k$ trees simultaneously. Loveard and Ciesielski [11] proposed five methods for solving multi-category classification problems including binary decomposition, static range selection, dynamic range selection, class enumeration, and evidence enumeration. Brameier and Banzhaf [3] used linear GP and MGP techniques. Individuals are represented as strings and can be transmitted between demes, i.e. subpopulations, according to their fitness value. Using functional expressions to represent individuals is effective in GP [4][7][10], and the binary tree is a common data structure for functional expressions. However, two problems occur when GP with functional expressions is used. First, it is difficult to choose appropriate operations for a given problem when its characteristics are completely unknown. If the operator set contains many operations, there is a greater possibility of discovering optimal solutions, but the searching space becomes larger and therefore may become impracticable. As shown in [7], GP with an operation set comprising only basic arithmetic operations, i.e. $\{+,-,\times,\div\}$, generates results
comparable to that with an operation set comprising additional operations like log, sine, and cosine. Second, it is difficult to know the proper length of an individual because there is no prior knowledge about optimal solutions. The predefined individual size, as the length of a string-expression individual or the depth of a tree-expression individual, is usually chosen according to heuristic or empirical assumptions. The following is an example of a classification problem containing 64 dimensional data, i.e., a datum \( x \) can be represented by 
\[ x = (a_1, a_2, \ldots, a_{64}). \]
Suppose that an optimal solution \( F \) is known and is quite easy: 
\[ F = \prod_{i=1}^{32} a_i. \]
\( F \) can be represented as a skew binary tree with a height of 64 or a balanced binary tree with a height of seven. An individual can contain at most \( 2^{32} - 1 \) nodes if the predefined maximum depth is 64. A population containing so many large trees is highly complex and is thereby impracticable. On the other hand, if the predefined maximum depth is fixed at seven, it is very difficult to generate the ideal balanced tree. Moreover, if the height of every individual is less than seven, the function \( F \) will never be obtained.

Using an acceptable and practicable individual size is a simple but dangerous way to avoid this problem. This problem motivates us to develop this work. Since a long function can be viewed as a composition of a number of small functions, it is possible to combine a number of small GP solutions into a large one. Therefore, it is desirable to generate those small solutions with a practicable size of individuals and then use them to compose a larger solution. For example, consider the above function \( F \) and two functions \( B = \prod_{i=1}^{16} a_i \) and \( C = \prod_{i=33}^{64} a_i \). Clearly, \( F \) can be represented as \( (B \times C) \).

Functional programs built up from \( B \) and \( C \) can be generated by two separate GPs and then are combined together to form \( F \). Here we attempt to develop a method by which we can determine a proper node to combine small functions.

The method proposed in this paper is called Layered Genetic Programming (LAGEP). It is a method based on MGP. LAGEP arranges populations in a layered architecture. All populations are independent of each other. Every layer can be viewed as an ISOLATED MGP system. Layers can have different numbers of populations. Populations in the same layer evolve with identical training sets and store the results of their best individuals into a dataset; this dataset becomes the training set of the following layer. After all layers have finished the evolution process, the output of the final layer is used as the result of LAGEP.

The rest of this paper is organized as follows. Section 2 describes the details of LAGEP. Section 3 presents and discusses the experimental results on selected classification problems. Conclusions are drawn in Section 4.

2 PROPOSED LAGEP METHOD

With this method we aim to generate short functions and combine these functions. We use a number of populations to generate short functions. The maximum number of nodes of individuals in these populations is restricted to a small value. In LAGEP, we arrange these populations into a layer and perform evolutionary processes simultaneously. The generated functions evaluate the training set and store their evaluation into a new dataset. This new dataset, which includes function values, will be used as the training set for successive layers. LAGEP works with a number of layers. Each layer performs as an isolated multipopulation GP system with a particular training set. The last layer contains only one population. The output of this last population is taken as the result of LAGEP.

This section describes the detailed definition of LAGEP. First, we introduce a single population, including definitions of the individual, population, and a method called adaptive mutation rate tuning (AMRT) used to increase mutation rate with generations. Then we bring in the layer concept and the multi-layer architecture.

2.1 Single population

We obtain a function from GP by evolving a population with a given training set. Let \( T \) be the training set and \( x_i \) be a training datum with \( m \) significant attributes, \( a_{1i}, a_{2i}, \ldots, a_{mi} \), and we have 
\[ T = \{ x_i | x_i = (a_{1i}, a_{2i}, \ldots, a_{mi}), m > 0, a_{ji} \in \mathbb{R}, 1 \leq i \leq n \}. \]
Individuals are potential solutions that are represented by predefined domain-dependent expressions. An individual is defined as a functional expression and its data structure is a binary tree. An individual \( I \) is formulated by three kinds of elements, \( \text{variables}, \text{constants}, \text{and operations} \), which belong to the variable set \( S_v \), the constant set \( S_c \), and the operation set \( S_o \), respectively. Variables are symbolic notations related to attributes of training data. \( S_v \) is a set of predefined constants. In order to reduce computation complexity, we use a small range of natural numbers. \( S_o \) contains only simple arithmetic operations \([7]\). \( S_v, S_c, S_o \) are defined by 
\[ S_v = \{ A_1, A_2, \ldots, A_m \}, S_c = \{ +, -, \times, / \}, S_o = \{ 1, 2, \ldots, 100 \}, \]
where \( / \) in \( S_o \) is a protected division. When the denominator equals zero, the division will be set to 1. The fitness function is a function used to evaluate the fitness of every individual. The number of maximum nodes in an individual is predefined and denoted as \( \text{LEN} \).
A population $P = \{I_1, I_2, ..., I_{|P|}\}$ is a set of individuals. A population evolves for a number of generations. It generates a best individual $\Lambda$ as its result. The term evolve mimics the natural selection mechanism by performing a systematic process on the population by genetic operators: crossover, mutation, and reproduction, that are performed based on predefined probabilities $W_r$, $W_m$, and $W_c$, respectively. In this work, we use the standard tree-based GP crossover and mutation methods [1]. In the evolutionary process, after fitness values are evaluated, two best individuals are reproduced directly to the next generation to keep surviving. The remaining individuals continue performing crossover and mutation. When individual(s) are selected to perform crossover, or mutation, we insert their offspring, or its mutant, into a new population if and only if the offspring, or the mutant, has better fitness values than the individuals.

Only the mutation operator is capable of generating individuals with new structures. However, in order to avoid random walk, $W_m$ is usually much lower than $W_c$. When $W_m$ is fixed at a small value, individuals may not have sufficient opportunity to mutate. As a result, the diversity of the population is limited. In particular, if some terminals are already good enough to classify samples, individuals may be stuck with such terminals. We raise $W_m$ to perform the mutation operator more frequently. We propose herein a method that changes $W_m$ when the generation increases. First, we define variables $\nu$ as the number of executed generations, and $G$ is the number of maximum generations. Then we can define $W_m$ at generation $\nu$ by

$$W_m(\nu) = W_m(0) \times \left( \frac{W_m}{W_m(0)} \right)^\nu,$$

where $f_{\text{MAX}}$ is the fitness value of the best individual in generation $\nu$, $f_{\text{AVERAGE}}$ is the average fitness value of all individuals in generation $\nu$. When the fitness value of the best individual is good enough, larger than the double of $f_{\text{AVERAGE}}$, $W_m$ is not changed. Otherwise, we assign a new value to $W_m$. At the last generation, $W_m$ could be equal to $W_c$. In other words, $W_m$ has a near 50% probability of being selected in later generations. Fig. 1 shows the curve of $W_m$ with $G=100$, $W_m(0)=0.05$, $W_c=0.5$, and $f_{\text{MAX}}$ is never larger than $2 \times f_{\text{AVERAGE}}$ during these 100 generations.

When we perform the training task of a classification problem, we need to know which class is the target class. The target class is the class label for which we train the system to find solutions. Training samples are divided into positive instances if they belong to the target class and negative instances if they do not. For a given training sample $x$, we say an individual $I_j$ recognizes or repels:

$I_j$ recognizes sample $x$ iff $I_j(x) \geq 0$,

$I_j$ repels sample $x$ iff $I_j(x) < 0$.

The classification results of an individual can be represented by four values as shown in TABLE 1 [15]. Note that $(TP + FP + FN + TN)$ is equal to the total amount of training data.

![Fig. 1. The curve of $W_m$ with $G=100$](image)

### TABLE 1

<table>
<thead>
<tr>
<th>$I_j$ is positive</th>
<th>Positive instance</th>
<th>Negative instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TP$</td>
<td></td>
<td>$FP$</td>
</tr>
<tr>
<td>$FN$</td>
<td></td>
<td>$TN$</td>
</tr>
</tbody>
</table>

The fitness function $F$ used in this work is made of sensitivity and specificity:

$$\text{Sensitivity} = \frac{TP}{(TP + FN)}, \text{Specificity} = \frac{TN}{(FP + TN)}.$$  

$$F(I_j) = (\text{Sensitivity} \times \text{Specificity})^2.$$  

We use such a fitness function because $F$ will be computed many times and therefore should be simple and efficient. The algorithm of an evolving single population is shown below.

### Algorithm 1: Evolving Single Population

1. Initialize $|P|$ randomly generated individuals; define $GEN \leftarrow 0$, $n_o \leftarrow 0$;
2. Evaluate fitness values for individuals;
3. Perform reproduction on best two individuals and insert them into a new population $P'$;
4. Tune the mutation rate by AMRT.
5. Select crossover or mutation based on $W_c$ and $W_m$.
   5.1 When crossover is selected, if $|P'| = |P| - 1$, jump to 5.2. Otherwise, perform the crossover operator on two individuals to generate offspring. $n_o \leftarrow 2$;
   5.2 If mutation is selected, then perform the mutation operator on a selected individual to generate the mutant. $n_o \leftarrow 1$;
   5.3 Insert $n_o$ best individuals from parents and offspring/mutant to $P'$;
6. Continue step 5 if $|P'| < |P|$. Otherwise, one generation is completed.
7) Replace $P$ by $P'$, and $GEN \leftrightarrow GEN+1$. Repeat steps 2) - 6) if $GEN < G$.
8) Output $\Lambda$, the optimal individual of $P$, as the result.

2.2. Layers and the Relationship between Layers

A layer is a set of populations. LAGEP uses multi-layer architecture, and the $i$th layer is defined by

$$L_i = \left\{ P_1, P_2, ..., P_{i(j)}, S_i' \right\},$$

where $l(i)$ is the number of populations in $L_i$ and $l(0)=m$.

We denote the original given training set by $P_0$. To simplify operations of populations in an identical layer, the populations work as an isolated multipopulation system [14]: no communications or individual exchanging mechanisms occur between these populations. For layer $L_i$, a particular training set $T_{i-1}$ is prepared and is used for the evolution processes of $L_i$’s populations. After $L_i$’s populations have performed the evolution processes and generated a vector of the best individuals ($\Lambda_1$, $\Lambda_2$, ..., $\Lambda_{|\Lambda|}$), a new training set $T_i$, and a new variable set $S_i'$ can be constructed from such a layer by

$$T_i = \left\{ x_j | x_j = \left( a_{i1}, a_{i2}, ..., a_{ik}, ..., a_{i(m)} \right), 1 \leq j \leq n_i \right\},$$

$$a_{ik} = \Lambda_{ik} \left( x_{i(i-1)}, \right), 1 \leq k \leq l(i), x_{i(i-1)} \in T_{i-1},$$

$$S_i' = \left\{ \Lambda_{i1}, \Lambda_{i2}, ..., \Lambda_{i(m)} \right\}.$$

When $T_i$ has been constructed, we say an era ends, and then layer $L_{i+1}$ is able to begin its evolution process with $T_i$.

According to the definitions provided so far, we define LAGEP as

$$\text{LAGEP} = \left\{ \left( L_i, T_i, T_{i+1} \right) \big| i = 1, 2, ..., \Gamma \right\},$$

where $\Gamma$ is the number of layers. After all the layers have finished their evolution processes, data stored in $T_{\Gamma}$ and generated by $L_{\Gamma}$ are the training results of LAGEP. Each datum in $T_{\Gamma}$ has a single dimension representing the training result of the current target class.

For a $K$-class classification problem LAGEP has to be trained $K$ times for $K$ different target classes. The $K$ different $T_i$ stands for training results of $K$ classes. The algorithm of LAGEP evolution is shown in Algorithm 2.

**Algorithm 2: LAGEP Evolution**

1) Let $T_0 \leftarrow 0, c \leftarrow 1.$
2) Perform Algorithm 1 for all populations in layer $L_0$, with training set $T_{0,COUNT-1}$.
3) Evaluate $\Lambda_1, \Lambda_2, ..., \Lambda_{|\Lambda|}$ with all data in $T_{c-1}$ and store them into $T_c$. An era is thus completed.
4) If $c < T$, then $c \leftarrow c + 1$. Jump to step 2).
5) Change target class to the next class label and jump to step 1).

LAGEP is capable of attribute selection. Single population GP can conduct attribute selection. Attributes not used in the final solution can be viewed as useless attributes. The first layer of LAGEP is composed by a number of single population GPs so that LAGEP still has the capability of attribute selection. A layer has a great probability of performing better than its previous layer. Consider $P_{i+1} \in L_{i+1}$ and $P_i \in L_i$. For any data $x$ in $T_i$, values of its terminal $\Lambda_{ij}$ are evaluated by $\Lambda_{ij}$. Assume that the prediction result of $\Lambda_{ij}$ is shown in TABLE 2. If an individual $i \in P_{i+1}$ and $i \neq \Lambda_{ij}$, we have

$$F(I) = \left( \frac{R_i}{R_i + R_t} \right)^{2} = F(\Lambda_{ij}).$$

Since an individual with a single terminal will very likely happen, the max fitness value of $(\Lambda_{i1}), (\Lambda_{i2}), ..., (\Lambda_{i|\Lambda|})$ will most likely be better than the maximum fitness value of $(\Lambda_1, \Lambda_2, ..., \Lambda_{|\Lambda|}).$

**TABLE 2**

<table>
<thead>
<tr>
<th>Prediction results of $\Lambda_{ij}$</th>
<th>Belong to target class</th>
<th>Not belong to target class</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Lambda_{ij} \geq 0$</td>
<td>$R_i$</td>
<td>$R_t$</td>
</tr>
<tr>
<td>$\Lambda_{ij} &lt; 0$</td>
<td>$R_i$</td>
<td>$R_t$</td>
</tr>
</tbody>
</table>

2.3. The Testing Phase and Z-value Measure

The set of test data is denoted as $V$. To predict the class of a test datum $y_i \in V$, we substitute $y_i$ to $K$ LAGEPs responding to $K$ different classes to determine the class of $y_i$. A problem called conflict occurs when $y_i$ is classified into two or more classes. This problem can be avoided by executing functions in the proper sequence, or the problem can be resolved by using additional techniques. In this paper, we use the method called Z-value measure published at [4][10].

3 EXPERIMENTS

In this section we describe the experiments and analyze classification results. To conduct the experiments described in this section, we developed a system executed under an ACER VT7600GL, which is equipped with 3.0GHz CPU and 1.5GB RAM.

3.1. Medical Datasets

In order to illustrate LAGEP we used six diagnostic problems selected from the PROBEN1 benchmark set of real-world problems [16], which was also used in [3]. Theses selected problems are cancer (CAN), diabetes (DBT), heart (HRT), horse (HRS),
gene (GEN), and thyroid (TRD). These problems have been preprocessed with completing missing values, normalizing to [0, 1], and 1-of-m coding methods by [16]. PROBEN1 divides every problem dataset into training set, validation set, and test set. The training set contains the first half of the samples. The validation set includes the next 25% of the samples. The last 25% of the samples are test data. In this work a validation set was not used. We summarize these problems in TABLE 3. Furthermore, PROBEN1 is prepared three different compositions with different orders of samples for each data set. This should increase confidence that classification results are not influenced by the sample distribution of the training set, validation set, and test set. In summary, we performed experiments on 18 problems.

3.2. Experiment Settings

We used three primary experiment settings in this paper. These settings are shown in TABLE 4. In TABLE 4, the variables $N$, $LEN$, and $G$ stand for the population size, individual length, and maximum number of generations for all populations, respectively. Nine subsettings are made with respect to the three primary settings. Every medical dataset is performed with each experiment setting 10 times. ES1.1, ES1.2, and ES1.3 are not of the LAGEP method. They are used to show the performance of a single large population GP and isolated MGP. ES2.1, ES2.2, and ES2.3 used two layers. ES3.1, ES3.2, and ES3.3 used three layers. We used them to show LAGEP behavior with respect to different conditions. For instance, ES2.2 used two layers. The first layer has three populations and the second layer has a single population. Each of the three populations has 1000 individuals and evolves at most 250 generations. Every individual has at most $2^{255} - 1$=255 nodes. The mutation weight and the crossover weight are 0.05 and 0.95, respectively.

<table>
<thead>
<tr>
<th>Settings</th>
<th>$\Gamma $</th>
<th>$l(i)$</th>
<th>$G$</th>
<th>$N$</th>
<th>$LEN$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES1.1</td>
<td>1</td>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>511</td>
</tr>
<tr>
<td>ES1.2</td>
<td>1</td>
<td>3</td>
<td>250</td>
<td>1500</td>
<td>511</td>
</tr>
<tr>
<td>ES1.3</td>
<td>1</td>
<td>$\lceil \lg(m) \rceil$</td>
<td>250</td>
<td>1000</td>
<td>511</td>
</tr>
<tr>
<td>ES2.1</td>
<td>2</td>
<td>2, 1</td>
<td>250</td>
<td>1000</td>
<td>255</td>
</tr>
<tr>
<td>ES2.2</td>
<td>2</td>
<td>3, 1</td>
<td>250</td>
<td>1000</td>
<td>255</td>
</tr>
<tr>
<td>ES2.3</td>
<td>2</td>
<td>$\lceil \lg(m) \rceil$, 1</td>
<td>250</td>
<td>800</td>
<td>255</td>
</tr>
<tr>
<td>ES3.1</td>
<td>3</td>
<td>2, 3, 1</td>
<td>250</td>
<td>800</td>
<td>127</td>
</tr>
<tr>
<td>ES3.2</td>
<td>3</td>
<td>3, 2, 1</td>
<td>250</td>
<td>800</td>
<td>127</td>
</tr>
<tr>
<td>ES3.3</td>
<td>3</td>
<td>3, 3, 1</td>
<td>250</td>
<td>800</td>
<td>127</td>
</tr>
</tbody>
</table>

3.3. Experimental Results and Discussion

In this section, we show accuracy comparisons in TABLE 5 and TABLE 6. These tables show the accuracy of ES1.1, linear GP [3], and neural network (NN) [3][16]. The values in each cell stand for the lowest error rate, the average error rate, the standard variation, and the average elapsed time of training all classes in seconds. We discuss performance by classification problems.

### TABLE 5

<table>
<thead>
<tr>
<th>Problem</th>
<th>Classes</th>
<th>features</th>
<th>Training samples</th>
<th>Test data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>2</td>
<td>9</td>
<td>350</td>
<td>174</td>
</tr>
<tr>
<td>DBT</td>
<td>2</td>
<td>8</td>
<td>384</td>
<td>192</td>
</tr>
<tr>
<td>HRT</td>
<td>2</td>
<td>35</td>
<td>152</td>
<td>75</td>
</tr>
<tr>
<td>GEN</td>
<td>3</td>
<td>120</td>
<td>1588</td>
<td>793</td>
</tr>
<tr>
<td>HRS</td>
<td>3</td>
<td>58</td>
<td>182</td>
<td>91</td>
</tr>
<tr>
<td>TRD</td>
<td>3</td>
<td>21</td>
<td>3600</td>
<td>1800</td>
</tr>
</tbody>
</table>

### CAN

NN achieves the lowest error rate in all CAN problems. In TABLE 5, ES1.2 performs well in CAN3. ES1.3 outperforms ES1.1 in CAN1 and CAN2. ES1.3 also outperforms linear GP in all CAN problems. We found that all ES2 outperform ES1.1 in CAN1 and CAN2. ES2.2 performs relatively poor in CAN3. Both ES2.1 and ES2.2 are better than linear GP in all CAN problems. In TABLE 5, we found that ES3 uses more populations but they do not achieve better accuracy. ES1.3 performs significantly better in CAN3. In CAN problems, using more populations does not mean better classification quality.

### DBT

NN obtains the lowest error rate in all DBT problems. Linear GP performs better than all experiment settings. In TABLE 5, ES1.2 and ES1.3 are not significantly better than ES1.1. ES1.3 performs better than ES1.2 in DBT1 only. However, ES1.3 takes less time than ES1.2 and almost half the time of ES1.1. ES2 executes better than ES1.1 in DBT2 and DBT3. ES2.1 uses the least time and achieves the lowest error rate against all experiment settings in DBT1 and DBT3. ES3.2 is poor in DBT1. ES1.1 is worse in DBT2 and DBT3. In DBT problems, using more populations or using more layers does not significantly improve performance.

### HRT

TABLE 5 shows that ES1.2 and ES1.3 obtain poor results in HRT1 and HRT3 problems. However, ES1.3 achieves significantly greater accuracy in HRT2. ES2.3 performs better than ES1.1, linear GP, and NN. ES2.1 and ES2.2 are worse in HRT1 problem. All ES3 settings are good in HRT2 problem but poor in HRT1 and HRT3 problems. In HRT problems, different data distribution has a great impact on accuracy. In HRT problems, using more populations helps obtain a lower error rate.

### HRS

...
From TABLE 6, we found that error rates obtained by ES1.1 are significantly better in HRS1 and HRS2. ES1.3 performs better than ES1.2 in all HRS problems. ES1.2 obtained better than ES1.1, ES2.2, and ES2.3. NN and linear GP are worse than ES2.1 in all HRS problems. The error rates of all ES3 settings are similar to ES1.1 in HRS2. ES3.1 performs relatively poorly in HRS problems. In HRS problems, ES2.1 obtained excellent results, indicating that using more populations or more layers may not cause any improvement in accuracy.

### TABLE 5

**Accuracy comparison with experiment settings, linear GP (Li-GP), and NN on 2-class medical problems**

<table>
<thead>
<tr>
<th></th>
<th>ES1.1</th>
<th>ES1.2</th>
<th>ES1.3</th>
<th>ES2.1</th>
<th>ES2.2</th>
<th>ES2.3</th>
<th>ES3.1</th>
<th>ES3.2</th>
<th>ES3.3</th>
<th>Li-GP</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN1</td>
<td>1.15</td>
<td>.58</td>
<td>.58</td>
<td>1.15</td>
<td>.58</td>
<td>.58</td>
<td>1.15</td>
<td>.58</td>
<td>1.15</td>
<td>.57</td>
<td>1.38</td>
</tr>
<tr>
<td>CAN2</td>
<td>3.45</td>
<td>5.17</td>
<td>4.02</td>
<td>4.02</td>
<td>4.60</td>
<td>4.02</td>
<td>4.60</td>
<td>5.17</td>
<td>4.02</td>
<td>4.02</td>
<td>4.77</td>
</tr>
<tr>
<td>CAN3</td>
<td>2.30</td>
<td>2.30</td>
<td>2.87</td>
<td>2.87</td>
<td>1.72</td>
<td>1.72</td>
<td>2.87</td>
<td>2.30</td>
<td>2.87</td>
<td>3.45</td>
<td>3.70</td>
</tr>
<tr>
<td>DBT1</td>
<td>23.96</td>
<td>24.34</td>
<td>24.48</td>
<td>23.44</td>
<td>22.40</td>
<td>22.92</td>
<td>23.96</td>
<td>25.00</td>
<td>22.92</td>
<td>21.35</td>
<td></td>
</tr>
<tr>
<td>DBT2</td>
<td>27.60</td>
<td>25.52</td>
<td>26.04</td>
<td>25.00</td>
<td>23.96</td>
<td>26.56</td>
<td>26.04</td>
<td>25.00</td>
<td>21.88</td>
<td>25.00</td>
<td></td>
</tr>
<tr>
<td>DBT3</td>
<td>29.95</td>
<td>27.92</td>
<td>29.48</td>
<td>27.81</td>
<td>27.66</td>
<td>28.75</td>
<td>28.49</td>
<td>27.24</td>
<td>28.02</td>
<td>27.85</td>
<td>26.42</td>
</tr>
<tr>
<td>HRT1</td>
<td>20.53</td>
<td>21.73</td>
<td>22.27</td>
<td>22.00</td>
<td>22.00</td>
<td>20.80</td>
<td>22.13</td>
<td>22.40</td>
<td>21.73</td>
<td>21.12</td>
<td>20.82</td>
</tr>
<tr>
<td>HRT2</td>
<td>2.56</td>
<td>1.87</td>
<td>1.72</td>
<td>1.84</td>
<td>1.47</td>
<td>2.26</td>
<td>1.04</td>
<td>1.53</td>
<td>1.91</td>
<td>1.27</td>
<td>2.23</td>
</tr>
<tr>
<td>HRT3</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
<td>10.67</td>
<td>13.98</td>
</tr>
</tbody>
</table>

### GEN

Linear GP obtains good results in all GEN problems. TABLE 6 shows that the error rates of ES1.2 and ES1.3 are much better than ES1.1. Due to the high dimension of GEN problem, ES1.3 takes much more time than other experiment settings. A similar situation occurs in ES2.3. We believe that ES1.3 and ES2.3 obtain better results because they used $\log_{120} \approx 7$ populations. The results of all ES3 are better than ES1.1. In GEN problems, using more populations should lead to improved accuracy.

### TRD

All of the experiment settings outperform linear GP and NN in all TRD problems. In TABLE 6, we found that using more populations leads to better performance. Although ES2.2 performed well in TRD2, the error rates of all ES2 settings are not good in this problem. The best accuracies among three TRD problems occur in ES3 settings. ES3 indicates that using more layers is desirable with TRD problems.

No single setting of LAGEP dominates the others. However, we found that ES2.1 performed better in most
of these medical problems. This phenomenon is hard to explain. Intuitively, because ES2.2 and ES2.3 have more than two populations in the first layer, they should generate better results than ES2.1. Such intuitive thinking was verified at ES3.3. ES3.3 outperformed ES3.1 and ES3.2. Therefore, we surmise that ES2.2 and ES2.3 did not perform better than ES2.1 because there were only two layers and the final layer contained only a single population; fewer populations in the first layer indicates fewer terminals, which makes the population in the final layer discover a good solution based on them more easily.

<table>
<thead>
<tr>
<th>TABLE 6</th>
<th>Accuracy comparison with experiment settings, linear GP (Li-GP), and NN on 3-class medical problems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES1.1</td>
</tr>
<tr>
<td>GEN2</td>
<td>14.25</td>
</tr>
<tr>
<td>GEN3</td>
<td>9.33</td>
</tr>
<tr>
<td>HRS1</td>
<td>23.08</td>
</tr>
<tr>
<td>HRS2</td>
<td>24.18</td>
</tr>
<tr>
<td>HRS3</td>
<td>29.67</td>
</tr>
<tr>
<td>TRD1</td>
<td>0.5</td>
</tr>
<tr>
<td>TRD2</td>
<td>1.33</td>
</tr>
<tr>
<td>TRD3</td>
<td>0.46</td>
</tr>
</tbody>
</table>

ES1.1 is time-consuming in almost all problems. Obviously, ES2.1 took the least time because it used only three small populations. ES3.3 used as much time as ES1.1 because it has many populations. Given a problem, LAGEP used a different amount of time with each class because of the sample distribution of classes.

From the error rate estimations and time records, we find that LAGEP can generate results comparable with a single large population GP. Using only two layers and a small population size, ES2.1 is capable of generating better results than ES1.1 and uses only half the time.

4 CONCLUSIONS AND FUTURE WORK

In this paper we propose a novel multi-population GP method, LAGEP, for solving real world classification problems - medical problems. LAGEP arranges a number of populations into a layer and
generates results through a number of layers. Every population can have short individuals but still generate good results. In order to prevent falling into a local optimum for a long time, a method for tuning the mutation rate is proposed. Several experiments illustrate that LAGEP can generate better results than a single large population GP or isolated multiple population GP (isolated MGP). Furthermore, we also show that LAGEP can outperform other evolutionary algorithms. Using LAGEP, we can obtain better classification results with only half the time that a single large population GP takes. The source code of LAGEP used in this paper is available in [17].

We intend to develop further research based on the LAGEP architecture. Our future work will focus on investigating LAGEP in real-world classification problems with feature generation. We attempt to generate new sample features (in addition to original input features) by LAGEP for reinforcing dissimilarity between samples in different classes. By making use of these new features, LAGEP can easily discover a discriminate hyperplane that separates samples into different classes.

Bibliography


