Framework for Optimization of Genetic Programming Evolved Arithmetic Classifier Expressions Using Particle Swarm Optimization for MultiClass Classification

Hajira Jabeen and Abdul Rauf Baig

hajira.jabeen@nu.edu.pk, rauf.baig@nu.edu.pk

Abstract

Evolution of Arithmetic Classifier Expressions (ACE) using Genetic Programming (GP) has received considerable attention due to numerous advantages like flexibility and transparency. However, GP suffers from few drawbacks like long training time and code growth. These properties are particularly undesirable for classification. In this paper, we describe a method to use Particle Swarm Optimization (PSO) for tuning of ACE for multiclass classification problems. ACE is evolved using GP and weights are introduced along each terminal in ACE for optimization. Each weight corresponds to the effect of a terminal in the output. These weight values are optimized using PSO. We have tested the performance of proposed method over several benchmark multi-class classification datasets. The results reveal that new hybrid classification approach performs better than the basic GP for all problems.

Keywords: Genetic Programming, Classification, Optimization, Particle Swarm Optimization.

Introduction

The task of classification ages from the ancient Greek scientist Aristotle who established the very first known classification of everything. Classification categorizes different objects into predefined groups having some similar properties. Initially, the training data is analyzed, and a model is built. Then, this model is used to predict the class labels of unseen data in the second step. Classification finds it applications in fraud detection, recognition, diagnosis and several others. This raises the need of efficient classification algorithms. It is a challenging task due to unpredictability and variety in data. Heuristics like evolutionary algorithms have been found successful in dealing with classification problems.

Genetic Programming is one of the evolutionary algorithms aimed to evolve computer programs automatically. It is a directed search technique based on mechanics of biological evolution. It works by initializing random programs, of variable size and length, and evolving these programs based on some evaluation criteria. The reproduction operators are applied recursively in search of fitter members. The process is repeated from generation to generation till a given termination criterion is satisfied. This process is similar to other evolutionary algorithms, but GP offers some outstanding features due to its different representation scheme. It can solve problems without explicit knowledge of structure or size of the solution in advance. This is opposed to other preset structured algorithms like neural networks or support vector machines. The inductively learned solutions in GP can model hidden relationships of data and remain unaffected by data distribution. GP based classifiers are transparent, portable, comprehensible and offer fast evaluation of results. A recent GP based classification method evolves of arithmetic classifier expressions (ACE). This ACE expresses an arithmetic relationship between different attributes of data. For example, an ACE can be of the form $(A_1+A_2)/(A_3-0.5)$ where A_1,A_2,A_3 are some attributes of the data and 0.5 is a constant. The positive or negative output of expressions performs the discriminative decision between the two classes. This technique has been found successful in solving various problems.

Besides numerous advantages, GP based classification suffers from a few limitations. GP is inherently computational expensive task, and classifier evolution requires long training time due to recursive evaluation procedure of the potential programs (population). The classifiers increase in their complexity during the evolution if necessary measures for avoiding code growth are not taken into account. GP yields results of varying structure and accuracy. Although common in many evolutionary algorithms, but we are interested in fitter classifiers only. This is also termed as lack of convergence property in GP. Lots of work has been done to overcome these limitations, but most of the solutions are not specifically addressed for the problem of classification.

In this paper, we have proposed Particle Swarm Optimization (PSO) based local search framework that stops the evolution process earlier and increases the performance of GP evolved ACE by optimization. A weight is appended to each terminal in ACE. These weights form a weight vector which is randomly initialized and optimized by PSO. We have chosen PSO for its several outstanding features. This includes its simplicity, ability to search for a better solution in less function evaluations and better global search mechanism [1].

The contributions of the proposed framework are listed as follows:-

1) A hybrid GP and PSO for classification

2) Novel optimization scheme applied to terminals of GP in search of optimal weights.

3) Better accuracy with less Function evaluations

This paper is aimed to present and analyze the proposed framework to optimize the GP evolved ACE. The set of weights is appended to all the terminals present in an expression and optimized for better accuracy. This is the first attempt towards optimization of GP evolved classifiers through attribute weighting. We have performed GP based classification on five benchmark datasets taken from UCI-ML repository. PSO based local search has increased the accuracy of classifiers in lesser number of function evaluations and yielded simpler classifiers as compared to increasing the GP evolution for a larger number of generations (due to bloat).

Related Work

GP is relatively new approach for automatic evolution of classifiers. It uses the Darwinian principle of survival of the fittest to search for fitter classifiers. It has gained importance in data mining community for its special ability to learn and represent implicit relationships in the data with its flexible structures. This is in contrast to the fixed models like neural networks or support vector machines.

GP has been applied for classification in several ways, which can be categorized into three types. First includes evolution of classification algorithms like decision trees, neural networks, petrinets, autonomous systems and rule induction algorithm [2-7]. Another approach is to evolve logical rules or expressions using logical operators for categorical or nominal data [8-12]. The third approach is evolution of Arithmetic classifier expressions (ACE) for real/integer valued data. This approach has been found efficient for several classification problems, and it is known as GP specific classification method. The output of an ACE is a real value. The boundary of positive and negative numbers serves as a threshold for binary classification problems. Where one class represents the positive values and other are represented by the negative output. The challenging task is to use ACE for multiple class classification. Two methods are frequently used for such a task. One is the binary decomposition method. Where, one classifier is evolve for each class and classification decision is obtained by evaluating each classifier and assigning the class with higher output or positive output. In this technique, a classifier for one class considers other classes of data as single ‘not desired’ class. Binary decomposition methods have been explored in [13-17].The second method is assigning thresholds to the output of ACE. The method includes assigning static thresholds [18, 19], dynamic thresholds [19, 20] and slotted thresholds [21]. A relatively different, GA inspired, method for multiclass classification has been proposed by Durga [22], an amalgamated chromosome (vector) of classifiers(for all classes) is evolved in single GP run. Other effective multiclass classification methods include Mende’s work [23] where two populations are evolved simultaneously, one population contains fuzzy rule sets and other population contains membership functions. Both populations are coevolved so that they can effectively adapt to each other. All the methods have been found efficient for classification problems. Any of these methods does not deal with important problems like lack of convergence and long training time.

Relatively less work has been done to optimize GP evolved intelligent structures. A unique method [24] performs gradient search for optimization of ephemeral constants or numeric constant values present in GP trees producing better results for symbolic regression. Zhang [25] applied offline and online learning method for learning of ephemeral constants in a GP tree using gradient descent for object recognition that outperformed traditional GP. In another motivating work, [26] weights are added to all the edges present in a GP expression tree and optimized using gradient descent. The method is found efficient in terms of accuracy. Both methods proposed by Zhang are successful but they are added into the evolutionary process adding the computational effort into an already expensive task. This raises the need for an optimization algorithm that is computationally efficient and yields better performance.

Optimization of GP expressions can be achieved by simplification and reduction in size of GP trees. Such methods have been addressed by several researchers. Zhang [27] has emphasized evolution of simple classifier expressions using GP. He uses an online simplification approach that search for algebraic expressions that can be reduced and apply the reduction formulas using a hashing method. The method has been found efficient in simplification of evolved classifier and reducing the time required to evolve complex structures.

PSO is a recently proposed efficient optimization algorithm. It works using a simple concept of moving birds and is easy to implement. It is computationally efficient and has been found effective on a wide variety of applications like Function optimization [28] or neural network training [29]. PSO has been widely compared with other optimization algorithms and found successful. PSO solves optimization problems by simulating bird flock/swarm flying together in n-dimensional space in search of some optimum place, adjusting their movements in the constrained environment. Considerable research has been done for optimization and efficient working of PSO. Several parameters have been introduced to improve the performance of PSO. Two important parameters are constriction coefficients and inertia weight. Constriction coefficients set the proportion to which we admire the previous best position of a particle and the global best particle of the swarm during the movement of one particular particle. A PSO variant with a constriction factor has been introduced by Clerc [30] who analyzed the convergence behavior of the PSO. Constriction factor guarantees the convergence and improves the exploration ability of swarm. The inertia weight determines the step size for movement. Shi [31] introduced the concept of linearly decreasing inertia weight which is much like the temperature in the simulated annealing context or we can correlate it with the fluidity of the medium. A fuzzy scheme to change the inertia weight, non-linearly, is proposed by Shi [32]. Optimal values of constriction coefficients and inertia weights proposed by Clerc reported by Poli in [1] are 0.7298 and C_0=C_1=1.49618.

To improve the performance of PSO, another research focus has been variations in PSO topology. Keneddy [33] proposed that PSO with smaller neighborhood performs better on complex problems and larger neighborhood would perform better on simpler problems. Suganthan [34] suggests dynamically increasing neighborhood, until it includes all the particles of the swarm. Parsopoulos uses a combination of the global version and local version to make a unified particle swarm optimizer (UPSO) [35]. Mendes introduce a fully informed PSO [36], in which all the neighbors of the particle are used to update the velocity. The influence of each particle on its neighbors is weighted based on its fitness value and the neighborhood size. Some variations in basic behavior of PSO are attractive repulsive PSO (ARPSO) [37], predator pray approach to PSO (PPO) [38] , species based PSO (SPSO) [39] and charged swarm [40]. The purpose of all these variations is to avoid premature convergence, and optimization of multi-modal functions.

PSO performs better when compared to other optimization algorithms [41, 42] due to its ability to search for global optima in lesser number of iterations. It can work efficiently with lesser population size as compared to Genetic Algorithms. It has the ability to escape local optima, contrary to gradient descent or other local search methods. Moreover, its simplicity and ease of understanding adds to its applicability.

We have not been able to find any optimization performed on GP evolved expressions using PSO. However, several hybrid algorithms of PSO and GP have been proposed in literature [43-46]. This makes our method unique in its approach. Next section explains the proposed optimization process preceded by the classification algorithm.

Methodology

A binary classifier is many to one mapping that takes a feature vector as its input and assigns a class label to it. In this paper, we are interested in finding an optimal arithmetic classifier expression using a hybrid approach. Such an expression outputs a real value which is translated into class label. A GP-ACE is evaluated for an input data, and if the output of the GP-ACE is ≥ 0, the input data is assigned to one class; otherwise, it is assigned to the other class. Thus, the desired output is '+1' for one class and '-1' for the other class in the training set. The best classifier during the genetic evolution is the one, which correctly classify the maximum of training samples. Although, this mapping is static but it has achieved good results for binary classification problems. For multi-class class classification by ACE, the well known technique is binary decomposition [13] or one-versus-all method. Figure 1 gives an overview of the proposed GP-PSO algorithm.

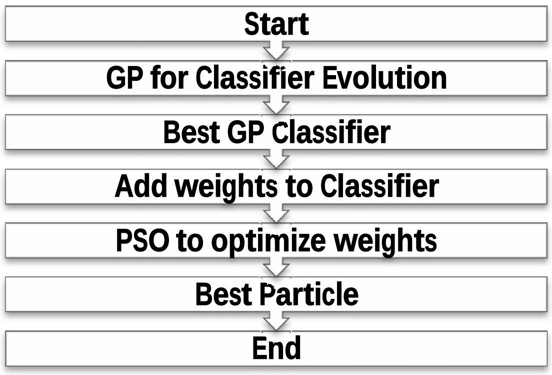

Figure 1 GP-PSO algorithm

GP is used to evolve ACE for classification. In case of multi-class classification, these ACE will be equal to the number of classes in the dataset. Where each ACE is trained to discriminate between one class and all the other classes. Each ACE is then prepared for tuning by the addition of weights. These weights are then optimized using PSO. The detailed description of GP-PSO is given in the *Algorithm GP-PSO*.

```
Step 1.  BEGIN
Step 2.  For each class ϵ data
      a. Apply GP to evolve ACE
      b. Add weights to ACE.
      c. Initialize all weight particles.
      d. Optimize these weights using PSO.
      e. Output Best result.
      f. Save best PSO ACE
Step 3.  End for
Step 4.  Integrate GP-PSO ACE of all classes
Step 5.  Check accuracy of combined PSO ACE.
Step 6.  END
```

Algorithm 1 GP-PSO

The ACE for each class is evolved using a separate GP run. The terminals are attributes of data and ephemeral constants, and the functions set is {+, - , /, *}. This method evolves arithmetic expressions with attributes as variables. This expression is trained to discriminate one class from the others by its real value output. Positive value corresponds to 'belong-to' the class, and negative output corresponds to 'not-belonging-to' the particular class. This technique has been explored in detail by Kishore et al [13] and Jabeen et al [48]. The difference in the later technique is the use of DepthLimited Crossover, which does not let classifiers increase in their complexity during evolution. Once we have obtained an ACE for a class using GP [48], it is optimized using PSO after the addition of weights.

The addition of weights along each terminal increases the complexity of ACE. Yet, this increase in complexity is less than uncontrolled bloat in GP evolution. GP-PSO stops the evolutionary process earlier and tunes the best GP classifiers. As mentioned earlier, there are usually two types of terminals present in the ACE. Attribute values [A_1,…..A_n] and an ephemeral constant [0-10]. We assign a weight to all the terminals present in ACE by addition of a '*' node and a weight node. Each node is assigned a different weight, irrespective of its value. This increase in complexity is shown in the Figure 2.Consider an example ACE

$$(A_1 + A_2) / A_3 \tag{1}$$

which becomes,

$$((A_1 * W_1)+(A_2 * W_2))/(A_3 * W_3) \tag{2}$$

after addition of weights. The weights added to an ACE combine to form a weight vector.

$$W = [W_1, W_2, W_3] \tag{3}$$

For an arbitrary ACE with t terminals, the increase in number of nodes in tuned tree is 2't' where t nodes are function nodes having value '*' and 't' nodes are terminal nodes having weights as their values. Then, the total number of nodes added to tree will be:-

$$\text{Number of new nodes} = 2*t \tag{4}$$

The weight vector will be:-

$$[W_t] \text{ where } t=1:t \tag{5}$$

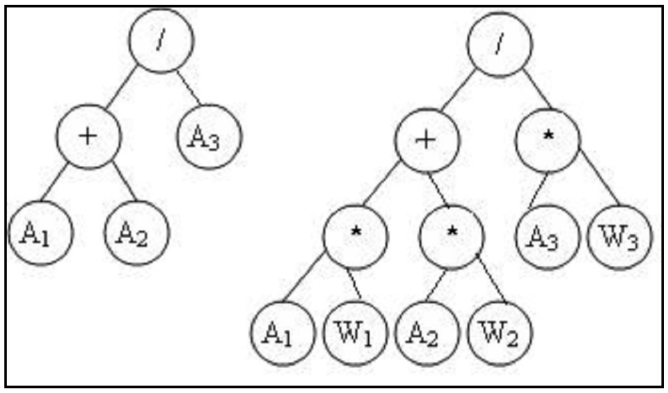

Figure 2 : Addition of weights for optimization

Each weight is associated to the node 't' of the tree for a particular class 'c'. An important point to note here is that the classifier remains intact if the values of all the added weights are set to '1'. Let A_0 be the original chromosome and A_w be the weight added chromosome, then

$$A_0 = A_w \text{ if all } [W_t]=1; \tag{6}$$

For optimization, a population of weight particles is initialized. These weights are assigned random values between -1 and 1. All random values may not lead to optimal results, therefore we have used a special initialization method. We check the accuracy of new classifier and do not accept the new particle if the accuracy is less than the original classifier. This ensures the initialization of PSO with fitter particles. Each particle is a multidimensional point in hyperspace that has as many dimensions as there are weights in a GP expression corresponding to each terminal. PSO is used to evolve these weights for optimal values. The fitness of each particle is calculated by putting the values of weights in their corresponding positions and evaluating the accuracy of the classifier. We have used cognitive-social model that keeps track of its previous best as well as the global best particle. The velocity of particles is updated by the following equation:

$$V_{i+1}= w V_i+C_o \, rand(0,1)(X_{lbest}-X_i)+C_1 \, rand(0,1)(X_{gbest}-X_i) \quad (7)$$

Where w is the inertia weight, C_0 and C_1 are the constriction coefficients, X_{lbest} and X_{gbest} are the local and global bests of the particle. The inertia weight can be interpreted as the fluidity of the medium in which a particle moves. Where as the constriction coefficients control the particle convergence and prevents explosion by eliminating the need for V_{max} parameter. [1]. We have used the clerc's constriction method which recommends following values

$$C_0=1.49618, \; C_1=1.49618 \text{ and } w=0.7298 \quad (8)$$

These particles have the ability to converge without any particular V_{max}. This results in a PSO with no problem specific parameters, making it the canonical PSO of today as reported by Poli. [1] X_{lbest} is the best position a particle has ever visited, whereas the X_{gbest} is the best position of whole swarm. X_{gbest} can be different for each particle based on the model used for PSO topology.

We have used the neighborhood model where a particle follows its neighboring best position as opposed to global best particle where all particles in the swarm follow same particle. This helps in exploring the search space effectively. This has been suggested by Kennedy [33] that the neighborhood of smaller size works better for complex problems.

After velocity update, the particles update their positions using the following equation:

$$X_{i+1}=X_i+V_i \quad (9)$$

All the particles in the swarm continue to update their positions and velocities using above equations, moving in the multidimensional hyperspace of solutions in search of optimal position until the termination condition is met. The condition is either a number of iterations, or certain threshold on fitness achieved by best particle. The optimization algorithm is explained in *Algorithm PSO.*

```
Step 1. Begin
Step 2. Initialize particles
Step 3. While termination criteria not met
     a. Calculate fitness of particles
     b. For each particle
              i.  Update particle $g_{best}$
             ii.  Update particle velocity using equation 7
            iii.  Update Particle Position using equation 9
     c. End particles
Step 4. End while
Step 5. Return best particle
Step 6. End
```

Algorithm 2 PSO

These weight particles are evolved for optimal position for a few generations until termination criteria is fulfilled. After optimization of ACE for each class the classification decision is made by evaluating each ACE and the ACE with positive response is declared winner. The conflicting decisions among multiple classifiers are resolved using the conflict resolution mechanism proposed by Muni [22].

Results

We have used five benchmark classification problems from UCI ML [49] repository to test the performance of proposed framework. Table 1 summarizes the properties of all the datasets used for classification. We have chosen data sets with varying number of classes from various dimensions of life with varying number of attributes and classes. These datasets are real valued datasets, which do not need any preprocessing for ACE based classification.

The parameters used for classifier evolution using GP are mentioned in Table 2. For classifier evolution we have used the simple binary decomposition method [13], [48]. These parameters have been empirically selected and detailed in our previous work [48].

Table 1 Datasets used for experimentation

Datasets	Referred as	Classes	Attributes	Type	Instances
Iris	IRIS	3	4	Real	150
Wine	WINE	3	13	Integer, Real	178
Statlog (Vehicle Silhouettes)	VEHICLE	4	18	Integer	946
Glass Identification	GLASS	6	10	Real	214
Yeast	YEAST	10	8	Real	1484

We are interested in showing the performance of PSO based local search, therefore we have not evolved GP for very long number of generations. GP can increase the performance of classifiers when evolved for longer number of generations. But this increase comes with the expense of very large number of function evaluations when compared to PSO based local optimization step.

Table 2 GP parameters used for classification

Population	600
Crossover Rate	0.50
Mutation rate	0.25
Reproduction Rate	0.25
Selection for DepthLimited cross over	Tournament selection with size 7
Selection for mutation	Random
Selection for reproduction	Fitness Proportionate selection
Mutation type	Point Mutation
Initialization method	Ramped half and half method
Function Set	+,-,*,/ (protected division,division by zero is zero)
Terminals	Data Attributes $A_1,A_2\ldots A_n,$ Ephemeral Constant [0,10]
Termination Criteria	User specified generations or 100% training accuracy of classifier

Table 3 presents the PSO parameters used for GP classifier tuning in GP-PSO approach. We have used smaller neighborhood as pointed by Kennedy that it works better for complex problems. Other parameters like constriction coefficients and inertia weight have been adopted from Clerc's analysis [1]. This confirms the convergence of PSO with these parameters. The initial weight values have been initialized between [-1,1] but not restricted to these values. PSO can search the optimal values beyond these initial limits. All these parameters have been selected after careful empirical analysis of the proposed approach.

Table 3 PSO parameters

Particles	100
Initial values	[-1,1]
Dimensions	Number of leaves
Iterations	30
C_0	1.49
C_1	1.49
W	0.7
Model	Lbest model
Neighborhood size	2

Before presenting the general results we will present the some initial results generated using GP-PSO in Table 4. For each dataset GP is evolved for 10 generations and best ACE is extracted for each class. Its test accuracy s noted and presented in the Table. On the same ACE the weight addition and optimization mechanism is applied. The test results are reported in GP-PSO column. We have repeated the weight addition and tuning process ten times for each ACE and averaged the results.

Table 4 Accuracies achieved using GP-PSO averaged for 10 PSO executions on a single GP classifier evolved for 10 generations

Datasets	GP	GP-PSO				
		Avg	Min	Max	Med	Avg Increase
IRIS	92.30	96.00 ± 0.39	96.00	97.77	97	3.7
WINE	73.33	84.20 ± 4.22	73.33	86.66	80	10.87
VEHICLE	44.57	53.48 ± 2.84	44.57	53.48	50.6	8.91
GLASS	47.36	50.73 ± 2.78	47.36	52.63	50.7	3.37
YEAST	31.26	36.60 ± 4.67	26.60	39.26	31	5.4

The GP column represents the accuracy of one ACE obtained after evolution of GP for 10 generations. GP-PSO gives the optimization result averaged for 10 runs of PSO on ACE. Min, Max and Med represent the minimum, maximum and median values of the GP-PSO results. The last column shows the average increase in accuracy achieved after application of GP-PSO ten times. We have been able to achieve 17% to 3% increase in accuracy for one classifier on average. We increased the GP evolution till 120 generations to study the effect of our proposed tuning method. The presented results have been produced as follows:-

1. The GP algorithm is executed ten times for tenfold cross validation for 120 generations.
2. An ACE is extracted in each run after 10 generations.
3. The PSO based optimization is performed 10 times on every ACE. Therefore, a result in GP column is average of 10 GP runs and result in GP-PSO is average of 1000 PSO runs.

The multiclass classification involves evolution and optimization of multiple classifiers (one for each class). We have optimized one classifier at a time for each class. Once the classifiers for all classes are optimized, they are combined to obtain the final classification decision.

Table 5 provides accuracies achieved for individual classifiers in case of WINE data. GP-PSO has successfully enhanced the performance of classifiers for each class that, in turn, increased the cumulative accuracy of the classifiers. However, the effect in the cumulative result is less obvious due to conflicts in multiple classifiers.

Table 5 Accuracies achieved using GP and GP-PSO for individual classes and combine result

WINE								
GP Generations	Class 1		Class 2		Class 3		Combined	
	GP	GP-PSO	GP	GP-PSO	GP	GP-PSO	GP	GP-PSO
10	89.87%	93.08%	86.79%	92.41%	95.58%	98.75%	77.9%	83.9%
20	88.62%	93.08%	89.29%	91.16%	95.62%	98.75%	78.5%	84.1%
30	90.50%	90.54%	89.33%	90.58%	98.12%	98.12%	78.6%	84.5%
40	91.12%	94.95%	89.33%	92.45%	96.25%	98.12%	79.3%	85.0%
50	91.12%	93.70%	88.66%	91.83%	97.50%	98.12%	80.3%	85.5%
60	92.37%	94.95%	88.04%	91.87%	97.50%	97.50%	81.4%	86.0%
70	92.37%	94.29%	88.04%	92.45%	96.87%	98.12%	81.4%	86.3%
80	92.37%	95.58%	89.29%	90.62%	96.87%	98.12%	81.5%	86.7%
90	91.16%	93.08%	89.91%	90.54%	96.87%	98.75%	81.6%	86.8%
100	90.50%	94.87%	88.66%	90.54%	96.87%	98.12%	81.7%	86.9%
110	91.12%	94.25%	85.50%	91.25%	96.25%	99.37%	82.0%	87.5%
120	91.08%	93.66%	84.87%	93.12%	97.50%	98.75%	82.7%	88.4%

Table 6 compares the classification results of GP and GP-PSO. The columns GP NFE and GP-PSO NFE columns represent the number of function evaluations (NFE) elapsed to achieve the results reported in that row. The GP and GP-PSO columns tell the classification accuracy obtained with GP alone and GP-PSO. We can see the GP-PSO outperformed GP in all the datasets. In many cases GP has not achieved the same accuracy till the end of evolution process. The GP-PSO results are better in accuracy and consume less number of function evaluations when compared in terms of accuracy. An important point note, here, is that NFE is directly proportional to time consumed in the experimentation. Larger NFE corresponds to more run time. Moreover, NFE provides system/code independent analysis.

Table 6 Comparison of GP and GP-PSO

Gen	GP NFE E+04	GP -PSO NFE E+04	IRIS		WINE		VEHICLE		GLASS		YEAST	
			GP	GP-PSO	GP	GP-PSO	GP	GP-PSO	GP	GP-PSO	GP	GP-PSO
10	0.6	0.9	92.5%	95.5%	77.9%	83.9%	42.3%	49.6%	44.7%	52.0%	38.7%	50.9%
20	1.2	1.6	93.1%	95.6%	78.5%	84.1%	43.3%	49.9%	48.1%	53.1%	40.9%	51.6%
30	1.8	2.2	93.2%	95.6%	78.6%	84.5%	44.9%	50.9%	52.2%	54.6%	45.9%	53.5%
40	2.4	2.8	93.7%	95.9%	79.3%	85.0%	46.6%	52.7%	52.3%	54.8%	49.1%	55.5%
50	3.0	3.4	93.7%	96.2%	80.3%	85.5%	47.1%	52.7%	52.4%	55.0%	50.0%	56.5%
60	3.6	4.0	94.1%	96.3%	81.4%	86.0%	47.2%	54.7%	52.9%	55.9%	50.8%	56.9%
70	4.2	4.6	94.6%	96.4%	81.4%	86.3%	47.6%	56.2%	53.1%	56.1%	52.3%	57.1%
80	4.8	5.2	95.0%	96.6%	81.5%	86.7%	47.9%	56.6%	54.0%	56.2%	54.3%	57.3%
90	5.4	5.8	95.0%	96.6%	81.6%	86.8%	48.7%	57.9%	54.3%	56.2%	55.8%	57.3%
100	6.0	6.4	95.0%	96.7%	81.7%	86.9%	48.8%	58.7%	54.4%	57.2%	56.4%	57.6%
110	6.6	7.0	95.0%	96.9%	82.0%	87.5%	48.8%	59.7%	54.5%	58.1%	56.6%	57.6%
120	7.2	7.6	95.0%	97.0%	82.7%	88.4%	49.0%	60.5%	54.7%	58.7%	56.8%	58.0%

Figure 3 presents the increase in accuracy for GLASS dataset. We can see that GP-PSO has effectively increased the classification accuracy in all cases. This increase in accuracy is higher in earlier and later generations as compared to 40-90 generations.

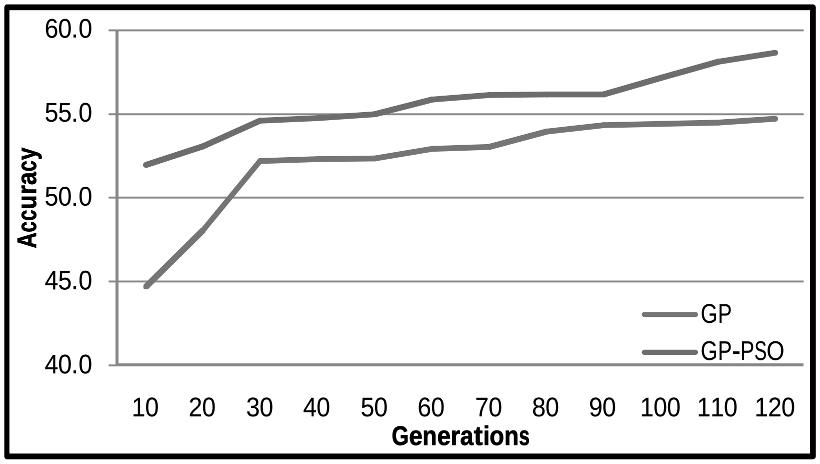

Figure 3 Increase in accuracy for GLASS data

Figure 4 displays the increase in accuracy averaged for all data sets. We can see that this increase in higher in earlier generations and later generations. This trend may be due to the fact that in earlier generations GP is searching for optimal solutions, which lead to higher increase. In later generations GP becomes more or less affected by bloat which in turn leaves more room for improvement by optimization.

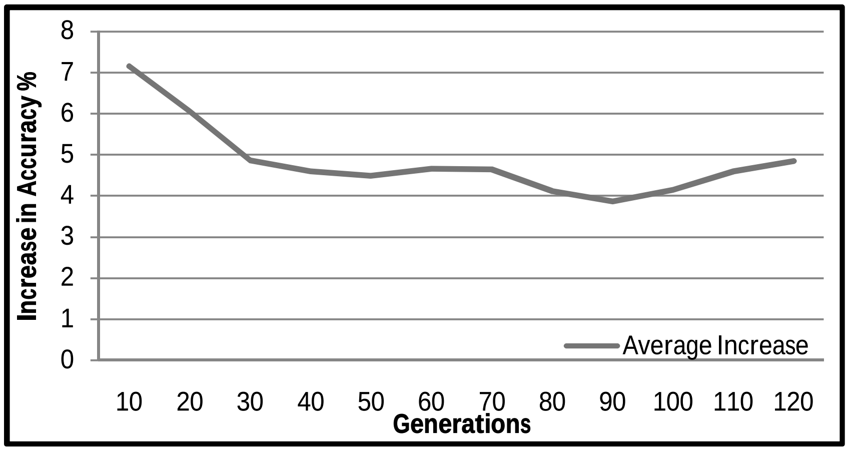

Figure 4 Average increase in accuracy for all datasets

Conclusions and Discussion

In this paper we have presented a novel hybrid framework for efficient tuning of GP evolved ACE. This method has successfully proved that addition of optimal weights to the terminals of a classifier increase their accuracy. This method can eliminate the need of evolving GP for large number of generations, which is a complex task. We can extract an ACE in a few generations of GP and increase its performance by addition and optimization of weights. This method has outperformed other GP based classification approaches. The future research directions include addition of weights along all the edges. Other optimization algorithms can also be explored for optimization of these weights.

References

[1] Poli, R, Kennedy, J and Blackwell, T., "Particle Swarm Optimization : An Overview." Swarm Intelligence, 2007, pp. 33-57.

[2] Konig, R, Johansson, U and Niklasson, L., "Genetic Programming - A Tool for Flexible Rule Extraction." 2007. IEEE Congress on Evolutionary Computation.

[3] Pappa, G A and Freitas, A A., "Evolving Rule Induction Algorithms with Multiobjective Grammer based Genetic Programming." Knowledge and Information Systems, 2008.

[4] Bojarczuk, C C, Lopes, H S and Freitas, A A., "Data mining with constrained-syntax genetic programming: applications to medical data sets." 2001. Proceedings Intelligent Data Analysis in Medicine and Pharmacology.

[5] Freitas, A A., "A Genetic Programming Framework For Two Data Mining Tasks : Classification And Generalized Rule Induction." CA , USA : Morgan Kaufmann, 1997. Genetic Programming. pp. 96-101.

[6] Eggermont, J., "Evolving Fuzzy Decision Trees for Data Classification." 2002. Proceedings of the 14th Belgium Netherlands Artificial Intelligence Conference.

[7] Tsakonas, A., "A comparison of classification accuracy of four genetic programming-evolved intelligent structures." Information Sciences, 2006, pp. 691-724.

[8] Engelbrecht, A P, Schoeman, L and Rouwhorst, S., "A Building Block Approach to Genetic Programming for Rule Discovery, in Data Mining: A Heuristic Approach." [book auth.] H A Abbass, R Sarkar and C Newton. *Data Mining.* s.l. : Idea Group Publishing, pp. 175-189.

[9] Carreno, E, Leguizamon, G and Wagner, N., "Evolution of Classification Rules for Comprehensible Knowledge Discovery." 2007. IEEE Congress on Evolutionary Computation. pp. 1261-1268.
[10] Falco, I D, Cioppa, A D and Tarantino, E., "Discovering Interesting Classification Rules With GP." Applied Soft Computing, 2002, pp. 257-269.
[11] Eggermont, J, Eiben, A E and Hemert, J I., "A comparison of genetic programming variants for data classification." 1999. Proceedings of the Eleventh Belgium Netherlands conference on Artificial Intelligence. pp. 253-254.
[12] Eggermont, J, Kok, J N and Kosters, W A., "GP For Data Classification , Partitioning The Search Space." 2004. Proceedings of the 2004 Symposium on Applied Computing. pp. 1001-1005.
[13] Kishore, J K, et al., "Application of Genetic Programming for Multicategory Pattern Classification." IEEE transactions on Eolutionary Computation, 2000.
[14] Loveard, T and Ciesielski, V., "Representing Classification Problems in Genetic Programming." 2001. IEEE Congress on Evolutionary Computation. pp. 1070-1077.
[15] Smart, W and Zhang, M., "Using Genetic Programming For Multiclass Classification By Simultaneously Solving Component Binary Classification Problems." Lecture Notes in Computer Science, 2005.
[16] Teredesai, A and Govindaraju, V., "Issues in Evolving GP based Classifiers for a Pattern Recognition Task." 2004. IEEE Congress on Evolutionary Computation. pp. 509-515.
[17] Bojarczuk, C C, Lopes, H S and Freitas, A A., "Genetic programming for knowledge discovery in chest-pain diagnosis." *IEEE Engineering in Medicine and Biology Magazine.* 2000, pp. 38-44.
[18] Zhang, M and Ciesielski, V., "Genetic Programming For Multiple Class object Detection." Australia : s.n., 1999. Proceedings of the 12th Australian Joint Conference on Artificial Intelligence. pp. 180–192.
[19] Parrott, D, Li, X and Ciesielski, V., "Multi-objective techniques in genetic programming for evolving classifiers." 2005. IEEE Congress on Evolutionary Computation. pp. 183–190.
[20] Smart, W R and Zhang, M., "Classification Strategies for Image Classification in Genetic Programming." 2003. Proceeding of Image and Vision Computing NZ International Conference. pp. 402-407.
[21] Zhang, M and Smart, W., "Multiclass object classification using genetic pro-gramming." Lecture notes in computer science, 2004, pp. 367–376.
[22] Muni, D P, Pal, N R and Das, J., "A Novel Approach To Design Classifiers Using GP." IEEE Transactions of Evolutionary Computation, 2004.
[23] Mendes, R R F, et al., "Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution." 2001. Genetic and Evolutionary Computation Conference, Late Breaking Papers.
[24] Topchy, A and Punch, W F., "Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values." 2001. Proceedings of the Genetic and Evolutionary Computation Conference. pp. 155-162.
[25] Zhang, M and Smart, W., "Genetic Programming with Gradient Descent Search for Multiclass Object Classification." 2004. 7th European Conference on Genetic Programming, EuroGP. pp. 399-408.
[26] Zhang, M and Smart, W R., "Learning Weights in Genetic Programs Using Gradient Descent for Object Recognition." 2005. Applications of Evolutionary Computing, EvoWorkshops. pp. 417-427.
[27] Zhang, M and Wong, P., "Genetic Programming for Medical Classification: A Program Simplification Approach." Genetic Programming and Evolvable Machines, 2008, pp. 229-255.
[28] Seo, J S, et al., "Multimodal function optimization based on particle swarm optimization." IEEE Transactions on Magnetics, 2006.
[29] Gudise, V G and Venayagamoorthy, G K., "Comparison of Particle Swarm Optimization and Back propagation as Training Algorithms for Neural Networks." 2003. Swarm Intelligence Symposium.
[30] Clerc, M and Kennedy, J., "The Particle Swarm Explosion, Stability, and Convergence In A Multidimensional Complex Space." IEEE Transaction on Evolutionary Computation, 2002, pp. 58–73.
[31] Shi, Y and Eberhart, R C., "A Modified Particle Swarm Optimizer." 1998. IEEE Congress on Evolutionary Computation. pp. 69–73.
[32] Shi, Y and Eberhart, R C., "Particle Swarm Optimization with Fuzzy Adaptive Inertia Weight." Indianapolis : s.n., 2001. Proceedings of the Workshop on Particle Swarm Optimization.
[33] Kennedy, J., "Small worlds and mega-minds: Effects of Neighborhood Topology on Particle Swarm Performance." 1931–1938. In Proceedings of Congress on Evolutionary Computation. p. 1999.
[34] Suganthan, P N., "Particle Swarm Optimizer with Neighborhood Operator." Washington : s.n., 1999. In Proceedings of Congress Evolutionary Computation. pp. 1958-1962.

[35] Parsopoulos, K E and Vrahatis, M N., "UPSO: A Unified Particle Swarm Optimization Scheme." 2004. Lecture Series on Computational Sciences. pp. 868–873.
[36] Mendes, R, Kennedy, J and Neves, J., "The Fully Informed Particle Swarm: Simpler, Maybe Better." IEEE Transactions on Evolutionary Computation, 2004, pp. 204–210.
[37] Riget, J and Vesterstrom, J, S, *A diversity-guided particle swarm optimizer - the ARPSO.* Department of Computer Science, University of Aarhus. 2002.
[38] Silva, A, Neves, A and Costa, E., "Chasing The Swarm: A Predator Pray Approach to Function Optimization." 2002. International Conference on Soft Computing.
[39] Parrot, D and Li, X., "Locating and Tracking Multiple Dynamic Optima by a Particle Swarm Model Using Speciation." IEEE Transactions on Evolutionary Computation, 2006.
[40] Blackwell, T M and Bentley, P J., "Dynamic Search with Charged Swarms." 2002. Proceedings of the Genetic and Evolutionary Computation Conference.
[41] Ercan, M F., "A performance comparison of PSO and GA in scheduling hybrid flow-shops with multiprocessor tasks." Brazil : ACM, 2008. Proceedings of the 2008 ACM symposium on Applied computing. 978-1-59593-753-7.
[42] Mohaghegi, S and Del Valle, Y., "A comparison of PSO and backpropagation for training RBF neural networks for identification of a power system with STATCOM." s.l. : IEEE, 2005. Proceedings 2005 IEEE Swarm Intelligence Symposium.
[43] Poli, R, Langdon, W B and Holland, O., "Extending Particle Swarm Optimization Via Genetic Programming." Springer, 2005. European Conference on Genetic Programming.
[44] Poli, R, Chio, C D and Langdon, W B., "Exploring Extended Particle Swarms:A Genetic Programming Approach." s.l. : Springer, 2005. Genetic and Evolutionary Computation Conference.
[45] Yan, L and Zeng, J., "Using Particle Swarm Optimization and Genetic Programming to Evolve Classification Rules." Dalina, China : s.n., 2006. 6th World Congress on Intelligent Control and Automation.
[46] Togelius, J, Nardi, R D and Moraglio, A., "Geometric PSO + GP = Particle Swarm Programming." Hong Kong : IEEE, 2008. IEEE World Congress on Computational Intelligence.
[47] Jabeen, H and Baig, A R., "Particle Swarm Optimization based Tuning of Genetic Programming Evolved Classifier Expressions." *Studies in Computational Intelligence.* Granada : Springer-Verlag, 2010.
[48] Jabeen, H and Baig, A R., "DepthLimited Crossover in Genetic Programming for Classifier Evolution." Computers in Human Behaviour, Ulsan, South Korea : Elsevier, 2009. Accepted.
[49] Asuncion, A and Newman, D J., Machine Learning Repository. [Online] 2007. http://archive.ics.uci.edu/ml/.