Program Evolution with Explicit Learning

Y. Shan, R. I. McKay, H. A. Abbass, D. Essam
School of Computer Science, University College
University of New South Wales Australia Defence Force Academy
Canberra, Australia
{shanyin,rim,abbass,daryl}@cs.adfa.edu.au

Abstract- In Genetic Programming (GP) and most other evolutionary computing approaches, the knowledge learned during the evolutionary processing is implicitly encoded in the population. A small family of approaches, known as Estimation of Distribution Algorithms, learn this knowledge directly in the form of probability distributions. In this research, we proposed a new approach for program synthesis – Program Evolution with Explicit Learning (PEEL), belonging to this family. PEEL learns probability distributions from previous generations and stochastically generates new populations according to this distribution. PEEL is intrinsically different from GP systems because it abandons conventional GP genetic operators and does not maintain a population. On the benchmark problems we have studied, this approach shows at least comparable performance to GP.

1 Introduction

Genetic Programming (GP) [6, 12] is a promising variant of genetic algorithms that typically evolves hierarchical structures, often described as programs. Through selection, unfit individuals are culled, and fitter individuals remain in the population. Genetic operators, such as crossover and mutation, vary these fitter individuals to explore their surrounding search space. In the evolutionary process, the population serves as a knowledge pool to implicitly encode the information describing the search space and characterising fitter individuals. The search space around the fitter individuals might be a promising search area, while building blocks, rather than totally randomly chosen symbols, exchanged among individuals might lead to efficient search. In this sense, GP can be regarded as a learning process. What is learned is the general salient characteristics of fitter individuals and promising search areas, implicitly encoded in the population.

Since we may take GP as a learning process which tries to characterize *what the good individual might look like*, why not make the learning explicit? The concept of explicit learning in GP has been previously studied. Some researchers have studied modules in GP in order to explicitly make use of building blocks, or to make the crossover more productive. Probabilistic Incremental Program Evolution (PIPE) [21] is a more radical approach. All these approaches attempt to explicitly characterize what good individuals might look like, although by different means.

In this research, we propose a new framework in this lineage – Program Evolution with Explicit Learning (PEEL). The Search Space Description Table (SSDT), a variant of stochastic parametric Lindenmayer Systems (L-system) [17], is used to describe the search space. A variant of Ant Colony Optimization (ACO) [8] is the learning method. Grammar refinement is employed to focus the search area. The search starts from a minimum SSDT, which describes the search space at a very coarse resolution. ACO updates the stochastic components of the SSDT. When this minimum SSDT is not adequate to focus the search to more promising, smaller areas of the search space, the grammar, which is represented as the SSDT, will be refined, i.e. certain rules in the SSDT will be split. On the benchmark problems we have studied, our new method shows at least comparable performance to Grammar Guided Genetic Programming (GGGP) [23], (which is probably the nearest equivalent implicit learning GP system) and to standard GP.

The paper is organized as follows. The next section is a brief survey of explicit learning in GP. In Section 3, our new approach is proposed. The section following contains the experimental details. The conclusions are given in the final section.

2 Explicit Learning in Genetic Programming

Explicit learning in GP is briefly surveyed in this section. In GP, the search mechanism is manipulating the individuals by applying genetic operators. The purpose is to find fitter individuals, and thus eventually the near-optimal or optimal solution. In canonical GP, there is no mechanism to describe *what the good individual might look like* although this is exactly what GP search tries to discover.

By explicit learning, we mean applying learning techniques to explicitly describe the characteristics of promising individuals. More precisely, we wish to describe promising areas of the search space. By more directly working on characterizing, finding and manipulating the promising search area, we hope to obtain better performance.

2.1 Modularity

Some research suggests, that if building blocks can be correctly identified and used, the performance of GP can be significantly improved. This line of research includes Automatically Defined Functions (ADF) [12], Genetic Library Builder (GliB) [3] and Adaptive Representation (AR) [19]. The building blocks or modules explicitly characterize what the promising individuals might look like and where the corresponding promising search areas might be.

2.2 Crossover

In [2], two self-adaptive crossover operators, selective self-adaptive crossover and self-adaptive multi-crossover, are proposed. These new operators adaptively determine where crossover will occur in an individual. Experimental results demonstrate that both of these self-adaptive operators perform as well or better than standard genetic programming crossover. Recombinative guidance for GP is proposed in [10]. For this, all the performance values for all the subtrees of a GP tree are calculated. These values are then used to decide which subtree will be chosen to apply GP operations. Although GP with recombinative guidance performs well on some problems, the definition of subtree value is problem dependent.

2.3 Probabilistic Incremental Program Evolution (PIPE)

PIPE [21] combines probability vector coding of program instructions, Population-Based Incremental Learning [4], and tree-coded programs. PIPE iteratively generates successive populations of functional programs according to an adaptive probability distribution, represented as a Probabilistic Prototype Tree (PPT), over all possible programs. Each iteration uses the best program to refine the distribution. Thus, the structures of promising individuals are learned and encoded in PPT. Good performance was reported over the problems which were studied.

2.4 Related Works in Genetic Algorithms

There is extensive similar work in Genetic Algorithms (GAs), but the relationships are too distant to cover the field in detail. Hence, we will mention only two highly relevant projects.

The first is the Learnable Evolution Model (LEM) [13]. The central engine of evolution in LEM is a machine learning mode, which creates new populations by employing hypotheses about high fitness individuals found in past populations. The machine learning mode seeks reasons why certain individuals in a population are superior to others, so as to form inductive hypotheses which explicitly characterize good individuals. In experimental evaluations, LEM significantly outperformed standard evolutionary methods in terms of the number of fitness evaluations.

The other is estimation of distribution algorithm (EDA) [14, 16]. EDA uses a probabilistic model of promising solutions to guide further exploration of the search space. EDA is a cluster of methods, ranging from methods that assume the genes in the chromosome are independent, through others that take into account pairwise interactions, to methods that can accurately model even a very complex problem structure with highly overlapping multivariate building blocks. Recently, there has been increasing research interest in this field.

3 Program Evolution with Explicit Learning

In Program Evolution with Explicit Learning (PEEL), traditional genetic operators are abandoned, and no population is maintained. Instead, a Search Space Description Table (SSDT) is used. The SSDT characterizes individuals with superior fitness and guides the search to promising areas. The higher level algorithm is sketched as follows:

```
1. Initialize Search Space Description
    Table (SSDT).
2. Generate new population according to SSDT.
3. Learn SSDT from the new population.
4. Refine SSDT if necessary (grammar
    refinement).
5. Mutate SSDT.
6. If termination condition is not satisfied,
    then go to 2,
    else output best individuals found.
```

The major data structures and procedures will be elaborated in the following subsections.

3.1 Representation of Search Space

The Search Space Description Table (SSDT) is used to represent the search space. It is a specialized stochastic parametric L-system. The SSDT consists of rules; each rule of the SSDT is defined as follows:

$$
\begin{aligned}
&predecessor(parameter : n, l) : \\
&\qquad (condition : n = depth\ and\ l = location) \\
&\quad \rightarrow successor_1(expression : n \leftarrow n+1, l'_1) \\
&\qquad successor_2(expression : n \leftarrow n+1, l'_2) \\
&\qquad \cdots \\
&\qquad successor_m(expression : n \leftarrow n+1, l'_m) \\
&\qquad (pheromone)
\end{aligned}
$$

where $predecessor$ and $successors$ are the same as in context-free grammars, $pheromone$, a term from Ant Colony Optimization, corresponds to the probability of choosing this production rule (pheromone is an unnormalized value; normalization gives us the probabilities)

```
expr = expr op expr | pre expr | x
op   = + | - | * | /
pre = sin | cos | e^ |ln
```

Figure 1: Context-free grammar for symbolic regression problem

and the two $parameters$ n and l are the depth and the relative location in the tree to be generated. n and l are used in the $conditions$ which test for matches. When matches occur, $expression$ increments the depth $parameter$ n so that next level of the tree can be generated. Note that the only major difference between this representation and that of L-systems is the term l'_i in the $expression$. Actually, in the SSDT, l'_i is *not* recorded since it is implicitly encoded in the individuals. The reason we have kept l'_i in the above formula is to keep the number of parameters of predecessor and successors consistent. The term *Left Hand Side* (LHS) stands for both the predecessor and its conditions. Therefore, that LHS matches means that both predecessors and conditions match. In the remaining part of this paper, sometimes we use the term $grammar$ to refer to the SSDT because the SSDT actually defines a grammar system.

As in Genetic Programming (GP), the individuals generated in PEEL are trees. Similarly to [7], the location information in the SSDT is coded as the relative coordinate in the tree. The position of each node in a rooted tree can be uniquely identified by specifying the path from the root to this specific node. A node's position can therefore be described by a tuple of n coordinates $T = (b_1, b_2, \cdots, b_n)$, where n is the depth of the node in the tree, and b_i indicates which branch to choose at level i. For example, in Fig 3, to reach the bottom right-most node in the left tree, from the root we need to choose the 2nd, 2nd and 0-th branch, i.e. 0220. Similarly, we can get the location of the left x at depth 3 in the left tree as 0200. (Note: in this paper the depth of the root is 0, the branch in each subtree which connects to the left-most child node is the 0th branch and the left-most child is the 0th child).

3.1.1 Example

For many symbolic regression problems, we may use the context-free grammar illustrated in Fig 1. $expr$ is the starting symbol. $e^\wedge$ and ln are exponential, and protected natural logarithm, respectively.

The initial SSDT representation for the first rule in Fig 1, which can be used to generate an individual at depth 0, may be as illustrated in Fig 2. Rules to generate other levels/depths of individuals can be defined in a similar way. The first condition $n = 0$ indicates these rules can only be used to generate the top level ($depth = 0$) of the tree. The pheromone levels are initialized uniformly, and are then

```
expr(n,l):(n=0 and l=#) = expr(n+1,0) op(n+1,1)
     expr(n+1,2) (pheromone:1)
expr(n,l):(n=0 and l=#) = pre(n+1,0) expr(n+1,1)
     (pheromone:1)
expr(n,l):(n=0 and l=#) = x(n+1,0) (pheromone:1)
```

Figure 2: Initial rules of the SSDT for generating depth 0 of tree

subject to update by the learning method – Ant Colony Optimization in this work. All the locations are initialized to the value $\#$, which means "don't care" or "match to any value". When necessary, the grammar will be refined and location information incrementally added. Note the predictable values for parameters of the terms on the right-hand side (always n+1, and then the location on the LHS with appropriate suffix). For clarity, in the following parts of the paper, these parameters will be omitted.

Controlling the size of the tree in terms of depth is not a trivial task for GGGP [20, 23]. However, this SSDT representation provides a natural way to do it. In this example, if the maximum depth is 14, we may simply set the pheromone levels of the following two rules to be 0 at the second last level ($n = 13$):

```
expr(n,l):(n=13 and l=0) = expr op expr (ph 0)
expr(n,l):(n=13 and l=1) = pre expr (ph 0)
```

Since these two rules are the only recursive rules, setting their pheromone levels to 0 simply stops recursive rule rewriting. Therefore, the tree will stop growing at the next level ($n = 14$).

3.2 Generation of Individuals

The generation of individuals, in this research, is similar to the initial generation of individuals for GGGP [23]. The two major differences are: firstly, Left Hand Side (LHS) matching, i.e. besides matching the predecessor, the conditions also need to be satisfied, and secondly, when deciding between rules with matching LHSs, one of the rules is chosen stochastically according to the attached probabilities.

For example, to generate an individual, given an initial SSDT derived from the context-free grammar shown in Fig 1, we need to rewrite the starting symbol $expr$, which is the root of the new individual. Since this $expr$ is the root, and no location requirement is available, we get $n = 0$ and $l = \#$. Searching the SSDT, we find three rules, illustrated in Fig 2, that match this left hand side, i.e. predecessor $expr$ and its conditions $n = 0$ and $l = \#$. Each of these three rules has the probability $1/(1 + 1 + 1)$ to be chosen. Assuming that the first rule in Fig 2 is chosen, then we have an incomplete individual of depth 1, which looks like the upper part of the left tree in Fig 3. For all non-terminal leaf nodes, this process is repeated until a closed individual is generated.

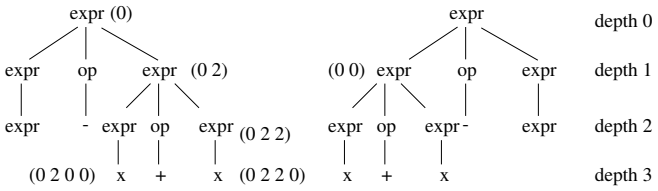

Figure 3: The generation of x-(x+x) and (x+x)-x

3.3 Learning Methods

In this work, Ant Colony Optimization (ACO) is used to update the pheromone levels. ACO is a kind of positive feedback system. A colony of ants concurrently and asynchronously move through adjacent states of the problem by moving through neighbor nodes. By moving, ants incrementally build solutions to the optimization problem. Once an ant has built a solution, or while the solution is being build, the ant evaluates the solution and deposits information about its goodness or fitness on the pheromone trails of the connections it used. This pheromone information then directs the search of the future ants. For a good introduction of ACO, [8] is recommended; [1] is a highly related work, which uses ACO to compose programs.

In PEEL, ACO is used as the major learning mechanism. For each iteration of the PEEL algorithm, we do two pheromone updates:

1. decrease the pheromone of those rules which were used to generate *each individual* in this iteration,
2. increase the pheromone of those rule which are used to generate *each elite individual* (the individuals which are the best so far; we maintain a group of elitists.)

In more detail, assume the probability of $rule_i$ needs to be updated according to $individual_j$, the probability update formula for increasing pheromone is:

$$\begin{aligned} pheromone_i = & pheromone_i \times decay\_rate \\ & + (1 - decay\_rate) \times fitness_j \end{aligned}$$

whilst, for decreasing pheromone, the formula is:

$$pheromone_i = pheromone_i \times decay\_rate$$

where $decay\_rate$ is a predefined value, $fitness_j$ is the fitness of $individual_j$, calculated as:

$$fitness_j = 1/(1 + \sum_{case} |estimated_{case,j} - actual_{case}|)$$

where $actual_{case}$ is the actual value of $case$ while $estimated_{case,j}$ is the result of evaluation of $individual_j$ over $case$. To obtain the probabilities, normalization needs to be applied to the $pheromone$. Normalization is applied independently whenever a Left Hand Side (LHS) must be matched. It is applied to all rules which match that particular LHS.

The rationale of these updates is quite clear. Decreasing pheromone directs ants away from the paths which have been visited and, thus toward exploring different parts of the search space, while increasing pheromone on good paths encourages exploitation of those parts of the search space.

3.4 Grammar Refinement

As can be seen, the initial SSDT represents the search space at a very coarse level. If the problem is complex, this coarse description is inadequate. However, if this occurs we can refine the grammar so that the search space is covered by a finer grid, thus enabling promising areas of the search space to be searched more intensively.

For example, assume we have the grammar illustrated in Fig 1. One of its initial rules in the SSDT representation is:

```
expr(n,l):(n=1 and l=#) = expr op expr (ph:p)
```

This rule would be applied to rewrite predecessor $expr$ at depth 1. $l = \#$ implies that no location information will be considered. Therefore, at depth 1, as long as predecessor $expr$ is matched, the rule would be applied with probability p no matter where the predecessor $expr$ is. As a result, this rule cannot distinguish between the derivations for $x-(x+x)$ and $(x+x)-x$ given in Fig 3 as these two trees must have the same probability of generation for any probability distribution over our original grammar.

Thus, it is necessary for the system to be able to increase the expressiveness of the grammar through refinement. For example, if we split the above rule into three:

```
expr(n,l):(n=1 and l=0) = expr op expr (ph:p0)
expr(n,l):(n=1 and l=1) = expr op expr (ph:p1)
expr(n,l):(n=1 and l=2) = expr op expr (ph:p2)
```

then the two trees in Fig 3 may have different generation probabilities, because at depth 1 ($n = 1$) the $expr$ at location 2 will be rewritten as $expr\ op\ expr$ with probability $p2$ while at location 0, it will be rewritten with probability $p0$.

Note that in PEEL, Left Hand Side (LHS) matching is partial matching. For example in the left tree in Fig 3, the complete location of the right-most $expr$ at depth 1 is (0 2). The rule:

```
expr(n,l):(n=1 and l=2) = expr op expr
```

can match even if its predecessor location is $l = 2$. In these circumstances, we start matching from the right of the complete location string since it is the closest (or most relevant) information. Therefore, in this example, for the complete location (0 2), both $l = 2$ and $l = 02$ can match, but not $l = 20$.

The key idea of grammar refinement used here, is to split the production rule using more location information. Two

steps are needed to refine the grammar: 1) detect the *most overloaded Left Hand Side (LHS)*, 2) split the most overloaded LHS.

The overloaded LHSs are those that are both highly influential and badly converged. More formally, we calculate the *burden* for each LHS in the grammar. The LHS with the largest burden is the most overloaded LHS. The burden of LHS L is defined as.

$$burden_L = (entropy_L)^a \times (pheromone_L)^b$$

The entropy of LHS L measures the convergence of this L, while the pheromone reflects the importance. a and b are system parameters used to balance the influence of the two terms. In this initial work, we set $a = b = 1$.

For example, the burden of LHS L with predecessor $expr$ and condition $n = 0$ and $l = \#$ in Fig 2 can be calculated as follows.

$$\begin{aligned} probability_L^i =& \frac{pheromone_L^i}{\sum_{i=1}^{3} pheromone_L^i} = \frac{1}{(1+1+1)} = \frac{1}{3} \\ burden_L =& (\sum_{i=1}^{3}(-probability_L^i \times \log probability_L^i))^a \\ & \times (\sum_{i=1}^{3} pheromone_L^i)^b \\ =& (-\frac{1}{3}\log_2\frac{1}{3} - \frac{1}{3}\log_2\frac{1}{3} - \frac{1}{3}\log_2\frac{1}{3}) \\ & \times (1+1+1) \approx 4.75 \end{aligned}$$

where $pheromone_L^i$ is the pheromone of i-th rule which has the same LHS L, i.e. has same predecessor $expr$ and satisfies condition $n = 0$ and $l = \#$, $probability_L^i$ is the corresponding probability, which is the normalized pheromone values.

Once the location information is added, one production can be split into several. For example, the following rule

```
expr(n,l):(n=1 and l=#) = expr op expr (ph:p)
```

might be split into three rules according to the grammar in Fig 1 :

```
expr(n,l):(n=1 and l=0) = expr op expr (ph:p)
expr(n,l):(n=1 and l=1) = expr op expr (ph:p)
expr(n,l):(n=1 and l=2) = expr op expr (ph:p)
```

Thus, initially the new rules each have the same pheromone as the original one. However, after further learning, i.e., updating the pheromone, these three rules might have different probabilities. Note that the location information is added incrementally – each time, only the minimum location information is added.

As can be seen, PEEL is an incremental learning approach. It starts from a minimum SSDT which describes the search space very roughly. Then, when necessary, rules are added to the SSDT to focus the search to a finer level.

3.5 Mutation of the SSDT

A key exploration mechanism of PEEL is mutation of the SSDT. The active rules which were used to generate the elitists (the best individuals found so far) and those rules which have the same LHS as these active rules, are mutated with probability $p_{mutation}$. There is a risk that the number of rules mutated might increase too fast as the size of individuals increases, so $p_{mutation}$ depends inversely on the total number of rules which are used for elitists:

$$p_{mutation} = c_{mutation}/n_{rules}$$

where $c_{mutation}$ is a predefined *mutation coefficient*, and n_{rules} is the number of rules that are to be mutated.

In this paper, the purpose of mutation is to allow the search to escape from the areas of previously learned knowledge, and thus explore different parts of the search area. Consequently, our mutation will force converged LHSs to partially lose their convergence, thus encouraging exploration. The mutation equation is defined as:

$$\begin{aligned} pheromone_i =& pheromone_i \\ & + mi \times (pheromone_i - pheromone_{average}) \end{aligned}$$

where $pheromone_i$ is the pheromone of $rule_i$ which is being mutated, mi is a predefined *mutation intensity* and $pheromone_{average}$ is the average pheromone of all rules which have the same LHS as $rule_i$.

4 Experiments

Two experiments have been conducted to test the performance of the PEEL. The context free grammar used by both experiments can be found in Figure 1.

4.1 Function regression

Function regression is often used in the GP literature for evaluating the performance of algorithms. In this work, the function to be approximated is adopted from [21], the function is:

$$f(x) = x^3 \times e^{-x} \times cos(x) \times sin(x) \times ((sin(x))^2 \times cos(x) - 1)$$

This complicated function is chosen to prevent the algorithm from simply guessing it. The following results compare PEEL and Grammar Guided Genetic Programming (GGGP). The results of PIPE in [21] over the same problem will also be briefly mentioned. However it is almost impossible to make a fair comparison, because the results given in [21] are in diagram format only, and in any case, PIPE adds ephemeral random constants while PEEL does not.

Algorithm	mean	std. dev.	max	median	min
PEEL	6.79	4.91	18.90	5.21	0.68
GGGP	7.87	3.54	14.00	7.56	0.95

Table 1: Comparison of PEEL and GGGP

The fitness cases were sampled at 101 equidistant points in the interval [0,10], 100,000 program evaluations were set for both PEEL and GGGP, the same as in [21]. The other settings for PEEL were population size 10, maximum depth 15, mutation coefficient $c_{mutation} = 0.5$, mutation intensity $mi = 0.2$, decay rate $decay\_rate = 0.95$, and the SSDT was refined every 50 generations, number of elitists 3. The settings for GGGP were population size 2000, maximum depth 15, crossover rate 0.9, mutation rate 0.1, tournament size 3. Thirty runs were generated for both PEEL and GGGP. The errors (sum of absolute value of error) of the best individual in each run are summarized in Table 1.

As can be seen from Table 1, when considering the mean, median and the minimum statistics, PEEL gets much better results than GGGP although both the standard deviation and range of results of PEEL is larger than GGGP. We also conducted a one-tailed student *t-test* to see whether PEEL significantly outperformed GGGP. However, the significance level is only at 0.2, i.e., in practice the results are not statistically significant.

Comparing PEEL with the results of PIPE and GP in [21], we find that PEEL is not worse than GP and PIPE in terms of mean of fitness of the best individuals, even though we use less tree depth and have no ephemeral constants in PEEL. However, PEEL with a smaller range of results is more stable than PIPE. Overall, in terms of the general accuracy, on this specific problem, PEEL has roughly similar performance to PIPE.

4.2 Time series predication

For this problem, we used the sunspot benchmark series for years 1700-1979 (The data can be obtained from ftp://ftp.santafe.edu/pub/Time-Series/data/). It contains 280 data points, divided into three subsets: the data points from years 1700-1920 are for training, while 1921-1955 and 1956-1970 are for testing.

The task of prediction is to map data points from lag space to an estimate of the future value:

$$\hat{x}_{t+1} = f(x_t, x_{t-1}, \ldots, x_{t-\delta})$$

where δ is the order of the model, which is set to 5 in this work, and t is the time index.

The accuracy of the model is measure by Average Relative Variance (ARV) [22]. Given data set S, ARV is defined as:

$$\begin{aligned} ARV(S) =& \frac{\sum_{t\in S}(target_t - prediction_t)^2}{\sum_{t\in S}(target_t - mean)^2} \\ =& \frac{1}{\sigma^2}\frac{1}{T}\sum_{t\in S}(x_t - \hat{x}_t)^2 \end{aligned}$$

where $target_t$ or x_t are the actual values, $prediction_t$ or $\hat{x}_t$ are the predicted values, N is the total number of data points, and σ is the variance of the total dataset (i.e. from year 1700-1979), it is set to 0.041056, the same as in [11].

Fifteen PEEL runs were generated using the same settings as in the function regression experiment, except that the number of programs generated was set to 600000 to make it comparable with the GP in [11]. The results are summarized in Table 2. The first row is the statistics of 15 PEEL runs. The second row is the best individual among all PEEL runs, in terms of ARV on the training data set, this best individual, in postfix notation, can be found in Figure 4. The third row is the GP result from [11]. The fourth row is the best GP result. The fifth row shows the results of back-propagation neural networks from [22]. The last row shows whether the differences between PEEL and GP are significant according to a student *t-test* (at significance level 0.05). The right-most column, size, is the number of nodes in the best individuals.

According to Table 2, PEEL, compared to GP, does not fit the training data set closely; however PEEL generalizes better. Note that the GP in [11] was carefully designed to alleviate overfitting, while neither such measures nor explicit parsimony pressure were used in PEEL. Given that, where does the parsimony pressure in PEEL come from? It seems that PEEL has a built-in Occam's Razor because of its probabilistic nature. This means that larger structures have less chance of preservation than smaller ones. This explains why the PEEL results tend to be much more compact than those from GP. As can be seen from Table 2, the average size of the best PEEL individuals is only around 21.9 nodes while GP's is as big as 66.5. These results are consistent with the general belief and theoretical understanding that compact solutions usually have better generalization performance. This is very clear from Table 2, which shows that while PEEL is worse on the training data than GP, it has similar performance on the testing data 1921-1955 and significantly outperforms GP on the testing data 1956-1979. Actually the best individual found by PEEL generalizes much better on the data set 1956-1979 than any other method cited in [11]. Much research has been done on parsimony pressure and control of individual complexity in GP [18, 9, 24]. It is notable that, in our work, although no complicated parsimony mechanism is used, the results still maintained impressively low complexity. Furthermore we did not observe any obvious manifestations of pre-mature

```
x6 x2 - x1 x2 / x4 + cos sin x2 - * e^ cos
x3 x2 sin + + x1 /
```

Figure 4: Best program found by PEEL on sunspot prediction

convergence, a perennial problem in GP with parsimony mechanisms. Although more accurate models of predicting sunspots have been obtained by GP [15], that method is highly specialized for non-linear function fitting, which lessens its usefulness for comparative purposes.

Algorithm	Training 1700-1920	Generalization 1921-1955	Generalization 1956-1979	size
PEEL	0.137±0.016	0.185±0.039	0.291±0.071	21.9±4.3
PEEL(best)	0.115	0.150	0.184	22
GP	0.125±0.006	0.182±0.037	0.37±0.06	66.5
GP(best)	0.118	0.127	0.329	60
BP-NN	0.082	0.086	0.35	
Significant	Yes	No	Yes	

Table 2: Comparison of PEEL and other methods on sunspot prediction

5 Conclusion and Future Work

Essentially, PEEL is a probabilistic approach which models the dependency among nodes. The SSDT represents a kind of restricted form of dependency tree. However, unlike Balujua's dependency tree [5] where the structure and probabilities are both learned from good individuals, the huge size of maximum individuals (in terms of number of nodes) simply prohibits the construction of a Balujua-style dependency tree. To permit this kind of learning, the SSDT fixes the structure and learns only the dependencies within this structure, i.e. given a specific location and parent node (i.e. the left hand side of rule), the probability of choosing a specific right hand side is specified. Although the SSDT models the most fundamental dependencies in the grammar constrained search space, it may not be sufficient for some problems. A more flexible dependency structure might model the dependency more fully, and therefore lead to better performance. This is one of our major future research issues. The SSDT also differs from pair-wise dependency models [5, 21] in that the dependency modeled can be higher order, because of the context/location information. In the SSDT, the probability of choosing a specific symbol depends not only on its immediate parent node, but also on its location or context. The expansion of the SSDT through grammar refinement also makes PEEL very different from PIPE.

In conclusion, Program Evolution with Explicit Learning (PEEL) is a novel approach for automatic program synthesis. The Search Space Description Table (SSDT) is employed to explicitly characterize fitter individuals and to thus guide the search process. Grammar refinement helps to make the SSDT more expressive, so that the search can be focused on a small promising search area. On the problems we have tested, PEEL shows at least comparable performance with GGGP and PIPE. However more analysis and experiments are required before we can draw general conclusions about the performance of PEEL.

In addition to the previously mentioned research issues, future work and research issues include the following:

- In this research, relative coordinates are used to encode location information for grammar refinement. Direct context information could also be used for this purpose.
- In this paper, the grammar is refined at fixed time steps. Clearly, this would be better done adaptively, and more research is needed into appropriate heuristics for the timing of grammar refinement.
- The learning method in this work is a variant of ACO. However there is no intrinsic difficulty in replacing it with other learning methods, such as ANN, Bayesian networks or rule induction.
- Identifying building blocks and incrementally incorporating them into the grammar may improve the scalability and convergence rate of PEEL.
- Since the knowledge about the search space is directly encoded in the SSDT, it would be interesting to extract that knowledge from the SSDT directly, for example, inducing production rules which may describe promising search area.
- The structured SSDT might make theoretical analysis, such as convergence, possible.
- The development of a hybrid approach, combining GGGP and PEEL.

Acknowledgment

Thanks to Dr. Kasper Peeters for his tree.hh, an STL-like C++ tree class.

Bibliography

[1] H. A. Abbass, N. X. Hoai, and R. I. McKay. AntTAG: A new method to compose computer programs using colonies of ants. In *The IEEE Congress on Evolutionary Computation*, 2002.

[2] P. J. Angeline. Two self adaptive crossover operations for genetic programming. In *Advances in Genetic Programming II*, pages 89–110. MIT Press, 1995.

[3] P. J. Angeline and J. B. Pollack. Coevolving high-level representations. In C. G. Langton, editor, *Artificial Life III*, volume XVII, pages 55–71, Santa Fe, New Mexico, 1994. Addison-Wesley.

[4] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Pittsburgh, PA, 1994.

[5] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In D. H. Fisher, editor, *Proc. 1997 International Conference on Machine Learning*, Tennessee, USA, 1997. Morgan Kaufmann.

[6] N. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proc. of an Intl. Conf. on Genetic Algorithms and their Applications*, pages 183–187, Carnegie-Mellon University, 1985.

[7] P. D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 1994. IEEE Press.

[8] B. E., M. Dorigo, and T. Theraulaz. *From Natural to Artificial Swarm Intelligence.* Oxford University Press, New York, 1999.

[9] H. Iba, H. de Garis, and T. Sato. Genetic programming using a minimum description length principle. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 265–284. MIT Press, 1994.

[10] H. Iba and H. D. Garis. Extending genetic programming with recombinative guidance. In *Advances in Genetic Programming II*. MIT Press, 1996.

[11] H. Jäske. One-step-aheah prediction of sunspots with genetic programming. In *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applicatios (2NWGA)*, Vaasa Finland, August 1996.

[12] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[13] R. S. Michalski. Learnable evolution model:evolutionary processes guided by machine learning. *Machine Learning*, 38:9–40, 2000.

[14] H. Müehlenbein and G. Paaß. From recombination of genes to the estimation of distributions i.binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV*, pages 178–187. 1996.

[15] N. Y. Nikolaev and H. Iba. Regularization approach to inductive genetic programming. *IEEE Transactions on Evolutionary Computation*, 2001.

[16] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. Technical Report IlliGAL Report No. 99018, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Sept 1999.

[17] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, 1990.

[18] J. P. Rosca. Analysis of complexity drift in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286–294, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[19] J. P. Rosca and D. H. Ballard. Genetic programming with adaptive representations. Technical Report TR 489, University of Rochester, Computer Science Department, Rochester, NY, USA, Feb. 1994.

[20] B. J. Ross. Logic-based genetic programming with definite clause translation grammars. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1236, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[21] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

[22] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting sunspots and exchange rates with connectionist networks. In *Nonlinear Modelling and Forecasting*, pages 395–432. Addison-Wesley, 1992.

[23] P. A. Whigham. Grammatically-based genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 1995.

[24] B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.