Finding Perceived Pattern Structures using Genetic Programming

Mehdi Dastani
Dept. of Mathematics
and Computer Science
Free University Amsterdam
The Netherlands
email: mehdi@cs.vu.nl

Elena Marchiori
Dept. of Mathematics
and Computer Science
Free University Amsterdam
The Netherlands
email: elena@cs.vu.nl

Robert Voorn
Dept. of Mathematics
and Computer Science
Free University Amsterdam
The Netherlands
email: rbvoorn@cs.vu.nl

Abstract

Structural information theory (SIT) deals with the perceptual organization, often called the 'gestalt' structure, of visual patterns. Based on a set of empirically validated structural regularities, the perceived organization of a visual pattern is claimed to be the most regular (simplest) structure of the pattern. The problem of finding the perceptual organization of visual patterns has relevant applications in multi-media systems, robotics and automatic data visualization. This paper shows that genetic programming (GP) is a suitable approach for solving this problem.

1 Introduction

In principle, a visual pattern can be described in many different ways; however, in most cases it will be perceived as having a certain description. For example, the visual pattern illustrated in Figure 1-A may have, among others, two descriptions as they are illustrated in Figure 1-B and 1-C. Human perceivers prefer usually the description that is illustrated in Figure 1-B. An empirically supported theory of visual perception is the Structural Information Theory (SIT) [Leeuwenberg, 1971, Van der Helm and Leeuwenberg, 1991,
Van der Helm, 1994]. SIT proposes a set of empirically validated and perceptually relevant structural regularities and claims that the preferred description of a visual pattern is based on the structure that covers most regularities in that pattern. Using the formalization of the notions of *perceptually relevant structure* and *simplicity* given by SIT, the problem of finding the simplest structure of a visual pattern (SPS problem) can be formulated mathematically as a constrained optimization problem.

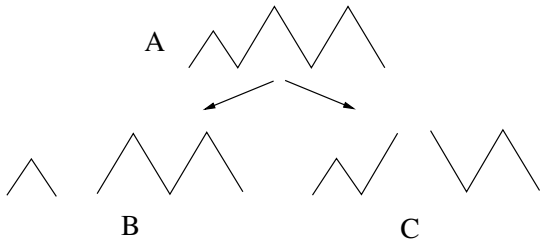


Figure 1: *Visual pattern A has two potential structures B and C.*

The SPS problem has relevant applications. For example, multimedia systems and image databases need to analyze, classify, and describe images in terms of constitutive objects that human users perceives in those images [Zhu, 1999]. Furthermore, autonomous robots need to analyze their visual inputs and construct hypotheses about possibly present objects in their environments [Kang and Ikeuchi, 1993]. Also, in the fields of information visualization the goal is to generate images that represent information such that human viewers extract that information by looking at the images [Bertin, 1981]. In all these applications, a model of gestalt perception is indispensable [Mackinlay, 1986, Marks and Reiter, 1990]. We focus on a simple domain of visual patterns and claim that an appropriate model of gestalt perception for this domain is an essential step towards a model of gestalt perception for more complex visual patterns that are used in the above mentioned real-world applications [Dastani, 1998].

Since the search space of possible structures grows exponentially with the complexity of the visual pattern, heuristic algorithms have to be used for solving the SPS problem efficiently. The only algorithm for SPS we are aware of is developed by [Van der Helm and Leeuwenberg, 1986]. This algo-

rithm ignores the important source of computational complexity of the problem and covers only a subclass of perceptually relevant structures. The central part of this partial algorithm consists of translating the search for a simplest structure into a shortest route problem. The algorithm is shown to have $O(N^4)$ computational complexity, where N denotes the length of the input pattern. To cover all perceptually relevant structures for not only the domain of visual line patterns, but also for more complex domains of visual patterns, it is argued in [Dastani, 1998] that the computational complexity grows exponentially with the length of the input patterns.

This paper shows that genetic programming [Koza, 1992] provides a natural paradigm for solving the SPS problem using SIT. A novel evolutionary algorithm is introduced whose main features are the use of SIT operators for generating the initial population of candidate structures, and the use of knowledge based genetic operators in the evolutionary process. The use of GP is motivated by the SIT formalization: structures can be easily described using the standard GP-tree representation. However, the GP search is constrained by the fact that structures have to characterize the same input pattern. In order to satisfy this constraint, knowledge based operators are used in the evolutionary process.

The paper is organized as follows. In the next section, we briefly discuss the problem of visual perception and explain how SIT predicts the perceived structure of visual line patterns. In Section 3, SIT is used to give a formalization of the SPS problem for visual line patterns. Section 4 describes how the formalization can be used in an automatic procedure for generating structures. Section 5 introduces the GP algorithm for SPS. Section 6 describes implementation aspects of the algorithm and reports some results of experiments. The paper concludes with a summary of the contributions and future research directions.

2 SIT: A Theory of Visual Perception

According to the structural information theory, the human perceptual system is sensitive to certain kinds of structural regularities within sensory patterns. They are called perceptually relevant structural regularities, which are specified by means of ISA operators: *Iteration, Symmetry* and *Alternations* [Van der Helm and Leeuwenberg, 1991]. Examples of string patterns that can be specified by these operators are *abab*, *abcba*, and *abgabpz*, respectively. A visual pattern can be described in different ways by applying different ISA operators. In order to disambiguate the set of descriptions and to decide on the perceived organization of the pattern, a simplicity measure, called *information load*, is introduced. The information load measures the amount of perceptually relevant regularities covered by pattern descriptions. It is claimed that the description of a visual pattern with the minimum information load reflects its perceived organization [Van der Helm, 1994].

In this paper, we focus on the domain of *linear line patterns* which are turtle-graphics, like line drawings for which the turtle starts somewhere and moves in such a way that the line segments are connected and do not cross each other. A linear line pattern is encoded as a letter string for which it can be shown that its simplest description represents the perceived organization of the encoded linear line pattern [Leeuwenberg, 1971]. The encoding process consists of two steps. In the first step, the successive line segments and their relative angles in the pattern are traced from the starting point of the pattern and identical letter symbols are assigned to identical line segments (equal length) as well as to identical angles (relative to the trace movement). In the second step, the letter symbols that are assigned to line segments and angles are concatenated in the order they have been visited during the trace of the first step. This results in a letter string that represents the pattern. An example of such an encoding is illustrated in Figure 2.

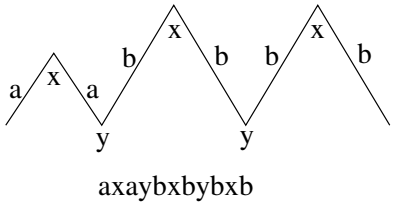


Figure 2: *Encoding of a line pattern into a string.*

Note that letter strings are themselves perceptual patterns that can be described in many different ways, one of which is usually the perceived description. The determination of the perceived description of string patterns is the essential focus of Hofstadter's Copycat project [Hofstadter, 1984].

3 The SPS Problem

In this section, we formally define the class of string descriptions that represent possible perceptually relevant organizations of linear line patterns. Also, a complexity function is defined that measures the information load of those descriptions. In this way, we can en-

code a linear line pattern into a string, generate the perceptually relevant descriptions of the string, and determine the perceived organization of the line pattern by choosing the string description which has the minimum information load.

The class of descriptions that represent possible perceptual organizations for Linear Line Patterns $\mathcal{LLP}$ is defined over the set $E = \{a, \ldots, z\}$ as follows.

1. For all $t \in E$, $t \in \mathcal{LLP}$
2. If $t \in \mathcal{LLP}$ and n is a natural number, then $iter(t, n) \in \mathcal{LLP}$
3. If $t \in \mathcal{LLP}$, then $symeven(t) \in \mathcal{LLP}$
4. If $t_1, t_2 \in \mathcal{LLP}$, then $symodd(t_1, t_2) \in \mathcal{LLP}$
5. If $t, t_1, \ldots, t_n \in \mathcal{LLP}$, then $altleft(t, < t_1, \ldots, t_n >) \in \mathcal{LLP}$ and $altright(t, < t_1, \ldots, t_n >) \in \mathcal{LLP}$
6. If $t_1, \ldots, t_n \in \mathcal{LLP}$, then $con(t_1, \ldots, t_n) \in \mathcal{LLP}$

The meaning of $\mathcal{LLP}$ expressions can be defined by the denotational semantics $[\![\ ]\!]$, which involves string concatenation ($\bullet$) and string reflection ($reflect(abcde) = edcba$) operators.

1. If $t \in E$, then $[\![t]\!] = t$
2. $[\![iter(t, n)]\!] = [\![t]\!] \bullet \ldots \bullet [\![t]\!]$ (n times)
3. $[\![symeven(t)]\!] = [\![t]\!] \bullet reflect([\![t]\!])$
4. $[\![symodd(t_1, t_2)]\!] = [\![t_1]\!] \bullet [\![t_2]\!] \bullet reflect([\![t_1]\!])$
5. $[\![altleft(t, < t_1, \ldots, t_n >)]\!] = [\![t]\!] \bullet [\![t_1]\!] \bullet \ldots \bullet [\![t]\!] \bullet [\![t_n]\!]$
6. $[\![altright(t, < t_1, \ldots, t_n >)]\!] = [\![t_1]\!] \bullet [\![t]\!] \bullet \ldots \bullet [\![t_n]\!] \bullet [\![t]\!]$
7. $[\![con(t_1, \ldots, t_n)]\!] = [\![t_1]\!] \bullet \ldots \bullet [\![t_n]\!]$

The *complexity function* C on $\mathcal{LLP}$ expressions, measures the complexity of an expression as the number of individual letters t occurring in it, i.e.

$C(t) = 1$
$C(f(T_1, \ldots, T_n)) = \sum_{i=1}^{n} C(T_i)$

During the last 20 years, Leeuwenberg and his co-workers have reported on a number of experiments that tested predictions based on the simplicity principle. These experiments were concerned with the disambiguation of ambiguous patterns. The predictions of the simplicity principle were, on the whole, confirmed by these experiments [Buffart et al., 1981, Van Leeuwen et al., 1988, Boselie and Wouterlood, 1989].

The following $\mathcal{LLP}$ expressions describe, among others, four different perceptual organizations of the pattern *axaybxbybxb*:

- $con(a, x, a, y, b, x, b, y, b, x, b)$,
- $con(symodd(a, x), y, symodd(b, x), y, symodd(b, x))$
- $con(symodd(a, x), iter(con(y, b, x, b), 2))$
- $con(symodd(a, x), iter(altright(b, < y, x >), 2))$

Note that these descriptions reflect four different perceptual organizations of the line pattern that is illustrated in Figure 2. The information load of these four descriptions are $11, 8, 6$, and 5, respectively. This implies that the last description reflects the perceived organization of the line pattern illustrated in Figure 2.

The *SPS problem* can now be defined as follows. Given a pattern p, find a $\mathcal{LLP}$ expression t such that

- $[\![t]\!] = p$ and
- $C(t) = \min\{C(s) \mid s \in \mathcal{LLP} \text{ and } [\![s]\!] = p\}$.

As mentioned in the introduction, the only (partial) algorithm for solving SPS problem is proposed by Van der Helm [Van der Helm and Leeuwenberg, 1986]. This algorithm finds only a subclass of perceptually relevant structures of string patterns by first constructing a directed acyclic graph for the given string pattern. If we place an index after each element in the string pattern, starting from the leftmost element, then each node in the graph would correspond to an index, and each link in the graph from node i to j corresponds to a gestalt for the subpattern starting at position i and ending at position j. Given this graph, the SPS problem is translated to a shortest route problem. Note that this algorithm is designed for one-dimensional string patterns and it is not clear how this algorithm can be applied to other domains of perceptual patterns. Instead, our formalization of the SPS problem can be easily applied to more complex visual patterns by extending the $\mathcal{LLP}$ with domain dependent operators such as Euclidean transformations for two-dimensional visual patterns [Dastani, 1998].

4 Generating $\mathcal{LLP}$ Expressions

In order to solve the SPS problem using genetic programming, a probabilistic procedure for generating $\mathcal{LLP}$ expressions, called BUILD-STRUCT, is used. This procedure takes as input a string, and generates a (tree structure of a) $\mathcal{LLP}$ expression for that string. The procedure is based on a set of probabilistic production rules.

The production rules are derived from the SIT definition of expressions, and are of the form
$\alpha \;\; t_1 \ldots t_n \; \beta \longrightarrow \alpha \;\; P(t_1 \ldots t_n) \; \beta$

where α and β are (possibly empty) $\mathcal{LLP}$ expressions, $t_1, \ldots, t_n$ are $\mathcal{LLP}$ expressions, and P is an ISA operator (of arity n). The triple $(\alpha, \;\; t_1 \ldots t_n, \;\; \beta)$ is called *splitting* of the sequence.

A snapshot of the set of production rules used in BUILD-STRUCT is given below.

$\alpha \; t \; t \; \beta \longrightarrow \alpha \; iter(t,2) \; \beta$
$\alpha \; t \; iter(t,n) \; \beta \longrightarrow \alpha \; iter(t,n+1) \; \beta$
$\alpha \; iter(t,n) \; t \; \beta \longrightarrow \alpha \; iter(t,n+1) \; \beta$
$\alpha \; t_1 \; t_2 \; \beta \longrightarrow \alpha \; con(t_1,t_2) \; \beta$
$\alpha \; con(t_1,..,t_n) \; t \; \beta \longrightarrow \alpha \; con(t_1,..,t_n,t) \; \beta$
$\alpha \; t \; con(t_1,..,t_n) \; \beta \longrightarrow \alpha \; con(t,t_1,..,t_n) \; \beta$

A production rule transforms a sequence of $\mathcal{LLP}$ expressions into a shorter one. In this way, the repeated application of production rules terminates after a finite number of steps and produces one $\mathcal{LLP}$ expression. There are two forms of non-determinism in the algorithm:

1. the choice of which rule to apply when more than one production rule is applicable,
2. the choice of a splitting of the sequence when more splittings are possible.

In BUILD-STRUCT both choices are performed randomly. BUILD-STRUCT employs a specific data structure which results in a more efficient implementation of the above described non-determinism. The BUILD-STRUCT procedure is used in the initialization of the genetic algorithm and in the mutation operator.

We conclude this section with an example illustrating the application of the production rules system. The $\mathcal{LLP}$ expression $iter(con(a,b,a),2)$ can be obtained using the above production rules starting from the pattern *abaaba* as follows, where an underlined substring indicates that an ISA operator will be applied to that substring:

$\underline{aba}\; aba \longrightarrow con(a,b,a)aba$
$con(a,b,a)\; \underline{aba} \longrightarrow con(a,b,a)con(a,b,a)$
$\underline{con(a,b,a)con(a,b,a)} \longrightarrow iter(con(a,b,a),2)$

Note in this example that the *iter* operator is applied to two structurally identical $\mathcal{LLP}$ expressions (i.e. $\underline{con(a,b,a)con(a,b,a)} \longrightarrow iter(con(a,b,a),2)$). In general, the ISA operators are not applied on the basis of structural identity of $\mathcal{LLP}$ expressions, but on the basis of their semantics, i.e. on the basis of the patterns that are denoted by the $\mathcal{LLP}$ expressions (i.e. $\underline{symodd(a,b)con(a,b,a)} \longrightarrow iter(symodd(a,b),2)$).

5 A GP for the SPS Problem

This section introduces a novel evolutionary algorithm for the SPS problem, called GPSPS (Genetic Programming for the SPS problem), which applies GP to SIT. A population of $\mathcal{LLP}$ expressions is evolved, using knowledge based mutation and crossover operators to generate new expressions, and using the SIT complexity measure as fitness function. GPSPS is an instance of the generational scheme, cf. e.g. [Michalewicz, 1996], illustrated below, where $P(t)$ denotes the population at iteration t and $|P(t)|$ its size.

```
PROCEDURE GPSPS
t <- 0
initialize P(t)
evaluate P(t)
 WHILE (NOT termination condition) DO
  BEGIN
   t <- t+1
   WHILE (|P(t)|<|P(t-1)|) DO
     BEGIN
     select two elements from P(t-1)
     apply crossover
     apply mutation
     insert in P(t)
     END
  END
END
```

We have used the Roulettewheel mechanism to select the elements for the next generation. Therefore the chance that an element of the original pool is selected is proportional to its fitness. Since we apply our system to a minimization problem, the fitness function has to be transformed. This is done with the function $newF(element) = maxF(pool) - F(element)$. This ensures that the element with the lowest fitness will

have the highest probability of being selected. We have also made our GP elitist to guarantee that the best element found so far will be in the actual population.

The main features of GPSPS are described in the rest of this section.

5.1 Representation and Fitness

GPSPS acts on $\mathcal{LLP}$ expressions describing the same string. A $\mathcal{LLP}$ expression is represented by means of a tree in the style used in Genetic Programming, where leaves are primitive elements while internal nodes are ISA operators. The fitness function is the complexity measure C as it is introduced in Section 3.

Thus, the goal of GPSPS is to find a chromosome (representing a structure of the a given string) which minimizes C. Given a string, a specific procedure is used to ensure that the initial population contains only chromosomes describing the same pattern. Moreover, novel genetic operators are designed which preserve the semantics of chromosomes.

5.2 Initialization

Given a string, chromosomes of the intial population are generated using the procedure BUILD-STRUCT. In this way, the initial population contains randomly selected (representations of) $\mathcal{LLP}$ expressions of the pattern.

5.3 Mutation

When the mutation operator is applied to a chromosome T, an internal node n of T is randomly selected and the procedure BUILD-STRUCT is applied to the (string represented by the) subtree of T starting at n. Figure 3 illustrates an application of the mutation operator to an internal node. Observe that each node (except the terminals) has the same chance of being selected. In this way smaller subtrees have a larger chance of being modified.

It is interesting to investigate the effectiveness of the heuristic implemented in BUILD-STRUCT when incorporated into an iterated local search algorithm. Therefore we have implemented an algorithm that mutates one single element for a large number of iterations and returns the best element that has been found over all iterations. Although some regularities are discovered by this algorithm, its performance is rather scarce if compared with GPSPS, even when the number of iterations is set to be bigger than the size of the population times the number of generations used by GPSPS.

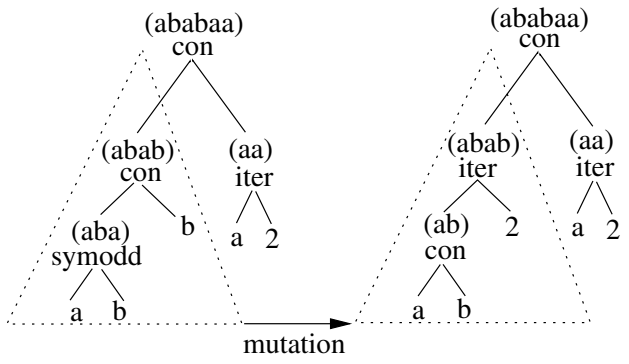


Figure 3: *Example of the mutation-operator.*

5.4 Crossover

The crossover operator cannot simply swap subtrees between two parents, like in standard GP, due to the semantic constraint on chromosomes (e.g. chromosomes have to denote the same string). Therefore, the crossover is designed in such a way that it swaps only subtrees that denote the same string. This is realized by associating with each internal node of the tree the string that is denoted by the subtree starting at that internal node. Then, two nodes of the parents with equal associated strings are randomly selected and the corresponding subtrees are swapped. An example of crossover is illustrated in Figure 4.

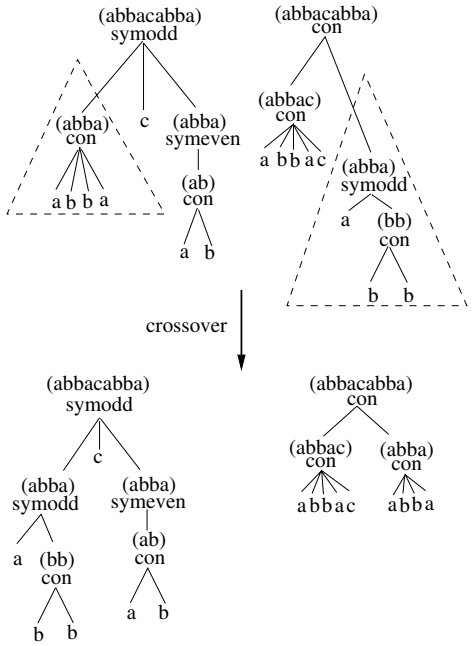


Figure 4: *Example of the crossover-operator.*

When a crossover-pair can not be found, no crossover takes place. Fortunately this happens only for a small portion of the crossovers. Usually there are more than one pair to choose from. This issue is further discussed in the next section.

5.5 Optimization

As discussed above, the mutation and crossover operators transform subtrees. When these operators are applied, the resulting subtrees may exhibit structures of a form suitable for optimization. For instance, suppose a subtree of the form $con(iter(b,2),a,con(b,b))$ is transformed by one of the operators in the subtree $con(iter(b,2),a,iter(b,2))$. This improves the complexity of the subtree. Unfortunately, based on this new subtree the expected $\mathcal{LLP}$ expression $symodd(iter(b,2),a)$ cannot be obtained.

The crossover operator is only helpful for this problem if there is already a subtree that encodes that specific substring with an *symodd* structure. This problem could in fact be solved by applying the mutation operator to the *con* structure. However, the probability that the application of the mutation operator will generate the *symodd* structure is small.

In order to solve this problem, a simple optimization procedure is called after each application of the mutation and crossover operators. This procedure uses simple heuristics to optimize the *con* structure. First, the procedure checks if the (entire) *con* structure is symmetrical and changes it into a *symodd* or *symeven* structure if possible. If this is not the case, the procedure checks if neighboring structures that are similar can be combined. For example, a structure of the form $con(c,iter(b,2),iter(b,3))$ can be optimized to $con(c,iter(b,5))$. This kind of optimization is also applied to $altleft$ and $altright$ structures.

6 Experiments

In this section we discuss some preliminary experiments. The example strings we consider are short and are designed to illustrate what type of structures are interesting for this domain. The choice of the values of the GP parameters used in the experiments is determined by the considered type of strings. Because the strings are short, a small pool size of 50 individuals is used. Making the size of the pool very large would make the GP perform better, but when the pool is initialized, it would probably already contain the most preferred structure. The number of iterations is also small to avoid generating all possible structures and is therefore set to 150. This allows us to draw preliminary conclusions about the performance of the GP.

Two important parameters of the GP are the mutation and crossover rates. We have done a few test runs to find a setting that produced good results. We have set the mutation-rate on 0.6 and the crossover-rate to 0.4. The mutation is deliberately set to a higher rate, because this operator is the most important for discovering structures. The crossover operator is used to swap substructures between good chromosomes.

We have chosen six different short strings that contain structures that are of interest to our search problem. Moreover, two longer strings are considered. For the two long strings the mutation and crossover rates above specified are used, but the poolsize and the number of generations are both set to 300. The eight strings are the code for the linear line patterns illustrated in Figure 5.

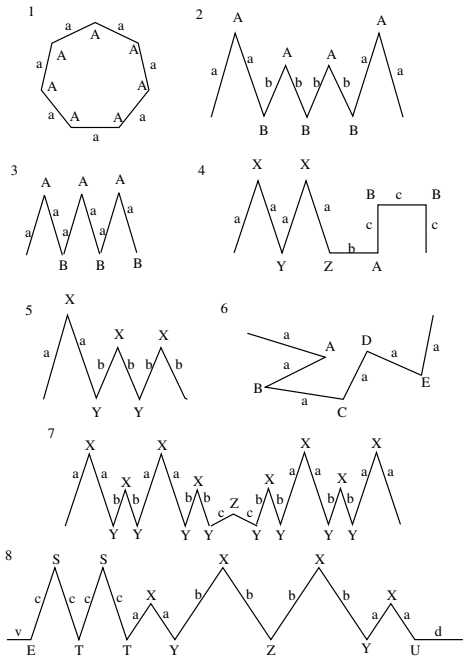


Figure 5: *Line drawings used in experiments.*

The algorithm is run on each string a number of times using different random seeds. The resulting structures are given in Figure 7, where the structure and fitnesses of the two best elements of the final population are reported. For each string GPSPS is able to find the optimal structure. The results of runs with different seeds are very similar, indicating the (expected) robustness of the algorithm on these strings.

Figure 6 illustrates how the best fitness and the mean fitness of the population vary in a typical run of GP-

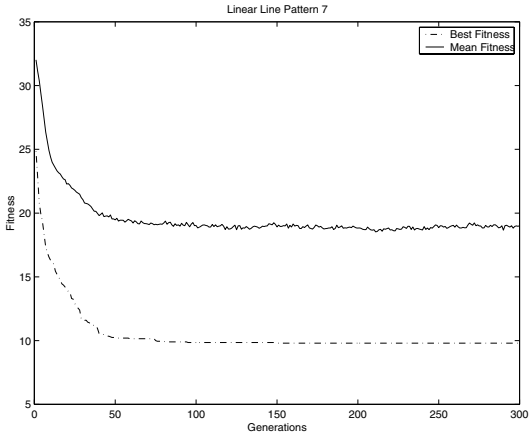


Figure 6: *Best and Mean Fitness.*

SPS on the line pattern number 7 of Figure 5. On this pattern, the algorithm is able to find a near optimum of rather good quality after about 50 generations, and it spends the other 250 generations to find the slighly improved structure. In this experiment about 12% of the crossovers failed. On average there were about 2.59 possible 'crossover-pairs' possible (with a standard deviation of 1.38) when the crossover operator was applicable.

The structures that are found are the most preferred structures as predicted by the SIT theory. The system is thus capable of finding the perceived organizations for these line drawings patterns.

7 Conclusion and Future Research

This paper discussed the problem of human visual perception and introduced a formalization of a theory of visual perception, called SIT. The claim of SIT is to predict the perceived organization of visual patterns on the basis of the simplicity principle. It is argued that a full computational model for SIT is computationally intractable and that heuristic methods are needed to compute the perceived organization of visual patterns.

We have applied genetic programming techniques to this formal theory of visual perception in order to compute the perceived organization of visual line patterns. Based on perceptually relevant operators from SIT, a pool of alternative organizations of an input pattern is generated. Motivated by SIT, mutation and crossover operations are defined that can be applied to these organizations to generate new organizations for the input pattern. Finally, a fitness function is defined that determines the appropriateness of generated organizations. This fitness function is directly derived from SIT and measures the simplicity of organizations.

In this paper, we have focused on a small domain of visual linear line patterns. The next step is to extend our system to compute the perceived organization of more complex visual patterns like two-dimensional visual patterns, which are defined in terms of a variety of visual attributes such as color, size, position, texture, shape.

Finally, we intend to investigate whether the class of structural regularities proposed by SIT is also relevant for finding meaningful organizations within patterns from biological experiments, like DNA sequences. For this task, we will need to modify GPSPS in order to allow a group of letters to be treated as a primitive element.

References

[Bertin, 1981] Bertin, J. (1981). *Graphics and Graphic Information-Processing*. Walter de Gruyter, Berlin NewYork.

[Boselie and Wouterlood, 1989] Boselie, F. and Wouterlood, D. (1989). The minimum principle and visual pattern completion. *Psychological Research*, 51:93–101.

[Buffart et al., 1981] Buffart, H., Leeuwenberg, E., and Restle, F. (1981). Coding theory of visual pattern completion. *Journal of Experimental Psychology: Human Perception and Performance*, 7:241–274.

[Dastani, 1998] Dastani, M. (1998). Ph.D. thesis, University of Amsterdam, The Netherlands.

[Hofstadter, 1984] Hofstadter, D. (1984). The copycat project: An experiment in nondeterministic and creative analogies. In *A.I. Memo 755, Artificial Intelligence Laboratory*, Cambridge, Mass. MIT.

[Kang and Ikeuchi, 1993] Kang, S. and Ikeuchi, K. (1993). Toward automatic robot instruction from perception: Recognizing a grasp from observation. In *IEEE Trans. on Robotics and Automation, vol. 9, no. 4*, pages 432–443.

[Koza, 1992] Koza, J. (1992). *Genetic Programming*. MIT Press.

[Leeuwenberg, 1971] Leeuwenberg, E. (1971). A perceptual coding language for visual and auditory patterns. *American Journal of Psychology*, 84:307–349.

[Mackinlay, 1986] Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. In *ACM Transactions on Graphics*, volume 5, pages 110–141.

[Marks and Reiter, 1990] Marks, J. and Reiter, E. (1990). Avoiding unwanted conversational implicatures in text and graphics. In *Proceeding AAAI*, Menlo Park, CA.

[Michalewicz, 1996] Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

[Van der Helm, 1994] Van der Helm, P. (1994). The dynamics of prägnanz. *Psychological Research*, 56:224–236.

[Van der Helm and Leeuwenberg, 1986] Van der Helm, P. and Leeuwenberg, E. (1986). Avoiding explosive search in automatic selection of simplest pattern codes. *Pattern Recognition*, 19:181–191.

[Van der Helm and Leeuwenberg, 1991] Van der Helm, P. and Leeuwenberg, E. (1991). Accessibility: A criterion for regularity and hierarchy in visual pattern code. *Journal of Mathematical Psychology*, 35:151–213.

[Van Leeuwen et al., 1988] Van Leeuwen, C., Buffart, H., and Van der Vegt, J. (1988). Sequence influence on the organization of meaningless serial stimuli: economy after all. *Journal of Experimental Psychology: Human Perception and Performance*, 14:481–502.

[Zhu, 1999] Zhu, S. (Nov, 1999). Embedding gestalt laws in markov random fields - a theory for shape modeling and perceptual organization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 21, No.11.

1	*string:* aAaAaAaAaAaAaA *structure:* a) iter(con(a,A),7) b) con(iter(con(a,A),2),iter(con(a,A),5)) *complexity* a) 2 b) 4
2	*string:* aAaBbAbBbAbBaAa *structure:* a) symodd(altleft(a,<A,con(B,symodd(b,A))>),B) b) symodd(con(symodd(a,A),altright(b,<B,A>)),B) *complexity* a) 6 b) 6
3	*string:* aAaBaAaBaAaB *structure:* a) iter(altleft(a,<A,B>),3) b) iter(con(symodd(a,A),B), 3) *complexity* a) 3 b) 3
4	*string:* aXaYaXaZbAcBcBc *structure:* a) altleft(symodd(a,X),<Y,altright(c,<con(Z,b,A),B,B>)) b) altleft(symodd(a,X),<Y, altright(c,<con(Z,b,A),symodd(B,c)>)) c) altleft(symodd(a,X),<Y,con(Z,b,A,c,iter(con(B,c),2))>) *scomplexity:* a) 9 b) 9 c) 9
5	*string:* aXaYbXbYbXb *structure:* a) altleft(a,<X,iter(con(Y,symodd(b,X)),2)>) b) altleft(a,<X,iter(altright(b,<Y,X>),2)>) *complexity:* a) 5 b) 5
6	*string:* aAaBaCaDaEa *structure:* a) altright(a,<altleft(a,<A,B>),C,D,E>) b) altleft(a,<A,B,C,D,con(E,a)>) *complexity:* a) 7 b) 7
7	*string:* axaybxbyaxaybxbyczcybxbyaxaybxbyaxa *structure:* a) symodd(con(iter(con(symodd(a,x), symodd(y,symodd(b,x))),2),c),z) b) symodd(con(iter(con(symodd(a,x), symodd(con(y,b),x)),2),c),z) *complexity:* a) 7 b) 7
8	*string:* vecsctcsctaxaybxbyzbxbyaxaud *structure:* a) con(v,altright(c,¡e,s¿),con(symodd(con(t,c),s), symodd(con(symodd(a,x),y,symodd(b,x)),z),u,d)) b) con(v,e,iter(altleft(c,¡s,t¿),2), symodd(con(symodd(a,x),y,symodd(b,x)),z),u,d) *complexity:* a) 13 b) 13

Figure 7: Results of experiments

Reducing Bloat and Promoting Diversity using Multi-Objective Methods

Edwin D. de Jong[1,2] **Richard A. Watson**[2] **Jordan B. Pollack**[2]
{edwin, richardw, pollack}@cs.brandeis.edu
[1]Vrije Universiteit Brussel, AI Lab, Pleinlaan 2, B-1050 Brussels, Belgium
[2]Brandeis University, DEMO Lab, Computer Science dept., Waltham, MA 02454, USA

Category: Genetic Programming

Abstract

Two important problems in genetic programming (GP) are its tendency to find unnecessarily large trees (bloat), and the general evolutionary algorithms problem that diversity in the population can be lost prematurely. The prevention of these problems is frequently an implicit goal of basic GP. We explore the potential of techniques from multi-objective optimization to aid GP by adding explicit objectives to avoid bloat and promote diversity. The even 3, 4, and 5-parity problems were solved efficiently compared to basic GP results from the literature. Even though only non-dominated individuals were selected and populations thus remained extremely small, appropriate diversity was maintained. The size of individuals visited during search consistently remained small, and solutions of what we believe to be the minimum size were found for the 3, 4, and 5-parity problems.

Keywords: genetic programming, code growth, bloat, introns, diversity maintenance, evolutionary multi-objective optimization, Pareto optimality

1 INTRODUCTION

A well-known problem in genetic programming (GP), is the tendency to find larger and larger programs over time (Tackett, 1993; Blickle & Thiele, 1994; Nordin & Banzhaf, 1995; McPhee & Miller, 1995; Soule & Foster, 1999), called *bloat* or *code growth*. This is harmful since it results in larger solutions than necessary. Moreover, it increasingly slows down the rate at which new individuals can be evaluated. Thus, keeping the size of trees that are visited small is generally an implicit objective of GP.

Another important issue in GP and in other methods of evolutionary computation is that of how diversity of the population can be achieved and maintained. A population that is spread out over promising parts of the search space has more chance of finding a solution than one that is concentrated on a single fitness peak. Since members of a diverse population solve parts of the problem in different ways, it may also be more likely to discover partial solutions that can be utilized through crossover. Diversity is not an objective in the conventional sense; it applies to the populations visited during the search, not to final solutions. A less obvious idea then is to view the contribution of individuals to population diversity as an objective.

Multi-objective techniques are specifically designed for problems in which knowledge about multiple objectives is available, see e.g. Fonseca and Fleming (1995) for an overview. The main idea of this paper is to use multi-objective techniques to add the objectives of size and diversity in addition to the usual objective of a problem-specific fitness measure. A multi-objective approach to bloat appears promising and has been used before (Langdon, 1996; Rodriguez-Vazquez, Fonseca, & Fleming, 1997), but has not become standard practice. The reason may be that basic multi-objective methods, when used with small tree size as an objective, can result in premature convergence to small individuals (Langdon & Nordin, 2000; Ekart, 2001). We therefore investigate the use of a size objective in combination with explicit diversity maintenance.

The remaining sections discuss the n-parity problem (2), bloat (3), multi-objective methods (4), diversity maintenance(5), ideas behind the approach, called FOCUS, (6), algorithmic details (7), results (8), and conclusions (9).

2 THE N-PARITY PROBLEM

The test problems that will be used in this paper are even n-parity problems, with n ranging from 3 to 5. A correct solution to this problem takes a binary sequence of length n as input and returns true (one) if

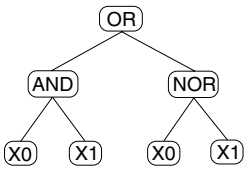


Figure 1: A correct solution to the 2-parity problem

the number of ones in the sequence is even, and false (zero) if it is odd. It is named *even* to avoid confusion with the related odd parity problem, which gives the inverse answer. Trees may use the following boolean operators as internal nodes: AND, OR, NAND, and NOR. Each leaf specifies an element of the sequence. The fitness is the fraction of all possible length n binary sequences for which the program returns the correct answer. Figure 1 shows an example.
The n-parity problem has been selected because it is a difficult problem that has been used by a number of researchers. With increasing order, the problem quickly becomes more difficult. One way to understand its hardness is that for any setting of the bits, flipping any bit inverts the outcome of the parity function. Equivalently, its Karnaugh map (Zissos, 1972) equals a checkerboard function, and thus has no adjacencies.

2.1 SIZE OF THE SMALLEST SOLUTIONS TO N-PARITY

We believe that the correct solutions to n-parity constructed as follows are of minimal size, but are not able to prove this. The principle is to recursively divide the bit sequence in half and, take the parity of each halve, and feed these two into a parity function. For subsequences of size one, i.e. single bits, the bit itself is used instead of its parity. When this occurs for one of the two arguments, the outcome would be inverted, and thus the odd 2-parity function is used to obtain the even 2-parity of the bits.
Let S be a binary sequence of length $|S| = n \geq 2$. S is divided in half yielding two subsequences L and R with, for even n, length $\frac{n}{2}$ or, for odd n, lengths $\frac{n-1}{2}$ and $\frac{n+1}{2}$. Then the following recursively defined function P(S) gives a correct expression for the even-parity of S for $|S| \geq 2$ in terms of the above operators:

$$P(S) = \begin{cases} S & \text{if} \quad |S| = 1 \\ \text{ODD}(P(L), P(R)) & \text{if} \quad |S| > 1 \wedge g(L, R) \\ \text{EVEN}(P(L), P(R)) & \text{otherwise} \end{cases}$$

where
ODD(A, B) = NOR(AND(A, B), NOR(A, B)),
EVEN(A, B) = OR(AND(A, B), NOR(A, B)), and

$$g(A, B) = \begin{cases} \text{TRUE} & \text{if} \quad (|A| = 1) \text{ XOR } (|B| = 1) \\ \text{FALSE} & \text{else} \end{cases}$$

Table 1: Length of the shortest solution to n-parity using the operators AND, OR, NAND, and NOR.

n	1	2	3	4	5	6	7
Length	3	7	19	31	55	79	103

The length $|P(S)|$ of the expression $P(S)$ satisfies:

$$|P(S)| = \begin{cases} 1 & \text{for} \quad |S| = 1 \\ 3 + 2|P(L)| + 2|P(R)| & \text{for} \quad |S| > 1 \end{cases}$$

For $n = 2^i, i > 0$, this expression can be shown to equal $2n^2 - 1$. Table 1 gives the lengths of the expressions for the first seven even-n-parity problems. For $|S| = 1$, the shortest expression is NOR(S, S); for $|S| > 1$, the length is given by the above expression. The rapid growth with increasing order stems from the repeated doubling of the required inputs.

3 THE PROBLEM OF BLOAT

A well-known problem, known as *bloat* or *code growth*, is that the trees considered during a GP run grow in size and become larger than is necessary to represent good solutions. This is undesirable because it slows down the search by increasing evaluation and manipulation time and, if the growth consists largely of non-functional code, by decreasing the probability that crossover or mutation will change the operational part of the tree. Also, compact trees have been linked to improved generalization (Rosca, 1996).
Several causes of bloat have been suggested. First, under certain restrictions (Soule, 1998), crossover favors smaller than average subtrees in removal but not in replacement. Second, larger trees are more likely to produce fit (and large) offspring because non-functional code can play a protective role against crossover (Nordin & Banzhaf, 1995) and, if the probability of mutating a node decreases with increasing tree size, against mutation. Third, the search space contains more large than small individuals (Langdon & Poli, 1998).
Nordin and Banzhaf (1995) observed that the length of the *effective* part of programs decreases over time. However, the total length of the programs in the experiments also increased rapidly, and hence it may be concluded that in those experiments bloat was mainly due to growth of ineffective code (*introns*).
Finally, it is conceivable that in some circumstances non-functional code may be useful. It has been suggested that introns may be useful for retaining code that is not used in the current individual but is a helpful building block that may be used later (Nordin, Francone, & Banzhaf, 1996).

Table 2: Properties of the basic GP method used.

Problem	3-Parity
Fitness	Fraction of correct answers
Operators	AND, OR, NAND, and NOR
Stop criterion	500,000 evaluations or solution
Initial tree size	Uniform [1..20] internal nodes
Cycle	generational
Population Size	1000
Parent selection	Boltzmann with T = 0.1
Replacement	Complete
Uniqueness check	Individuals occur at most once
P(crossover)	0.9
P(mutation)	0.1
Mutation method	Mutate node with P = $\frac{1}{n}$

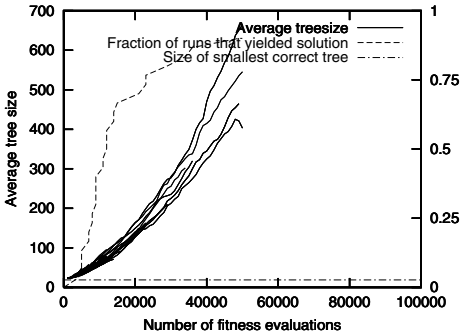


Figure 2: Average tree sizes of ten different runs (solid lines) using basic GP on the 3-parity program.

3.1 OBSERVATION OF BLOAT USING BASIC GP

To confirm that bloat does indeed occur in the test problem of n-parity using basic GP, thirty runs where performed for the 3-parity problem. The parameters of the run are shown in Table 2. A run ends when a correct solution has been found. Figure 2 shows that average tree sizes increase rapidly in each run. If a solution is not found at an early point in the run, bloating rapidly increases the sizes of the trees in the population, thus increasingly slowing down the search. A single run of 111,054 evaluations already took more than 15 hours on a current PC running Linux due to the increasing amount of processing required per tree as a result of bloat. The population of size-unlimited trees that occurred in the single 4-parity run that was tried (with trees containing up to 6,000 nodes) filled virtually the entire swap space and caused performance to degrade to impractical levels. Clearly, the problem of bloat must be addressed in order to solve these and higher order versions of the problem in an efficient manner.

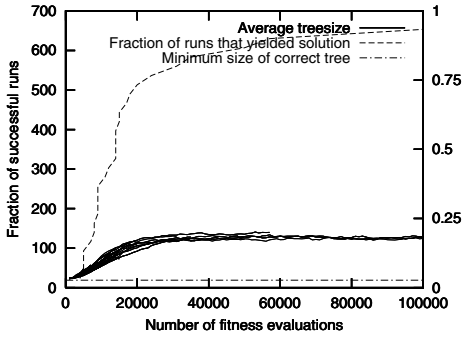


Figure 3: Average tree sizes and fraction of successful runs in the 3-parity problem using basic GP with a tree size limit of 200. Tree sizes are successfully limited, of course, but the approach is not ideal (see text).

3.2 USING A FIXED TREE SIZE LIMIT

Probably the most common way to avoid bloat is to simply limit the allowed tree size or depth (Langdon & Poli, 1998; Koza, 1992), although the latter has been found to lead to loss of diversity near the root node when used with crossover (Gathercole & Ross, 1996). Figure 3 shows the effect of using a limit of 200 on 3-parity. This limit is well above the minimum size of a correct solution, but not too high either since several larger solutions were found in the unrestricted run. The average tree size is around 140 nodes.
On the 4-parity problem (with a tree size limit of 200), the average tree size varied around 150. However, whereas on 3-parity 90% of the runs found a solution within 100,000 evaluations, on 4-parity only 33% of the runs found a solution within 500,000 evaluations, testifying to the increased difficulty of this order of the parity problem. For 5-parity, basic GP found no solutions within 1,000,000 evaluations for any of the 30 runs. Thus, our version of GP with fixed tree size limit does not scale up well. Furthermore, a fundamental problem with this method of preventing bloat is that the maximum tree size has to be selected before the search, when it is often unknown.

3.3 WEIGHTED SUM OF FITNESS AND SIZE

Instead of choosing a fixed tree size limit in advance one would rather like to have the algorithm search for trees that can be as large as they need to be, but not much larger. A popular approach that goes some way towards this goal is to include a component in the fitness that rewards small trees or programs. This is mostly done by adding a component to the fitness, thus making fitness a linear combination of a performance measure and a parsimony measure (Koza, 1992; Soule, Foster, & Dickinson, 1996). However, this approach is not without its own problems (Soule & Fos-

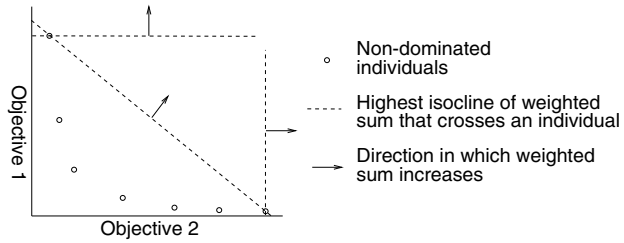


Figure 4: Schematic rendition of a concave tradeoff surface. This occurs when better performance in one objective means worse performance in the other, *vice versa*. The lines mark the maximum fitness individuals for three example weightings (see vectors) using a linear weighting of the objectives. No linear weighting exists that finds the in-between individuals, with reasonable performance in both objectives.

ter, 1999). First, the weight of the parsimony measure must be determined beforehand, and so a choice concerning the tradeoff between size and performance is already made before the search. Furthermore, if the tradeoff surface between the two fitness components is concave[1] (see Fig. 4), a linear weighting of the two components favors individuals that do well in one of the objectives, but excludes individuals that perform reasonably in both respects (Fleming & Pashkevich, 1985).

Soule and Foster (1999) have investigated why a linear weighting of fitness and size has yielded mixed results. It was found that a weight value that adequately balances fitness and size is difficult to find. However, if the required balance is different for different regions in objective space, then adequate parsimony pressure cannot be specified using a single weight. If this is the case, then methods should be used that do not attempt to find such a single balance. This idea forms the basis of multi-objective optimization.

4 MULTI-OBJECTIVE METHODS

After several early papers describing the idea of optimizing for multiple objectives in evolutionary computation (Schaffer, 1985; Goldberg, 1989), the approach has recently received increasing attention (Fonseca & Fleming, 1995; Van Veldhuizen, 1999). The basic idea is to search for multiple solutions, each of which satisfy the different objectives to different degrees. Thus, the selection of the final solution with a particular combination of objective values is postponed until a time when it is known what combinations exist.

A key concept in multi-objective optimization is that of dominance. Let individual x_A have values A_i for the n objectives, and individual x_B have objective values B_i. Then A dominates B if

$$\forall i \in [1..n] : A_i \geq B_i \wedge \exists i : A_i > B_i$$

Multi-objective optimization methods typically strive for *Pareto optimal* solutions, i.e. individuals that are not dominated by any other individuals.

[1]Since fitness is to be *maximized*, the tradeoff curve shown is concave.

5 DIVERSITY MAINTENANCE

A key difference between classic search methods and evolutionary approaches is that in the latter a *population* of individuals is maintained. The idea behind this is that by maintaining individuals in several regions of the search space that look promising (*diversity maintenance*), there is a higher chance of finding useful material from which to construct solutions.

In order to maintain the existing diversity of a population, evolutionary methods typically keep some or many of the individuals that happen to have been generated and have relatively high fitness, but lower than that found so far. In the same way, evolutionary multi-objective methods usually keep some dominated individuals in addition to the non-dominated individuals (Fonseca & Fleming, 1993). However, this appears to be a somewhat arbitrary way of maintaining diversity. In the following section, we present a more directed method. The relation to other diversity maintenance methods is discussed.

6 THE *FOCUS* METHOD

We propose to do diversity maintenance by using a basic multi-objective algorithm and including an objective that actively promotes diversity. To the best of our knowledge, this idea has not been used in other work, including multi-objective research. If it works well, the need for keeping arbitrary dominated individuals may be avoided. To test this, we use the diversity objective in combination with a multi-objective method that only keeps non-dominated individuals, as reported in section 8.

The approach strongly directs the attention of the search towards the explicitly specified objectives. We therefore name this method FOCUS, which stands for Find Only and Complete Undominated Sets, reflecting the fact that populations only contain non-dominated individuals, and contain all such individuals encountered so far. Focusing on non-dominated individuals combines naturally with the idea that the objectives are responsible for exploration, and this combination defines the FOCUS method.

The concept of diversity applies to populations, meaning that they are dispersed. To translate this aim into an objective for individuals, a metric has to be defined that, when optimized by individuals, leads to diverse populations. The metric used here is that of average

squared distance to the other members of the population. When this measure is maximized, individuals are driven away from each other.

Interestingly, the average distance metric strongly depends on the current population. If the population were centered around a single central peak in the fitness landscape, then individuals that moved away from that peak could survive by satisfying the diversity objective better than the individuals around the fitness peak. It might be expected that this would cause large parts of the population to occupy regions that are merely far away from other individuals but are not relevant to the problem. However, if there are any differences in fitness in the newly explored region of the search space, then the fitter individuals will come to replace individuals that merely performed well on diversity. When more individuals are created in the same region, the potential for scoring highly on diversity for those individuals diminishes, and other areas will be explored. The dynamics thus created are a new way to maintain diversity.

Other techniques that aim to promote diversity in a directed way exist, and include fitness sharing (Goldberg & Richardson, 1987; Deb & Goldberg, 1989), deterministic crowding (Mahfoud, 1995), and fitness derating (Beasley, Bull, & Martin, 1993). A distinguishing feature of the method proposed here is that in choosing the diversity objective, problem-based criteria can be used to determine which individuals should be kept for exploration purposes.

7 ALGORITHM DETAILS

The algorithm selects individuals if and only if they are not dominated by other individuals in the population. The population is initialized with 300 randomly created individuals of 1 to 20 internal nodes. A cycle proceeds as follows. A chosen number n of new individuals (300) is generated based on the current population using crossover (90%) and mutation (10%). If the individual already exists in the population, it is mutated. If the result also exists, it is discarded. Otherwise it is added to the population. All individuals are then evaluated if necessary. After evaluation, all population members are checked against other population members, and removed if dominated by any of them.

A slightly stricter criterion than Pareto's is used: A dominates B if $\forall i \in [1..n] : A_i \geq B_i$. Of multiple individuals occupying the same point on the tradeoff surface, precisely one will remain, since the removal criterion is applied sequentially. This criterion was used because the Pareto criterion caused a proliferation of individuals occupying the same point on the trade-off surface when no diversity objective was used[2].

[2]In later experiments including the diversity objective, this proliferation was not observed, and the standard Pareto criterion also worked satisfactorily.

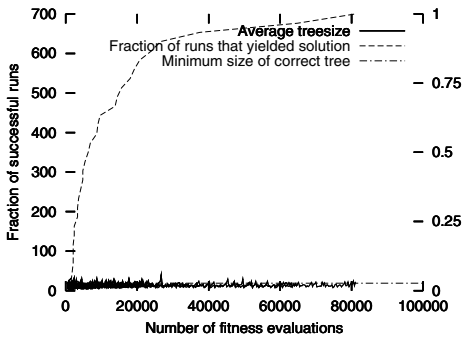


Figure 5: Average tree size and fraction of successful runs for the [fitness, size, diversity] objective vector on the 3-parity problem. The trees are much smaller than for basic GP, and solutions are found faster.

The following distance measure is used in the diversity objective. The distance between two corresponding nodes is zero if they are identical and one if they are not. The distance between two trees is the sum of the distances of the corresponding nodes, i.e. nodes that overlap when the two trees are overlaid, starting from the root. The distance between two trees is normalized by dividing by the size of the smaller tree of the two.

8 EXPERIMENTAL RESULTS

In the following experiments we use fitness, size, and diversity as objectives. The implementation of the objectives is as follows. Fitness is the fraction of all 2^n input combinations handled correctly. For size, we use 1 over the number of nodes in the tree as the objective value. The diversity objective is the average squared distance to the other population members.

8.1 USING FITNESS, SIZE, AND DIVERSITY AS OBJECTIVES

Fig. 5 shows the graph of Fig. 3 for the method of using fitness, size, and diversity as objectives. The average tree size remains extremely small. In addition, a glance at the graphs indicates that correct solutions are found more quickly. To determine whether this is indeed the case, we compute the *computational effort*, i.e. the expected number of evaluations required to yield a correct solution with a 99% probability, as described in detail by Koza (1994).

The impression that correct solutions to 3-parity are found more quickly for the multi-objective approach (see Figure 6) is confirmed by considering the computational effort E; whereas GP with the tree size limit requires 72,044 evaluations, the multi-objective approach requires 42,965 evaluations. For the 4-parity problem, the difference is larger; basic GP needs

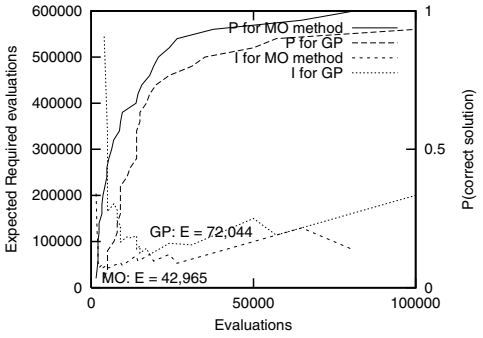


Figure 6: Probability of finding a solution and computational effort for 3-parity using basic GP and the multi-objective method.

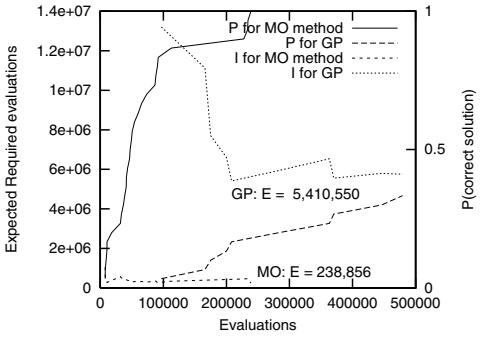


Figure 7: Probability of finding a solution and computational effort for 4-parity for basic GP and the multi-objective method. The performance of the multi-objective method is considerably superior.

5,410,550 evaluations, whereas the multi-objective approach requires only 238,856. This is a dramatic improvement, and demonstrates that our method can be very effective.

Finally, experiments have been performed using the even more difficult 5-parity problem. For this problem, basic GP did not find any correct solutions within a million evaluations. The multi-objective method did find solutions, and did so reasonably efficiently, requiring a computational effort of 1,140,000 evaluations.

Table 3 summarizes the results of the experiments. Considering the average size of correct solutions on 3-parity, the multi-objective method outperforms all methods that have been compared, as the first solution it finds has 30.4 nodes on average. What's more, the multi-objective method also requires a smaller number of evaluations to do so than the other methods. Finally, perhaps most surprisingly, it finds correct solutions using extremely small populations, typically containing less than 10 individuals. For example, the average population size over the whole experiment for 3-parity was 6.4, and 8.5 at the end of the experiment,

Table 3: Results of the experiments (GP and Multi-Objective rows). For comparison, results of Koza's (1994) set of experiments (population size 16,000) and the best results with other configurations (population size 4,000) found there. E: computational effort, S: average tree size of first solution, Pop: average population size.

3-parity	E	S	Pop
GP	72,044	93.67	1000
Multi-objective	42,965	30.4	6.4
Koza GP	96,000	44.6	16,000
Koza GP-ADF	64,000	48.2	16,000
4-parity	E	S	Pop
GP	5,410,550	154	1000
Multi-objective	238,856	68.5	15.8
Koza GP	384,000	112.6	16,000
Koza GP-ADF	176,000	60.1	16,000
5-parity	E	S	Pop
GP	∞ [1]	n.a.	n.a
Multi-objective	1,140,000	218.7	49.7
Koza GP	6,528,000	299.9	16,000
Koza GP	1,632,000	299.9	4,000
Koza GP-ADF	464,000	156.8	16,000
Koza GP-ADF	272,000	99.5	4,000

[1] No solutions were found for 5-parity using basic GP.

and the highest population size encountered in all 30 runs was 18. This suggests that the diversity maintenance achieved by using this greedy multi-objective method in combination with an explicit diversity objective is effective, since even extremely small populations did not result in premature convergence.

Considering 4 and 5-parity, the GP extended with the size and diversity objectives outperforms both basic GP methods used by Koza (1994) and the basic GP method tested here, both in terms of computational effort and tree size. The Automatically Defined Function (ADF) experiments performed by Koza for these and larger problem sizes perform better. These probably benefit from the inductive bias of ADFs, which favors a modular structure. Therefore, a natural direction for future experiments is to also extend ADFs with size and diversity objectives.

For comparison, we also implemented an evolutionary multi-objective technique that does keep some dominated individuals. It used the number of individuals by which an individual is dominated as a rank, similar to the method described by Fonseca and Fleming (1993). The results were similar in terms of evaluations, but the method keeping strictly non-dominated individuals worked faster, probably due to the calculation of the distance measure. Since this is quadratic in the population size, the small populations of multi-objective save much time (about a factor 7 for 5-parity), which made it preferable.

As a control experiment, we also investigated whether the diversity objective is really required by using only fitness and size as objectives using the algorithm that was described. The individuals found are small (around 10 nodes), but the fitness of the individuals found was well below basic GP, and hence the diversity objective was indeed performing a useful function in the experiments.

8.2 OBTAINING STILL SMALLER SOLUTIONS

Finally, we investigate whether the algorithm is able to find smaller solutions, after finding the first. After the first correct solution is found, we monitor the smallest correct solution. Although the first solution size of 30 was already low compared to other methods, the algorithm rapidly finds smaller correct solutions. The average size drops to 22 within 4,000 additional evaluations, and converges to around 20. The smallest tree (found in 12 out of 30 runs) was 19, i.e. equalling the presumed minimum size. On 4-parity, solutions dropped in size from the initial 68.5 to 50 in about 10,000 evaluations, and to 41 on average when runs were continued longer (85,000 evaluations). In 12 of the 30 runs, minimum size solutions (31 nodes) were found. Using the same method, a minimum size solution to 5-parity (55 nodes) was also found.

The quick convergence to smaller tree sizes shows that at least for the problem at hand, the method is effective at finding small solutions when it is continued running after the first correct solutions have been found, in line with the seeding experiments by Langdon and Nordin (2000).

9 CONCLUSIONS

The paper has discussed using multi-objective methods as a general approach to avoiding bloat in GP and to promoting diversity, which is relevant to evolutionary algorithms in general. Since both of these issues are often implicit goals, a straightforward idea is to make them explicit by adding corresponding objectives. In the experiments that are reported, a size objective rewards smaller trees, and a diversity objective rewards trees that are different from other individuals in the population, as calculated using a distance measure.

Strongly positive results are reported regarding both size control and diversity maintenance. The method is successful in keeping the trees that are visited small without requiring a size limit or a relative weighting of fitness and size. It impressively outperforms basic GP on the 3, 4, and 5-parity problem both with respect to computational effort and tree size. Furthermore, correct solutions of what we believe to be the minimum size have been found for all problem sizes examined, i.e. the even 3, 4, and 5-parity problems.

The effectiveness of the new way of promoting diversity proposed here can be assessed from the following, which concerns the even 3, 4, and 5-parity problems. The multi-objective algorithm that was used only maintains individuals that are not dominated by other individuals found so far, and maintains all such individuals (except those with identical objective vectors). Thus, only non-dominated individuals are selected after each generation, and populations (hence) remained extremely small (6, 16, and 50 on average, respectively). In defiance of this uncommon degree of greediness or elitism, sufficient diversity was achieved to solve these problems efficiently in comparison with basic GP method results both as obtained here and as found in the literature. Control experiments in which the diversity objective was removed (leaving the fitness and size objectives) failed to maintain sufficient diversity, as would be expected.

The approach that was pursued here is to make desired characteristics of search into explicit objectives using multi-objective methods. This method is simple and straightforward and performed well on the problem sizes reported, in that it improved the performance of basic GP on 3 and 4-parity. It solved 5-parity reasonably efficiently, even though basic GP found no solutions on 5-parity. For problem sizes of 6 and larger, basic GP is no longer feasible, and more sophisticated methods must be invoked that make use of modularity, such as Koza's Automatically Defined Functions (1994) or Angeline's GLiB (1992). We expect that the multi-objective approach with size and diversity as objectives that was followed here could also be of value when used in combination with these or other existing methods in evolutionary computation.

Acknowledgements

The authors would like to thank Michiel de Jong, Pablo Funes, Hod Lipson, and Alfonso Renart for useful comments and suggestions concerning this work. Edwin de Jong gratefully acknowledges a Fulbright grant.

References

Angeline, P. J., & Pollack, J. B. (1992). The evolutionary induction of subroutines. In *Proceedings of the fourteenth annual conference of the cognitive science society* (p. 236-241). Bloomington, Indiana, USA: Lawrence Erlbaum.

Beasley, D., Bull, D. R., & Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, *1*(2), 101–125.

Blickle, T., & Thiele, L. (1994). Genetic programming and redundancy. In J. Hopf (Ed.), *Genetic algorithms within the framework of evolutionary computation (workshop at ki-94, saarbrücken)* (pp. 33–38). Im Stadtwald, Building

44, D-66123 Saarbrücken, Germany: Max-Planck-Institut für Informatik (MPI-I-94-241).

Deb, K., & Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In J. D. Schaffer (Ed.), *Proceedings of the 3rd international conference on genetic algorithms* (pp. 42–50). George Mason University: Morgan Kaufmann.

Ekart, A. (2001). Selection based on the Pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, *2*, 61-73.

Fleming, P. J., & Pashkevich, A. P. (1985). Computer-aided control system design using a multiobjective optimization approach. In *Proceedings of the iee international conference — control '85* (pp. 174–179). Cambridge, UK.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In S. Forrest (Ed.), *Proceedings of the fifth international conference on genetic algorithms (ICGA'93)* (pp. 416–423). San Mateo, California: Morgan Kauffman Publishers.

Fonseca, C. M., & Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, *3*(1), 1–16.

Gathercole, C., & Ross, P. (1996). An adverse interaction between crossover and restricted tree depth in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, & R. L. Riolo (Eds.), *Genetic programming 1996: Proceedings of the first annual conference* (pp. 291–296). Stanford University, CA, USA: MIT Press.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette (Ed.), *Genetic algorithms and their applications : Proc. of the second Int. Conf. on Genetic Algorithms* (pp. 41–49). Hillsdale, NJ: Lawrence Erlbaum Assoc.

Koza, J. R. (1992). *Genetic programming.* Cambridge, MA: MIT Press.

Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable programs.* Cambridge, MA: MIT Press.

Langdon, W. B. (1996). Advances in genetic programming 2. In P. J. Angeline & K. Kinnear (Eds.), (p. 395-414). Cambridge, MA: MIT Press. (Chapter 20)

Langdon, W. B., & Nordin, J. P. (2000). Seeding GP populations. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, & T. C. Fogarty (Eds.), *Genetic programming, proceedings of eurogp'2000* (Vol. 1802, pp. 304–315). Edinburgh: Springer-Verlag.

Langdon, W. B., & Poli, R. (1998). Fitness causes bloat: Mutation. In W. Banzhaf, R. Poli, M. Schoenauer, & T. C. Fogarty (Eds.), *Proceedings of the first european workshop on genetic programming* (Vol. 1391, pp. 37–48). Paris: Springer-Verlag.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms.* Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL, USA. (IlliGAL Report 95001)

McPhee, N. F., & Miller, J. D. (1995). Accurate replication in genetic programming. In L. Eshelman (Ed.), *Genetic algorithms: Proceedings of the sixth international conference (icga95)* (pp. 303–309). Pittsburgh, PA, USA: Morgan Kaufmann.

Nordin, P., & Banzhaf, W. (1995). Complexity compression and evolution. In L. Eshelman (Ed.), *Genetic algorithms: Proceedings of the sixth international conference (icga95)* (pp. 310–317). Pittsburgh, PA, USA: Morgan Kaufmann.

Nordin, P., Francone, F., & Banzhaf, W. (1996). Explicitly defined introns and destructive crossover in genetic programming. In P. J. Angeline & K. E. Kinnear, Jr. (Eds.), *Advances in genetic programming 2* (pp. 111–134). Cambridge, MA, USA: MIT Press.

Rodriguez-Vazquez, K., Fonseca, C. M., & Fleming, P. J. (1997). Multiobjective genetic programming: A nonlinear system identification application. In J. R. Koza (Ed.), *Late breaking papers at the 1997 genetic programming conference* (pp. 207–212). Stanford University, CA, USA: Stanford Bookstore.

Rosca, J. (1996). Generality versus size in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, & R. L. Riolo (Eds.), *Genetic programming 1996: Proceedings of the first annual conference* (pp. 381–387). Stanford University, CA, USA: MIT Press.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the 1st international conference on genetic algorithms and their applications* (pp. 93–100). Pittsburgh, PA: Lawrence Erlbaum Associates.

Soule, T. (1998). *Code growth in genetic programming.* Unpublished doctoral dissertation, University of Idaho.

Soule, T., & Foster, J. A. (1999). Effects of code growth and parsimony presure on populations in genetic programming. *Evolutionary Computation*, *6*(4), 293–309.

Soule, T., Foster, J. A., & Dickinson, J. (1996). Code growth in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, & R. L. Riolo (Eds.), *Genetic programming 1996: Proceedings of the first annual conference* (pp. 215–223). Stanford University, CA, USA: MIT Press.

Tackett, W. A. (1993). Genetic programming for feature discovery and image discrimination. In S. Forrest (Ed.), *Proceedings of the 5th international conference on genetic algorithms, icga-93* (pp. 303–309). University of Illinois at Urbana-Champaign: Morgan Kaufmann.

Van Veldhuizen, D. A. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* Unpublished doctoral dissertation, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Zissos, D. (1972). *Logic design algorithms.* London: Oxford University Press.

Adaptive Genetic Programs via Reinforcement Learning

Keith L. Downing
Department of Computer Science
The Norwegian University of Science and Technology (NTNU)
7020 Trondheim, Norway
tele: (+47) 73 59 18 40
email: keithd@idi.ntnu.no

Abstract

Reinforced Genetic Programming (RGP) enhances standard tree-based genetic programming (GP) [7] with reinforcement learning (RL)[11]. Essentially, leaf nodes of GP trees become monitored action-selection points, while the internal nodes form a decision tree for classifying the current state of the problem solver. Reinforcements returned by the problem solver govern both fitness evaluation and intra-generation learning of the proper actions to take at the selection points. In theory, the hybrid RGP system hints of mutual benefits to RL and GP in controller-design applications, by, respectively, providing proper abstraction spaces for RL search, and accelerating evolutionary progress via Baldwinian or Lamarckian mechanisms. In practice, we demonstrate RGP's improvements over standard GP search on maze-search tasks

1 Introduction

The benefits of combining evolution and learning, while largely theoretical in the biological sciences, have found solid empirical verification in the field of evolutionary computation (EC). When evolutionary algorithms (EAs) are supplemented with learning techniques, general adaptivity improves such that the learning EA finds solutions faster than the standard EA [3, 16]. These enhancements can stem from biologically plausible mechanisms such as the Baldwin Effect [2, 14], or from disproven phenomena such as Lamarckianism [8, 4].

In most learning EAs, the data structure or program in which learning occurs is divorced from the structure that evolves. For example, a common learning EA is a hybrid genetic-algorithm (GA) - artificial neural network (ANN) system in which the GA encodes a basic ANN topology (plus possibly some initial arc weights), and the ANN then uses backpropagation or hebbian learning to gradually modify those weights [17, 10, 6]. A Baldwin Effect is often evident in the fact that the GA-encoded weights improve over time, thus reducing the need for learning [1]. Lamarckianism can be added by reversing the morphogenic process and back-encoding the ANN's learned weights into the GA chromosome prior to reproduction [12].

Our primary objective is to realize Baldwinian and Lamarckian adaptivity within standard tree-based genetic programs [7], without the need for a complex morphogenic conversion to a separate learning structure. Hence, as the GP program runs, the tree nodes can adapt, thereby altering (and hopefully improving) subsequent runs of the same program. Thus, the typical problem domain is one in which each GP tree executes many times during fitness evaluation, for example, in control tasks.

2 RGP Overview

Reinforced Genetic Programming combines reinforcement learning [11] with conventional tree-based genetic programming [7]. This produces GP trees with reinforced action-choice leaf nodes, such that successive runs of the same tree exhibit improved performance on the fitness task. These improvements may or may not be reverse-encoded into the genomic form of the tree, thus facilitating tests of both Baldwinian and Lamarckian enhancements to GP.

The basic idea is most easily explained by example. Consider a small control program for a maze-wandering agent:

```
(if (between 0 x 5)
    (if (between 0 y 5)
            (choice (move-west) (move-north))      R1
            (choice (move-east) (move-south)))     R2
    (if (between 6 x 8)
            (choice (move-west) (move-east))       R3
            (choice (move-north) (move-south))))   R4
```

Figure 1 illustrates the relationship between this program and the 10x10 maze. Variables x and y specify the agents current maze coordinates, while the *choice* nodes are monitored action decisions. The *between* predicate simply tests if the middle argument is within the closed range specified by the first and third arguments, while the *move* functions are discrete one-cell jumps. So if the agent's current location falls within the southwest region, R1, specified by the (between 0 x 5) and (between 0 y 5) predicates of the decision tree, then the agent can choose between a westward and a northward move; whereas the eastern edge gives a north-south option.

During fitness testing, the agent will execute its tree code on each timestep and perform the recommended action in the maze, which then returns a reinforcement signal. For example, hitting a wall may invoke a small negative signal, while reaching a goal state would garner a large positive payback.

Initially, the choice nodes select randomly among their possible actions, but as the fitness test proceeds, each node accumulates reinforcement statistics as to the relative utility of each action (in the context of the particular location of the choice node in the decision tree, which reflects the location of the agent in the maze). After a fixed number of random *free trials*, which is a standard parameter in reinforcement-learning systems (RLSs), the node begins making stochastic action choices based on the reinforcement statistics. Hence, the node's initial exploration gives way to exploitation.

Along with determining the tree's internal decisions, the evolving genome sets the range for RL exploration by specifying the possible actions to the choice nodes; the RLS then fine-tunes the search. By including alternate forms of choice nodes in GP's primitive set, such as choice-4, choice-2, choice-1 (direct action), where the integer denotes the number of action arguments, the RGP's learning effort comes under evolutionary control. Over many evolutionary generations, the genomes provide more appropriate decision trees and more restricted (yet more relevant) action options to the RLS.

In the maze domain, learning has an implicit cost due to the nature of the fitness function, which is based on

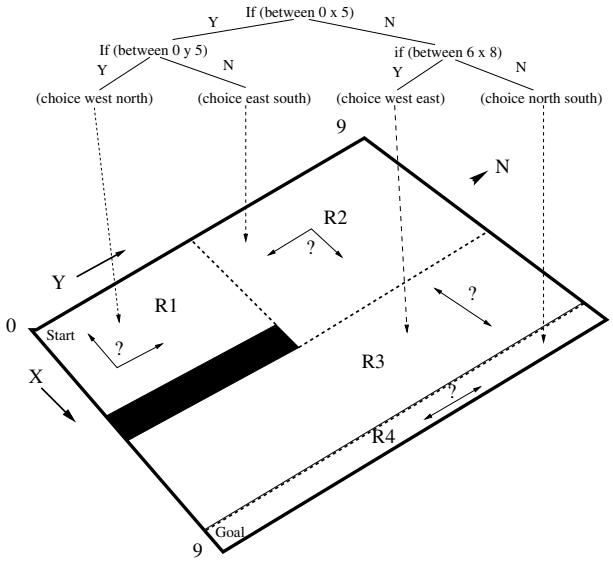


Figure 1: The genetic program determines a partitioning of the reinforcement-learning problem space.

the average reinforcement per timestep of the agent. So an agent that moves directly to a goal location (or follows a wall without any explorative "bumps" into it) will have higher average reinforcement than one that investigates areas off the optimal path. Initially, explorative learning helps the agent find the goal, but then evolution further hones the controllers to follow shorter paths to the goal, with little or no opportunity for stochastic action choices. Hence, the average reinforcement (i.e. fitness) steadily increases, first as a result of learning (phase I of the Baldwin Effect) and then as a result of genomic hard-wiring (phase II) encouraged by the implicit learning cost [9].

To exploit Lamarckianism, RGP can replace any choice node in the genomic tree with a direct action function for the action that was deemed best for that node. Hence, if the choice node for R1 in Figure 1 learns that north is the best move from this region (while choices for R2 and R3 find eastward moves most profitable, and R4 learns the advantage of southward moves), then prior to reproduction, the genome can be specialized to:

```
(if (between 0 x 5)
    (if (between 0 y 5) (move-north) (move-east))
    (if (between 6 x 8) (move-east) (move-south)
```

This represents an optimal control strategy for the example, with no time squandered on exploration.

3 Reinforcement Learning in RGP

Reinforcement Learning comes in many shapes and forms, and the basic design of RGP supports many of these variations. However, the examples in this paper use Q-learning [15] with eligibility traces.

Q-learning is an *off-policy temporal differencing* form of RL. In conventional RL terminology, Q(s,a) denotes the value of choosing action a while in state s. Temporal differencing implies that to update Q(s,a) for the current state, s_t, and most recent action, a_t, utilize the difference between the current value of $Q(s_t, a_t)$, and the sum of a) the reward, r_{t+1}, received after executing action a in state s, and b) the discounted value of the new state that results from performing a in s. For the new state, s_{t+1}, its value, $V(s_{t+1})$ is based on the best possible action that can be taken from s_{t+1}, or $max_a Q(s_{t+1}, a)$. Hence, the complete update equation is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

Here, γ is the discount rate and α is the step size or learning rate. The expression in brackets is the *temporal-difference error*, δ_t. Thus, if performing a in s leads to positive (negative) rewards and good (bad) next states, then $Q(s, a)$ will increase (decrease), with the degree of change governed by α and γ.

To implement these Q(s,a) updates (the core activity of Q-learning) within GP trees, RGP employs *qstate* objects, one per choice node. Each qstate houses a list of *state-action pairs* (SAPs), where the *value* slot of each SAP corresponds to Q(s,a). For each GP tree, a *qtable* object is generated. It keeps track of all qstates in the tree, as well as those most recently visited and the the latest reinforcement signal.

In conventional RL, all possible states, Σ, are determined prior to any learning, with each state typically a point in a space whose dimensions are the relevant environmental factors and internal state variables of the agent. So for a maze-wandering robot, the dimensions might be discretized x and y coordinates along with the robot's energy level. Conversely, in RGP, each individual GP trees determines its own Σ in a manner that generally partitions a standard RL state space into coarser regions. Whereas a basic Q-learner would divide an NxM maze into NM cell states and then try to learn optimal actions to perform in each cell, an RGP individual divides the same maze into a number (normally much less than NM) of region states and uses RL to learn a mutual proper action for every cell in each region. Thus, evolution proposes state-space partitions and possible actions for each partition, while learning finds the most appropriate of those actions.

In RGP, the trail through a program tree from the root to a choice node embodies an RL state. In other words, the Q-learning state of the agent-environment duo can only be found by running the tree in the current context and registering the choice node that gets activated. The program thus serves as a state-classification tree with action options at the leaves. Hence, during Q-learning, the temporal-difference update of $Q(s_t, a_t)$ must wait until the succeeding run of the tree, since only then is s_{t+1} known.

This basic scheme will then support a wide array of reinforcement-learning mechanisms, which typically differ in their methods of estimating $V(s_{t+1})$ and then updating $V(s_t)$ or $Q(s_t, a_t)$ [11]. Furthermore, a few simple additions to the SAP objects enable eligibility tracing and full backups, both of which greatly speed the convergence of Q-learning to an optimal control strategy.

Figure 3 graphically illustrates this basic process, wherein the GP tree sends a *move* command to the simulator/problem-solver, which makes the move and returns a reinforcement to the RLS qtable, which stores it and waits until the next run of the GP tree to determine the abstract state, $s_{t+1} = R3$ of the problem solver. The RLS then computes the temporal difference error and sends it to the most recently activated SAP, (R2, North), which relays a decayed (via the eligibility trace) version to its predecessor, and so on back through the sequence of active SAPs.

The pseudocode of Figure 2 gives a rough sketch of the combination of RL and GP in RGP.

3.1 Maze Search Examples

Maze searching is a popular task in the RL literature, partly due to the clear mapping from states and actions to 2d graphic representations of optimal strategies (i.e., grids with arrows). Despite this graphic simplicity, the underlying search problem is quite complex, since the agent lacks any remote sensing capabilities, let alone a birds-eye view of the maze. So trial and error is the only feasible approach, and learning from these errors is essential for success.

Figure 4 shows a 10x10 maze with a start point in the southwest and goal site on the eastern edge. The maze includes a few subgoals along the optimal path, so agents have opportunities for gaining partial credit. Reinforcements are 10 for the main goal, 2 for each

```
For generation = 1 to max-generations
  ∀a ∈ agent-population
    steps = 0
    For episode = 1 to max-episodes
      SAP_old = ∅, reward_old = ∅
      ps-state(a) = start
      Repeat
        SAP_new = run-GP-tree(a)
        [reward_new, ps-state(a)] = do-action(SAP_new)
        do-temp-diff(SAP_old ,reward_old ,SAP_new)
        predecessor(SAP_new) = SAP_old; for elig trace
        SAP_old = SAP_new, reward_old = reward_new
        steps = steps + 1
      Until ps-state(a) = goal or timeout
    Fitness(a) = total-reward(a) / steps
```

Figure 2: Pseudocode overview of RGP

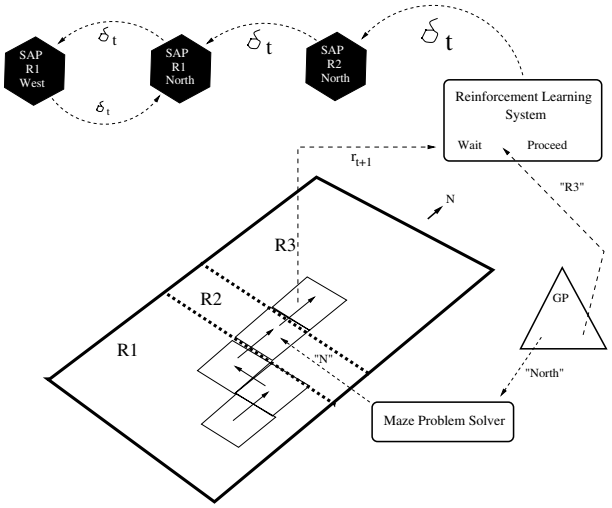


Figure 3: The basic control flow in RGP: The GP tree sends a movement command to the problem solver, which carries it out and returns the reinforcement to the RLS. After waiting to receive the next state from the GP, the RLS computes the temporal difference, δ_t and passes it down the chain of recently-active SAPs. The SAPs are separated from the GP tree only for illustrative purposes.

subgoal, -1 for hitting a wall, and 0 for all other moves. Agents are also penalized -1 for repeating any cell that occured within the past 20 moves (i.e., *minimum loop* = 21). The optimal path has 20 steps, with a total payoff of 18 (1 goal plus 4 subgoals). Thus, any agent who takes the shortest path will have an average reinforcement per timestep, $\bar{R}$, of 0.9. Agent fitness is computed as $e^{\bar{R}}$, so maximum fitness is 2.46 in this maze.

The RGP functions (with number of arguments in parentheses) are: 1) Logical functions: and(2), or(2), not(1), in-region(4); 2) Conditionals: if(3); 3) Monitored Actions: mve(0), mvw(0), mvn(0), mvs(0); and 4) Monitored Choices: pickmove(0)

The *in-region* predicate, in-region(x1,x2,y1,y2), returns true iff the x coordinate of the agent's location is in the closed range [x1, x2] and the y coordinate is within [y1, y2]. The 4 *move* actions are for moving east, west, north and south, respectively. These actions expand into single-action choice nodes so that the resulting reinforcement signals can be propagated through the reinforcement learning system to the other choice nodes. *Pickmove* is the only true trial-and-error learning function. It expands into a choice node with all 4 action possibilities. The *if*, *and*, *or* and *not* functions are standard. Terminals for an NxN maze are the integers 0 through N-1; all maze indexing is 0-based. Strong typing of the RGP trees insures that action and choice nodes occur only at the leaves. The GP uses two-individual-tournament selection with single-individual elitism.

During fitness testing, each agent gets 3 attempts at the maze, i.e., 3 reinforcement-learning *episodes*, with a maximum of 50 steps per attempt (i.e., *max-steps*=50). Each choice node selects actions randomly during the first 16 visits (i.e., *free trials*=16), after which the SAP with highest value gets priority. The discount, γ, and decay, λ, rates for RL are both 0.9, while $\alpha = 0.1$ is the learning rate (i.e., step-size parameter). Many RL systems use a much higher α value, but a lower value seems more appropriate for the non-Markovian situations incurred by RGP's coarse state-space abstractions: it is dangerous to allow the reinforcement of any one move to have excessive influence on a Q(s,a) value when it is unclear whether action a in state s will yield anything close to the same result on another occasion. Table 1 summarizes these details.

Objective:	Find optimal strategy for traversing the maze from start to goal.
Terminal set:	0...N-1 (for an NxN maze)
Function set:	and, or, not, in-region, if, mve, mvw, mvn, mvs, pickmove
Standard fitness:	e^R
GP Parameters:	population = 500, generations = 400, minimum loop = 21, p_{mut} = 0.5, p_{cross} = 0.7
RL Parameters:	α = 0.1 , γ = 0.9, λ = 0.9 , episodes = 3, max-steps = 50, free trials = 16, penalty = -1, goal reward = 10, subgoal reward = 2

Table 1: Tableau for RGP used for the 10x10 maze-search problem

Figure 4 shows the maze along with the fittest strategy for the final generation, as depicted by arrows. Figure 7 displays a logically-simplified, intron-free version of the code for this strategy; the original contained approximately 150 internal nodes. Figures 5 and 6 show a fitness graph and a plot of the average learning effort per generation. The latter is simply the average number of decisions made at all of the active choice nodes in the population, where "active" means that control comes to the node at least once during fitness evaluation. An average near 4 reveals a majority of *pickmove* nodes, while values closer to 1 indicate the dominance of single-action choice nodes.

Note the very slow progress in the first 100 generations, followed by a rapid increase from generation 100 to 175. Since the GP uses elitism, the rugged maximum-fitness plots in these transient periods reflect stochastic behavior, which has only one source: *pickmove*. Hence, the agents use learning to evolutionary advantage, as is characteristic of the first stage of the Baldwin Effect. But then, near generation 175, an optimally hard-wired agent emerges and fitness shoots up to the maximum value. The stability of the maximum curve after this ascent entails a total absence of active learning nodes in the highest-fitness individuals.

The learning graph of Figure 6 shows the classic Baldwinian progression, with an initial increase in learning rate followed by a gradual decline as learned strategy components become hard-wired. The learning drop correlates with the fitness increase, with the final plunge occuring during convergence: the lack of exploratory moves on the path to the goal facilitates a maximum average reward.

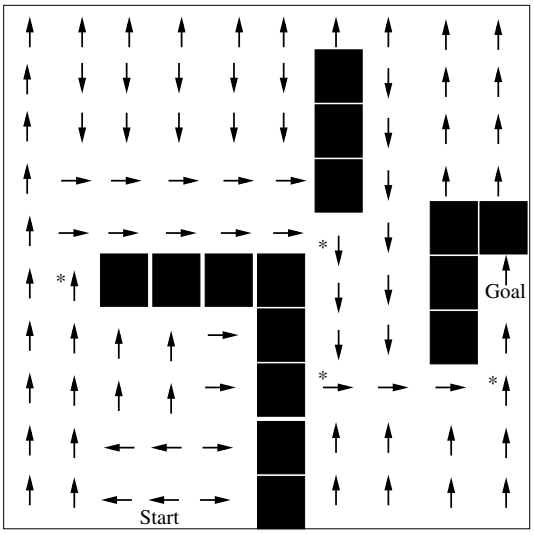


Figure 4: The 10x10 test maze. Asterisks denote subgoal cells.

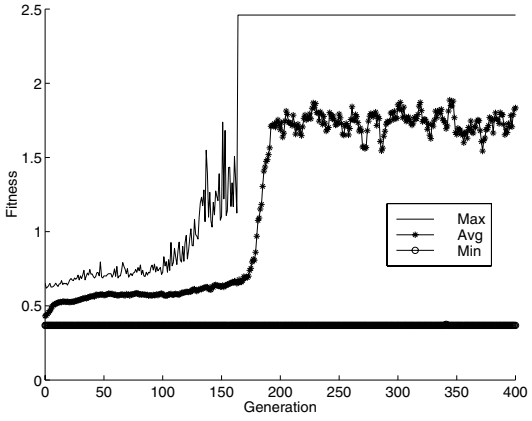


Figure 5: Fitness progression in a standard run of RGP on the 10x10 maze of Figure 4

3.1.1 Performance Comparison

We compare the performance of four Evolutionary Algorithms: 1) a standard GP, 2) a standard GP with one extra function-set member: randmove(0), 3) an RGP, and, 4) an RGP with 20% Lamarckianism.

As shown in Table 2, the RGP employs the same function set as in the previous example, while the standard GP lacks a *pickmove* equivalent, plus its four move functions are not monitored. For the second EA, *randmove* is a function that randomly selects a move in one of the 4 directions. It does not keep track of reinforcements nor send information to previously-called randmove nodes. Hence, it represents the stochastic exploration of the early stages of RL, but without the credit assignment and adaptivity.

In Lamarckian RGP, reverse encoding of learned moves into the genome is on a per-individual basis, so 20% of the maze walkers have all of their active multiple-move choice nodes converted into single-action nodes (for the action that gave the best results for that choice node during the run) immediately prior to reproduction.

The four EAs were tested on three 5x5 mazes, the most difficult of which appears in Figure 8. The performance metric is the average of the best-of-generation fitnesses for 100 runs of 50 individuals over 50 generations. On the two easier mazes (not shown), Lamarckian RGP finds optimal solutions much faster on average than the other 3 EAs, with basic RGP outperforming the two GP variants. However, in the most difficult test, RGP overtakes Lamarckian RGP, as shown in Figure 9, while the two GP variants lose ground to the RGP versions. In general, the three comparisons

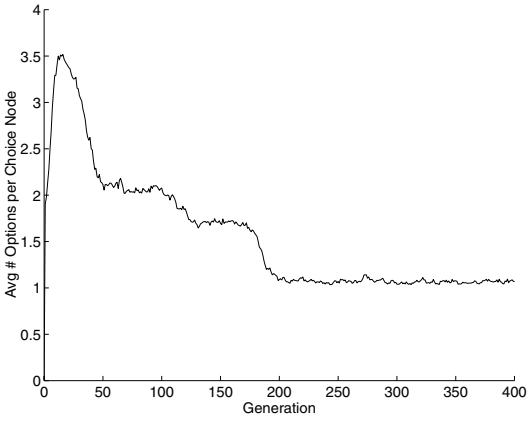


Figure 6: Progression of population-averaged learning effort in an RGP run on the 10x10 maze of Figure 4

```
(if (in-region 1 3 0 4)
  (if (in-region 1 1 5 5)
      (if (not (in-region 1 8 1 2))
          (if (in-region 5 9 8 8) (pickmove) (mve))
          (mvw))
      (if (in-region 2 3 0 1) (mvw) (mvn)))
  (if (in-region 6 6 7 8)
      (if (in-region 0 2 9 9) (mve) (mvw))
      (if (or (in-region 4 8 2 2) (in-region 1 5 0 6))
          (mve)
          (if (in-region 1 7 2 8) (mvs) (mvn)))))
```

Figure 7: Logically-simplified, intron-free Lisp code for the strategy of the most fit individual of generation 400 of the 10x10 maze search.

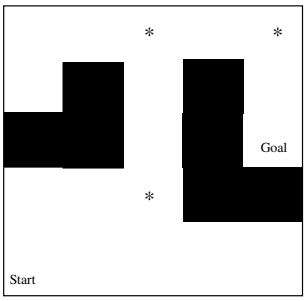


Figure 8: The most difficult of the three 5x5 mazes used in the EA comparison tests. Asterisks denote subgoal locations.

reveal a significant advantage to the reinforced GPs with respect to total evolutionary effort (i.e., fitness gain per individual tested), whether via Baldwinian or Lamarckian processes.

Objective:	Find optimal strategy for traversing the maze from start to goal.
Terminal set:	0...4
Function set:	and, or, not, in-region, if, mve, mvw, mvn, mvs, pickmove
Evol. Algs.:	GP, GP + Random Nodes, RGP, Lamarckian RGP
Standard fitness:	$e^{\bar{R}}$
Runs:	100 per algorithm per maze
GP Parameters:	population = 50, generations = 50, minimum loop = 11, $p_{mut} = 0.5$, $p_{cross} = 0.7$ $p_{lamarck} = 0.2$
RL Parameters:	$\alpha = 0.1$, $\gamma = 0.9$, $\lambda = 0.9$, episodes = 10, max-steps = 15 or 20, free trials = 8, goal reward = 10, subgoal reward = 2, penalty = -1

Table 2: Tableau for Evolutionary Algorithms used in the comparative runs of the 5x5 maze in Figure 8

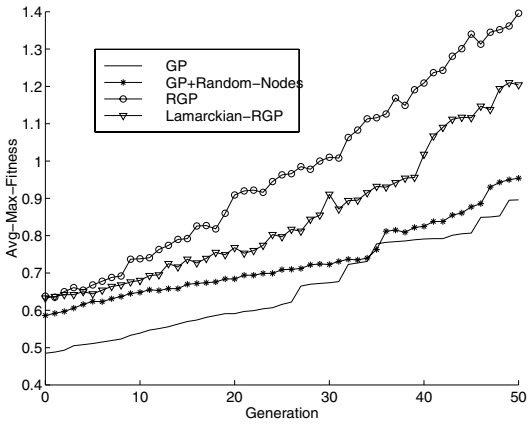


Figure 9: Comparative average fitness progressions of 100 runs each of the 4 EAs on the maze of Figure 8.

However, the addition of RL increases the computational effort of fitness testing by about 50% for a single-episode learning test. But for multiple-episode learning, the effort/episode ratio decreases substantially, since a) the cost of generating the RL data structures is paid only for the first episode, and b) as learning progresses, fewer actions are chosen stochastically, more efficient solutions are discovered, and hence fewer episode time-outs occur. In other tests, RGP permitted monitored choices at internal nodes of the GP tree. The results were similar to the best curves of Figure 9, but the computational effort was an order of magnitude worse than RGP. In general, further testing on a variety of problems is necessary to assess the computational tradeoffs of RGP versus standard GP.

4 Related Work

To date, the only direct combination of tree-based GP and RL is Iba's QGP system [5]. It uses GP to generate

a structured search space for Q-Learning. Given a set of possible state variables (e.g. w,x,y,z), QGP evolves Q-tables with variable combinations as the dimensions. For example, the genotype (TAB (* x y) (+ z 5)) specifies a 2-d table with xy as one dimension and z+5 as the other. The individual states in this table have the same level of abstraction and scope: each circumscribes the same volume in the underlying continuous state space. In several multi-agent maze-navigation tasks, QGP generates useful Q-tables to simplify RL, and in situations with many possible state variables, QGP outperforms standard RL, which flounders in an exponential search space.

In contrast to QGP, which applies GP to improve RL, RGP uses RL to enhance GP. While Iba constrains his GP trees to a small set of functions and terminals to generate well-formed Q-tables, RGP sanctions the evolution of amorphous decision trees that embody heterogeneous abstractions of the RL search space. One qstate in RGP may represent a single maze cell, while another, in the same GP tree, can encompass several rows and columns or even a concave region or a set of disjoint regions. This reflects the philosophy that the proper abstractions are not necessarily homogeneous partitions of a select quadrant of the search space. Unfortunately, our approach incurs a much larger evolutionary search cost than Iba's, making the present RGP an unlikely aid to standard RL. But for improving standard GP, RGP holds some promise, since it endows GP trees with behavioral flexibility.

Whereas QGP strongly couples GP and RL, RGP allows evolution to determine the degree of learning needed for a particular problem, thus facilitating the standard Baldwininan transition from early plasticity to later hard-wiring in static problem domains.

In the other previous GP/RL hybrid, Teller's use of credit assignment in neural programming [13] more closely matches the goals of our RGP research: to supplement genetic programming with internal reinforcements in order to increase search efficiency. However, the differences between RGP trees and neural programs are quite extreme, as are the associated reinforcement mechanisms. While RGP trees are typically control-flow structures, neural programs involve data flow between distributed neural processors. Internal reinforcement of neural programs (IRNP) closely ressembles supervised learning in conventional artificial neural networks: discrepancies between desired and actual system outputs over a training set govern internal updates. Conversely, RGP is designed for reinforcement learning in the standard machine-learning sense [11]: situations where the environmental feedback signals constitute rewards or punishments but do not explicitly indicate the correct problem-solver action. The two key characteristics of RL: trial-and-error search and (potentially) delayed rewards, are intrinsic to RGP. This makes it amenable to a host of control tasks, whereas IRNP appears more tailored for classification problems.

The collective results of QGP, RGP and IRNP indicate that combinations of GP and credit-assignment harbor potential benefits for the whole spectra of adaptive systems, from supervised and reinforced learners to evolutionary algorithms.

5 Discussion

RGP supplements evolutionary search with reinforcement learning, providing a hybrid approach for situations in which each GP tree runs several times during fitness evaluation, e.g., control tasks. Ideally, RGP should benefit both GP and RL. As shown above, the added plasticity that RL gives to GP trees can speed evolutionary convergence to good solutions via Baldwinian and/or Lamarckian mechanisms. Conversely, using GP to determine proper state abstractions for RL may yield a huge savings for RL systems that get bogged down in immense fine-grained search spaces.

Of course, the hybrid bears added computational costs. The learning GP trees require more space and time to execute than standard GP trees, and although a single RL session in the abstracted state space often runs much faster than in the detailed state space, the evolutionary effort to find the proper abstraction can dominate total run-time complexity. This does not preclude the possibility of mutual improvements for both RL and GP, but the potential for such is clearly problem specific and probably only empirically ascertained.

Essentially, RGP inverts the typical control flow of a tree-based genetic program. For example, Koza [7] attacks the broom-balancing-on-a-moving-cart problem with a set of primitives whose composite programs return an action value from the top of the tree. However, the corresponding RGP solution involves primitives that attempt to classify the current problem state (in terms of the cart's velocity, the broom's angle, etc.) and thereby funnel control to a leaf node, which houses a cart command or a monitored, reinforced choice of such commands. Thus, RGP enforces a different modelling scheme, one which typically requires strong typing of the primitive functions. As with standard GP, designing function sets is more of an art than a science in RGP, but the task is no more complicated,

and quite possibly more natural, when viewed from RGP's classify-and-act perspective.

References

[1] David H. Ackley and Michael L. Littman. Interactions between learning and evolution. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 487–509. Addison, 1992.

[2] J. Mark Baldwin. A new factor in evolution. *The American Naturalist*, 30:441–451, 1896. reprint in: Adaptive Individual in Evolving Populations: Models and Algorithms, R. K. Belew and M. Mitchell (eds.), 1996, pp. 59–80, Reading, MA: Addison Wesley.

[3] Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987. reprint in: Adaptive Individuals in Evolving Populations: Models and Algorithms, R. K. Belew and M. Mitchell (eds.), 1996, pp. 447–454, Reading, MA: Addison Wesley.

[4] Christopher R. Houck, Jeffery A. Joines, Michael G. Kay, and James R. Wilson. Empirical investigation of the benefits of partial Lamarckianism. *Evolutionary Computation*, 5(1):31–60, 1997.

[5] Hitoshi Iba. Multi-agent reinforcement learning with genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 167–172, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[6] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4):461–476, 1990.

[7] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, 1992.

[8] Jean Baptiste Lamarck. Of the influence of the environment on the activities and habits of animals, and the influence of the activities and habits of these living bodies in modifying their organization and structure. In Jean Baptiste Lamarck, editor, *Zoological Philosophy*, pages 106–127. Macmillan, London, 1914. Reprint in: Adaptive Individuals in Evolving Populations: Models and Algorithms, ed. R. K. Belew and M. Mitchell.

[9] Giles Mayley. Landscapes, learning costs and genetic assimilation. *Evolutionary Computation*, 4(3), 1996. Special edition: Evolution, Learning, and Instinct: 100 Years of the Baldwin Effect.

[10] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hedge. Designing neural networks using genetic algorithms. In *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 379–384. Kaufmann, 1989.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[12] Stewart Taylor. Using Lamarckian evolution to increase the effectiveness of neural network training with a genetic algorithm and backpropagation. In John R. Koza, editor, *Artificial Life at Stanford 1994*, pages 181–186. Stanford Bookstore, Stanford, California, 94305-3079 USA, June 1994.

[13] Astro Teller. The internal reinforcement of evolving algorithms. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 14, pages 325–354. MIT Press, Cambridge, MA, USA, June 1999.

[14] Peter Turney, L. Darrell Whitley, and Russell W. Anderson. Introduction to the special issue: Evolution, learning, and instinct: 100 years of the Baldwin effect. *Evolutionary Computation*, 4(3):iv–viii, 1997.

[15] C.J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[16] Darrell L. Whitley, V. Scott Gordon, and Keith E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard M"anner, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 6–15, Berlin, 1994. Springer. Lecture Notes in Computer Science 866.

[17] Larry Yaeger. Computational genetics, physiology, metabolism, neural systems, learning, vision and behavior or polyworld: Life in a new context. In C. G. Langton, editor, *Artificial Life III, Proceedings Volume XVII*, pages 263–298. Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, 1994.

Evolving Strategies for Global Optimization - A Finite State Machine Approach

Clemens Frey
Department of Mathematics
Darmstadt University of Technology
frey@mathematik.tu-darmstadt.de

Günter Leugering
Department of Mathematics
Darmstadt University of Technology
leugering@mathematik.tu-darmstadt.de

Abstract

In this work Genetic Programming methods are used to find certain transition rules for two-step discrete dynamical systems. This issue is similar to the well-known artificial-ant problem. Here we seek the dynamic system to produce a trajectory leading from given initial values to a maximum of a given spatial functional. This problem is recast into the framework of input-output relations for controllers, and the optimization is performed on program trees describing input filters and finite state machines incorporated by these controllers simultaneously. Reinterpreting the resulting optimal discrete dynamical system as an algorithm for finding the maximum of a functional under constraints, we have derived a paradigm for the automatic generation of adapted optimization algorithms via optimal control. We provide numerical evidence on key properties of resulting strategies.

1 INTRODUCTION

It is well-known in constrained optimization that certain algorithms resulting from augmented Lagrangians are equivalent to operator splitting schemes for appropriate differential equations (Glowinski and LeTallec, 1989). Thus, the optimal point is achieved as the limit point of a dynamical system's trajectory. Indeed, any iterative process designed to achieve a maximum of a given functional can formally be interpreted as a discrete dynamical system with some memory for its iteration history. A well-known experiment in the Genetic Programming context, namely the artificial-ant problem, can be interpreted in this sense, too. There the functional to be maximized consists of the number of food pieces found by the ant, and the dynamic system is given by the initial position of the ant on the grid and by its strategy. If it is modelled by a finite state machine (Jefferson et al., 1991) the memory is established by set of internal states of the machine.

It is tempting to optimize search strategies performing a maximization (or minimization) task on objective functions in an analogous way. We are, therefore, looking for an optimal dynamical system such that a trajectory starting at given values terminates at a global maximum of a function Φ. Reaching for this we have to confine ourselves to a certain class of dynamical systems which we can parametrize in a suitable way. In fact, in this paper we concentrate on systems which are based on deterministic Mealy machines, i.e. on discrete dynamical systems on finite sets of states and inputs.

Since observations gathered from evaluating an objective function are typically non-discrete, the formalization of the system must also include some kind of input filter which maps non-discrete function values to discrete input for the finite state machine. Eventually, by detecting policies for searching maxima on objective functions, we relate global optimization with global optimal control.

It is important to note that the experiments suggest robusteness of the optimal policies with respect to changes in the cost functional and in the initial conditions (i.e. the starting points) of the search procedure. This is the main reason not to retreat to the mere search for optimal solutions directly instead of searching for dynamic systems leading to such solutions; furthermore, this illustrates one of the main differences to the artificial-ant experiments. Additionally, as opposed to the latter, here strategies have to deal with multimodal environments and non-discrete gradient information.

These features are typical for constrained global op-

timization problems, but also for the problem a biological plant, e.g., generally faces when searching its environment for resources. Thus this problem can be seen as a subproblem of modelling the behaviour of resource acquisition of single plants, and solving it in terms of an evolutionary approach plays an important role in modelling an evolving, herbal ecoystem as a whole (cf. (Hauhs and Lange, 1996), (Lange, 1999)).

The controller design proposed in this paper is, indeed, used as a basis for calculating growth directions at the shoot level of a model for the simulation of plant growth. Used in that context, instead of the model functions that are incorporated here functions describing a plant's environment come into action. By using model functions here, however, we hope to acquire some insight into the controllers' fundamental behaviour. The important observation in ecological respect is that neither excessive resources nor exceptionally rich designs are necessary in order to obtain a whole variety of suitable strategies.

We first pose the problem in section 2. In the following section we propose the design of controllers as a composition of finite state machines and sensors filtering input information. Section 4 outlines the Genetic Programming algorithm involved and defines the evolvable structures subject to its operators. Sections 5 and 6 show the experimental setup and the results we have obtained. The final section gives a concluding discussion of this work.

2 PROBLEM SETTING

We consider spatial problems of the following type. First of all let a non-negative function $\Phi : \Omega \to \mathbb{R}_{\geqslant 0}$ on a subdomain $\Omega \subset \mathbb{R}^2$ be given; this function will be called *objective function* from now on. The discrete dynamical system shall be determined by an update rule

$$x_{t+1} = x_t + f\big(\Phi(x_t), \Phi(x_{t-1})\big), \qquad (1)$$

where x_0 and x_1 are given and $(x_t)_{t\geqslant 0} \subset \Omega$. We look at this two-step recursion up to a finite horizon $T > 1$ while our objective is to maximize

$$\Psi(f,T) := \frac{\nu(\Phi(x_T)+1)}{\max_{0\leqslant t\leqslant T}\Phi(x_t)+1} + \max_{0\leqslant t\leqslant T}\Phi(x_t) \qquad (2)$$

by choosing f in an optimal way.

Laying aside the mathematical notations, we seek a strategy of motion for a device which uses the information from both of the positions having been visited most recently (1). This strategy shall be optimal in a sense that a maximum of Φ should be reached (second addend in (2)) and retained up to the final step T (first addend in (2)). Since there is a tradeoff between generally retaining the last value x_t found and searching for better values of Φ we have introduced a constant $\nu \in \mathbb{R}$ into definition (2) which tunes the respective weight of the optimization criteria.

The function $\Psi(f,T)$ will be used as the fitness function in the Genetic Programming setting. The design of the controllers, i.e. the set from which f can be chosen, will be specified next.

3 DESIGN OF CONTROLLERS

Any controller being evolved which uses a certain update rule f will have the overall design depicted in figure 1. It consists of a *Mealy machine* acting as the control kernel, a *sensor device* which maps objective function values to a finite set of input values for the finite state machine, and an *actor device* which turns the finite state machine's output into action. Essentially these actions are movements of the device on a two-dimensional grid. The principal task of our controllers consists of building up a trajectoriy in Ω. This is done in the following way.

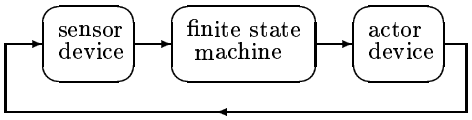


Figure 1: Overall Controller Design

The actor device always contains values x_t and x_{t-1} defining its current and previous positions, respectively, on a grid over the objective function's domain Ω. Additionally, it holds a current direction vector d_t on this grid, which spans one or more mesh lengths. Values for x_0, x_1 and d_1 are given as initial values of the dynamic system. Depending on the output coming from the Mealy automaton, the actor device may stay at the current position x_t or move in one of the four orthogonal directions on the grid in order to attain x_{t+1}. The step size may, in addition to that, be doubled or bisected down to a minimum, i.e. the grid's mesh length. Possible moves are symbolized by

$$Y = \{\mathbf{F}, \mathbf{B}, \mathbf{L}, \mathbf{R}, \mathbf{S}, \mathbf{2F}, \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{F}\}\,.$$

They have, in the same order, the following meanings: move one step d_t forward, backward, left or right, stay at the current position, or move forward with doubled or bisected step length.

The sensor device in turn gets two objective values $\Phi(x_{t-1})$ and $\Phi(x_t)$ and maps them to the finite input alphabet (denoted by X) of the Mealy machine $\mathcal{A}$. Together with the the Mealy automaton, this mapping $\sigma : \mathbb{R}^2 \to X$ will undergo the Genetic Programming process.

Canonically a Mealy machine $\mathcal{A}$ is defined in terms of its input and output alphabets X and Y, an initial internal state s_1 from the set S, and finally its state transition function $\delta : X \times S \to S$ and output function $\gamma : X \times S \to Y$. Each of the sets is finite. As mentioned above, γ actually decides over which action is taken by the actor device.

At last, f from (1) is knocked down into the action of the three devices from figure 1. While the mapping of Y into actions, i.e. the actor device, is defined a-priori, the sensor mapping σ as well as the functions δ and γ will be subject to the Genetic Programming process.

4 METHODICAL ASPECTS

4.1 CODING OF PROGAM TREES

We try to solve the problem from section 2 by means of Genetic Programming; we choose to code both the function σ of the sensor device and the transition and output function of the Mealy machine in a single program tree. The nodes in this tree obey the syntactic rules defining a context-free grammar displayed in Backus-Naur-form in table 1:

cond	::=	$\geqslant_0(\langle\text{arith}\rangle, \{\langle\text{rule}\rangle\}, \{\langle\text{rule}\rangle\})$
rule	::=	$\boldsymbol{\gamma}(\langle\text{state}\rangle, \langle\text{output}\rangle) \mid \boldsymbol{\delta}(\langle\text{state}\rangle, \langle\text{state}\rangle) \mid \langle\text{cond}\rangle$
arith	::=	$\langle\text{arith}\rangle + \langle\text{arith}\rangle \mid \langle\text{arith}\rangle - \langle\text{arith}\rangle \mid \langle\text{arith}\rangle * \langle\text{arith}\rangle \mid \langle\text{arith}\rangle \,\%\, \langle\text{arith}\rangle \mid c \in \mathbb{R} \mid \boldsymbol{A} \mid \boldsymbol{B}$
state	::=	$s \in S$
output	::=	$y \in Y$

Table 1: Syntactic Rules for the Program Trees

The program trees are parametrized by formal variables $\boldsymbol{A}$ and $\boldsymbol{B}$ which are, during evaluation, replaced by actual values from $\Phi(x_t)$ and $\Phi(x_{t-1})$, respectively; c represents real-valued constants. The root node of every program tree we consider contains a conditional expression symbolized by $\geqslant_0$; depending on whether the embodied arithmetic expression is greater or equal to 0 or not, either the first or the second rule list is evaluated. Any such rule list may again contain conditional expressions, but it also contains output as well as state transition specifications wrapped in $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ expressions. States from S and outputs from Y are treated as terminal symbols. Arithmetic expressions are built from standard arithmetic operators common in Genetic Programming (cf. (Koza, 1992), (Michalewicz, 1996), (Langdon, 1998)). A simple program tree built in concordance with the upper rules is shown in figure 2. For the ease of presentation it just has state transitions, i.e. $\boldsymbol{\delta}$-specifications.

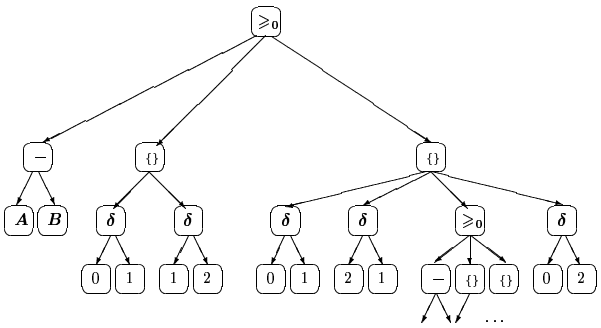


Figure 2: Sample Program Tree

As soon as a specific program tree has been evaluated, there is a set of state transitions and output instructions which are in turn used for the action of the Mealy machine. Since we are dealing with finite trees, there are only finite many cases which can be distinguished, that is, there are only finite many sets of output and state transition instructions. If we associate a certain input symbol with each of these sets, we are indeed dealing with finite state machines whose input alphabet's size changes during the evolutionary process.

There might be some confusion when more than one transition or output instruction is present for a single state of the machine. In the tree shown in figure 2, e.g., there are two instructions for state 0 in case of $\boldsymbol{A} - \boldsymbol{B} < 0$; these are $\boldsymbol{\delta}(0,1)$ and $\boldsymbol{\delta}(0,2)$. This conflict is resolved simply by letting the leftmost instruction precede.

4.2 GENETIC OPERATORS

The general procedure of Genetic Programming was slightly modified here in order to suit our purposes. It is a version of the breeding procedure from (Koza et al., 2000). The syntactic rules from table 1 impose that we have to use strongly typed operators (cf. (Montana, 1995), (Angeline, 1998)). We are engaging standard crossover and standard mutation as main operators; both of them affect randomly chosen subtrees with (in case of crossover) matching types of the subtree root nodes. Because the order of instructions does matter (cf. the previous paragraph), we additionally employ structural mutation operators, namely dupli-

cation, deletion and inversion. They affect the instruction lists by duplicating or deleting subtrees and by changing the order of subtrees. These secondary operators were mentioned by (Goldberg, 1989) and (Holland, 1992) in the Genetic Algorithm context and, e.g., by (Jacob, 1995) in the Genetic Programming context.

Because of limited computer power we have used a very small population size of 100 individuals and have made the Genetic Programming algorithm run over just 300 generations. These values are small compared to those of Koza (Koza, 1992) who routinely has several thousand individual programs in one population. Mutation probability was set to 60 %; crossover probability was ten times the probabilities of the structural mutation operators and only half the probability set for standard mutation.

5 EXPERIMENTAL ENVIRONMENTS

5.1 TEST FUNCTIONS

This section presents some sample experiments we have undertaken. The Genetic Algorithms settings have already been introduced in the previous section, so we will focus on the objective functions. In the next section we will present results that have been obtained in conjunction with the following well-known test functions Φ_k (where $k \in \{1, 2, 3\}$) from global optimization defined by (cf. (Hock and Schittkowski, 1981), (Törn and Žilinskas, 1989))

$$\Phi_1(x) := -50\mu_1 \sum_{i=1}^{n} x_i \sin\sqrt{50x_i}$$

$$\Phi_2(x) := \mu_2 \sum_{i=1}^{n} \left(\frac{1}{4}x_i^2 - 10\cos(\pi x_i) + 10\right)$$

$$\Phi_3(x) := \mu_3 \sum_{i=1}^{n-1} 100\left(\frac{1}{5}x_{i+1} - \frac{1}{25}x_i^2)\right)^2 + \left(\frac{1}{5}x_i - 1\right)^2$$

for values $x \in \Omega := [0, 10]^2$ and $n = 2$. These functions are scaled versions of functions which have already been used to evaluate Genetic Algorithms' and Evolutionary Programming's performance, too. Φ_1 is a modification of Schwefels function, Φ_2 corresponds to Rastrigins function; they are multimodal functions, cf. (Salomon, 1996). Φ_3 is a scaled Rosenbrock function which in this general form was taken from (Chellapilla and Fogel, 1997); its minimum is located on top of a narrow, bended ridge. The surfaces of Φ_1 and Φ_3 are depicted in figure 3.

Further on, we used scaling parameters μ_i such that the range of functions Φ_i equals $[0, 1]$. The fitness

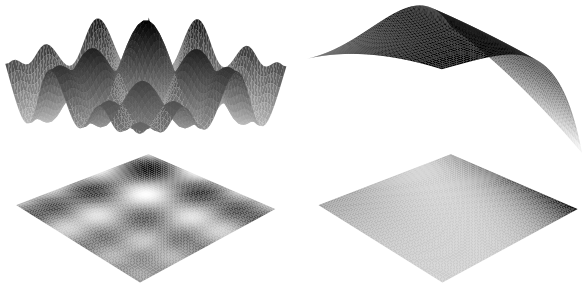


Figure 3: Test Functions Φ_1 and Φ_3

measures Ψ_k incorporated a value of $\nu = 10$; strategies were run over $T = 50$ steps, starting from 10 randomly chosen points as initial values in Ω. The Mealy machines themselves had a set of 10 internal states disposable.

5.2 COMPARISON TO OTHER EXPERIMENTS

It was mentioned earlier that (Jefferson et al., 1991), (Koza, 1992) and (Jacob, 1995) have undertaken so-called artificial ant experiments which in some sense are similar to our experimental work. In these experiments 'the task of navigating an artificial ant attempting to find all the food lying along an irregular trail' is considered.

While (Jefferson et al., 1991) have employed binary string-encoded finite state automata and neural networks to solve this problem, the latter have applied syntactic expressions. The basic differences to our experimental setting is that the ant had just one sensor which could differentiate between food, non-food and pheromone cells laying ahead, thus a comparison between sensor values was not available for the ant. In our experiments, however, controller are able to get an approximation for directional derivatives on their path by comparing function values $\Phi_i(t)$ and $\Phi_i(t-1)$, respectively.

The main difference from the experimental point of view lies in the fact that each ant was always beginning to move at the same position on the field relative to the point where the trail starts. So there was only one test case in contrast to our varying sets of test cases. (Koza, 1992) claims, however, that he relies 'on the various states of the ant that actually arise along the ant's actual trajectory to be sufficiently representative of the general trail following problem'; this targets on a peculiarity of the artificial ant's problem, that is to say, that an ant in any case has to learn how to bridge gaps or knight's moves in order to follow the trail.

As a matter of fact, however, the artificial ant problem can be inserted in the above framework. To this end, a discrete objective function Φ mapping the ant's grid of movement to three values has to be defined. The three values indicate if there is nothing, a food piece or a pheromone piece at the respective mesh position. Although the update rule (1) must be altered to

$$x_{t+1} = x_t + f\big(\Phi(x_t + d_t), \Phi(x_t)\big)$$

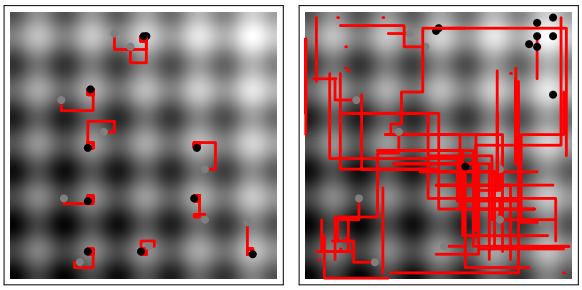
because the ant is always looking one step ahead, the underlying controller design can remain unchanged. The set Y of motions is broader that the set of motions originally used in artificial ant experiments.

6 RESULTING STRATEGIES

Now we exemplify strategies which have been evolved for the objectives Φ_k using fitness functions Ψ_k. To this end we, on the one hand side, display trajectories that have been produced by these strategies according to equation (1) as $(x_t)_{t=0,\dots,T} \subset \Omega$. Initial positions are marked by grey dots, whereas final positions are marked by black ones. We also plot average objective function values, where the average is taken over the actual positions within all of the 10 trajectories.

6.1 LOCAL OPTIMIZATION

First of all, we look at two example strategies s_1^* and s_2^* which were evolved using Ψ_2 as the fitness function. The trajectories of these strategies are plotted in figure 4a. and 4b. The first one is a typical local search strategy which is often evolved by the Genetic Programming method in this context, cf. figure 5.



a. Trajectories of s_1^* b. Trajectories of s_2^*

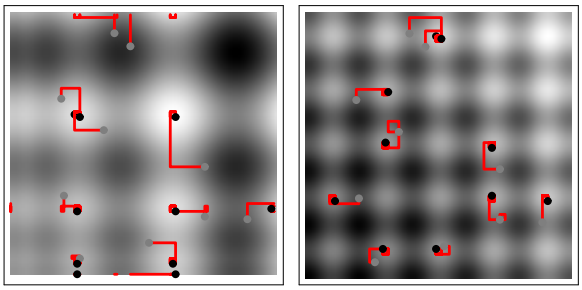
Figure 4: Strategies s_1^*, s_2^* Evolved Against Φ_2

These strategies find a peak near their starting position and retain this position from then on; they loop around peaks. But as a consequence, the average values obtained are far below the theoretical maximum of 1, since the strategies also loop around local maxima of low order. These are located near the lower left corner of the figures; generally such maxima are represented by relatively dark regions in the background of the trajectory diagrams.

6.2 GLOBAL OPTIMIZATION

The second strategy s_2^* which was evolved against Φ_2, too, is a very interesting one since its averages get much closer to 1 than the ones of s_1^*. This is due to the fact that it takes much bigger steps than s_1^* does at the beginning; so by successively doubling the length of d_t as described in section 3, this strategy is able to scan much larger areas of Φ_2.

As an effect, although it uses the same initial positions, 9 out of 10 final positions are located in the upper right quarter of Ω (i.e. where the 'better' local maxima are), 6 are located even in the immediate neighborhood of the global maximum. Any local optimizer could iterate into the global maximum from these points. As a result, this strategy has the potential of a global optimizer on Φ_2.



a. Trajectories of s_3^* on Φ_1 b. Trajectories of s_3^* on Φ_2

Figure 5: Strategy s_3^* Evolved Against Φ_1

It is worthwhile to analyze s_2^* in more detail. It is defined by the following expression:

$$\begin{aligned} s_2^* = {} & \geqslant_0\big(3\boldsymbol{A} - 2.282, \\ & \{\boldsymbol{\delta}(9,8), \boldsymbol{\delta}(4,5), \boldsymbol{\gamma}(6,\mathbf{2F}), \boldsymbol{\gamma}(5,\mathbf{S}), \boldsymbol{\delta}(6,3), \dots\} \\ & \{\boldsymbol{\delta}(4,0), \boldsymbol{\delta}(7,0), \boldsymbol{\delta}(0,1), \boldsymbol{\gamma}(5,\mathbf{L}), \\ & \quad \geqslant_0\big(2\boldsymbol{B}, \{\boldsymbol{\delta}(3,3), \boldsymbol{\delta}(1,9), \boldsymbol{\delta}(9,7), \boldsymbol{\gamma}(0,\mathbf{2F}), \\ & \qquad \boldsymbol{\delta}(5,0), \boldsymbol{\gamma}(9,\mathbf{R}), \boldsymbol{\gamma}(6,\tfrac{1}{2}\mathbf{F})\}, \\ & \qquad \{\dots\}\big), \\ & \boldsymbol{\gamma}(7,\mathbf{L}), \dots\}\big), \end{aligned}$$

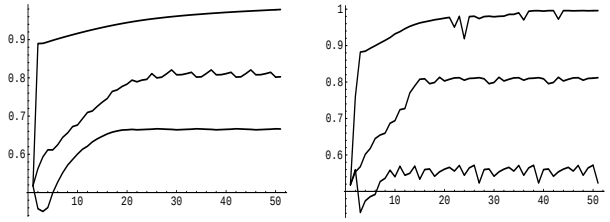
where the ellipsis ... stands for redundant rules because of shadowing. The reader may observe that state 6 will never be attained by s_2^*. For **S** is the default action if nothing is specified, s_2^* will stay at its current position if $3\Phi(x_t) \geqslant 2.282$ (**A** is replaced by $\Phi(x_t)$ during evaluation). Otherwise the controller will change from its standard initial value 0 into state 1 with **2F** being output (**B** is replaced by $\Phi(x_{t-1})$

and $2\Phi(x_{t-1}) \geqslant 0$ is true in any case). If $3\Phi(x_t) < 2.282$ still holds, the consecutive states are 9, 7 and 0 again, with outputs **S**, **R** and **L**. This produces the zig-zagging behaviour the reader may observe in figure 4 b. It stops as soon as $\Phi(x_t)$ exceeds a threshold value of 0.761.

The analysis of s_2^* shows that this strategy does not only use its sensors to compare objective function values. Rather it has learned that the the maxima of Φ_2 are located on an orthogonal grid which can be searched effectively by the above-mentioned zig-zagging strategy.

6.3 ROBUSTNESS

Here we indicate a strategy as a robust one if it shows reasonable good performance if tested against an objective function differing from the one the strategy originally was evolved on. Of course, the functions have to be similar in some respect: all the Φ_i are differentiable, deterministic and are scaled to the same range $[0, 1]$. They mainly differ in the number of local extrema and their (non-)insularity; the multimodal ones (Φ_1, Φ_2) differ in the scale of the respective distances between extrema, but are similar in that their extrema are located on orthogonal grids. So we expect that at least local optimizers (which are not specifically adjusted to the distances between local extrema of Φ_1 and Φ_2) should work well on either of these functions. Is is quite surprising that a strategy evolved against Φ_1 can also do well when operating on Φ_3, because the latter function has only one maximum which is (in contrast to the extrema of Φ_1) not at all insulated.

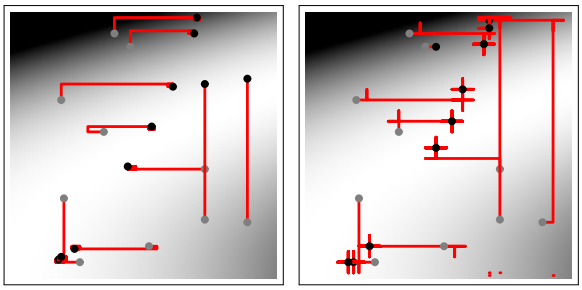


a. Averages for s_3^* b. Averages for s_4^*

Figure 6: Average Performances

We compare the performance of strategies s_3^* and s_4^*. s_3^* was evolved on Schwefel's function Φ_1; it can be interpreted as a local optimization strategy finding and looping on peaks near the initial position (see figure 5). On Rastrigin's function its behaviour looks very similar, although this function has more local maxima being closer to each other than the maxima of function Φ_1 are. But what is most impressing is the fact that s_3^* does almost as good on Φ_3 as the 'specialist' s_4^* does which actually has been evolved against Φ_3; the reader may observe this by comparing the top lines in figure 6a. and 6b. The lowermost lines in these figures represent averages on Rastrigin's function, and the lines in the middle are performance averages for Schwefel's function.

The trajectories of s_3^* and s_4^* on Φ_3 can be compared in figure 7. Obviously, during its evolution on Φ_1 strategy s_4^* has learned to perform big steps initially, which is a valuable capability on Φ_3, too.



a. Trajectories of s_3^* on Φ_3 b. Trajectories of s_4^* on Φ_3

Figure 7: Strategies s_3^*, s_4^* Tested Against Φ_3

7 CONCLUSIONS

In this work we have formulated an optimization problem as a problem of optimal control of Mealy machines. We have given the design and coding of a controller and a suitable objective in order to be used in conjunction with methods of Genetic Programming. These methods have produced some illustrative examples of strategies which were robust against changes regarding initial positions as well as regarding the overall objective function to be maximized; a potential global optimization strategy was found, too.

Deliberately the experiments have been performed with limited effort regarding the controllers' capabilities as well as the computational effort (as compared to other Genetic Programming efforts) since the controllers establish just one part of a more extensive ecological model. After all, we observed a learning effect of the controllers concerning special features of the involved functionals as well as concerning optimization strategies. Further testings of the design, and a successive refinement of its capabilities and a better understanding of the theoretical background of Genetic Programming in the ecological simulation context is subject to current research.

Acknowledgements

The first author thanks the *Evangelisches Studienwerk Villigst e.V.* deigning a scholarship for a PhD project this work is part of. We thank the anonymous reviewers for their useful hints and suggestions that helped refining this paper.

References

Angeline, P. J. (1998). A historical perspective on the evolution of executable structures. *Fundamenta Informaticae*, 36(1–4):179 – 195.

Chellapilla, K. and Fogel, D. (1997). Two new mutation operators for enhanced search and optimization in evolutionary programming. In B.Bosacchi, J.C.Bezdek, and D.B.Fogel, editors, *Applications of Soft Computing*, volume 3165 of *Proc. SPIE*, pages 260–269.

Glowinski, R. and LeTallec, P. (1989). *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. SIAM, Philadelphia.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Redwood City.

Hauhs, M. and Lange, H. (1996). Ecosystem dynamics viewed from an endoperspective. *The Science of the Total Environment*, 183:125–136.

Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*, volume 187 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems - An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge.

Jacob, C. (1995). *MathEvolvica, Simulierte Evolution von Entwicklungsprogrammen der Natur*. Arbeitsberichte des Instituts für mathematische Maschinen und Datenverarbeitung (Informatik), Universität Erlangen.

Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., and Wang, A. (1991). Evolution as a theme in artificial life: The genesys/tracker system. volume 10 of *SFI Studies in the Sciences of Complexity*, pages 549 – 578, Redwood City. Addison-Wesley.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

Koza, J. R., Keane, M. E., Yu, J., Bennett III, F. H., and Mydlowec, W. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1:121 –164.

Langdon, W. (1998). *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Kluwer Academic Publishers, Amsterdam.

Lange, H. (1999). Are ecosystems dynamical systems ? *International Journal of Computing Anticipatory Systems*, 3:169 – 186.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, 3 edition.

Montana, M. J. (1995). Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199 – 230.

Salomon, R. (1996). Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. *BioSystems*, 39(3):263–278.

Törn, A. and Žilinskas, A. (1989). *Global Optimization*. Number 350 in Lecture Notes in Computer Science. Springer Verlag, Berlin.

Genetic Programming solution of the convection-diffusion equation

Daniel Howard and **Simon C. Roberts**
Software Evolution Centre,
Systems and Software Engineering Centre,
Defence Evaluation and Research Agency,
Malvern, WORCS WR14 3PS, UK.
dhoward@dera.gov.uk, Tel:+44 1684 894480

Abstract

A version of Genetic Programming (GP) is proposed for the solution of the steady-state convection-diffusion equation which neither requires sampling points to evaluate fitness nor application of the chain rule to GP trees for obtaining the derivatives. The method is successfully applied to the equation in one space dimension.

1 Introduction

This paper proposes a way to use Genetic Programming (GP) to model the interaction between convective and diffusive processes. Modelling this interaction is vital to the fields of Heat Transfer, Fluid Dynamics, and Combustion, and remains one of the most challenging tasks in the numerical approximation of differential equations.

The new method is applied to the simplest model problem, the steady-state version of the convection-diffusion equation in one space dimension. This linear differential equation has two Dirichlet boundary conditions at the end points of the interval $0 \leq x \leq 1$,

$$\begin{aligned} \frac{d^2T}{dx^2} - Pe\,\frac{dT}{dx} &= 0 \\ T(0) &= 1 \\ T(1) &= 0 \end{aligned} \qquad (1)$$

It involves derivatives of the temperature (T) and the Peclet number (Pe), which is a measure or ratio of convection to diffusion and a parameter which determines the sharpness of the boundary layer at $x = 1$.

In 1992, Koza (Koza, 1992) described a GP method to find the solution of a differential equation. It evolved a GP tree or solution to the equation, and applied the chain rule to the GP tree to obtain its derivatives. The fitness measure used a weight factor to balance the ability of the function to satisfy the initial condition with the ability of the derivatives of the function to satisfy the differential equation at a number of sampled points. The technique was illustrated with reference to initial-value problems.

In common with all other numerical methods, a straightforward application of this method to search for the numerical approximation to the solution of the convection-diffusion equation ($\hat{T}$), suffers with numerical difficulties. Very early in the run, the division operator produces steep gradients and approximations with high fitness of the form:

$$\hat{T} = 1 - \frac{x}{x + \epsilon}$$

emerge which for arbitrarily small $\epsilon > 0$ result in a temperature that becomes zero almost everywhere and which also exactly satisfies the boundary conditions. Such solutions dominate and it becomes extremely difficult to adjust the weight factor to accentuate the large error in the satisfaction of the differential equation at $x = 0$ (Howard, 1998).

Although approximations to the true solution, rather than to this kind of trivial solution, were achieved by removing the protected division, leaving $+$, $-$, and $*$ in the function set, the method was slow to converge and chain rule evaluation of derivatives of GP trees was an expensive step.

2 Proposed approach

The analytical solution of equation (1) is:

$$T = \frac{\exp(xPe) - \exp(Pe)}{1 - \exp(Pe)} \qquad (2)$$

The task is to find $\hat{T}$, an approximation to T. A polynomial p is evolved, and by polynomial division it can

be transformed such that the resulting expression for $\hat{T}$ always satisfies the boundary conditions exactly, e.g.

$$\hat{T} = x(1-x)p + (1-x) \tag{3}$$

and by the Remainder Theorem all polynomials are guaranteed. The derivatives of $\hat{T}$ are given by:

$$\frac{d\hat{T}}{dx} = x(1-x)\frac{dp}{dx} + (1-2x)p - 1$$
$$\frac{d^2\hat{T}}{dx^2} = x(1-x)\frac{d^2p}{dx^2} + 2(1-2x)\frac{dp}{dx} - 2p$$

and a GP method can take the negative of the square integral of the left hand side of the differential equation as its Darwinian fitness F,

$$F = -\int \left(\frac{d^2\hat{T}}{dx^2} - Pe\,\frac{d\hat{T}}{dx} \right)^2 dx \tag{4}$$

which is a least squares measure of the error of approximation. The integral expression for the fitness F can be obtained analytically because $\hat{T}$ is polynomial and this is a point of difference with the method in (Koza, 1992) which used a number of points in the domain to sample the fitness.

Although from a theoretical standpoint the uniform norm or infinity norm:

$$\left\| \frac{d^2\hat{T}}{dx^2} - Pe\,\frac{d\hat{T}}{dx} \right\|_\infty = \max^{x\in[0,1]} \left| \frac{d^2\hat{T}}{dx^2} - Pe\,\frac{d\hat{T}}{dx} \right|$$

is preferable in F because it can warn of δ like spikes, it requires a separate optimisation procedure to find the maximum. Numerical experiments, however, were successful with F as defined in equation 4.

3 Representation

Considering equation 1, a GP method can combine ephemeral random constants to evolve the coefficients

$$[a_0, a_1, a_2, ...] \tag{5}$$

to obtain the univariate polynomial p

$$p = a_0 + a_1x + a_2x^2 + a_3x^3.... \tag{6}$$

that can be substituted into equation 3. Evaluation of the integral in equation 4 requires expressions for:

$$\int \frac{d\hat{T}}{dx}\frac{d\hat{T}}{dx}dx,\ \int \frac{d^2\hat{T}}{dx^2}\frac{d^2\hat{T}}{dx^2}dx \text{ and } \int \frac{d\hat{T}}{dx}\frac{d^2\hat{T}}{dx^2}dx$$

all of which can be obtained by shifting and modifying the coefficients in equation 5 and by multiplication of these small vectors. The limits of integration are at $x = 0$ and at $x = 1$ which means that there is no loss of accuracy involved in computing the integral even if p is a polynomial of high order.

Table 1: GP terminals, functions and variables

parameter	setting
functions	+, -, *, / ($x/0 = 1$), ADD, $BACK$, $WRITE$, Wm_1, Wm_2, Rm_1, Rm_2
terminals	S_C
globals	variable length solution vector pointers L and C memories m_1 and m_2
max tree size	1000 nodes

Although at first glance the Genetic Algorithm seemed a good choice, the requirement to generate a variable length vector of very precise coefficients favoured the Genetic Programming method.

The GP formulation of table 1 was devised. In this formulation, the GP tree generates the required variable length vector as it is being evaluated by combining ephemeral constants to produce very accurate coefficients; furthermore, the return value of the GP tree has no meaning. During evaluation, functions in the GP tree manipulate a vector of coefficients (equation 5) in global memory. The functions, as described in the next paragraph, manipulate L and C, two global pointers to the element or position in the vector of coefficients. Pointer L stands for "last index" or tail position, and pointer C stands for "current" position. Prior to the evaluation of the GP tree, L and C are both set to zero.

Functions *ADD*, *BACK* and *WRITE* are functions of two arguments. They return one of the arguments, e.g. *ADD* returns its second, *BACK* its first and *WRITE* its second argument, the choice is arbitrary. Function *ADD* writes its first argument to the vector element pointed by L. It increments L provided $L < L_{MAX}$ and enforces $C = L$. Function *BACK* decrements pointer C provided $C > 0$. Function *WRITE* overwrites the vector element at C with its first argument. Also, if $C < L_{MAX}$ it increments this pointer and if $C > L$ it increments pointer L.

The function set is enhanced with two memories m_1 and m_2 manipulated by functions again of two arguments. Functions Wm_I return their first argument and over write their second argument to memory location m_I. Functions Rm_I simply return the value of the memory at m_I and ignore both of their arguments.

Table 2: GP run parameters

parameter	setting
population	8000
kill tournament	size 2 for steady-state GP
breed tournament	size 4 for steady-state GP
regeneration	80% x-over, 20% clone,
fitness measure	$-\int(\frac{d^2\hat{T}}{dx^2} - Pe\ \frac{d\hat{T}}{dx})^2 dx$

Table 3: Information about highly successful GP runs

Pe	pop	gens	best F	avg tree	mins
5	8000	42	-0.000175	55	24.23
20	8000	80	-0.003145	348	72.11
50	8000	80	-0.042962	830	139.61
100	2000	300	-0.258476	958	228.86

An ephemeral random constant C_S is stored as one byte and can represent up to 256 values. These are equally spaced and obtained by dividing the numbers 0 to 255 by 255, to obtain values in the range $[0, 1]$.

4 Moderate Peclet Numbers

Parallel independent runs of steady-state Genetic Programming obtain solutions for a Peclet number with parameters as in Table 2. The search becomes progressively difficult with Peclet number because the desired polynomial is of higher and higher order. Information for some of the more successful runs carried out on an 850MHz Pentium III PC is provided in table 3.

There is a steady increase in the average size of tree with Pe as well as steady increase in time required to obtain an acceptable solution. There is an increase in the number of coefficients also. At $Pe = 5$ only seven coefficients are obtained, see table 4, while for $Pe = 20$ twenty six coefficients are produced, see table 5.

The nature of the approximation is reflected in Figure 1, i.e. an approximation driven by a least squares process. As is typical, the approximation is characterised by minute oscillations (apparent in the magnified graph at the bottom of the figure). However, that is fine as the scheme is developed for quantitative accuracy, not for qualitative shape, and aims to locate the boundary layer at the expense of maintaining a property such as monotonicity, for example. If a desired shape property were to be required, this might be accomplished by modification of the fitness measure, or by evolution of the coefficients to a more complex type of base function which enjoys and enforces the desired property.

Table 4: $Pe = 5$, table 3, evolved 7 coefficients.

I	a_{I+0}	a_{I+1}	a_{I+2}	a_{I+3}
a_0	0.964706	0.882353	0.713725	0.717647
a_4	0.170588	0.137255	0.447377	

Table 5: $Pe = 20$, table 3, evolved 26 coefficients.

I	a_{I+0}	a_{I+1}	a_{I+2}	a_{I+3}
a_0	0.996078	0.972549	1.24314	0.615686
a_4	0.752941	1.77528	0.980392	0.588235
a_8	0.745098	0.611765	2.24359	0.931927
a_{12}	0.586275	0.858824	0.462745	0.92549
a_{16}	0.74902	0.374761	0.374761	0.733333
a_{20}	0.374761	0.410748	0.0313725	0.588235
a_{24}	0.0313725	0.0313725		

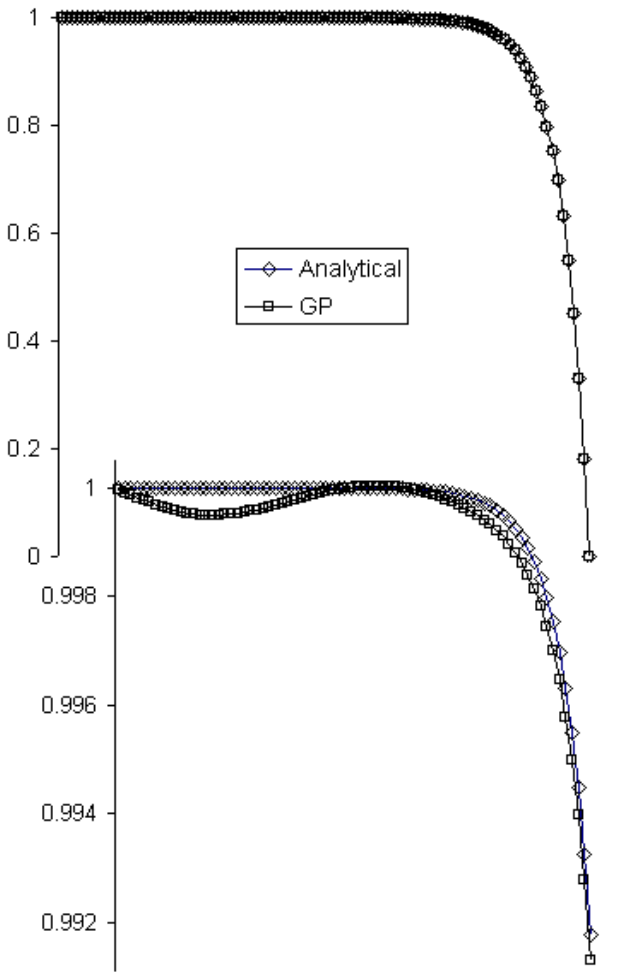


Figure 1: Approximation at $Pe = 20$.

Different combinations of ephemeral random constants were tested but no clearly superior choice emerged. For example, when constants were varied from 0.0 to 0.001 the resulting coefficients were much smaller than when constants were varied from -1 to 1, but without appreciable difference in accuracy or effort required to obtain a solution.

5 Further work

The remainder of this paper presents ideas and motivations for developing this approach further.

5.1 High Peclet Numbers

For high Peclet number, e.g. $Pe > 50$, an adequate approximation to the solution or equation 2 is:

$$T = 1 - x^{Pe}$$

which corresponds to the following polynomial for p,

$$p = 1 + x + x^2 + x^3 + ... + x^{Pe-2}$$

For the very large Pe, such as $Pe = 1000$, the global fitness maximum resides where the vector in equation 5 has circa 1000 coefficients. At high Pe a local maximum at $p = 0$, i.e. at $T = 1 - x$ attracts the search. Polynomials with far fewer number of coefficients than 1000 are attracted to this local maxima. Thus, unsuccessful approximations for high Peclet number try to improve on $T = 1 - x$ through a relatively small number of coefficients. They use $T = 1 - sx$ (the slope s is near one) over a significant portion of the domain and exhibit a small boundary layer behaviour near $x = 1$ for example.

For very large Pe, the present scheme is not producing enough genetic material to generate a sufficient number of coefficients (equation 5) to enable the evolutionary process to see the global minimum. The following tactics may help overcome this, with the motivation of solving practical engineering problems:

1. The convection-diffusion problem at a Peclet number lower than required is solved, and the resulting population is used as a starting point to evolve the solution to the desired Peclet number. This is called *continuation* and can be implemented in a variety of ways.

2. Build individuals in the initial population with sufficient genetic material to allow them to generate vectors (equation 5) with close to the required number of terms for the required Peclet number.

3. Use of evolutionary techniques which maintain genetic diversity and prevent similarity of solutions. Special mutation operations could copy material to increase the resulting number of coefficients.

4. Changing the landscape. The fitness measure could be replaced by the logarithm of equation 4 to diminish the effect of the ridge or hump in the fitness landscape. However, such a choice would not have effect with tournament selection for instance because it cannot alter selection which is based on ranking.

A GP formulation with ADFs was experimented with but did not significantly improve performance.

5.2 Other polynomials

Simple polynomials are not the only option. Chebyshev and Legendre polynomials are popular for high order regression and could serve as the basis functions ϕ for the scheme where $p = a_i\phi_i$, and can easily be analytically differentiated and integrated.

5.3 Improved functions

The GP functions ADD, $BACK$ and $WRITE$ could be enhanced with more powerful data manipulation functions that could introduce or modify more than one coefficient at a time or apply an operator, e.g. to sort groups of coefficients. The list of pointers L and C could be enhanced with more complex pointers.

5.4 Evolution of the phenotype

The coefficients (equation 5) can be considered the phenotype, and the GP trees the genotype. An evolutionary algorithm could be applied directly to improve upon a group of successful phenotype. This as a final post-processor because there is no way to incorporate the improvement back into the genotype, i.e. the evolutionary process is not Lamarckian.

5.5 Partial differential equations (PDEs)

Extension of the method to solve the steady-state convection-diffusion method in two space variables would open the road for application to the steady-state Heat Transport and Navier-Stokes equations. This section suggests a way to achieve this for problems which possess a regular geometry.

The steady-state convection-diffusion equation in two space variables can be handled in a similar way to the equation in one space variable. For illustration,

consider a square heated on one of its sides:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} - \mathrm{Pe}(\frac{\partial T}{\partial x} + \frac{\partial T}{\partial y}) = 0$$
$$T(x=0) = 1.0$$
$$T(x=1) = 0.0$$
$$T(y=0) = 0.0$$
$$T(y=1) = 0.0$$

A Dirichlet boundary condition on a line or curve defined by the function $g(x, y)$ can be enforced with an exponential term such as $\exp(-\Gamma g^2)$, where Γ is a large constant. The following expression for $\hat{T}$ would then seem appropriate,

$$\hat{T} = xy(1-x)(1-y)p + \exp(-\Gamma x^2)$$

where perhaps $\Gamma > 10^4$, such that the term to which it belongs is effectively zero except for $x = 0$ when it becomes unity. Polynomial p is in $x^i y^j$ with GP evolution of a_{ij} its coefficients. However, evaluation of F involves cross multiplication of $x^i y^j$ terms with the exponential term in T, and requires an analytical expression for the following integral,

$$I_n = \int x^n \exp(-\Gamma x^2) dx \tag{7}$$

It can be approximately integrated by exploiting a recursive relationship hence the label I_n. For $n = 0$ and $n = 1$ the integral I_0 can be computed with the error function erf(x), and the integral for I_1 is straight forward,

$$I_0 = \int \exp(-\Gamma x^2) dx = \frac{1}{\Gamma} \int \Gamma \exp(-\Gamma x^2) dx$$
$$= \Gamma \int \exp(-u^2) du = \frac{\Gamma\sqrt{\pi}}{2} \mathrm{erf}(1.0)$$
$$I_1 = \int x \exp(-\Gamma x^2) dx = -\frac{\exp(-\Gamma x^2)}{2\Gamma}$$

the error function, erf (x=1) can be calculated approximately by carrying the series to an appropriate number of terms,

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \Big(x - \frac{x^3}{3 \cdot 1!} + \frac{x^5}{5 \cdot 2!} - \frac{x^7}{7 \cdot 3!} + ... \Big)$$

The recursive relationship to compute I_n is

$$2\Gamma I_n = -x^{n-1} \exp(-\Gamma x^2) + (n-1) I_{n-2}$$

Alternatively, an expression for $\hat{T}$ that is valid only for $x \geq 0$ and which seems appropriate is

$$\hat{T} = xy(1-x)(1-y)p + \exp(-\Gamma x)$$

F now requires the analytical solution (Abramowitz and Stegun, 1964) to the following integral

$$\int x^n \exp(-\Gamma x) dx =$$
$$\frac{\exp(-\Gamma x)}{\Gamma^{n+1}} \Big[(\Gamma x)^n - n(\Gamma x)^{n-1} + n(n-1)(\Gamma x)^{n-2}$$
$$+ ... + (-1)^{n-1} n(\Gamma x) + (-1)^n n \Big]$$

Such algebraic expressions are tedious to implement (see Appendix) but once coded result in an effective algorithm. More work is also required to handle problems with mixed boundary conditions and complex geometry.

6 Why GP?

The reader may ask himself: "why investigate GP solution of differential equations when many popular commercial packages already exist to solve these equations?"

Such packages use the weighted residuals method (WRM). Popular WRMs are the finite differences method (FDM), the finite volume method (FVM), the finite element method (FEM), and the Boundary Element Method (BEM).

The only motivation for investigating an evolutionary method is for approximating the solution to non-self-adjoint multi-dimensional equations, e.g. Navier-Stokes equations, because the WRM cannot always conclusively solve these problems. The remaining sections outline potential advantages of the evolutionary method with respect to the WRM.

6.1 Mathematics

Numerical solution of self-adjoint differential equations (e.g. elliptic equations with even order derivatives) via WRM, (e.g. Galerkin FEM, cell centered FVM method, central difference FDM) is "optimal". This means that schemes converge to the analytical solution uniformly as the mesh is refined, and/or as the order of approximation of the functions in the WRM is increased. Applications relate to engineering design of edifices, structures and bridges with the WRM, and in particular with the Galerkin FEM.

The WRM, however, looses its "optimal" behaviour when applied to non-self-adjoint boundary value differential equations essential to Heat Transfer, Fluid Dynamics, and Combustion and results in unstable solutions containing "wiggles" (Gresho, 1981). The

numerical difficulty is linear in nature and cannot really be analysed for non-linear PDEs, e.g. the Navier-Stokes equations.

Both engineers and mathematicians have postulated special methods for dealing with these equations in the WRM framework. Notable examples are Petrov-Galerkin FEM, cell vertex FVM, and upwind differencing FDM. These methods are only optimal for the linear equation in one space variable (Morton, 1995). Application to PDEs, only in the most specialised but trivial of cases is optimal, e.g. if the scheme coincides with a clear directional characteristic of the solution.

Using such special methods in more than one space variable, i.e. on PDEs, finds solution but to a more diffuse PDE than that intended. Approximations can look deceptively smooth. Consequently, a physical experiment is required to calibrate the numerical method, when the original objective was for the numerical method to predict the outcome of the equivalent physical experiment.

It is very important to realize that the GP approximation is free from this fundamental mathematical drawback of WRM. Accuracy is an important motivation to investigate new solution approximation methods.

6.2 Memory

Every WRM requires either a mesh composed of a number of mesh points, or the presence of internal points (in the case of the Boundary Element Method) to solve the non-self-adjoint boundary value problem. The larger the number of points the more accurate will be the result.

The mesh and points introduce computational complexities and trade-offs: cell aspect ratio distortion, indirect memory addressing, rapid growth in the number of operations required to solve the matrix system, conditioning of the matrix in the case of iterative matrix solution methods, etc. Finally, adaptive methods for mesh refinement must be devised to track a solution by correcting the mesh most economically.

The GP scheme presented in this paper does not use any sampling points and does not require a mesh. Consequently, complicated algorithms to handle memory addressing are not required.

6.3 Order of approximation

High order WRM (quadratic finite elements and high order finite differences) increase the bandwidth of the resulting matrix system to be solved precluding their practical use in solution of equations in three space variables (3D problems). In addition, Petrov-Galerkin methods and multi-grid methods are next to impossible to construct in 3D with higher order FEMs.

The least squares FEM essentially squares the equations to restore ellipticity, is a credible alternative to a Petrov-Galerkin method, and handles higher order elements in a straight forward manner (Bochev, 1998). However, squaring the PDE must cause a very significant increase in the matrix bandwidth.

Consequently, and for 3D problems, WRM usually requires millions of mesh points with the low order linear approximation.

If exponential functions, or equivalent high order polynomials, could be precisely located in boundary layers then very few mesh points would be required - a panacea for WRM practitioners.

The GP method proposed in this paper, shares with the method proposed by Koza (Koza, 1992), an ability to discover and to construct for itself whatever order of approximation is required to solve the problem that is presented to it.

6.4 Parallel computing

Parallelization of the WRM is problematic, and normally achieved with domain decomposition methods which must carefully balance processor communication, process startup time, and work load.

In contrast, Genetic Programming easily lends itself to efficient parallel implementation (Koza, 1992) and when combined with the method in (Nordin, 1994) can achieve significant performance gains.

7 Conclusions

A novel GP method is developed to model convection-diffusion problems, which evolves a variable length vector of polynomial coefficients. Its fitness uses the integral of squared error, which has the advantage of not requiring sampling points nor derivatives of GP trees. Experiments solve the steady convection-diffusion equation in one space variable. This copes easily from $Pe = 5$ to $Pe = 100$ but encounters computational difficulty for higher Pe. Even so, potentially, the method has advantages over popular WRMs.

This method cannot be recommended as a serious alternative for solving these problems until schemes are found to obtain results at higher Pe; to develop techniques for solution on complex geometries in two and three space variables (Irons, 1966); and to handle both Neumann and Dirichlet boundary conditions.

Acknowledgments

This paper has benefited from the comments and suggestions of Robert Whittaker, Richard Brankin, Bill Langdon, and Joseph Kolibal.

References

M. Abramowitz and I. A. Stegun (1964): Handbook of Mathematical Functions, Dover Publications Inc., New York.

P. Bochev and M. Gunzburger (1998): Finite Element methods of least squares type, SIAM Review 40, 789-837.

P. Gresho and R. L. Lee (1981): Don't Suppress the Wiggles: They are telling you something, Comput. & Fluids, 9, 223-253.

D. Howard (1998): Late Breaking Papers of the GP 98 conference, Madison, Wisconsin.

B. M. Irons (1966): Engineering Application of Numerical Integration in Stiffness Method, Journal of the American Institute of Aeronautics and Astronautics, 14, 2035-2037.

J. R. Koza (1992): Genetic Programming: on the programming of computers by means of natural selection. MIT Press.

K. W. Morton (1995): Numerical Solution of Convection Diffusion Problems, Applied Mathematics and Computation 12, Chapman and Hall.

P. Nordin (1994): A Compiling Genetic Programming System that Directly Manipulates the Machine Code, Advances in Genetic Programming, ed. Kenneth Kinnear Jr., MIT Press.

Appendix

This appendix pertains to section 5.5 and the possibility of extending the method to PDEs.

The calculation of F for the steady-state convection-diffusion equation in two space variables (square box heated on one of its sides) requires algebraic manipulation. A change of notation makes for a less cluttered presentation, i.e. the polynomial coefficients a_{ij} are represented as a_A^B:

$$p = a_0^0 + a_1^0 x + ... + a_A^B x^A y^B + ...$$

Expressing the temperature T as

$$T = T_P + e$$

$$\begin{aligned} T &= gp + e \\ T_P &= gp \\ e &= \exp(-\Gamma x^2) \\ g &= xy(1-x)(1-y) \end{aligned}$$

spatial derivatives for temperature can be obtained by the chain rule:

$$\begin{aligned} \frac{\partial T_P}{\partial x} &= p\frac{\partial g}{\partial x} + g\frac{\partial p}{\partial x} \\ \frac{\partial T_P}{\partial y} &= p\frac{\partial g}{\partial y} + g\frac{\partial p}{\partial y} \\ \frac{\partial^2 T_P}{\partial x^2} &= p\frac{\partial^2 g}{\partial x^2} + 2\,\frac{\partial g}{\partial x}\frac{\partial p}{\partial x} + g\frac{\partial^2 p}{\partial x^2} \\ \frac{\partial^2 T_P}{\partial y^2} &= p\frac{\partial^2 g}{\partial y^2} + 2\,\frac{\partial g}{\partial y}\frac{\partial p}{\partial y} + g\frac{\partial^2 p}{\partial y^2} \end{aligned}$$

and when applying the chain rule to g, to p and to e a number of expressions follow:

$$\begin{aligned} g &= xy - x^2 y - xy^2 + x^2 y^2 \\ \frac{\partial g}{\partial x} &= y - 2xy - y^2 + 2xy^2 \\ \frac{\partial g}{\partial y} &= x - 2xy - x^2 + 2x^2 y \\ \frac{\partial^2 g}{\partial x^2} &= -2y + 2y^2 \\ \frac{\partial^2 g}{\partial y^2} &= -2x + 2x^2 \\ \frac{\partial e}{\partial x} &= -2\,\Gamma x \exp(-\Gamma x^2) \\ \frac{\partial^2 e}{\partial x^2} &= -2\,\Gamma \exp(-\Gamma x^2) + 4\,\Gamma^2 x^2 \exp(-\Gamma x^2) \end{aligned}$$

Those terms which are polynomial can be expressed in terms of coefficients a_A^B of the polynomial p which is evolved by Genetic Programming. After algebraic manipulation coefficient expressions are arrived at:

$$\begin{aligned} \Big[p\frac{\partial g}{\partial x}\Big]_A^B &= 2\big(a_{A-1}^{B-2} - a_{A-1}^{B-1}\big) + \big(a_A^{B-1} - a_A^{B-2}\big) \\ \Big[g\frac{\partial p}{\partial x}\Big]_A^B &= {}_{(A-1)}\big(a_{A-1}^{B-2} - a_{A-1}^{B-1}\big) + {}_A\,\big(a_A^{B-1} - a_A^{B-2}\big) \\ \Big[p\frac{\partial^2 g}{\partial x^2}\Big]_A^B &= 2\big(a_A^{B-2} - a_A^{B-1}\big) \\ \Big[\frac{\partial g}{\partial x}\frac{\partial p}{\partial x}\Big]_A^B &= {}_{(A+1)}\big(a_{A+1}^{B-1} - a_{A+1}^{B-2}\big) + \\ &\qquad {}_{2A}\big(a_A^{B-2} - a_A^{B-1}\big) \\ \Big[g\frac{\partial^2 p}{\partial x^2}\Big]_A^B &= {}_{A(A-1)}\big(a_A^{B-2} - a_A^{B-1}\big) + \\ &\qquad {}_{A(A+1)}\big(a_{A+1}^{B-1} - a_{A+1}^{B-2}\big) \end{aligned}$$

giving expressions in the polynomial coefficients:

$$-\mathrm{Pe}\Big[\frac{\partial T_P}{\partial x}\Big]_A^B = \mathrm{Pe}_{(A+1)}\big(a_{A-1}^{B-1} - a_{A-1}^{B-2} + a_A^{B-2} - a_A^{B-1}\big)$$

$$\Big[\frac{\partial^2 T_P}{\partial x^2}\Big]_A^B = {}_{(A^2+3A+2)}\big(a_{A+1}^{B-1} - a_{A+1}^{B-2} + a_A^{B-2} - a_A^{B-1}\big)$$

A new notation Δ or difference of a pair of coefficients a_A^B is introduced, e.g.:

$$\Delta_{+K}^A = a_{A+K}^{B-1} - a_{A+K}^{B-2} \text{ and } \Delta_{+K}^B = a_{A-1}^{B+K} - a_{A-2}^{B+K}$$

and used to define c_A^B polynomial coefficients of:

$$c_A^B = \Big[-\mathrm{Pe}\frac{\partial T_P}{\partial x} + \frac{\partial^2 T_P}{\partial x^2}\Big]_A^B = (\mathrm{Pe}A + \mathrm{Pe})\Delta_{-1}^A + (A^2 + 3A + 2 + \mathrm{Pe}A + \mathrm{Pe})\Delta_0^A + (A^2 + 3A + 2)\Delta_{+1}^A$$

Similar expressions can be obtained for the derivatives in the second space variable:

$$\Big[p\frac{\partial g}{\partial y}\Big]_A^B = 2\big(a_{A-2}^{B-1} - a_{A-1}^{B-1}\big) + \big(a_{A-1}^B - a_{A-2}^B\big)$$

$$\Big[g\frac{\partial p}{\partial y}\Big]_A^B = {}_{(B-1)}\big(a_{A-2}^{B-1} - a_{A-1}^{B-1}\big) + {}_B\,\big(a_{A-1}^B - a_{A-2}^B\big)$$

$$\Big[p\frac{\partial^2 g}{\partial y^2}\Big]_A^B = 2\big(a_{A-2}^B - a_{A-1}^B\big)$$

$$\Big[\frac{\partial g}{\partial y}\frac{\partial p}{\partial y}\Big]_A^B = {}_{(B+1)}\big(a_{A-1}^{B+1} - a_{A-2}^{B+1}\big) + {}_{2B}\big(a_{A-2}^B - a_{A-1}^B\big)$$

$$\Big[g\frac{\partial^2 p}{\partial y^2}\Big]_A^B = {}_{B(B-1)}\big(a_{A-2}^B - a_{A-1}^B\big) + {}_{B(B+1)}\big(a_{A-1}^{B+1} - a_{A-2}^{B+1}\big)$$

yielding expressions in the polynomial coefficients:

$$-\mathrm{Pe}\Big[\frac{\partial T_P}{\partial y}\Big]_A^B = {}_{\mathrm{Pe}(B+1)}\big(a_{A-1}^{B-1} - a_{A-2}^{B-1} + a_{A-2}^B - a_{A-1}^B\big)$$

$$\Big[\frac{\partial^2 T_P}{\partial y^2}\Big]_A^B = {}_{(B^2+3B+2)}\big(a_{A-1}^{B+1} - a_{A-2}^{B+1} + a_{A-2}^B - a_{A-1}^B\big)$$

and by using Δ the polynomial coefficients as before:

$$\Big[-\mathrm{Pe}\frac{\partial T_P}{\partial y} + \frac{\partial^2 T_P}{\partial y^2}\Big]_A^B = (\mathrm{Pe}B + \mathrm{Pe})\Delta_{-1}^B + (B^2 + 3B + 2 + \mathrm{Pe}B + \mathrm{Pe})\Delta_0^B + (B^2 + 3B + 2)\Delta_{+1}^B$$

The GP fitness measure F is given by:

$$\int \Big[\frac{\partial^2 T_P}{\partial x^2} + \frac{\partial^2 e}{\partial x^2} + \frac{\partial^2 T_P}{\partial y^2} - \mathrm{Pe}\Big(\frac{\partial T_P}{\partial x} + \frac{\partial e}{\partial x} + \frac{\partial T_P}{\partial y}\Big)\Big]^2 d\Omega$$

The following integrals

$$\int_0^1\int_0^1 \Big[\frac{\partial^2 T_P}{\partial x^2} - \mathrm{Pe}\frac{\partial T_P}{\partial x}\Big]\Big[\frac{\partial^2 T_P}{\partial x^2} - \mathrm{Pe}\frac{\partial T_P}{\partial x}\Big]dx\,dy$$

$$\int_0^1\int_0^1 \Big[\frac{\partial^2 T_P}{\partial y^2} - \mathrm{Pe}\frac{\partial T_P}{\partial y}\Big]\Big[\frac{\partial^2 T_P}{\partial y^2} - \mathrm{Pe}\frac{\partial T_P}{\partial y}\Big]dx\,dy$$

$$2\int_0^1\int_0^1 \Big[\frac{\partial^2 T_P}{\partial x^2} - \mathrm{Pe}\frac{\partial T_P}{\partial x}\Big]\Big[\frac{\partial^2 T_P}{\partial y^2} - \mathrm{Pe}\frac{\partial T_P}{\partial y}\Big]dx\,dy$$

can be obtained simply by multiplication of the coefficients already derived and shifting of the coefficients in the resulting vector to obtain an integrated expression which can then be evaluated by substitution of the limits of integration: 0 and 1, and integration of the cross term, and of the purely exponential terms:

$$2\int_0^1\int_0^1 \Big[\frac{\partial^2 e}{\partial x^2} - \mathrm{Pe}\frac{\partial e}{\partial x}\Big]\Big[\frac{\partial^2 T_P}{\partial y^2} - \mathrm{Pe}\frac{\partial T_P}{\partial y}\Big]dx\,dy$$
$$\int_0^1\int_0^1 \Big[\frac{\partial^2 e}{\partial x^2} - \mathrm{Pe}\frac{\partial e}{\partial x}\Big]\Big[\frac{\partial^2 e}{\partial x^2} - \mathrm{Pe}\frac{\partial e}{\partial x}\Big]dx\,dy$$

are both similarly accomplished. However,

$$2\int_0^1\int_0^1 \Big[\frac{\partial^2 e}{\partial x^2} - \mathrm{Pe}\frac{\partial e}{\partial x}\Big]\Big[\frac{\partial^2 T_P}{\partial x^2} - \mathrm{Pe}\frac{\partial T_P}{\partial x}\Big]dx\,dy$$

is not straightforward, and can be expressed as two integrals $L_0 + L_1$:

$$L_0 = 2\int_0^1\int_0^1 -\mathrm{Pe}\frac{\partial e}{\partial x}\Big[\frac{\partial^2 T_P}{\partial x^2} - \mathrm{Pe}\frac{\partial T_P}{\partial x}\Big]dx\,dy$$
$$L_1 = 2\int_0^1\int_0^1 \frac{\partial^2 e}{\partial x^2}\Big[\frac{\partial^2 T_P}{\partial x^2} - \mathrm{Pe}\frac{\partial T_P}{\partial x}\Big]dx\,dy$$

Relations already obtained can be substituted for:

$$L_0 = 4\,\mathrm{Pe}\,\Gamma\int_0^1\int_0^1 \sum_{AB} c_A^B\, x^{A+1} y^B\, \exp(-\Gamma x^2)dx\,dy$$

$$L_1 = -4\Gamma\int_0^1\int_0^1 \sum_{AB} c_A^B\, x^A y^B\, \exp(-\Gamma x^2)dx\,dy + 8\,\Gamma^2\int_0^1\int_0^1 \sum_{AB} c_A^B\, x^{A+2} y^B\, \exp(-\Gamma x^2)dx\,dy$$

each term in the sum contributes an integral which can be approximated with the erf(x) function and the recursive relationship identified in section 5.5.

Adaptive Logic Programming

M. Keijzer & V. Babovic
DHI — Water & Environment
Hørsholm, Denmark
{mak|vmb}@dhi.dk

C. Ryan & M. O'Neill
University of Limerick
Limerick, Ireland
{conor.ryan|michael.oneill}@ul.ie

M. Cattolico
Tiger Mountain Scientific Inc.
Kirkland, WA U.S.A
mike@TigerScience.com

Abstract

A new hybrid of Evolutionary Automatic Programming which employs logic programs is presented. In contrast with tree-based methods, it employs a simple GA on variable length strings containing integers. The strings represent sequences of choices used in the derivation of non-deterministic logic programs. A family of Adaptive Logic Programming systems (ALPs) are proposed and from those, two promising members are examined. A proof of principle of this approach is given by running the system on three problems of increasing grammatical difficulty. Although the initialization routine might need improvement, the system as presented here provides a feasible approach to the induction of solutions in grammatically and logically constrained languages.

1 Introduction

Logic Programming [3] makes a rigorous distinction between the declarative aspect of a computer program and the procedural part. The declarative part defines everything that is 'true' in the specific domain, while the procedural part derives instances of these 'truths'.

The programming language Prolog [16] fills in the procedural aspect by employing a strict depth-first search-strategy through the rules (clauses) defined by a logic program. In this paper an alternative search strategy is examined. This employs a variable length genetic algorithm that specifies the choice to make at each choice-point in the derivation of a query. The search strategy operates on logic programs that define simple to more constrained languages. This hybrid of a variable length genetic algorithm operating on logic programs is given the name Adaptive Logic Programming.

The paper is organized by first giving a short introduction of logic programming and Prolog, followed by a description of the non-deterministic modifications we propose. A section with related work of applying genetic programming to logic programs follows in section 4. The system thus described is tested on three problems with increasingly more involved grammatical constraints. A discussion and conclusion finish the paper.

2 Logic Programming

A logic program consists of clauses consisting of a head and a body. In Prolog notation, identifiers starting with an uppercase character are considered to be logic variables, while lowercase characters are atoms or function symbols. The logic program

```
sym(x).
sym(y).
sym(X + Y) :- sym(X), sym(Y).
sym(X * Y) :- sym(X), sym(Y).
```

defines a single predicate *sym*. The derivation symbol *:-/2* should be read as an inverse implication sign. In predicate logic the third *clause* can then be interpreted as

$$\forall X, Y : sym(X) \wedge sym(Y) \rightarrow sym(X + Y)$$

The query

```
?- sym(X).
```

can be interpreted as the inquiry $\exists X : sym(X)$[1] and produces in Prolog the following sequence of solutions:

```
X = x;
X = y;
X = x + x;
X = x + y;
X = x + (x + x);
X = x + (x + y);
X = x + (x + (x + x));
...
```

Extrapolating this sequence it is easy to see that without bounds on the depth or size of the derivation, the depth-first clause selection with backtracking strategy employed in Prolog will never generate an expression that contains the multiplication character. Therefore, while the depth-first selection of clauses may be *sound*, it is not *complete* w.r.t. an arbitrary logic program[2].

Logic programming is a convenient paradigm for specifying languages and constraints. A predicate can have several *attributes* and these attributes can be used to constrain the search space. For example, the logic program and query

```
sym(x,1).
sym(y,1).
sym(X+Y,S) :-
   sym(X,S1), sym(Y,S2), S is S1+S2+1.
sym(X*Y,S) :-
   sym(X,S1), sym(Y,S2), S is S1+S2+1.

?-sym(X, S), S<10.
```

specifies all expressions of size smaller than 10. With such terse yet powerful descriptiveness, it is therefore no surprise that attribute logic and constraint logic programming are more often than not implemented in Prolog. It is this convenient representation of data or program structures together with constraints that we are trying to exploit in this paper.

Formally, a Logic Programming system is defined by Selected Literal Definite clause resolution (or SLD-resolution for short), and an *oracle* function that selects the next clause or the next literal[3]. This *oracle* function is in Prolog implemented as:

- Select first clause
- Select first literal
- Backtrack on failure

3 Grammatical Evolution and Logic Programming

Grammatical Evolution [13] aims at inducing arbitrary computer programs based on a context-free specification of the language. It employs a variable length integer representation that specifies a sequence of choices made in the context-free grammar to generate an expression. Due to the specific representation of a sequence of choices, no type information needs to be maintained in the evolving strings, and no custom mutation and crossover operators need to be designed. The variable length one-point crossover employed in GE was shown to have an elegant interpretation in closed grammars in [7].

In this paper we similarly use a sequence of choices as the base representation, but rather than choosing between the production rules of a context-free grammar, they are used to make a choice between clauses in a logic program. The sequence of choices thus represents one part of the selection function operating together with SLD-resolution on the logic program. Furthermore, backtracking is implemented in the system together with an alternative strategy on failure: restarting the original query.

As an example of the mapping process, consider the grammar defined above in Section 2, and an evolutionary induced sequence of choices $[2, 1, 3, 0, 1]$. The derivation of an instance then proceeds as follows:

```
?- sym(X).
?- sym(X1), sym(X2). [(X1 + X2)/X]  2
?- sym(y),  sym(X2). [y/X1]         1
?- sym(X3), sym(X4). [(X3 * X4)/X2] 3
?- sym(x),  sym(X4). [x/X3]         0
?- sym(y).           [y/X4]         1
```

Applying all bindings made, this produces the symbolic expression: $y + x * y$. The values from the sequence of choices are in this example conveniently chosen to lie between 0 and 3 inclusive; in practice a number encountered in the genotype can be higher than the number of choices present. The choice will then be taken modulo the number of available choices.

In this example, the depth-first clause selection of Prolog is replaced by a guided selection where choices are drawn from the genotype. The first unresolved literal is still chosen to be the first to derive. It is possible to replace this with guided selection as well, be it in the

[1] Formally the negation of this formula is disproven, thus proving this formula.

[2] A depth-first strategy is however far more efficient than the breadth-first alternative

[3] A literal is a single predicate call in the body of a clause or query. In the query above, $sym(X, S)$ and $S < 10$ are literals.

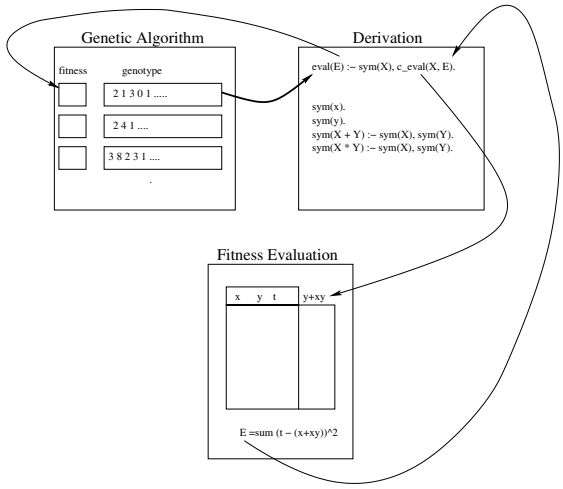


Figure 1: Overview of the ALP system: the sequence of choices is used in the derivation process to derive a specific instance for *sym(X)*, this instance is passed to the evaluation function. The calculated fitness is returned to the genetic algorithm.

same string or in a seperate string. Together with a choice whether to do backtracking or not, this leads to Table 1 which gives an overview of the parts of the Prolog engine that can be replaced. Table 1 thus defines a family of adaptive logic programming systems. Enumerating them, ALP-0 will correspond with a Prolog system, while ALP-1 (modified clause selection) and ALP-4 (modified clause selection without backtracking) correspond with the systems examined here.

Selection	Prolog	Modification
Clause	First Found	From Genotype
Literal	First Found	From Genotype
On Failure	Backtrack	Restart

Table 1: The possible modifications to the selection function.

We've chosen to focus on ALP-1 and ALP-4 as there are some practical problems associated with replacing literal selection. In many applications, a logic program consists of a mix of non-deterministic predicates (such as the *sym/1 and sym/2* predicates above) and deterministic predicates (such as the assignment function *is/2*). The deterministic predicates often assume some variables to be bound to ground terms, evaluating them out of order would then lead to runtime exceptions. Section 6 will show that for languages with a nontrivial set of constraints, backtracking is necessary to obtain solutions reliably.

A logic program is thus used as a formal specification of the language, the sequence of choices is used to steer the resolution process and a small external program is used to evaluate the expressions generated. See Figure 1 for the typical flow of information. The scope of the system are then logic programs where there is an abundance of solutions that satisfy the constraints, which are subsequently evaluated for performance on a problem domain.

3.1 Backtracking

In ALP-1, at every step in the derivation process, a list is maintained of clauses that are not tried yet. When a query fails at a certain point, the selection function will be asked to pick a new choice out of the remaining clauses. This choice is removed and when all are exhausted, the branch reports failure to the previous level where this procedure starts again.

ALP-4 does not use backtracking; on failure it will restart the original, top-level, query, while the reading continues from where it left of.

If the sequence runs out of choices, i.e., the end of the genotype is reached, the derivation is cut off and the individual gets the worst performance value available. This will be labelled a failure.

3.2 Initialization

Initialization is performed by doing a random walk through the grammar, maintaining the choices made, backtracking on failure (ALP-1) or restarting (ALP-4). After a successful derivation is found, the shortest, non-backtracking path to the complete derivation is calculated. An occurence check is performed and if the path is not present in the current population, a new individual is initialized with this shortest non-backtracking path. Individuals in the initial population will thus consist solely of non-backtracking derivations to sentences.

Typically a depth limit is employed.

3.3 Performance Evaluation

Performance is typically evaluated in a special module, written in a compiled language such as C. This program walks through the tree structure and evaluates each node. This is however not necessary if the fitness can be readily evaluated in the logic program itself. The query investigated typically has the form: find that derivation for $sentence(X)$, such that $fitness\_eval(X, F)$ returns the maximal or minimal F.

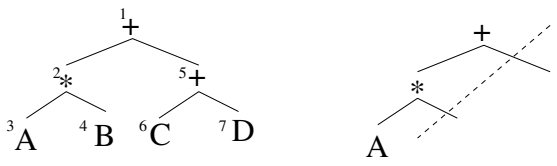


Figure 2: An individual in the form of a derivation tree. Vacant sites are filled by sub-trees from the other parent.

3.4 Variational Operators

Crossover is implemented as a simple variable length string crossover. Two independent random points are chosen in the strings and strings starting at those points are swapped. The two points are chosen within the *expressed* code of a string — code that is used in the derivation.

The effects of the crossover in this case is quite different from that of subtree crossover. This is because the derivation tree is created in a pre-order fashion, i.e., the left-most literal of a goal is always mapped to completion before the rest of the goal is processed.

Crossover operates on the linear structure, and single point crossover thus divides an individual into a partially mapped tree, and a stack of choices. In general, all subtrees to the right of the crossover site are removed, as in Figure 2, leaving multiple vacant sites on the derivation tree. These sites are said to *ripple* up from the crossover site.

An integer in the genome is said to be *intrinsically polymorphic*, meaning that it can be interpreted (or reinterpreted) by any node in a derivation tree in whatever context. By adding codons from the other parent to the incomplete derivation tree in Figure 2, the sites vacated by the crossover event are again filled with new subtrees of the appropriate type.

In contrast with subtree crossover, the percentage of genetic material exchanged is on average 50% and it has been shown that this crossover is quite effective in exploring the search space of possible programs as it is less susceptible to premature convergence [7].

Although many mutations can be defined on a string of integers, the one used here simply replaces a randomly selected integer from the string with a randomly drawn integer lower than 2^{16}.

3.5 Special Predicates

All Prolog built-in clauses such as assignment (*is/2*) are evaluated in Prolog directly. This is done as often such clauses are deterministic and depend on the Prolog depth-first search strategy. Also calls to libraries etc., are evaluated directly.

A special predicate *ext_int/2* is employed that, when encountered in the derivation, binds the first argument with an integer drawn from the genotype modulo the second argument (which therefore needs to be grounded). Using this technique, floating point constants can be specified as part of the logic program. The floating point grammar used in this paper is:

```
fp_unsigned(X) :-
  ext_int(Num,256),
  ext_int(Denom,256),
  X is Num / (Denom + 1).

fp_unsigned(X) :-
  fp_unsigned(First),
  fp_unsigned(Second),
  X is First * Second.

fp(X) :-
  ext_int(S,2),
  Sign is (S-0.5) * 2,
  fp_unsigned(Y),
  X is Sign * Y.
```

There is nothing particularly innovative or clever about this program. Although it specifies up to machine precision floating points, it can only model rational numbers for which the numerator and denominator are factors of primes smaller than 256. It does show however, how intricate calculations can be made a part of the language. A call to *fp/1* will bind the argument to a floating point value instead of an expression. Future versions of ALPs will undoubtedly support floating point numbers that evolve together with the list of choices, so that specialized mutation operators can be used.

4 Related Work

Wong and Leung [17] hybridized inductive logic programming and genetic programming in their system LOGENPRO. The representation that is being manipulated by the genetic operators consist of derivation trees. LOGENPRO first applies a preprocessing step that transforms a logic grammar (a Definite Clause Grammar) into a logic program. Apart from expressions in the specified language, this logic program also

produces a symbolic representation of the derivation tree. This derivation tree is subsequently manipulated by the genetic operators. Some fairly intricate crossover and mutation operators are used which, together with semantic validation, ensure that the resulting derivation tree specifies a valid instantiation of the logic grammar. Because the logic program is able to parse derivation trees, semantic verification reduces to checking whether Prolog accepts the derivation tree.

Ross [15] describes a similar system that uses Definite Clause Translation Grammars. This representation is also translated into a logic program that is able to parse and generate derivation trees in the language defined by the grammar. The crossover described in [15] seems to only use type information contained in the predicate names and arity at the heads of the clauses and swaps derivation subtrees that contain the same head. A semantic verification (running the Prolog program on the derivation tree), is subsequently performed.

Even for typed crossovers, semantic validation is necessary as the body of a clause can introduces additional constraints, not related to the type but to the actual values found in the derivation. An additional problem for strongly typed crossover occurs when the number of distinct types grows. As the operator will only swap subtrees that have the same type, every type needs to be present multiple times with different derivations in the population to make the operator swap something other than identical trees. If a specific type disappears from a population, or only has a single distinct instance, the system has to rely on mutation to re-introduce instances. Every additional type or constraint thus partitions the search space further and thereby restricts the crossover.

Yet another problem with subtree crossover is that it will process an increasingly smaller percentage of genetic material as the size of the individuals grows [1], while the crossover employed here will always swap on average half of the genetic material [7].

In contrast with the systems described above, the ALP systems do not use an explicit representation of the derivation tree, thus being time and memory efficient. In the systems described above, every step in the derivation process is recorded in a node together with the bindings that are made, effectively doubling the size of an expression tree. In ALPs, no pre-processing step is necessary, it works on logic programs directly. Also no bookkeeping is necessary when trying crossovers and mutations. The downside of this is that the ALPs can generate invalid individuals, i.e., strings of choices that have no valid derivation. However, such a failed derivation is equivalent with a failed semantic validation in the systems described above. The rate at which this happens is ultimately bound to the language and constraints used.

5 Proof of Principle

The system outlined above was implemented using SWI-Prolog[4], mainly because of the two-way C API that it implements. A steady-state genetic algorithm using a tournament size of 5 was implemented using the evolutionary objects library[5]. Crossover and mutation were applied with rates 0.9 and 0.1 respectively. What follows are three experiments with grammars of increasing degrees of complexity. The purpose of these experiments is to present a proof of the principle that a variable length GA can indeed be used to successfully induce sentences in both easy and difficult languages.

The experiments were run for 100 generations using both ALP-1 and ALP-4. For the symbolic regression and Santa Fe trail problem, 100 runs were performed, the results on the sediment transportation experiments are reported on the basis of 500 runs. As a baseline test, for each problem, 10 million random individuals were generated using the initialization procedure from ALP-1 (denoted by ALP-1R). Also 10 million individuals were generated by Prolog (ALP-0). As Prolog was not able to produce a single correct individual for any of the problems, these results are further omitted. For all methods, the same depth limit was set.

5.1 Symbolic Regression: $0.3xsin(2\pi x)$

From this function 100 equally spaced points in the interval [-1,1] were generated. This problem has been studied in [6] with data in the range [0,1]. For the experiments a population size of 1000 was used. A success was determined to be a root mean squared error less than 0.01.

5.2 An Artificial Ant on the Santa Fe Trail

The artificial ant problem has been studied intensively in [10] for a closed grammar. Here a context free grammar is employed like in [7].

A population of size 500 was used. The best performance achievable was 89 food pellets eaten.

[4] http://www.swi.psy.uva.nl/projects/SWI-Prolog
[5] http://www.sourceforge.net/projects/eodev

5.3 Units of Measurement: Sediment Transport

The units of measurement problem used here has been studied previously in [2]. In contrast with [2] the system is constrained to generate only dimensionally correct equations. Another approach for this class of problems is studied in [14] where a context free grammar is generated that models a subset of the language of units of measurement.

The desired output for this problem is a dimensionless quantity, a concentration. Two experiments were performed, one where the desired output is given and one experiment where no desired output is given. These are denoted in Table 2 as *Sed1* and *Sed2* respectively. The second experiment thus seeks for a dimensionally consistent formulation stated in any units. It is quite common for empirical equations to multiply the resulting equation with a constant stated in some units to obtain an equation stated in the desired units of measurement[6], this is usually a residual coefficient that tries to describe some unmodelled phenomena.

The parameters were set at the same values as the symbolic regression problem above. A successful run was determined by comparing the error produced to that of a benchmark model, which was an equation induced by a scientist [5]. Because success rates were low, 500 runs were performed for this problem.

6 Results

For all problems, solutions were found, Table 2 summarizes the results. Although the differences between ALP-1 and ALP-4 are not significant ($\alpha = 0.05$) on the symbolic regression problem[7] and the Sante-Fe problem, the failure of ALP-4 to find any solutions on any of the sediment transport problems clearly shows the need for backtracking. The sediment transport problem involves non-trivial constraints, and inspection of the expressions produced by ALP-4 showed that it got very quickly trapped into derivations of shallow depth, often converging on a single constant. It is hypothesized that the use of backtracking allows the genotype to specify a particular start of the derivation process, relying on backtracking as a local search operator to find feasible solutions.

Confidence intervals were calculated around the 99% computational effort statistic proposed by Koza ([8] p. 194). The first fifty percent of the runs were used to find the generation that maximized the effort statistic, the results reported were subsequently calculated on the latter (independent) half of the runs. As the confidence interval calculated for the sediment transportation problem included a 0% success rate, the upper bound of the confidence interval is infinite. This is to be expected, as the success predicate demanded that the system should improve upon an equation proposed by an expert in the field of sediment transport. Interestingly enough, for the second sediment transportation problem (that allows dimensionally consistent equations that do not produce the desired dimensionless output), the success rate is significantly higher. This illustrates the dangers of providing too much bias to a weak search algorithm such as ALPs.

	ALP-1	ALP-4	ALP-1R
S. R.	4253(9%) [2351, 11642]	5508(6%) [2924, 16868]	inf (0%)
S. F.	185(37%) [124, 305]	284(28%) [172, 584]	1279(3.6e − 4%) [852, 2302]
Sed1	10997(1.6%) [3629, inf]	inf (0%)	inf (0%)
Sed2	1610(26%) [1300, 2054]	inf (0%)	inf (0%)

Table 2: Computational Effort divided by 1000 for solving the three problems. Overall success rate in round brackets. Numbers in square brackets denote 95% confidence intervals around the effort statistic calculated above. Confidence intervals are calculated with resampling statistics, using a bootstrap sample of 10000. The success rates are calculated on the final (100th) generation.

The confidence intervals were calculated in response to a question posed by Miller [11] on the value of this statistic on experiments with a low success rate. Table 2 shows that indeed, for a low success rate such as 1.6%, the statistic can only give a lower (highly optimistic) bound on the number of individuals to process. It also shows that the statistic is highly volatile even for moderate success rates. For the Santa-Fe problem that has an overall success rate of 37%, the width of the confidence interval (i.e., the uncertainty around the statistic) is nearly as large as the value of the computational effort itself. The confidence intervals clearly show that a straightforward comparison of computational effort, even differing in an order of magnitude, is not possible.

Figure 3 shows the average fail ratio for ALP-1. As

[6] A famous example is Chezy's roughness coefficient, stated in the unit $m^{1/2}/s$.

[7] A control run using a strongly typed subtree crossover on the symbolic regression problem resulted in a success rate of 4%, lower than either ALP-1 or ALP-4.

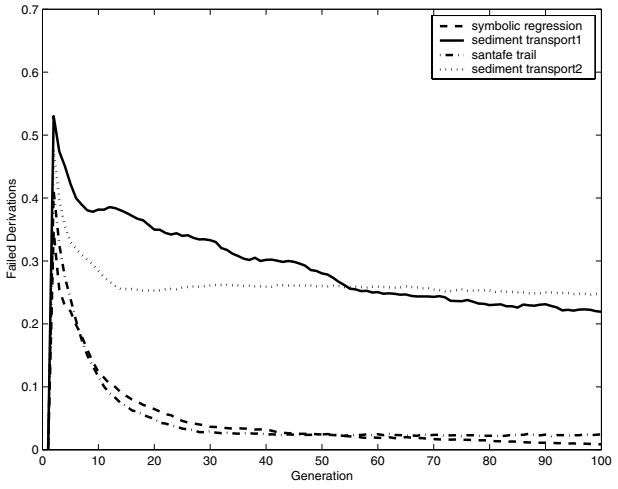


Figure 3: Failed derivations over the total number of derivations per generation for ALP-1, averaged over the number of runs.

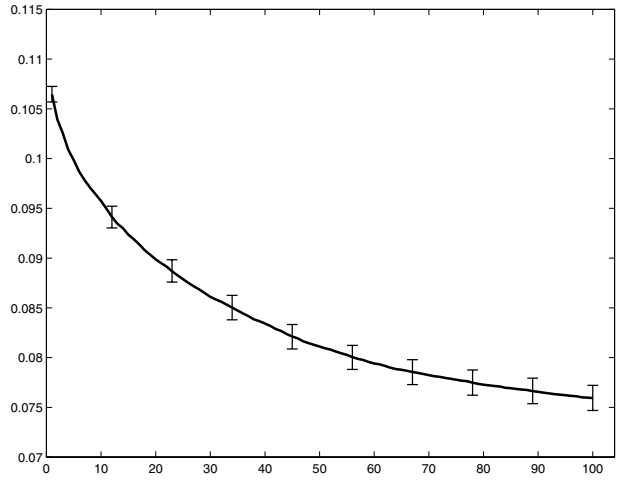


Figure 4: Average performance of ALP-1 on the sediment transport problem Sed1. Although the failure rate is high (see Figure 3), improvements keep on being found. Notice that the performance has not leveled off yet at 100 generations.

the initial generation includes only valid individuals, the ratio is zero. It is clear from the figure that this initial population is not well adapted to produce valid individuals. For the less constraint problems, the percentage of failed derivations quickly drops to low values. For the problems involving units of measurement, the level of failed derivations does not drop that quickly: even after 100 generations, more than one in five crossover and/or mutation events results in a failed derivation.

Although it might seem that the crossover and mutation employed here are very destructive, and might even lead to the hasty conclusion that a strongly typed crossover is necessary, this is in our opinion not warranted. The high fail rates are a symptom of the highly constrained nature of this search space. A strongly typed crossover would have this same problem, it would either obscure it by only swapping identical subtrees, or by a high failure rate in the semantic validation. Figure 4 shows that despite this high failure rate, the system is still able to perform significant optimization. It would however be instructive to see how well a strongly typed system would fair on this problem.

7 Discussion

The system presented here is the first prototype for evolving sentences in languages with constraints. It has proven to be able to optimize all the problems described here, including a difficult language such as the units of measurement grammar.

The initialization procedure as is described here does not provide an optimal starting point for the ALP systems. The initialization procedure consists of non-backtracking points to derivations, with no unexpressed code. It is an avenue of future research to find a better initialization procedure. However, the highly explorative nature of the crossover used here, enables the system to overcome this and even with a non-optimal starting point, it is able to find competitive solutions to the problems presented to it.

The main benefit of the ALPs system in contrast with strongly typed genetic programming systems is that the variational operators do not depend as heavily on the grammar that is used. A strongly typed crossover is constrained to search in the space of available types in the population, thus having a strong macro-mutation flavor [1]. The ALP systems, borrowing the mapping process from Grammatical Evolution, is in principle not thus constrained. New instances of types can be created during the run.

Although this paper has focussed on *expression induction*, due to the general nature of logic programs, we also expect to be able to perform optimization on transformational problems [12], as well as on constructional (embryonic) problems [4, 9].

8 Conclusion

An implementation and proof of principle is given for an adaptive logic programming system called ALPs.

It modifies the standard Prolog clause selection to a selection strategy that is guided by a variable length genotype. The system was tested on three different problems of increasing difficulty and was able to produce solutions to these problems.

Although backtracking did not seem necessary for the simpler grammars, it made a significant difference in the difficult grammar of units of measurement.

Acknowledgements

The first two authors would like to acknowledge the Danish Technical Research Council (STVF) for partly funding Talent Project 9800463 entitled "Data to Knowledge – D2K" http://www.d2k.dk

References

[1] P. J. Angeline. Subtree crossover: Building block engine or macromutation? 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 9–17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[2] V. Babovic and M. Keijzer. Genetic programming as a model induction engine. *Journal of Hydro Informatics*, 2(1):35–61, 2000.

[3] E. Burke and E. Foxley. *Logic and Its Applications.* Prentice Hall, 1996.

[4] F. Gruau. Genetic micro programming of neural networks. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 24, pages 495–518. MIT Press, 1994.

[5] J.A.Zyserman and J. Fredsœ. Data analysis of bed concentration of suspended sediment. *Journal of Hydraulic Engineering*, (9):1021–1042, 1994.

[6] M. Keijzer and V. Babovic. Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 76–90, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[7] M. Keijzer, C. Ryan, M. O'Neill, M. Catollico, and V. Babovic. Ripple crossover in genetic programing. In J. Miller, editor, *Proceedings of EuroGP 2001*, 2001.

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

[9] J. R. Koza, David Andre, F. H. Bennett III, and M. Keane. *Genetic Programming 3: Darwinian Invention and Problem Solving.* Morgan Kaufman, Apr. 1999.

[10] W. B. Langdon and R. Poli. Why ants are hard. Technical Report CSRP-98-4, University of Birmingham, School of Computer Science, Jan. 1998. Presented at GP-98.

[11] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[12] P. Nordin and W. Banzhaf. Genetic reasoning evolving proofs with genetic search. 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 255–260, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[13] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Trans. Evolutionary Computation*, 2001.

[14] A. Ratle and M. Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, Sept. 16-20 2000. Springer Verlag. LNCS 1917.

[15] B. Ross. Logic based genetic programming with definite clause translation grammars. Technical report, Department of Computer Science, Brock University, Ontario Canada, 1999.

[16] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT press, 1994.

[17] M. L. Wong and K. S. Leung. Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180, summer 1997.

Evolution of Genetic Code on a Hard Problem

Robert E. Keller
Leiden Institute of Advanced Computer Science
Leiden University
The Netherlands
Robert.E.Keller@WEB.DE

Wolfgang Banzhaf
Computer Science Department
Dortmund University
Germany
banzhaf@icd.de

Abstract

In most Genetic Programming (GP) approaches, the space of genotypes, that is the search space, is identical to the space of phenotypes, that is the solution space. Developmental approaches, like Developmental Genetic Programming (DGP), distinguish between genotypes and phenotypes and use a genotype-phenotype mapping prior to fitness evaluation of a phenotype. To perform this mapping, DGP uses a genetic code, that is, a mapping from genotype components to phenotype components. The genotype-phenotype mapping is critical for the performance of the underlying search process which is why adapting the mapping to a given problem is of interest. Previous work shows, on an easy synthetic problem, the feasibility of code evolution to the effect of a problem-specific self-adaptation of the mapping.The present empirical work delivers a demonstration of this effect on a hard synthetic problem, showing the real-world potential of code evolution which increases the occurrence of relevant phenotypic components and reduces the occurrence of components that represent noise.

1 INTRODUCTION AND OBJECTIVE

Genetic programming (Koza 1992, Banzhaf *et al.* 1998) is an evolutionary algorithm that, for the purpose of fitness evaluation, represents an evolved individual as algorithm. Most GP approaches do not distinguish between a genotype, that is, a point in search space, and its phenotype, that is, a point in solution space. Developmental approaches, however, like (Keller and Banzhaf 1996, O'Neill and Ryan 2000, Spector and Stoffel 1996), make a distinction between the search space and the solution space. Thus, they employ a genotype-to-phenotype mapping (GPM) since the behavior of the phenotype defines its fitness which is used for selection of the corresponding genotype. This mapping is critical to the performance of the search process: the larger the fraction of the search space that a GPM maps onto good phenotypes, the better the performance. In this sense, a mapping is said to be "good" if it maps a "large" fraction of search space onto good phenotypes. This is captured in the formal measure of "code fitness" which is defined in (Keller and Banzhaf 1999). That work shows, on an easy synthetic problem, the effect of code evolution: genetic codes, i.e., information that controls the genotype-phenotype mapping and that is carried by individuals, get adapted such that problem-relevant symbols are increasingly being used for the assembly of phenotypes, while irrelevant symbols are less often used. This implies that the approach can adapt the mapping to the problem, which eliminates the necessity of having a user define a problem-specific mapping. This in itself would often be impossible when facing a new problem, since the user does not yet understand the problem well enough. From an abstract point of view, code evolution adapts fitness landscapes, since a certain mapping defines that landscape. (Keller and Banzhaf 1999) also shows that, during evolution, it is mostly better individuals who carry better codes, and it is mostly better codes that are carried by better individuals. However, the computation of code fitness is only feasible for small search spaces, that is, easy problems, why it is of interest to test whether the effect of code evolution also takes place on a hard problem, which is the objective of this work.

First, developmental genetic programming (DGP) (Keller and Banzhaf 1996, Keller and Banzhaf 1999)

is introduced as far as needed in the context of this article, and the concept of a genetic code as an essential part of a mapping is defined. Second, the principle of the evolution of mappings as an extension to developmental approaches is presented in the context of DGP. Here, the genetic code is subjected to evolution which implies the evolution of the mapping. Third, the objective mentioned above is being followed by investigating the progression of phenotypic-symbol frequencies in codes during evolution. Finally, conclusions and objectives of further work are discussed.

2 DEVELOPMENTAL GENETIC PROGRAMMING

All subsequently described random selections of an object from a set of objects occur under equal probability unless mentioned otherwise.

2.1 ALGORITHM

A DGP variant uses a common generational evolutionary algorithm, extended by a genotype-phenotype mapping prior to the fitness evaluation of the individuals of a generation.

2.2 GENOTYPE, PHENOTYPE, GENETIC CODE

The output of a GP system is an algorithm in a certain representation. This representation often is a computer program, that is, a word from a formal language. The representation complies with structural constraints which, in the context of a programming language, are the syntax of that language. DGP produces output compliant with the syntax defined by an arbitrary context-free LALR(1) (look-ahead-left-recursive, look ahead one symbol) grammar. Such grammars define the syntax of real-world programming languages like ISO-C. A phenotype is represented by a syntactically legal symbol sequence with every symbol being an element of either a function set F or a terminal set T that both underlie a genetic-programming approach. Thus, the solution space is the set of all legal symbol sequences.

A codon is a contiguous bit sequence of $b > 0$ bits length which encodes a symbol. In order to provide for the encoding of all symbols, b must be chosen such that for each symbol there is at least one codon which encodes this and only this symbol. For instance, with $b = 3$, the codon 010 may encode the symbol a, and 2^3 symbols at most can be encoded. A genotype is a fixed-size codon sequence of $n > 0$ codons, like 011 010 000 111 with size $n = 4$. By definition, the leftmost codon is codon 0, followed by codon 1 up to codon $n - 1$.

A genetic code is a codon-symbol mapping, that is, it defines the encoding of a symbol by one or more codons. An example is given below with codon size 3.

000	001	010	011	100	101	110	111
a	b	c	d	+	*	-	/

The "symbol frequency" of a symbol in a code is the number m of occurrences of the symbol in the code, which means that m different codons are mapped onto this symbol.

2.3 GENOTYPE-PHENOTYPE MAPPING

In order to map a genotype onto a phenotype, the genotype gets transcribed into a *raw sequence* of symbols, using a *genetic code*. Transcription scans a genotype, starting at codon 0, ending at codon $n - 1$. The genotype 101 101 000 111, for instance, is mapped onto "$**a/$" by use of the above sample code.

For the following examples, consider the syntax of arithmetic expressions. A symbol that represents a syntax error at a given position in a given symbol sequence is called *illegal*, else *legal*. A genotype is mapped either onto a legal or, in the case of "$**a/$", illegal raw symbol sequence. An illegal raw sequence gets repaired according to the syntax, thus yielding a legal symbol sequence. To that end, several repair algorithms are conceivable. A comparatively simple mechanism is introduced here, called "deleting repair". Intron splicing (Watson *et al.* 1992), that is the removal of genetic information which is not used for the production of proteins, is the biological metaphor behind this repair mechanism. Deleting repair scans a raw sequence and deletes each illegal symbol, which is a symbol that cannot be used for the production of a phenotype, until it reaches the sequence end. If a syntactic unit is left incomplete, like "$a-$", it deletes backwards until the unit is complete. For instance, the above sample raw sequence gets repaired as follows: "$**a/ \rightarrow *a/ \rightarrow a/$", then a is scanned as a legal first symbol, followed by / which is also legal. Next, the end of the sequence is scanned, so that "$a/$" is recognized as an incomplete syntactic unit. Backward deleting sets in and deletes /, yielding the sequence a, which is legal, and the repair algorithm terminates. Note that deleting repair works for arbitrarily long and complex words from any LALR(1) language.

If the entire sequence has been deleted by the repair mechanism, like it would happen with the phenotype "$++++$", the worst possible fitness value is assigned

to the genotype. This is appropriate from both a biological and a technical point of view. In nature, a phenotype not interacting with its environment does not have reproductive success, the latter being crudely modeled by the concept of "fitness" in evolutionary algorithms. In a fixed-generation-size EA, like the DGP variant used for the empirical investigation described here, an individual with no meaning is worthless but may not be discarded due to the fixed generation size. It could be replaced, for instance, by a meaningful random phenotype. This step, however, can be saved by assigning worst possible fitness so it is likely to be replaced by another individual during subsequent selection and reproduction.

The produced legal symbol sequence represents the phenotype of the genotype which has been the input to the repair algorithm. Therefore, theoretically, the GPM ends with the termination of the repair phase. Practically, however, the legal sequence must be mapped onto a phenotype representation that can be executed on the hardware underlying a GP system in order to evaluate the fitness of the represented phenotype. This representation change is performed by the following phases.

Following repair, *editing* turns the legal symbol sequence into an *edited symbol sequence* by adding standard information, e.g., a main program frame enclosing the legal sequence. Finally, the last phase of the mapping, which can be compilation of the edited symbol sequence, transforms this sequence into a machine-language program processable by the underlying hardware. This program is executed in order to evaluate the fitness of the corresponding phenotype. Alternatively, interpretation of the edited symbol sequence can be used for fitness evaluation.

2.4 CREATION, VARIATION, REPRODUCTION, FITNESS AND SELECTION

Creation builds a fixed-size genotype as a sequence of n codons random-selected from the codon set. Variation is implemented by point genotype mutation where a randomly selected bit of a genotype is inverted. The resulting mutant is copied to the next generation. Reproduction is performed by copying a genotype to the next generation. An *execution probability* p of a reproduction or variation operator designates that the operator is randomly selected from the set of variation and reproduction operators with probability p. An execution probability is also called a rate. Fitness-based tournament selection with tournament size two is used in order to select an individual for subsequent reproduction or variation. Adjusted fitness (Koza 1992) is used as fitness measure. Thus, all possible fitness values exist in $[0, 1]$, and a perfect individual has fitness value 1.

3 CODE EVOLUTION

3.1 BIOLOGICAL MOTIVATION

The mapping employed by DGP is a crude metaphor of protein synthesis that produces proteins (phenotype) from DNA (genotype). In molecular biology, a codon is a triplet of nucleic acids which uniquely encodes one amino acid, at most. An amino acid is a part of a protein and thus corresponds to a symbol. Like natural genotypes have evolved, the genetic code has evolved, too, and it has been argued that selection pressure works on code properties necessary for the evolution of organisms (Maeshiro 1997). Since artificial evolution gleaned from nature works for genotypes, the central hypothesis investigated here is that artificial evolution works for genetic codes, too, producing such codes that support the evolution of good genotypes.

3.2 TECHNICAL MOTIVATION

In DGP, the semantics of a phenotype is defined by its genotype, the specific code, repair mechanism and semantics of the employed programming language. Especially, different codes mean different genotypic representations of a phenotype and therefore different fitness landscapes for a given problem. Finally, certain landscapes differ extremely in how far they foster an evolutionary search. Thus, it is of interest to evolve genetic codes during a run such that the individuals carrying these codes find themselves in a beneficial landscape. This situation would improve the convergence properties of the search process. A related aspect is the identification of problem-relevant symbols in the F and T sets. In order to investigate and analyze the effects of code evolution, an extension to DGP has been defined and implemented, which will be described next.

3.3 INDIVIDUAL GENETIC CODE

DGP may employ a *global code*, that is, all genotypes are mapped onto phenotypes by use of the same code. This corresponds to the current situation in organic evolution, where one code, the standard genetic code, is the basis for the protein synthesis of practically all organisms with very few exceptions like mitochondrial protein synthesis.

(Keller and Banzhaf 1999) introduces the algorithm of genetic-code evolution. If evolution is expected to oc-

cur on the code level, the necessary conditions for the evolution of any structure must be met. Thus, there must exist a structure population, reproduction and variation of the individuals, a fitness measure, and a fitness-based selection of individuals. A code population can be defined by replacing the global genetic code by an *individual code*, that is, each individual carries its own genetic code along with its genotype. During creation, each individual receives a random code. An instance random code is shown:

000	001	010	011	100	101	110	111
*	/	*	a	a	d	+	a

Note that a code, since it is defined as an arbitrary codon-symbol mapping, is allowed to be redundant with respect to certain symbols., i.e., it may map more than one codon onto the same symbol. This is not in contradiction to the role of a code, since also a redundant code can be used for the production of a phenotype. Actually, redundancy is important, as the empirical results will show.

3.4 VARIATION, REPRODUCTION, CODE FITNESS AND SELECTION

A point code mutation of a code is defined as randomly selecting a symbol of the code and replacing it by a different symbol random-selected from the symbol set. Point code mutation has a certain execution probability. Reproduction of a code happens by reproducing the individual that carries the code. The same goes for selection.

4 EMPIRICAL ANALYSIS

The announced major objective of the present work is to empirically test whether the effect of code evolution takes place on a hard problem, i.e., whether the codes are adapted in a problem-specific way that is beneficial to the search process. To this end, a run series is performed on a hard synthetic problem. Evolution means a directed change of the structures of interest, which are, in the present case, the genetic codes of the individuals. In the context of the present work, the phenomenon of interest is the change of the symbol frequencies of the target symbols. If the effect of code evolution takes place on a hard problem, this must show as a shift of symbol frequencies such that the resulting codes map codons on relevant symbols rather than on other symbols.

According with the objective of the present work, a hard problem has to be designed, and problem-relevant as well as irrelevant symbols, which represent noise, have to be contained in the symbol set. Note that the objective is not to solve the problem but to observe code evolution during the DGP runs on the problem. There are several conditions for a problem that is hard for an evolutionary algorithm, and one of the most prominent ones is that the search space is by many orders of magnitude larger than the set of individuals generated by the algorithm during its entire run time.

The problem to be considered is a symbolic function regression of an arithmetic random-generated function on a real-valued parameter space.

All function parameters come from $[0, 1]$, and the real-valued problem function is given by

$$f(A,B,a,b,..,y,z) = j+x+d+j*o+e*r-t-a+h-k*u+a-k-s*o*i-h*v-i-i-s+l-u*n+l+r-j*j*o*v-j+i+f*c+x-v+n-n*v-a-q*i*h+d-i-t+s+l*a-j*g*v-i-p*q*u-x+e+m-k*r+k-l*u*x*d*r-a+t-e*x-v-p-c-o-o*u*c*h+x+e-a*u+c*l*r-x*t-n*d+p*x*w*v-j*n-a-e*b+a.$$

Accordingly, the terminal set used by the system for all of its runs is given as $\{A, B, a, b, .., y, z\}$, and the four parameters A, B, y, z do not occur in the expression that defines the problem function, that is, they represent noise in the problem context. In order to provide for noise in the context of the function set, too, this set shall be given as $\{+, -, *, /\}$. As the division function / does not occur in the expression that defines the problem function, it represents noise. As only 5 symbols, – i.e., about 15% –, of all 32 symbols represent noise, identifying those by chance is unlikely.

Due to the resulting real-valued 28-dimensional parameter space, a fitness case consists of 28 real-valued input values and one real output value. Let the training set consist of 100 random-generated fitness cases. A population size of 1,000 individuals is chosen for all runs, and 30 runs shall be performed, each lasting for exactly 200 generations. That is, there is no run termination when a perfect individual is found so that phenomena of interest can be measured further until a time-out occurs after the evolution of the 200th generation.

As there are 32 target symbols, the size of the codons must be set to five, at least, in order to have codes that can accomodate all symbols, and for the run series, the size is fixed at five. As $2^5 = 32$, the space of all possible genetic codes contains 32^{32} elements, or approximately $1.5 * 10^{48}$ codes, including 32! or about $2.6 * 10^{35}$ codes with no redundancy. Genotype size 400 is chosen, i.e., 400 codons make up an individual genotype, while the length of the problem func-

tion, measured in target symbols, is about 200. This over-sizing of the genotype size strongly enlarges the search space, making the problem at hand very hard. As the codon size equals five and the genotype size equals 400, the search space contains 2^{400*5} individuals, or 10^{602}, and as the single-bit-flip operator is the only genotypic variation operator, this corresponds to a 2000-dimensional search space. According to the experimental parameters, $6 * 10^6$ individuals are evaluated during the run series, so that the problem search space as well as the space of all codes are significantly larger than the set of search trials, that is, individuals, generated by the approach.

The execution probabilities are 0.85 for genotype reproduction, 0.12 for point genotype mutation, and 0.03 for point code mutation. Note that the point code mutation rate is only 25 percent of the genotype point mutation rate. This has been set to allow the approach to evolve the slower changing codes by use of several different individuals that carry the same code, like genotypes are evolved by use of several different, usually static, fitness cases. We hypothesize that these differing time scales are needed by the approach to distinguish between genotypes and codes.

The codes of the individuals of an initial generation are randomly created, so that each of the 32 symbol frequencies is about one in generation 0.

5 RESULTS AND DISCUSSION

Subsequently, "mean" refers to a value averaged over all runs, while "average" designates a value averaged over all individuals of a given generation.

Top down, figure 1 shows the progression of the mean best fitness and the mean average fitness.

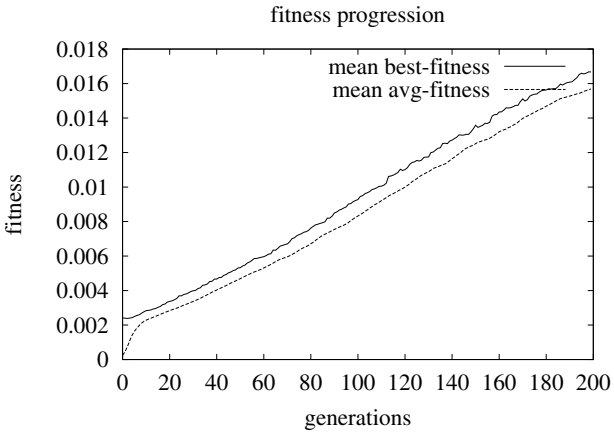


Figure 1: Top down, the curves show the progression of the mean best fitness and mean average fitness.

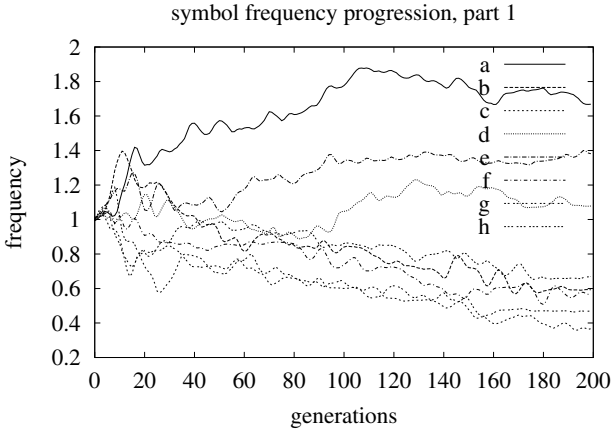


Figure 2: Progression of the mean symbol frequency in the code population.

Both curves rise, indicating convergence of the search process, which is relevant to the hypothesized principle of code evolution that is given below.

The following four figures together illustrate the progression of the mean symbol frequencies for all 32 symbols, while each figure, for reasons of legibility, displays information for eight symbols only.

As for the interpretation of figures 2 to 5, the frequency value F for a symbol S in generation G says that, over all runs, S occurs, on average, F times in a genetic code of an individual from G. As there are 32 positions in each code, F theoretically comes from $[0, .., 32]$, while practically the extreme values of the range will not be reached due to point code mutation. A value below one indicates the rareness of S in most codes of the generation, while a value above one signals redundancy of S, that is, on average, more than one codon of a genotype gets mapped onto S, or, put differently, S gets more often used for the build-up of a phenotype. Note that, due to the random creation of the codes for generation 0, all curves in all figures approximately begin in $(0, 1)$, since there are 32 codes and 32 positions in each code.

A general impression to be gained from all figures is that, after an initial phase of strong oscillation of the frequencies, the frequency distribution stabilizes. This phenomenon is typical for learning processes in the field of evolutionary algorithms, where after an initial exploratory phase a phase of exploitation sets in. It can be observed for fitness progressions, where well-performing individuals are of interest, and it can also be oberved for the presented symbol-frequency distri-

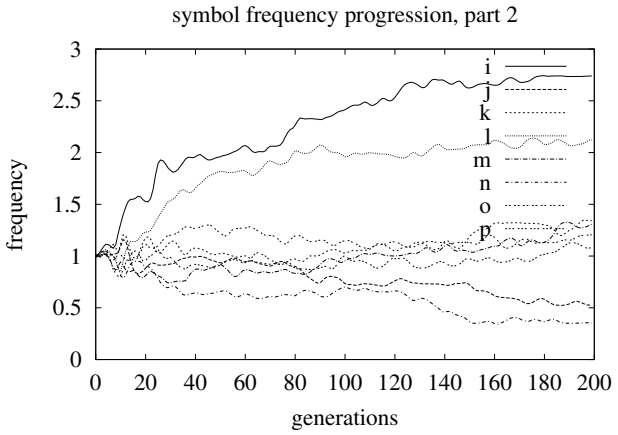


Figure 3: Progression of the mean symbol frequency in the code population.

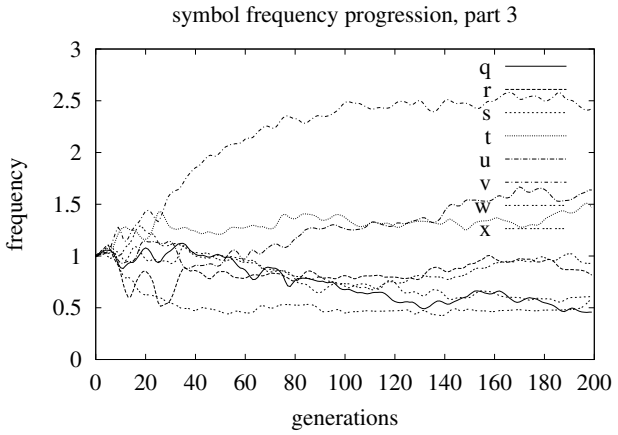


Figure 4: Progression of the mean symbol frequency in the code population.

butions, where a beneficial genotype-phenotype mapping is of interest.

Specifically, the figures show a classification of the symbols with respect to their relevance for the solving of the problem, as will be argued next. Due to initial oscillation, more reliable results are to be gained from late generations, which is why the frequencies of the final 200th generation shall be considered. In order to accomodate for variance of the mean average frequency values, symbols with a frequency of 0.8 or lower shall be designated as clearly under-represented in number in the genetic codes. As levels of statistical significance mostly come from $[0.9, .., 0.99]$, 0.8 represents a safe upper threshold for insignificance.

These symbols are $A, B, b, c, f, g, h, j, n, q, s, w, y, /$, which implies that four of five, that is, 80%, of

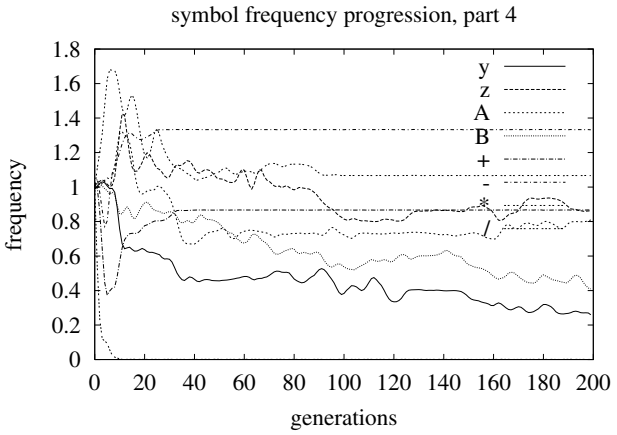


Figure 5: Progression of the mean symbol frequency in the code population. Note that the arithmetic-operator frequencies stabilize very fast and stay very stable. This is not an artefact.

the noise-representing symbols $A, B, y, z, /$ are under-represented, while 63% of the problem-relevant symbols, that is, 17 of 27 symbols, are represented with a frequency of one and higher.

The frequency of a symbol in a code heavily influences the frequency of the occurrence of the symbol in the phenotype onto which a genotype carrying the code is mapped. Thus, if non-noise symbols do and noise symbols do not become elements of the phenotype, this situation increases the likelihood that the phenotype has an above-average fitness. Therefore, the presented result represents the objective of the present work, as it verifies that the effect of code evolution also takes place on a hard problem in a way beneficial to the search process.

As for the principle of code evolution, we hypothesize that, for a certain problem, some individual code W, through a point code mutation, becomes better than another individual code L. Thus, W has a higher probability than L that its carrying individual has a genotype together with which W yields a good phenotype. Therefore, since selection on individuals is selection on codes, W has a higher probability than L of being propagated over time by reproduction and being subjected to code mutation. If such a mutation results in even higher code fitness, then the argument that worked for W works for W's mutant, and so forth.

6 CONCLUSIONS

It has been shown empirically that the effect of code evolution works on a hard problem, that is, genetic codes carried by individuals get adapted such that, during run time, problem-relevant phenotypic symbols are increasingly being used while irrelevant symbols are less often used.

7 FUTURE RESEARCH

Several hypotheses must be investigated, among them the claim that DGP with code evolution outperforms non-developmental approaches on hard problems. We argue especially that there is a high potential in code evolution for the application to data-mining problems, since, in this domain, a "good" composition of a symbol set is typically unknown since the functional relations between the variables are unknown due to the very nature of data-mining problems. We hypothesize that code evolution, through generation of redundant codes, enhances the learning of significant functional relations by biasing for problem-specific key data and filtering out of noise. Last not least, the hypothesized principle of code evolution, that is, the co-operative co-evolution of individuals and codes, shall be investigated.

References

Banzhaf, Wolfgang, Peter Nordin, Robert E. Keller and Frank D. Francone (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann, dpunkt.verlag.

Keller, Robert E. and Wolfgang Banzhaf (1996). Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In: *Genetic Programming 1996: Proceedings of the First Annual Conference* (John R. Koza, David E. Goldberg, David B. Fogel and Rick L. Riolo, Eds.). MIT Press, Cambridge, MA. Stanford University, CA. pp. 116–122.

Keller, Robert E. and Wolfgang Banzhaf (1999). The evolution of genetic code in genetic programming. In: *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 13-17, 1999, Orlando, Florida USA* (W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela and R.E. Smith, Eds.). Morgan Kaufmann. San Francisco, CA.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, Cambridge, MA.

Maeshiro, Tetsuya (1997). Structure of Genetic Code and its Evolution. PhD thesis. School of Information Science, Japan Adv. Inst. of Science and Technology. Japan.

O'Neill, M. and C. Ryan (2000). Crossover in grammatical evolution: A smooth operator?. In: *Genetic Programming* (Riccardo Poli et al., Ed.). Number 1802 In: *LNCS*. Springer.

Spector, Lee and Kilian Stoffel (1996). Ontogenetic programming. In: *Genetic Programming 1996: Proceedings of the First Annual Conference* (John R. Koza, David E. Goldberg, David B. Fogel and Rick L. Riolo, Eds.). MIT Press, Cambridge, MA.. Stanford University, CA. pp. 394–399.

Watson, James D., Nancy H. Hopkins, Jeffrey W. Roberts, Joan A. Steitz and Alan M. Weiner (1992). *Molecular Biology of the Gene.* Benjamin Cummings. Menlo Park, CA.

Genetic Programming for Combining Classifiers

W. B. Langdon and **B. F. Buxton**
Computer Science, University College, London, Gower Street, London, WC1E 6BT, UK
{W.Langdon,B.Buxton}@cs.ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/W.Langdon, /staff/B.Buxton
Tel: +44 (0) 20 7679 4436, Fax: +44 (0) 20 7387 1397

Abstract

Genetic programming (GP) can automatically fuse given classifiers to produce a combined classifier whose Receiver Operating Characteristics (ROC) are better than [Scott *et al.*, 1998b]'s "Maximum Realisable Receiver Operating Characteristics" (MR-ROC). I.e. better than their convex hull. This is demonstrated on artificial, medical and satellite image processing bench marks.

1 INTRODUCTION

[Scott *et al.*, 1998b] has previously suggested the "Maximum Realisable Receiver Operating Characteristics" for a combination of classifiers is the convex hull of their individual ROCs. However the convex hull is not always optimal [Yusoff *et al.*, 1998]. We show, on the problems used by [Scott *et al.*, 1998b], that genetic programming can evolve a combination of classifiers whose ROC are better than the convex hull of the supplied classifier's ROCs.

The next section gives the back ground to data fusion, Section 3 summarises Scott's work, his three bench marks are described in Section 4. The genetic programming system and its results are given in Sections 5 and 6. Finally we finish in Sections 7 and 8 with a discussion and conclusions.

2 BACKGROUND

There is considerable interest in automatic means of making large volumes of data intelligible to people. Arguably traditional sciences such as Astronomy, Biology and Chemistry and branches of Industry and Commerce can now generate data so cheaply that it far outstrips human resources to make sense of it. Increasingly scientists and Industry are turning to their computers not only to generate data but to try and make sense of it. Indeed the new science of Bioinformatics has arisen from the need for computer scientists and biologists to work together on tough data rich problems such as rendering protein sequence data useful. Of particular interest are the Pharmaceutical (drug discovery) and food preparation industries.

The terms Data Mining and Knowledge Discovery are commonly used for the problem of getting information out of data. There are two common aims: 1) to produce a summary of all or an interesting part of the available data 2) to find interesting subsets of the data buried within it. Of course these may overlap. In addition to traditional techniques, a large range of "intelligent" or "soft computing" techniques, such as artificial neural networks, decision tables, fuzzy logic, radial basis functions, inductive logic programming, support vector machines, are being increasingly used. Many of these techniques have been used in connection with evolutionary computation techniques such as genetic algorithms and genetic programming.

We investigate ways of combining these and other classifiers with a view to producing one classifier which is better than each. Firstly we need to decide how we will measure the performance of a classifier. In practise when using any classifier a balance has to be chosen between missing positive examples and generating too many spurious alarms. Such a balancing act is not easy. Especially in the medical field where failing to detect a disease, such as cancer, has obvious consequences but raising false alarms (false positives) also has implications for patient well being. Receiver Operating Characteristics (ROC) curves allow us to show graphically the trade off each classifier makes between its "false positive rate" (false alarms) and its "true positive rate" [Swets *et al.*, 2000]. (The true positive rate is the fraction of all positive cases correctly classified. While the false positive rate is the fraction of negative cases incorrectly classified as positive). Ex-

ample ROC curves are shown in Figures 1 and 3. We treat each classifier as though it has a sensitivity parameter (e.g a threshold) which allows the classifier to be tuned. At the lowest sensitivity level the classifier produces no false alarms but detects no positive cases, i.e. the origin of the ROC. As the sensitivity is increased, the classifier detects more positive examples but may also start generating false alarms (false positives). Eventually the sensitivity may become so high that the classifier always claims each case is positive. This corresponds to both true positive and false positive rates being unity, i.e. the top right hand corner of the ROC. On average a classifier which simply makes random guesses will have an operating point somewhere on the line between the origin and 1,1 (see dotted line in Figure 3).

Naturally we want our classifiers to have ROC curves that come as close to a true positive rate of one and simultaneously a false positive rate of zero. In Section 5 we score each classifier by the area under its ROC curve. An ideal classifier has an area of one. We also require the given classifiers, not only to indicate which class they think a data point belongs to, but also how confident they are of this. Values near zero indicate the classifier is not sure, possible because the data point lies near the classifier's decision boundary.

Arguably "Boosting" techniques combine classifiers [Freund and Schapire, 1996]. However Boosting is normally applied to only one classifier and produces improvements by iteratively retraining it. Here we will assume the classifiers we have are fixed, i.e. we do not wish to retrain them. Similarly Boosting is normally applied by assuming the classifier is operated at a single sensitivity (e.g a single threshold value). This means on each retraining it produces a single pair of false positive and true positive rates. Which is a single point on the ROC rather than the curve we require.

3 "MAXIMUM REALISABLE" ROC

[Scott *et al.*, 1998b] describes a procedure which will create from two existing classifiers a new one whose performance (in terms of its ROC) lies on a line connecting the performance of its two components. This is done by choosing one or other of the classifiers at random and using its result. E.g. if we need a classifier whose false positive rate vs. its true positive rate lies on a line half way between the ROC points of classifiers A and B, then the Scott's composite classifier will randomly give the answer given by A half the time and that given by B the other half, see Figure 1. (Of course persuading patients to accept such a random diagnose may not be straightforward).

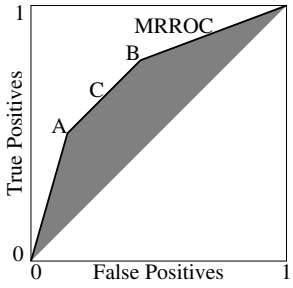


Figure 1: Classifier C is created by choosing equally between the output of classifier A and classifier B. Any point in the shaded area can be created. The "Maximum Realisable ROC" is its convex hull (solid line).

The performance of the composite can be readily set to any point along the line simply by varying the ratio between the number of times one classifier is used relative to the other. Indeed this can be readily extended to any number of classifiers to fill the space between them. The better classifiers are those closer to the zero false positive axis or with a higher true positive rate. In other words the classifiers lying on the convex hull.

Often classifiers have some variable threshold or tuning parameter whereby their trade off between false positives and true positives can be adjusted. This means their Receiver Operating Characteristics (ROC) are now a curve rather than a single point. Scott applied his random combination method to each set of points along the curve. So the "maximum realisable" ROC is the convex hull of the classifier's ROC. Indeed, if the ROC curve is not convex, an improved classifier can easily be created from it [Scott *et al.*, 1998b] (see Figure 4). The nice thing about the MRROC, is that it is always possible. But as we show it may be possible to do better automatically.

4 DEMONSTRATION PROBLEMS

[Scott *et al.*, 1998b] contains three benchmarks. Three of the following sections (4.2, 4.3 and 4.5) describe the preparation of the datasets. Sections 4.1 and 4.4 describe the two classifiers Scott used.

4.1 LINEAR CLASSIFIERS

In the first two examples (Sections 4.2 and 4.3) we use a tunable linear classifier for each data attribute (dimension). This classifier has a single decision value (a threshold). If examples of the class lie mostly at high values then, if a data point is above the threshold, the classifier says the data point is in the class. Otherwise it says it isn't. To produce a ROC curve

the threshold is varied from the lowest possible value of the associated attribute to the highest.

To use a classifier in GP we adopt the convention that non-negative values indicate the data is in the class. We also require the classifier to indicate its "confidence" in its answer. In our GP, it does this by the magnitude of the value it returns.

(The use of the complex plane would allow extension of this signalling to more than two classes. Absolute magnitude would continue to indicate the classifiers confidence. While the complex plane could be divided into (possibly unequal) angular segments, one for each class. An alternative would be to allocate each class a point in the complex plane. The designated class would be the one closest in the complex plane. But if two class origins were a similar distance from the value returned by GP this would indicate the classifier was not sure which of the two classes to choose).

The linear classifier splits the training set at the threshold. When predicting, it uses only those examples which are the same side of the threshold as the point to be classified and chooses the class to which most of them belong. Its "confidence" is the difference between the number of training examples below the threshold in each class divided by their sum. Note the value returned to GP lies in the range $-1 \ldots +1$.

4.2 OVERLAPPING GAUSSIAN

Following [Scott *et al.*, 1998b, Section 3.1 and Figure 3] we created a training and a verification dataset, each containing 5000 randomly chosen data points. The points are either in class 1 or class 2. 1250 values were created using Gaussian distributions each with a standard deviation of 0.5. Those of class 1 had means of 3 and 7. While those used to generate class 2 data had means of 5 and 9. Note this gives rise to interlocking regions with some degree of overlap at their boundaries, see Figure 2.

Clearly a linear classifier (LC) with only a single decision point can not do well on this problem. Figure 3 shows its performance in terms of the trade off between false positives and true positives.

4.3 THYROID

The data preparation for the Thyroid problem follows Scott's. The data was down loaded from the UCI machine learning repository[1]. `ann.train` was used for the training set and `ann.train2` for the verification set.

[1] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/thyroid-disease`

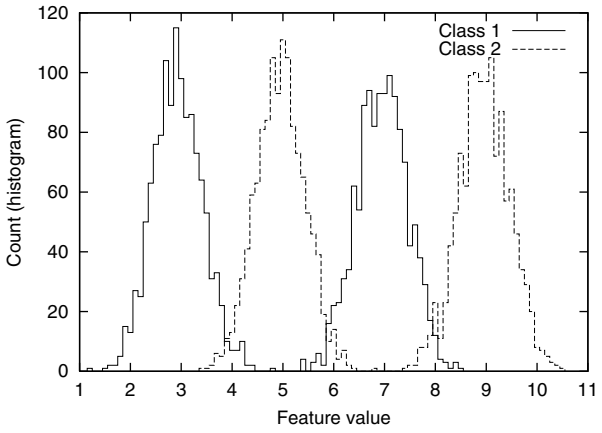


Figure 2: Example of a two class multi-modal data designed to be difficult for a linear classifier (Section 4.1).

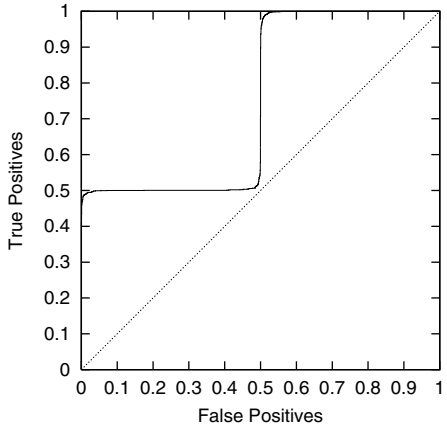


Figure 3: The Receiver Operating Characteristics curve produced by moving the decision boundary along the x-axis of Figure 2. The ROC are stepped as the classifier (Sect. 4.1) cannot capture the nature of the data.

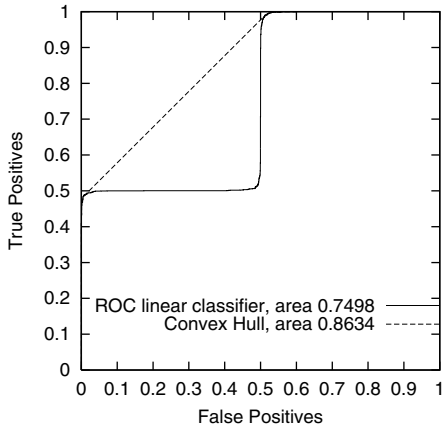


Figure 4: The convex hull of the ROC curve of Figure 3. Note a tunable classifier is improved by combining with itself, if its ROC are not convex.

(Both contain 3800 records). Originally it is a three class problem, the two classes for abnormal thyroids (79 and 199 records each in `ann.train`) were combined into one class. The GP is limited to using the two attributes (out of a total of 21) that Scott used. (Using all the attributes makes the problem much easier). Following strange floating point behaviour, both attributes were rescaled by multiplying by 1000. Rescaling means most numbers are integers between 1 and 200 (cf. Figure 10). Scott does not report rescaling. Two linear classifiers (LC18 and LC19) were trained, one on each attribute (D18 and D19) using the training set.

4.4 NAIVE BAYES CLASSIFIERS

The Bayes [Ripley, 1996; Mitchell, 1997] approach attempts to estimate, from the training data, the probability of data being in each class. Its prediction is the class with the highest estimated probability. We extend it 1) to include a tuning parameter to bias its choice of class and 2) to make it return a confidence based upon the difference between the two probabilities.

Naive Bayes classifiers are based on the assumption that the data attributes are independent. I.e. the probabilities associated with a data point are calculated by multiplying the estimates of the probabilities associated with each of its attributes.

The probabilities estimates of each class are based upon counting the number of instances in the training set for each attribute (dimension) that match both the point to be classified and the class, and dividing by the total number of instances which match regardless of the class. The estimates for each attribute are then multiplied together to give the probability of the data point being in a particular class.

The functions $P_{0,a}$ and $P_{1,a}$ use to estimate the probabilities for classes from training set attributes a

$$P_{c,a}(E) \quad = \quad Pr(\text{class} = c) \prod_{j \in a} Pr(X_j = v_j | \text{class} = c)$$

As an example, consider the data point $E = (6, 7, 8, 9, 10, 11, 12, 13)$ and a classifier using the set of attributes $a = \{2, 3, 5\}$. Then the probability E is in class 0, $P_{0,a}(E)$, is estimated to be, the probability of class 0 times, the probability that attribute 2 is 7 given the data is in class zero times, the probability attribute 3 is 8 (given the class is zero) times, the probability attribute 5 is 10 (given the class is zero). The calculation is repeated for the other classes (i.e. for class 1). The classifier predicts that E belongs to the class with the highest probability estimate. I.e. if $P_{0,a}(E) < P_{1,a}(E)$ then the Naive Bayes classifier (working on the set a of attributes) will predict the example data point E is in class 1, otherwise 0.

If there is no training data for a given class/attribute value combination, we follow [Kohavi and Sommerfield, 1996, page 11] and estimate the probability based on assuming there was actually a count of 0.5. ([Mitchell, 1997] suggests a slightly different way of calculating the estimates).

Since the denominators in $P_{c,a}$ are the same for all classes we can remove them and instead work with B

$$B_{c,a}(E) = \text{Number}(\text{class} = c) \prod_{j \in a} \text{Number}(X_j = v_j \cap \text{class} = c)$$

A threshold T ($0 \leq T \leq 1$), allows us to introduce a bias. I.e. if $(1 - T) \times B_{0,a}(E) < T \times B_{1,a}(E)$ then our Bayes classifier will predicts E is in class 1, otherwise 0. Finally we define the classifiers "confidence" to be $\frac{|B_{0,a}(E) - B_{1,a}(E)|}{(B_{0,a}(E) + B_{1,a}(E))}$.

4.5 GREY LANDSAT

Despite some care we have not been able to reproduce exactly the graphical results pictured in [Scott *et al.*, 1998a] and [Scott *et al.*, 1998b]. The Naive Bayes classifiers on the data we have appear to perform some what better. This makes the problem more challenging since there is less scope for improvement. [Scott *et al.*, 1998a] and [Scott *et al.*, 1998b] show considerable crossings in the ROC curves of the five classifiers they use. The absence of this in our data may also make it harder (see Figure 11).

The Landsat data comes from the Stalog project via the UCI machine learning repository[2]. The data is spilt into training (`sat.trn` 4425 records) and test (`sat.tst` 2000). Each record has 36 continuous attributes (8 bit integer values nominally in the range 0–255) and 6 way classification. (Classes 1, 2, 3, 4, 5 and 7). Following Scott; classes 3, 4 and 7 were combined into one (positive, grey) while 1, 2 and 5 became the negative examples (not-grey). `sat.tst` was kept for the holdout set.

The 36 data values represent intensity values for nine neighbouring pixels and four spectral bands (see Figure 5). While the classification refers to just the central pixel. Since each pixel has eight neighbours and

[2] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/statlog/satimage`

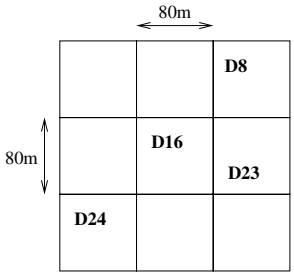


Figure 5: Each record contains data from nine adjacent Landsat pixels. Scott's five classifiers (nb16, nb16,23 nb16,23,24 nb23,24 and nb8,23,24) together use four attributes, Three (8, 16, 24) use spectral band 0 and the other (23) uses band 3. Notice how they straddle the central pixel in a diagonal configuration. However nb23,24 (which straddles both the area and the spectrum) has the best performance of Scott's Naive Bayes classifiers.

each may be in the dataset, data values appear multiple times in the data set. But when they do, they are presented as being different attributes each time. The data come from a rectangular area approximately five miles wide.

After reducing to two classes, the continuous values in `sat.trn` were partitioned into bins before it was used by the Naive Bayes classifier. Following [Scott *et al.*, 1998a, page 8], we used entropy based discretisation [Kohavi and Sommerfield, 1996], implemented in MLC++ `discretize.exe`[3], with default parameters. (Giving between 4 and 7 bins per attribute). To avoid introducing bias, the holdout data (`sat.tst`) was partitioned using the same bin boundaries.

`sat.trn` was randomly split into training (2956 records) and verification (1479) sets. The Bayes classifiers use the discrete data. In some experiments, the GP system was able to read data attributes values directly. In which case it used the continuous (floating point) value, rather than the attribute bin number.

5 GP CONFIGURATION

The GP is set up to signal its prediction of the class of each data value in the same was as the classifiers it can use. I.e. by return a floating point value, whose sign indicates the class and whose magnitude indicates the "confidence". (Note confidence is not constrained to lie in a particular range).

Following earlier work [Jacobs *et al.*, 1991; Soule, 1999; Langdon, 1998] each GP individual is composed of five trees. Each of which is capable of acting as a classifier. The use of signed numbers makes it natural to combine classifiers by adding them. I.e. the classification of the "ensemble" is the sum of the answers given by the five trees. Should a single classifier be very confident about its answer this allows it to "out vote" the all others.

We have not systematically experimented with the number of trees or alternative methods of combining them. The simplest problem can be solved with only one. Also in many individuals one or more of the trees appear to have little or a very basic function, such as always returning the same value or biasing the result by the threshold parameter.

5.1 FUNCTION AND TERMINAL SETS

The function set includes the four binary floating arithmetic operators (+, ×, − and protected division), maximum and minimum and absolute maximum and minimum. The latter two return the (signed) value of the largest, (or smallest) in absolute terms, of their inputs. IFLTE takes four arguments. If the first is less than or equal to the second, IFLTE returns the value of its third argument. Otherwise it returns the value of its fourth argument. INT returns the integer part of its argument, while FRAC(e) returns e - INT(e).

The classifiers are represented as floating point functions. Their threshold is supplied as their single argument. As described in Sections 4.1 and 4.4.

The terminal T yields the current value of the threshold being applied to the classifier being evolved by GP. In some experiments the terminals Dn were used. These contain the value of attribute n. Finally the GP population was initially constructed from a number of floating point values. These constants do not change as the population evolves. However crossover and mutation do change which constants are used and in which parts of the program. GPQUICK limits the number of constants to about 200.

5.2 FITNESS FUNCTION

Each new individual is tested on each training example with the threshold parameter (T) taking values from 0 to 1 every 0.1 (i.e. 11 values). So, depending upon the problem, it is run 55000, 41800 or 32516 times. For each threshold value the true positive rate is calculated. (The number of correct positive cases divided by the total number of positive cases). If a floating point exception occurs its answer is assumed to be wrong. Similarly its false positive rate is given by the no. of negative cases it gets wrong divided by the total no. of negative cases. It is possible to do worse

[3] `http://www.sgi.com/Technology/mlc`

than random guessing. When this happens, i.e. the true positive rate is less than the false positive rate, the sign of the output is reversed. This is common practise in classifiers.

Since a classifier can always achieve both a zero success rate and 100% false positive rate, the points (0,0) and (1,1) are always included. These plus the eleven true positive and false positive rates are plotted and the area under the convex hull is calculated. The area is the fitness of the individual GP program. Note the GP individual is not only rewarded for getting answers right but also for using the threshold parameter to get a range of high scores. Cf. Table 1.

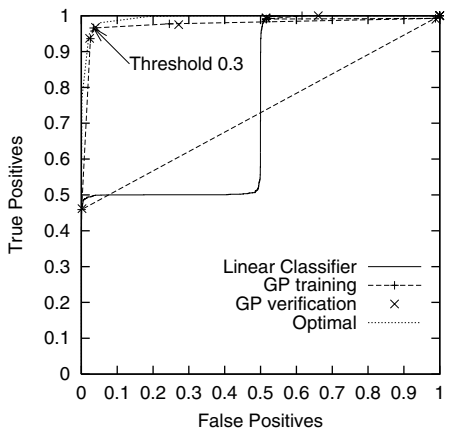


Figure 6: The ROC of GP (generation 21) classifier on interlocking Gaussians. Note it has near optimal performance.

6 RESULTS

6.1 OVERLAPPING GAUSSIAN

In the first run the best fitness score (on the training data) was 0.981556. The first individual with this score was found in generation 21 and was treated as the output of the GP. Its total size (remember it has five trees) is 92. On another 5000 random data points its fitness was 0.981607. Its ROC are shown in Figure 6. (The linear classifier's convex hull area is 0.85).

Since we know the under lying distribution in this (artificial) example, we can calculate the optimal ROC curve, see Figure 6. The optimal classifier requires three decision boundaries, which correspond to the overlap between the four interlocking Gaussians. Figure 6 shows this GP individual has near optimal behaviour. Its output for one threshold setting (0.3) is given in Figure 7. Figure 7 shows GP has been able to use the output of the linear classifier to create three decision points (remember the linear classifier has just one) and these lie at the correct points.

Figure 8 shows, in each of the problems, little change in program size occurs after the first five generations or so. This is despite little or no improvement in the best fitness. This may be due to "size fair crossover" [Langdon, 2000].

6.2 THYROID

In one run the best fitness rose steadily to a peak of 0.838019 at generation 50. The program with this fitness has a total size of 60. On the verification set it has a fitness of 0.860040. Its ROC are shown in Figure 9.

Its bulk behaviour is to combine the two given (single attribute, single threshold) classifiers to yield a rectangular area near the origin. As the threshold is increased, the rectangle grows to include more data points. Thus increasing the number of true positives, albeit at the expense of also increasing the number of false positive. Eventually with a threshold of 1, the rectangle covers all thyroid disease cases. Figure 10 shows the decision boundary for a threshold of 0.5. The superior performance of the GP classifier arises, at least in part, because it has learnt to recognise regularities in the training data. In particular it has spotted columns of data which are predominantly either all negative or positive and adjusted its decision boundary to cover these.

6.3 GREY LANDSAT

In the first GP run fitness rose quickly in the first six generations but much slower after that. The best training fitness was 0.981855 which was first discovered in generation 49. The ROC of this individual are shown in Figure 11. The area of its convex hull is bigger than all of those of its constituent classifiers. On the holdout set, its ROC are better than all of them, except for one threshold value where it has 3 false negatives v. 1 for the best of the Naive Bayes classifiers.

7 DISCUSSION

So far we have used simple classifiers with few parameters that are learnt. This appears to make them robust to over fitting. In contrast one often needs to be careful when using GP to avoid over fitting. In these experiments we have seen little evidence of over fitting. This may be related to the problems themselves or the choice of multiple tree programs or the absence of "bloat". The absence of bloat may be due

Table 1: GP Parameters (Variations between problems given in brackets or on separate lines)

Objective:	Evolve a function with Maximum Convex Hull Area	
Function set:	INT FRAC Max Min MaxA MinA MUL ADD DIV SUB IFLTE	
common plus	*Gaussians*	LC
	Thyroid	LC17 LC18
	Grey Landsat	nb16 nb16,23 nb16,23,24 nb23,24 nb8,23,24
Terminal set:	*Gaussians*	T, 0, 1, 200 unique constants randomly chosen in $-1 \ldots +1$
	Thyroid	T, D17, D18, 0, 0.1, 1, 212 unique constants randomly chosen from the test set.
	Grey Landsat	T 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Fitness:	Area under convex hull of 11 ROC points. (5000, 3800, 2956) randomly chosen test points	
Selection:	generational (non elitist), tournament size 7	
Wrapper:	$\geq 0 \Rightarrow$ positive, negative otherwise	
Pop Size:	500	
No size or depth limits		
Initial pop:	ramped half-and-half (2:6) (half terminals are constants)	
Parameters:	50% size fair crossover [Langdon, 2000], 50% mutation (point 22.5%, constants 22.5%, shrink 2.5% subtree 2.5%)	
Termination:	generation 50	

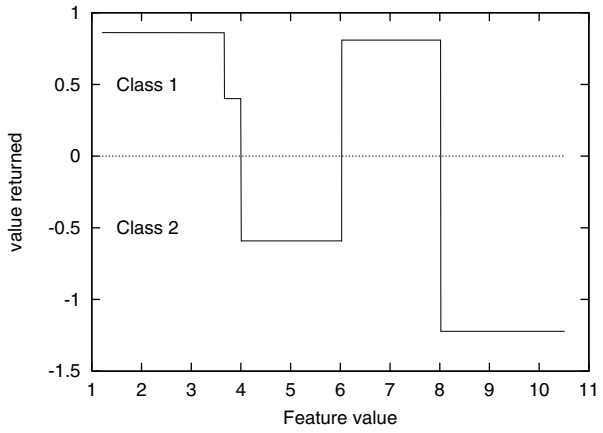


Figure 7: Value returned by evolved classifier (threshold=0.3) evolved on the interlocking Gaussians problem. High fitness comes from GP being able to use given classifier to distinguish each of the Gaussians. Note zero crossings align with Gaussians, Figure 2.

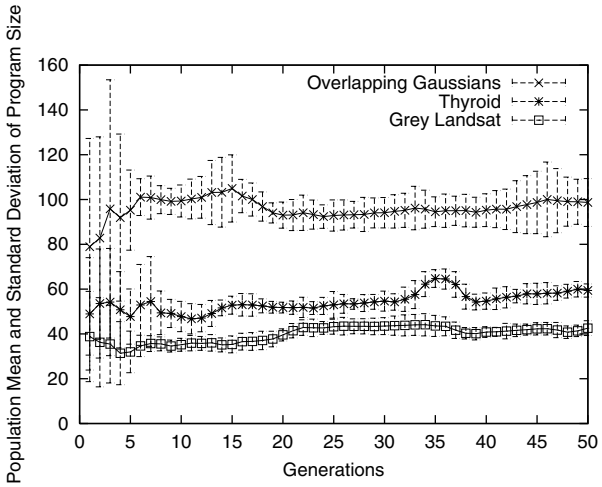


Figure 8: Evolution of total program size in one GP run of each the three problems.

to our choice of size fair crossover and a high mutation rate. Our intention is to evaluate this GP approach on more sophisticated classifiers and on harder problems. Here we expect it will be essential to ensure the classifiers GP uses do not over fit, however this may not be enough to ensure the GP does not.

8 CONCLUSIONS

[Scott *et al.*, 1998b] has proved one can always combine classifiers with variable thresholds to yield a composite with the "Maximum Realisable Receiver Operating Characteristics" (MRROC). Scott's MRROC is the convex hull of the Receiver Operating Characteristics of the individual classifiers. Previously we showed [Langdon and Buxton, 2001] genetic programming can in principle do better automatically. Here we have shown, using Scott's own bench marks, that GP offers a systematic approach to combining classifiers which may exceed Scott's MRROC. (Using [Scott *et al.*, 1998b]'s proof, we can ensure GP does no worse than the MRROC).

Mutation and size fair crossover [Langdon, 2000] mean there is little bloat.

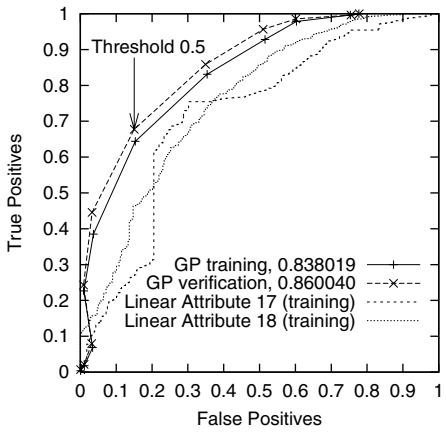


Figure 9: The ROC produced by GP (gen 50) using threshold values $0, 0.1, \ldots, 1.0$ on the Thyroid data.

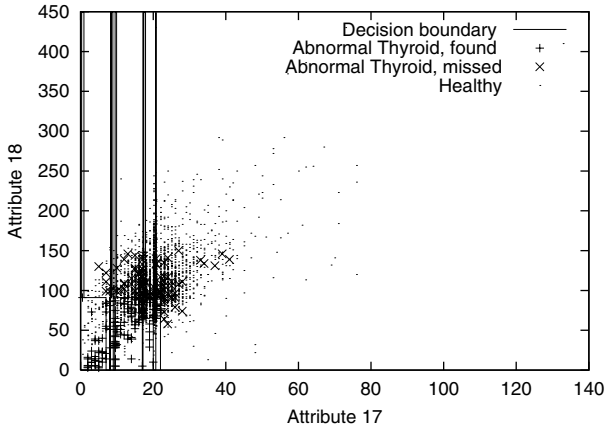


Figure 10: Decision boundary (threshold 0.5) for the Thyroid data produced by GP. The origin side of the boundary are abnormal (179 found, missed 99). 2982 correctly cleared, 540 false alarms.

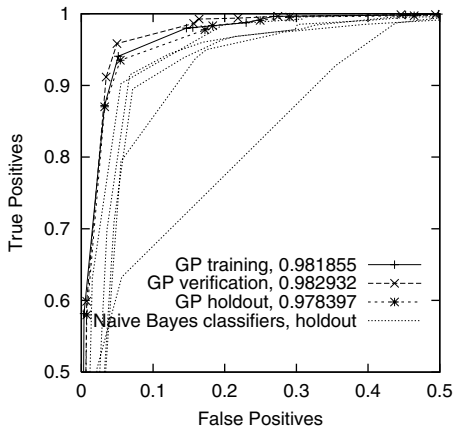


Figure 11: The ROC produced by GP (generation 49) using threshold values $0, 0.1, \ldots, 1.0$ on the Grey Landsat data. The ROC of the five given Naive Bayes classifiers are given on the holdout set.

References

[Freund and Schapire, 1996] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proc.* 13^{th} *International Conference*, pp 148–156. Morgan Kaufmann.

[Jacobs *et al.*, 1991] R. A. Jacobs, M. I. Jordon, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[Kohavi and Sommerfield, 1996] R. Kohavi and D. Sommerfield. MLC++: Machine learning library in C++. Technical report, `http://www.sgi.com/Technology/mlc/util/util.ps`.

[Langdon and Buxton, 2001] Evolving receiver operating characteristics for data fusion. In J. F. Miller *et al.*, eds., *EuroGP'2001*, *LNCS 2038*, pp 87–96, Springer-Verlag.

[Langdon, 1998] W. B. Langdon. *Data Structures and Genetic Programming*. Kluwer.

[Langdon, 2000] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming & Evolvable Machines*, 1(1/2):95–119.

[Mitchell, 1997] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Ripley, 1996] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press.

[Scott *et al.*, 1998a] M. J. J. Scott, M. Niranjan, and R. W. Prager. Parcel: feature subset selection in variable cost domains. Technical Report CUED/F-INFENG/TR.323, Cambridge University, UK.

[Scott *et al.*, 1998b] Realisable classifiers: Improving operating performance on variable cost problems. In P. H. Lewis and M. S. Nixon, eds., *Ninth British Machine Vision Conference*, pages 304–315,

[Soule, 1999] T. Soule. Voting teams: A cooperative approach to non-typical problems using genetic programming. In W. Banzhaf *et al.*, eds., *GECCO*, pages 916–922. Morgan Kaufmann.

[Swets *et al.*, 2000] J. A. Swets, R. M. Dawes, and J. Monahan. Better decisions through science. *Scientific American*, pages 70–75, October.

[Yusoff *et al.*, 1998] Combining multiple experts for classifying shot changes in video sequences. In *IEEE Int. Conf. on Multimedia Computing and Systems.*

When Short Runs Beat Long Runs

Sean Luke
George Mason University
http://www.cs.gmu.edu/∼sean/

Abstract

What will yield the best results: doing one run n generations long or doing m runs n/m generations long each? This paper presents a technique-independent analysis which answers this question, and has direct applicability to scheduling and restart theory in evolutionary computation and other stochastic methods. The paper then applies this technique to three problem domains in genetic programming. It discovers that in two of these domains there is a maximal number of generations beyond which it is irrational to plan a run; instead it makes more sense to do multiple shorter runs.

1 INTRODUCTION

Research in stochastic search has long struggled to determine how best to allocate precious resources to find the best possible solution. This issue has not gone away with increases in computer power: rather, the difficulty of our optimization problems has more than kept up with our new computational muscle. And the rise of massive parallelism has added an additional constraint to how we may divvy up our total evaluations.

Studies in resource allocation have attacked different aspects of the problem. One popular area of study in genetic algorithms is *online restart determination*. This area asks: while in the midst of a stochastic run and with no *a priori* knowledge, should I restart now and try again? This used to be a critical issue for GAs because of the spectre of premature convergence. Detecting the approach of premature convergence during a run saved valuable cycles otherwise wasted. There has been much work in this area; for a few examples, see [Goldberg, 1989, Collins and Jefferson, 1991, Eshelman and Schaffer, 1991]. This work usually assumes certain heuristics about convergence which may or may not be appropriate. Commonly the work relies on variance within a population or analysis of change in performance over time. These techniques are ad-hoc, but more problematic, they are often domain-specific. For example, they would not work in general on genetic programming.

In some sense, detecting premature convergence is an analysis of time-to-failure. A more cheerful focus in evolutionary computation, *convergence velocity*, is not directly involved in resource allocation but has many important ties. Evolutionary strategies analysis can demonstrate the rates at which specific techniques are expected to move towards the optimum, either in solution space or in fitness space [Bäck, 1996]. Since different population sizes can be considered different techniques, this analysis can shed light on resource allocation issues.

One area which directly tackles resource allocation is *scheduling* [Fukunaga, 1997]. A schedule is a plan to perform n runs each l generations long. The idea is to come up with a schedule which best utilizes available resources, based on past knowledge about the algorithm built up in a database. Typically this knowledge is derived from previous applications of the algorithm to various problem domains different from the present application. [Fukunaga, 1997] argues that previous problem domains are a valid predictor of performance curves in new domains, for genetic algorithms at least.

Outside of evolutionary computation, there is considerable interest in *restart methods* for global optimization. For difficult problems where one expects to perform many runs before obtaining a satisfactory solution, one popular restart method is to perform random restarts [Hu et al., 1997, Ghannadian and Alford, 1996]. If the probability density function of probability of convergence at time t is known then it is also possible to derive the *optimum restart time* such that, as the number of evaluations approaches infinity, the algorithm converges with the most rapid possible rate [Magdon-Ismail and Atiya, 2000].

Lastly, much genetic programming work has assumed that

the optimum can be discovered. A common metric of time-to-optimal-discovery is called *cumulative probability of success* [Koza, 1992]. However, this metric does not directly say anything about the rate of success nor whether or not shorter runs might yield better results.

The analysis presented in this paper takes a slightly different tack. It attempts to answer the question: is it rational to try a single run n generations long? Would it be smarter to instead try m runs each $\frac{n}{m}$ generations long? As it turns out, this question can be answered with a relatively simple procedure derived from a manipulation of order statistics. The procedure is entirely problem-independent; in fact it can be easily applied to any stochastic search method.

Unlike some of the previous methods, this analysis does not attempt to determine how long it takes to discover the optimum, nor the probability of discovering it, nor how fast the system converges either globally or prematurely. It is simply interested in knowing whether one schedule is likely to produce better net results than another schedule.

This paper will first present this analysis and prove it. It will then apply the analysis to three problems in genetic programming, an evolutionary computation approach which is notorious for requiring large populations and short runlengths. It then discusses the results.

2 PRELIMINARIES

We begin with some theorems based on order statistics, which are used to prove the claims in Section 3. These theorems tell us what the expected value is of the highest quality (fitness) found among of some n samples picked with replacement from a population. The first theorem gives the continuous case (where the population is infinite in size). The second theorem gives the discrete case.

Theorem 1 *Let $X_1, ..., X_n$ be n independent random variables representing n selections from a population whose density function is $f(x)$ and whose cumulative density function is $F(x)$. Let X_{max} be the random variable representing the maximum of the X_i. Then the expected value of X_{max} is given by the formula $\int_{-\infty}^{\infty} xnf(x)(F(x))^{n-1}dx$.*

Proof Note that for any given x, $X_{max} \leq x$ if and only if for all i, $X_i \leq x$. Then the cumulative density function $F_{X_{max}}(x)$ of the random variable X_{max} is as follows: $F_{X_{max}}(x) = P(X_{max} \leq x) = P(X_1 \leq x)P(X_2 \leq x)...P(X_n \leq x) = F(x)F(x)...F(x) = (F(x))^n$. The density function $f_{X_{max}}(x)$ for X_{max} is the derivative of this, so $f_{X_{max}}(x) = nf(x)(F(x))^{n-1}$. The expected value of any density function $G(x)$ is defined as $\int_{-\infty}^{\infty} xG(x)dx$, so the expected maximum value of the n random variables is equal to $\int_{-\infty}^{\infty} xf_{X_{max}}dx = \int_{-\infty}^{\infty} xnf(x)(F(x))^{n-1}dx$. ■

Lemma 1 *Given n selections with replacement from the set of numbers $\{1, ..., m\}$, the probability that r is the maximum number selected is given by the formula $\frac{r^n-(r-1)^n}{m^n}$. The sum of probabilities for all such r is 1.*

Proof Consider the set S_r of all possible events for which, among the n numbers selected with replacement, r is the maxiumum number. These events share the two following criteria. First, for each selection x among the n selections, $x \leq r$. Second, there exists a selection y among the n for which $y \geq r$. The complement to this second criterion is that for each selection x among the n selections, $x \leq (r-1)$. Since this complement is a strict subset of the first criterion, then S_r is the set difference between the first criterion and the complement, thus the probability of P_r of an event in S_r occuring is the difference between the probability of the first criterion and the probability of the complement, that is, $P_r = P(\forall x : x \leq r) - P(\forall x : x \leq (r-1))$.

For a single selection with replacement from the set of numbers $\{1, ..., m\}$, the probability that the selection is less than or equal to some value q is simply $\frac{q}{m}$. Thus for n independent such selections, the probability that all are $\leq q$ is $\frac{q^n}{m^n}$. Substituting into the solution above, we get $P_r = \frac{r^n}{m^n} - \frac{(r-1)^n}{m^n} = \frac{r^n-(r-1)^n}{m^n}$. Further, the sum of such probabilities for all r is $\sum_{r=1}^{m} \frac{r^n-(r-1)^n}{m^n} = \frac{1^n-0^n}{m^n} + \frac{2^n-1^n}{m^n} + \cdots + \frac{m^n-(m-1)^n}{m^n} = \frac{m^n-0^n}{m^n} = 1$ ■

Theorem 2 *Consider a discrete distribution of m trials, with each trial r having a quality $Q(r)$, sorted by Q so that trial 1 has the lowest quality and trial m has the highest quality. If we pick n trials with replacement from this distribution, the expected value of the maximum quality among these n trials will be*

$$\sum_{r=1}^{m} Q(r)\frac{r^n - (r-1)^n}{m^n}$$

Proof The *rank* of a trial is its position $1, ..., m$ in the sorted order of the m trials. The expected value of the maximum quality among the n selected trials is simply the sum, over each rank r, of the probability that r will be the highest rank among the selected trials, times the quality of r. This probability is given by Lemma 1. Hence the summation is $\sum_{r=1}^{m} Q(r)\frac{r^n-(r-1)^n}{m^n}$. ■

3 SCHEDULES

These order statistics results make possible the creation of tools that determine which of two techniques A and B is expected to yield the best results. This paper discusses a specific subset of this, namely, determining whether evo-

lutionary technique A run m_1 generations n_1 times (commonly 1 time) is superior the same technique A run m_2 generations n_2 times, where $n_1 m_1 = n_2 m_2$. We begin with some definitions.

Definition 1 *A* **schedule** *S is a tuple $\langle n_S, l_S \rangle$, representing the intent to do n_S independent runs of length l_S each.*

Definition 2 *Let S, T be two schedules. Then S* **reaches** *T if n_S runs of length l_S are expected to yield as good as or higher quality than n_T runs of length l_T. Define the predicate operator $S \succeq T$ to be true if and only if S reaches T.*

The following two theorems assume that higher quality is represented by higher values. In fact, for the genetic programming examples discussed later, the graphs shown have *lower* fitness as higher quality; this is rectified simply by inverting the fitness values.

Theorem 3 *Let $p_t(x)$ be the probability density function and $P_t(x)$ the cumulative probability density function of the population of all possible runs, reflecting their quality at time t (assume higher values mean higher quality). Then $S \succeq T$ if and only if:*

$$\int_{-\infty}^{\infty} x n_S p_{l_S}(x)(P_{l_S}(x))^{n_S-1}dx \geq \int_{-\infty}^{\infty} x n_T p_{l_T}(x)(P_{l_T}(x))^{n_T-1}dx$$

Proof Both sides of this inequality are direct results of Theorem 1. ■

The continuous case above is not that useful in reality, since we rarely will have an infinite number of runs to draw from! However, if we perform many runs of a given runlength, we can estimate the expected return from doing n runs at that runlength, and use this to determine if some schedule outperforms another schedule. The estimate makes the assumption that the runs we performed (our sample) is *exactly representative* of the full population of runs of that runlength.

Theorem 4 *Given a schedule $S = \langle n_S, l_S \rangle$, consider a random sample, with replacement, of m_S runs from all possible runs of runlength l_S. Let these runs be sorted by quality and assigned ranks $1, ..., m_S$, where a run's rank represents its order in the sort, and rank 1 is the lowest quality. Further, let $Q_S(r)$ be the quality of the run from the sample whose rank is r; $Q_S(r)$ should return higher values for higher quality. For another schedule T, similarly define m_T and $Q_T(r)$. Then an estimate of reaching is as follows. $S \succeq T$ if and only if:*

$$\sum_{r=1}^{m_S} Q_S(r)\frac{r^{n_S}-(r-1)^{n_S}}{{m_S}^{n_S}} \geq \sum_{r=1}^{m_T} Q_T(r)\frac{r^{n_T}-(r-1)^{n_T}}{{m_T}^{n_T}}$$

Proof Both sides of this inequality are direct results of Theorem 2. ■

These theorems give tools for determining whether one schedule reaches another. We can use this to estimate what schedule is best for a given technique. If we wanted to examine a technique and determine its best schedule, we have two obvious options:

1. Perform runs out to our maximum runlength, and use run-data throughout the runs as estimates of performance at any given time t. The weakness in this approach is that these estimates are not statistically independent.
2. Perform runs out to a variety of runlengths. The weakness in this approach is that it requires $O(n^2)$ evaluations.

A simple compromise adopted in this paper is to do runs out to 1 generation, a separate set of runs out to 2 generations, another set of runs out to 4 generations, etc., up to some maximal number of generations. This is $O(n)$, yet still permits runlength comparisons between statisticatlly independent data sets.

Two statistical problems remain. First, these comparisons do not come with a difference-of-means test (like a t-test or ANOVA). The author is not aware of the existence of any such test which operates over order statistics appropriate to this kind of analysis, but hopes to develop (or discover!) one as future work. This is alleviated somewhat by the fact that the result of interest in this paper is often not the hypothesis but the null hypothesis. Second, the same run data for a schedule is repeatedly compared against a variety of other schedules; this increases the alpha error. To eliminate this problem would necessitate $O(n^3)$ evaluations (!) which is outside the bounds of the computational power available at this time.

4 ANALYSIS OF THREE GENETIC PROGRAMMING DOMAINS

Genetic Programming is an evolutionary computation field with traditionally short runlengths and large population sizes. Some of this may be due to research following in the footsteps of [Koza, 1992, 1994] which used large populations (500 to 1000 individuals) and short runlengths (51 generations). Are such short runlengths appropriate? To

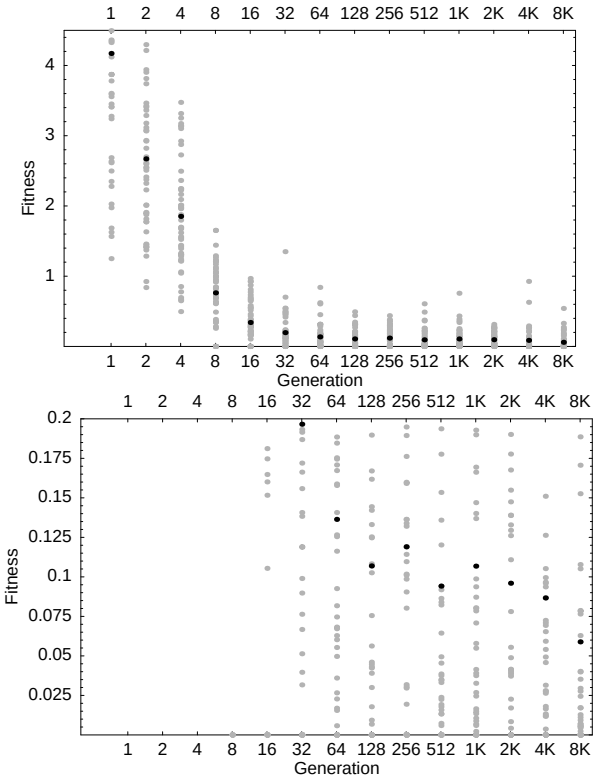


Figure 1: Runlength vs. Fitness, Symbolic Regression Domain (Including Detail)

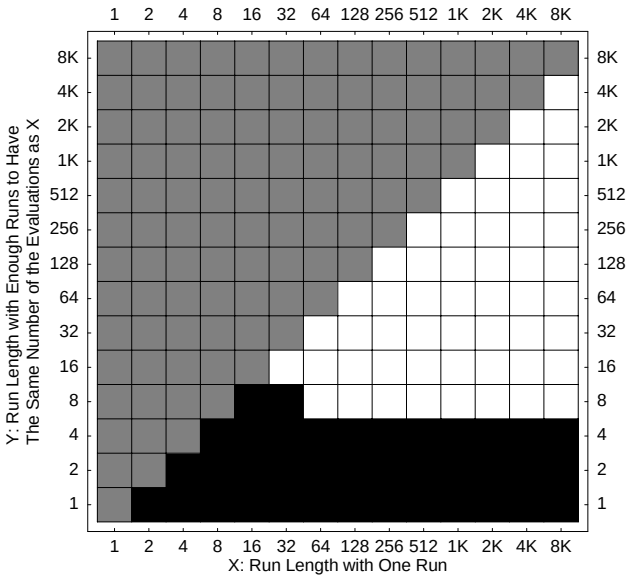


Figure 2: Runlength Analysis of Symbolic Regression Domain. Areas are black where X is a superior strategy to Y and white where Y is as good or better than X. Gray regions are out of bounds.

consider this, I analyzed three GP problem domains: Symbolic Regression, Artificial Ant, and Even 10-Parity. These three domains have very different dynamics.

In all three domains, I performed 50 independent runs for runlengths of 2^i generations ranging from 2^0 to some 2^{max}. Because these domains differ in evaluation time, max varied from domain to domain. For Symbolic Regression, $2^{max} = 8192$. For Artificial Ant, $2^{max} = 2048$. For Even 10-Parity, $2^{max} = 1024$. For all three domains, lower fitness scores represent better results. The GP system used was ECJ [Luke, 2000].

The analysis graphs presented in this paper compare single-run schedules with multiple-run schedules of shorter length. However additional analysis comparing n-run schedules with nm-run schedules of shorter length has yielded very similar results.

4.1 Symbolic Regression

The goal of the Symbolic Regression problem is to find a symbolic expression which best matches a set of randomly-chosen target points from a predefined function. Ideally, Symbolic Regression discovers the function itself. I used the traditional settings for Symbolic Regression as defined in [Koza, 1992], with a population size of 500 and tournament selection with a tournament of size 7. The function to be fitted was $x^4 + x^3 + x^2 + x$.

Unlike the other two problems, Symbolic Regression operates over a continuous fitness space; if cannot find the optimal solution, it will continue to find incrementally smaller improvements. Although Symbolic Regression very occasionally will discover the optimum, usually it tends towards incrementalism. As such, Symbolic Regression fitness values can closely approach 0 without reaching it, so Figure 1 shows both zoomed-out and zoomed-in versions of the same data. Grey dots represent individual best-of-run results for each run; black dots represent means of 50 runs of that runlength.

As can be seen, the mean continues to improve all the way to runlengths of 8192. But is it rational to plan to do a run out to 8192 generations? Figure 2 suggests otherwise.

The runlength analysis graphs can be confusing. On the graph, the point $(X, Y), X > Y$ indicates the result of comparing a schedule $A = \langle 1, X \rangle$ with the schedule $B = \langle \frac{X}{Y}, Y \rangle$, which has the same total number of evaluations. The graph is white if $B \succeq A$, black otherwise. This is a lower-right matrix: gray areas are out-of-domain regions.

Figure 2 shows that the expected quality of a single run of length $\geq$32 is reached by doing some n runs of length 16 which total the same number of evaluations. Another

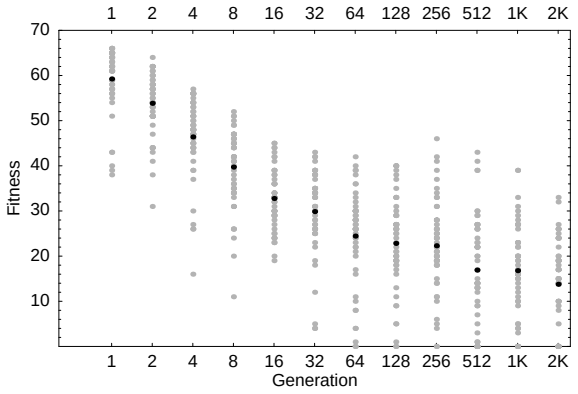


Figure 3: Runlength vs. Fitness, Artificial Ant Domain

interesting feature is that there is a minimum acceptable runlength: under no circumstances could multiple runs less than 8 generations reach a single run of larger size.

What about comparing a schedule $A = \langle c, X \rangle$ with schedules $B = \langle \frac{cX}{Y}, Y \rangle$? Even with values of $c = 2, 4, 8$, the resultant runlength analysis graphs were almost identical.

4.2 Artificial Ant

Artificial Ant moves an ant across a toroidal world, attempting to follow a trail of food pellets and eat as much food as possible in 400 moves. I used the traditional Artificial Ant settings with the Santa Fe trail as defined in [Koza, 1992], with a population size of 500 and tournament selection using a tournament of size 7.

As shown in Figure 3, the mean Artificial Ant best-of-run fitness improved monotonically and steadily with longer runlengths clear out to 2048 generations. But this did not mean that it was rational to plan to do a run out that far. Figure 4 suggests that single runs of runlengths beyond 64 generations were reached by multiple runs with shorter runlengths but the same number of total evaluations.

This is very similar to the Symbolic Regression results. Also similar was the existence of a minimum acceptable runlength: runs less than 4 could not reach a single run of larger size. Lastly, runlength analysis graphs with values of $c = 2, 4,$ or 8 were very similar.

4.3 Even-10 Parity

The last problem analyzed was Even-10 Parity, a very difficult problem for Genetic Programming. Even-10 Parity evolves a symbolic boolean expression which correctly identifies whether or not, in a vector of 10 bits, an even number of them are 1. This is a large and complex function and necessitates a large GP tree. To make the problem even harder, I used a small population (200), but otherwise

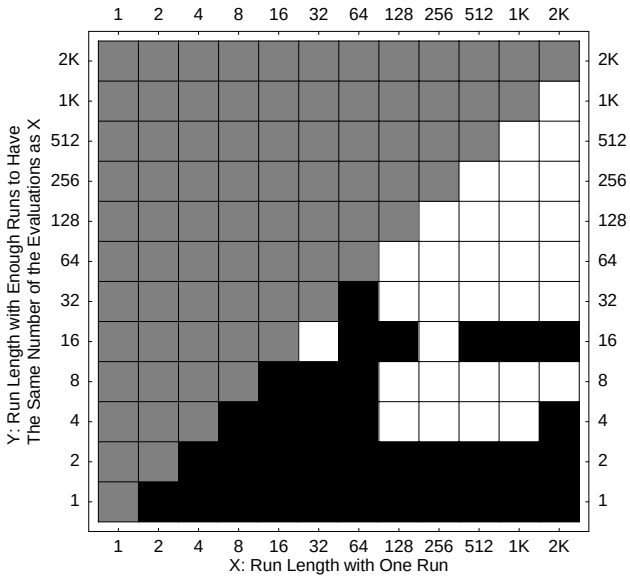


Figure 4: Runlength Analysis of Artificial Ant Domain. Areas are black where X is a superior strategy to Y and white where Y is as good or better than X. Gray regions are out of bounds.

followed the specifications for the Parity problem family as outlined in [Koza, 1992].

Figure 5 shows just how difficult it is for Genetic Programming to solve the Even-10 Parity problem. Even after 1024 generations, no run has reached the optimum; the mean best-of-run fitness has improved by only 25% over random solutions. The curve does not resemble the logistic curve of the other two GP domains.

One might suppose that in a domain where 1024 generations improves little over 1 generation, runlength analysis would argue for the futility of long runs. Yet the results were surprising: a single run of any length was always consistently superior to multiple runs of shorter lengths. Even though Even-10 Parity is very difficult for Genetic Programming to solve, it continues to plug away at it. It is conceivable that, were we to run out far enough, we might see a maximal rational runlength in the Even-10 Parity domain. Nonetheless, it is surprising that even at 1024 generations, Even-10 Parity is still going strong.

5 DISCUSSION

As the Symbolic Regression and Artificial Ant domains have shown, there can be a runlength beyond which it seems irrational to plan to do runs, because more runs of shorter length will do just as well if not better. I call this

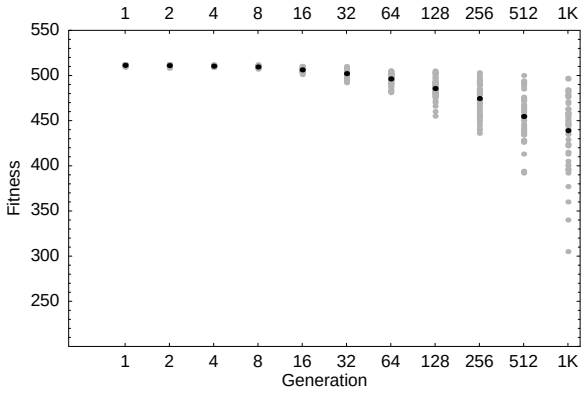


Figure 5: Runlength vs. Fitness, Even-10 Parity Domain

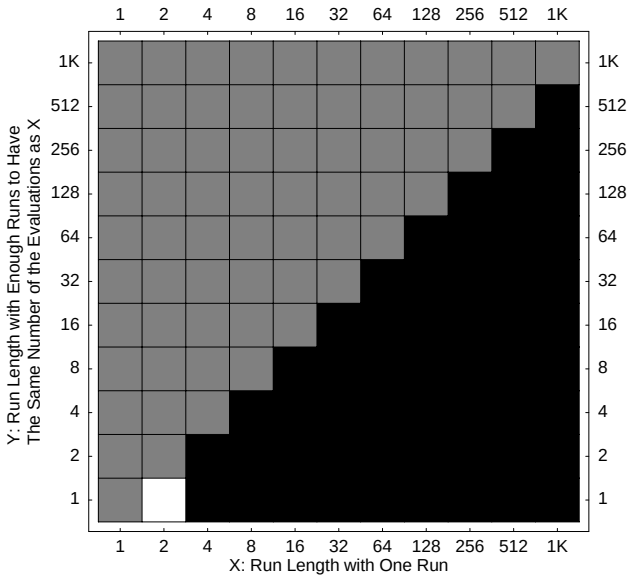


Figure 6: Runlength Analysis of Even-10 Parity Domain. Areas are black where X is a superior strategy to Y and white where Y is as good or better than X. Gray regions are out of bounds.

runlength a *critical point*. The location of the critical point suggests interesting things about the ability of the technique to solve the problem at hand. As the critical point approaches 1, the technique becomes less and less of an improvement over blind random search.

Symbolic Regression only occasionally finds the optimum, but if it is lost, around generation 64 it seems to begin to search for incrementally smaller values. One is tempted to suggest that this is why it is irrational to continue beyond about generation 32 or so. However, while the curve flattens out, as the detail shows, it still makes improvements in fitness. The critical feature is that the *variance* among the runs stays high even though the mean improves only slowly. This is what makes it better to do 2 runs of length 32 (or 8 of 8) than 1 run of length 64, for example.

Artificial Ant demonstrates a similar effect. Even though the mean improves steadily, the variance after generation 32 stays approximately the same. As a result, 4 runs of 32 will handily beat out 1 run of 128 despite a significant improvement in the mean between 32 and 128 generations.

The interesting domain is Even 10-Parity. In this domain the mean improves and the variance also continues to increase. As it turns out, the mean improves just enough to counteract the widening variance. Thus even though this is a very difficult problem for genetic programming to solve, it never makes sense to do multiple short runs rather than one long run!

Symbolic Regression and Artificial Ant also suggest that there can exist a *minimum* runlength such that any number of runs with fewer generations are inferior to a single run of this runlength. In some sense it is also irrational to do multiple runs with fewer generations than this minimum runlength instead of (at least) one run at the minimum runlength. Thus there is a window between the minimum and maximum rational runlengths. If one has enough evaluations, it appears to makes most sense to spend them on runs within this window of rationality.

One last item that should be considered is *evaluation time*, which for genetic programming is strongly influenced by the phenomenon of *code bloat* . As a genetic programming run continues, the size of its individuals grows dramatically, and so does the amount of time necessary to breed and particularly to evaluate them. So far we have compared schedules in terms of total number of evaluations; but in the case of genetic programming it might make more sense to compare them in terms of *total runtime*. The likely effect of this would be to make the maximally rational runtime even shorter. In the future the author hopes to further explore this interesting issue.

6 CONCLUSION

Genetic programming has traditionally not done runs longer than 50 generations or so, at least for the common cannonical problems. Instead it prefers larger population sizes. The results of this analysis suggest one reason why this might be: beyond a very small runlength (16 for Symbolic Regression, about 32 or 64 for Artificial Ant) the diminishing returns are such that it makes more sense to divvy up the total evaluations into multiple smaller runs.

But “rapidly diminishing returns” is not the same thing as “difficult problem”. In a hard problem like Even-10 Parity,

it *still* makes sense on average to press forward rather than do many shorter runs.

This paper presented a formal, heuristic-free, domain-independent analysis technique for determining the expected quality of a given schedule, and applied it to three domains in genetic programming, with interesting results. But this analysis is applicable to a wide range of stochastic techniques beyond just GP, and the author hopes to apply it to other techniques in the future.

Acknowledgements

The author wishes to thank Ken DeJong, Paul Wiegand, Liviu Panait, and Jeff Bassett for their considerable help and insight.

References

T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

R. J. Collins and D. R. Jefferson. Selection in massively parallel genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 249–256, 1991.

L. J. Eshelman and J. D. Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 115–122, 1991.

A. Fukunaga. Restart scheduling for genetic algorithms. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, 1997.

F. Ghannadian and C. Alford. Application of random restart to genetic algorithms. *Intelligent Systems*, 95:81–102, 1996.

D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.

X. Hu, R. Shonkwiler, and M. Spruill. Random restart in global optimization. Technical Report 110592-015, Georgia Tech School of Mathematics, 1997.

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

John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

Sean Luke. ECJ: A Java-based evolutionary computation and genetic programming system. Available at http://www.cs.umd.edu/projects/plus/ecj/, 2000.

M. Magdon-Ismail and A. Atiya. The early restart algorithm. *Neural Computation*, 12(6):1303–1312, 2000.

A Survey and Comparison of Tree Generation Algorithms

Sean Luke
George Mason University
http://www.cs.gmu.edu/∼sean/

Liviu Panait
George Mason University
http://www.cs.gmu.edu/∼lpanait/

Abstract

This paper discusses and compares five major tree-generation algorithms for genetic programming, and their effects on fitness: RAMPED HALF-AND-HALF, PTC1, PTC2, RANDOMBRANCH, and UNIFORM. The paper compares the performance of these algorithms on three genetic programming problems (11-Boolean Multiplexer, Artificial Ant, and Symbolic Regression), and discovers that the algorithms do not have a significant impact on fitness. Additional experimentation shows that tree size does have an important impact on fitness, and further that the ideal initial tree size is very different from that used in traditional GP.

1 INTRODUCTION

The issue of population initialization has received surprisingly little attention in the genetic programming literature. [Koza, 1992] established the GROW, FULL, and RAMPED HALF-AND-HALF algorithms, only a few papers have appeared on the subject, and the community by and large still uses the original Koza algorithms.

Some early work was concerned with algorithms similar to GROW but which operated on derivation grammars. [Whigham, 1995a,b, 1996] analyzed biases due to population initialization, among other factors, in grammatically-based genetic programming. [Geyer-Schulz, 1995] also devised similar techniques for dealing with tree grammars.

The first approximately uniform tree generation algorithm was RAND-TREE [Iba, 1996], which used Dyck words to choose uniformly from all possible tree structures of a given arity set and tree size. Afterwards the tree structure would be populated with nodes. [Bohm and Geyer-Schulz, 1996] then presented an exact uniform algorithm for choosing among all possible trees of a given function set.

Recent tree generation algorithms have focused on speed. [Chellapilla, 1997] devised RANDOMBRANCH, a simple algorithm which generated trees approximating a requested tree size. After demonstrating problems with the GROW algorithm, [Luke, 2000b] modifed GROW to produce PTC1 which guaranteed that generated trees would appear around an expected tree size. [Luke, 2000b] also presented PTC2 which randomly expanded the tree horizon to produce trees of approximately the requested size. All three of these algorithms are linear in tree size.

Both [Iba, 1996] and [Bohm and Geyer-Schulz, 1996] argued for the superiority of their algorithms over the Koza standard algorithms. [Whigham, 1995b] showed that biasing a grammar-based tree-generation algorithm could dramatically improve (or hurt) the success rate of genetic programming at solving a given domain, though such bias must be hand-tuned for the domain in question.

In contrast, this paper examines several algorithms to see if any of the existing algorithms appears to make much of a difference, or if tree size and other factors might be more significant.

2 THE ALGORITHMS

This paper compares five tree generation algorithms from the literature. These algorithms were chosen for their widely differing approaches to tree creation. The chief algorithm not in this comparison is RAND-TREE [Iba, 1996]. This algorithm has been to some degree subsumed by a more recent algorithm [Bohm and Geyer-Schulz, 1996], which generates trees from a truly uniform distribution (the original unachieved goal of RAND-TREE).

The algorithms discussed in this paper are:

2.1 Ramped Half-And-Half and Related Algorithms

RAMPED HALF-AND-HALF is the traditional tree generation algorithm for genetic programming, popularized by

[Koza, 1992]. RAMPED HALF-AND-HALF takes a tree depth range (commonly 2 to 6 – for this and future references, we define "depth" in terms of number of nodes, not number of edges). In other respects, the user has no control over the size or shape of the trees generated.

RAMPED HALF-AND-HALF first picks a random value within the depth range. Then with 1/2 probability it uses the GROW algorithm to generate the tree, passing it the chosen depth; otherwise it uses the FULL algorithm with the chosen depth.

GROW is very simple:

GROW(depth d, max depth)
Returns: a tree of depth $\leq - d$
If $d =$, return a random terminal
Else
Choose a random function or terminal f
If f is a terminal, return f
Else
For each argument a of f,
Fill a with GROW($d + 1$,)
Return f with filled arguments

GROW is started by passing in 0 for d, and the requested depth for . FULL differs from GROW only in the line marked with a . On this line, FULL chooses a nonterminal function only, never a terminal. Thus FULL only creates full trees, and always of the requested depth.

Unlike other algorithms, because it does not have a size parameter, RAMPED HALF-AND-HALF does not have well-defined computational complexity in terms of size. FULL always generates trees up to the depth bound provided. As [Luke, 2000b] has shown, GROW without a depth bound may, depending on the function set, have an expected tree size of infinity.

2.2 PTC1

PTC1 [Luke, 2000b] is a modification of the GROW algorithm which is guaranteed to produce trees around a finite expected tree size. A simple version of PTC1 is described here. PTC1 takes a requested expected tree size and a maximum legal depth. PTC1 begins by computing p, the probability of choosing a nonterminal over a terminal in order to maintain the expected tree size as:

$$p = \frac{1 - \frac{1}{}}{\sum_{n} \frac{1}{}_{n}}$$

where is the set of all nonterminals and $_n$ is the arity of nonterminal n. This computation can be done once offline. Then the algorithm proceeds to create the tree:

PTC1(precomputed probability p, depth d, max depth)
Returns: a tree of depth $\leq - d$
If $d =$, return a random terminal
Else if a coin-toss of probability p is true,
Choose a random nonterminal n
For each argument a of n,
Fill a with PTC1(p,$d + 1$,)
Return n with filled arguments
Else return a random terminal

PTC1 is started by passing in p, 0 for d, and the maximum depth for . PTC1's computational complexity is linear or nearly linear in expected tree size.

2.3 PTC2

PTC2 [Luke, 2000b] receives a requested tree size, and guarantees that it will return a tree no larger than that tree size, and no smaller than the size minus the maximum arity of any function in the function set. This algorithm works by increasing the tree horizon at randomly chosen points until it is sufficiently large. PTC2 in pseudocode is big, but a simple version of the algorithm can be easily described.

PTC2 takes a requested tree size S. If $S = 1$, it returns a random terminal. Otherwise it picks a random nonterminal as the root of the tree and decreases S by 1. PTC2 then puts each unfilled child slot of the nonterminal into a set , representing the "horizon" of unfilled slots. It then enters the following loop:

1. If $S \leq$, break from the loop.
2. Else remove a random slot from . Fill the slot with a randomly chosen nonterminal. Decrease S by 1. Add to every unfilled child slot of that nonterminal. Goto #1.

At this point, the total number of nonterminals in the tree, plus the number of slots in , equals or barely exceeds the user-requested tree size. PTC2 finishes up by removing slots from one by one and filling them with randomly chosen terminals, until is exhausted. PTC2 then returns the tree.

PTC2's computational complexity is linear or nearly linear in the requested tree size.

2.4 RandomBranch

RANDOMBRANCH [Chellapilla, 1997] is another interesting tree-generation algorithm, which takes a requested tree size and guarantees a tree of that size or "somewhat smaller".

Problem Domain	Algorithm	Parameter	Avg. Tree Size
11-Boolean Multiplexer	RAMPED HALF-AND-HALF	(No Parameter)	21.2
11-Boolean Multiplexer	RANDOMBRANCH	Max Size: 45	20.0
11-Boolean Multiplexer	PTC1	Expected Size: 9	20.9
11-Boolean Multiplexer	PTC2	Max Size: 40	21.4
11-Boolean Multiplexer	UNIFORM-even	Max Size: 42	21.8
11-Boolean Multiplexer	UNIFORM-true	Max Size: 21	20.9
Artificial Ant	RAMPED HALF-AND-HALF	(No Parameter)	36.9
Artificial Ant	RANDOMBRANCH	Max Size: 90	33.7
Artificial Ant	PTC1	Expected Size: 12	38.5
Artificial Ant	PTC2	Max Size: 67	35.3
Artificial Ant	UNIFORM-even	Max Size: 65	33.9
Artificial Ant	UNIFORM-true	Max Size: 37	36.8
Symbolic Regression	RAMPED HALF-AND-HALF	(No Parameter)	11.6
Symbolic Regression	RANDOMBRANCH	Max Size: 21	11.4
Symbolic Regression	PTC1	Expected Size: 4	10.9
Symbolic Regression	PTC2	Max Size: 18	11.1
Symbolic Regression	UNIFORM-even	Max Size: 19	11.2
Symbolic Regression	UNIFORM-true	Max Size: 11	10.8

Table 1: Tree Generation Parameters and Resultant Sizes

RANDOMBRANCH(requested size)
Returns: a tree of size $\leq$
If a nonterminal with arity $\leq$ does not exist
Return a random terminal
Else
Choose a random nonterminal n of arity $\leq$
Let $_n$ be the arity of n
For each argument a of n,
Fill a with RANDOMBRANCH($\frac{}{n}$)
Return n with filled arguments

Because RANDOMBRANCH evenly divides up among the subtrees of a parent nonterminal, there are many trees that RANDOMBRANCH simply cannot produce by its very nature. This makes RANDOMBRANCH the most restrictive of the algorithms described here. RANDOMBRANCH's computational complexity is linear or nearly linear in the requested tree size.

2.5 Uniform

UNIFORM is the name we give to the exact uniform tree generation algorithm given in [Bohm and Geyer-Schulz, 1996], who did not name it themselves. UNIFORM takes a single requested tree size, and guarantees that it will create a tree chosen *uniformly* from the full set of all possible trees of that size, given the function set. UNIFORM is too complex an algorithm to describe here except in general terms.

During tree-generation time, UNIFORM's computational complexity is nearly linear in tree size. However, UNIFORM must first compute various tables offline, including a table of numbers of trees for all sizes up to some maximum feasibly requested tree sizes. Fortunately this daunting task can be done reasonably quickly with the help of dynamic programming.

During tree-generation time, UNIFORM picks a node selected from a distribution derived from its tables. If the node is a nonterminal, UNIFORM then assigns tree sizes to each child of the nonterminal. These sizes are also picked from distributions derived from its tables. UNIFORM then calls itself recursively for each child.

UNIFORM is a very large but otherwise elegant algorithm; but it comes at the cost of offline table-generation. Even with the help of dynamic programming, UNIFORM's computational complexity is superlinear but polynomial.

3 FIRST EXPERIMENT

[Bohm and Geyer-Schulz, 1996] claimed that UNIFORM dramatically outperformed RAMPED HALF-AND-HALF, and argued that the reason for this was RAMPED HALF-AND-HALF's highly non-uniform sampling of the initial program space. Does uniform sampling actually make a significant difference in the final outcome? To test this, the first experiment compares the fitness of RAMPED HALF-AND-HALF, PTC1, PTC2, RANDOMBRANCH, and two different versions of UNIFORM (UNIFORM-true and

UNIFORM-even, described later). It is our opinion that the "uniformity" of sampling among the five algorithms presented is approximately in the following order (from most uniform to least): UNIFORM (of course), PTC2, RAMPED HALF-AND-HALF, PTC1, RANDOMBRANCH.

The comparisons were done over three canonical genetic programming problem domains, 11-Boolean Multiplexer, Artificial Ant, and Symbolic Regression. Except for the tree generation algorithm used, these domains followed the parameters defined in [Koza, 1992], using tournament selection of size 7. The goal of 11-Boolean Multiplexer is to evolve a boolean function on eleven inputs which performs multiplexing on eight of those inputs with regard to the other three. The goal of the Artificial Ant problem is to evolve a simple robot ant algorithm which follows a trail of pellets, eating as many pellets as possible before time runs out. Symbolic Regression tries to evolve a symbolic mathematical expression which best fits a training set of data points.

To perform this experiment, we did 50 independent runs for each domain using the RAMPED HALF-AND-HALF algorithm to generate initial trees. From there we measured the mean initial tree size and calibrated the other algorithms to generate trees of approximately that size. This calibration is not as simple as it would seem at first. For example, PTC1 can be simply set to the mean value, and it should produce trees around that mean. However, an additional complicating factor is involved: duplicate rejection. Usually genetic programming rejects duplicate copies of the same individual, in order to guarantee that every initial individual is unique. Since there are fewer small trees than large ones, the likelihood of a small tree being a duplicate is correspondingly much larger. As a result, these algorithms will tend to produce significantly larger trees than would appear at first glance if, as was the case in this experiment, duplicate rejection is part of the mix. Hence some trial and error was necessary to establish the parameters required to produce individuals of approximately the same mean size as RAMPED HALF-AND-HALF. Those parameters are shown in Table 1.

In the PTC1 algorithm, the parameter of consequence is the expected mean tree size. For the other algorithms, the parameter is the "maximum tree size". For PTC2, RANDOMBRANCH, and UNIFORM-even, a tree is created by first selecting an integer from the range 1 to the maximum tree size inclusive. This integer is selected uniformly from this range. In UNIFORM-true however, the integer is selected according to a probability distribution defined by the number of trees of each size in the range. Since there are far more trees of size 10 than of 1 for example, 10 is chosen much more often than 1. For each remaining algorithm, 50 independent runs were performed with both problem domains. ECJ [Luke, 2000a] was the genetic programming system used.

Fisher LSD	Algorithm	Tukey
	PTC2	
	PTC1	
	RAMPED HALF-AND-HALF	
	UNIFORM-true	
	UNIFORM-even	
	RANDOMBRANCH	

Table 2: ANOVA Results for Symbolic Regression. Algorithms are in decreasing order by average over 50 runs of best fitness per run. Vertical lines indicate classes with statistically insignificant differences.

Figures 1 through 6 show the results for the various algorithms applied to 11-Boolean Multiplexer. Figures 8 through 13 show the results for the algorithms applied to Artificial Ant. As can be seen, the algorithms produce surprisingly similar results. ANOVAs at 0.05 performed on the algorithms for both the 11-Boolean Multiplexer problem and the Artificial Ant problem indicate that there is no statistically significant difference among any of them. For Symbolic Regression, an ANOVA indicated statistically significant differences. The post-hoc Fisher LSD and Tukey tests, shown in Figure 2, reveal that UNIFORM fares worse than all algorithms except RANDOMBRANCH!

4 SECOND EXPERIMENT

If uniformity provides no statistically significant advantage, what then accounts for the authors' claims of improvements in fitness? One critical issue might be average tree size. If reports in the literature were not careful to normalize for size differences (very easy given that RAMPED HALF-AND-HALF has no size parameters, and duplicate rejection causes unforseen effects) it is entirely possible that significant differences can arise.

The goal of the second experiment was to determine how much size matters. Using UNIFORM-even, we performed 30 independent runs each for the following maximum-size values: 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30, 40, 50, 60, 80, 100. The test problem domains were again 11-Boolean Multiplexer, Artificial Ant, and Symbolic Regression with two features modified. First, the population size was reduced from 500 (the standard in [Koza, 1992]) to 200, to speed up runtime. Second, the runs were only done for eight generations, rather than 50 (standard for [Koza, 1992]). The reasoning behind this is that after eight generations or so the evolutionary system has generally settled down after initial "bootstrapping" effects due to the tree-generation algorithm chosen.

Figures 7, 14, and 21 show the results of this experiment. The light gray dots represent each run. The dark gray dots represent the means of the 30 runs for each maximum-size value. Because of duplicate rejection, runs typically have mean initial tree sizes somewhat different from the values predicted by the provided maximum-size. Also note the apparent horizontal lines in the 11-Boolean Multiplexer data: this problem domain has the feature that certain discrete fitness values (multiples of 32) are much more common than others.

These graphs suggest that the optimal initial tree size for `UNIFORM-even` for both domains is somewhere around 10. Compare this to the standard tree sizes which occur due to `RAMPED HALF-AND-HALF`: 21.2 for 11-Boolean Multiplexer and 36.9 for Artificial Ant!

5 CONCLUSION

The tree generation algorithms presented provide a variety of advantages for GP researchers. But the evidence in this paper suggests that improved fitness results is probably not one of those advantages. Why then pick an algorithm over `RAMPED HALF-AND-HALF` then? There are several reasons. First, most new algorithms permit the user to *specify* the size desired. For certain applications, this may be a crucial feature, not the least because it allows the user to create a size distribution more likely to generate good initial individuals. Fighting bloat in subtree mutation also makes size-specification a desirable trait.

Second, some algorithms have special features which may be useful in different circumstances. For example, `PTC1` and `PTC2` have additional probabilistic features not described in the simplified forms in this paper. Both algorithms permit users to hand-tune exactly the likelihood of appearance of a given function in the population, for example.

The results in this paper were surprising. Uniformity appears to have little consequence in improving fitness. Certainly this area deserves more attention to see what additional features, besides mean tree size, *do* give evolution that extra push during the initialization phase. Lastly, while this paper discussed effects on *fitness*, it did not delve into the effects of these algorithms on *tree growth*, another critical element in the GP puzzle, and a worthwhile study in its own right.

Acknowledgements

The authors wish to thank Ken DeJong, Paul Wiegand, and Jeff Bassett for their considerable help and insight.

References

Walter Bohm and Andreas Geyer-Schulz. Exact uniform initialization for genetic programming. In Richard K. Belew and Michael Vose, editors, *Foundations of Genetic Algorithms IV*, pages 379–407, University of San Diego, CA, USA, 3–5 August 1996. Morgan Kaufmann.

Kumar Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216, September 1997.

Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness*. Physica-Verlag, Heidelberg, 1995.

Hitoshi Iba. Random tree generation for genetic programming. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 144–153, Berlin, Germany, 22-26 September 1996. Springer Verlag.

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

Sean Luke. ECJ: A Java-based evolutionary computation and genetic programming system. Available at http://www.cs.umd.edu/projects/plus/ecj/, 2000a.

Sean Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions in Evolutionary Computation*, 4(3), 2000b.

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

P. A. Whigham. Inductive bias and genetic programming. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 461–466, Sheffield, UK, 12-14 September 1995b. IEE.

P. A. Whigham. Search bias, language bias, and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

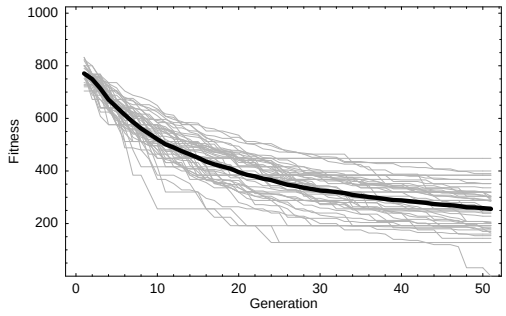


Figure 1: Generation vs. Fitness, `RAMPED HALF-AND-HALF`, 11-Boolean Multiplexer Domain

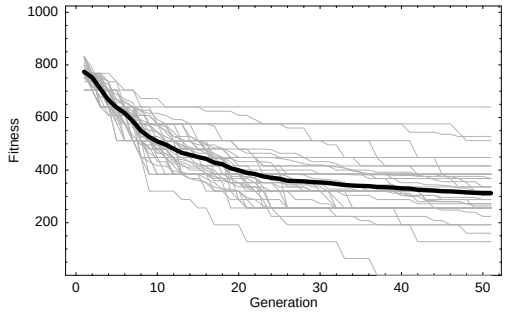


Figure 2: Generation vs. Fitness, `PTC1`, 11-Boolean Multiplexer Domain

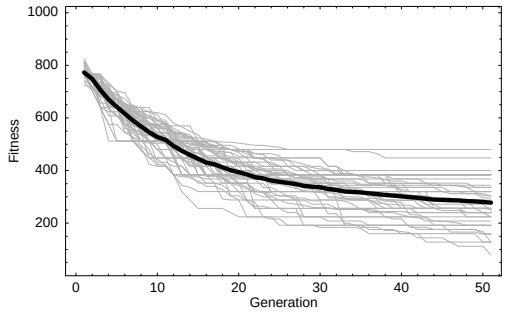


Figure 3: Generation vs. Fitness, `PTC2`, 11-Boolean Multiplexer Domain

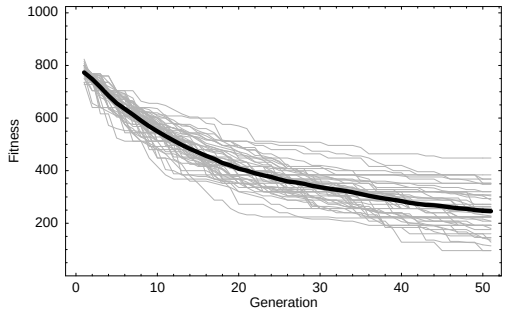


Figure 4: Generation vs. Fitness, `RANDOMBRANCH`, 11-Boolean Multiplexer Domain

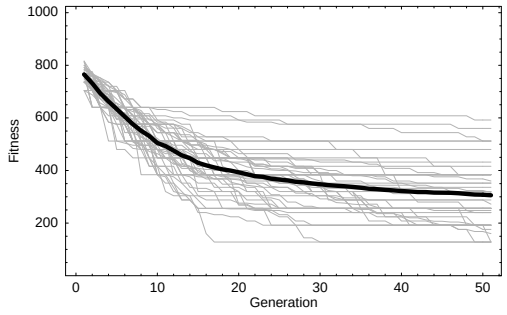


Figure 5: Generation vs. Fitness, `UNIFORM-even`, 11-Boolean Multiplexer Domain

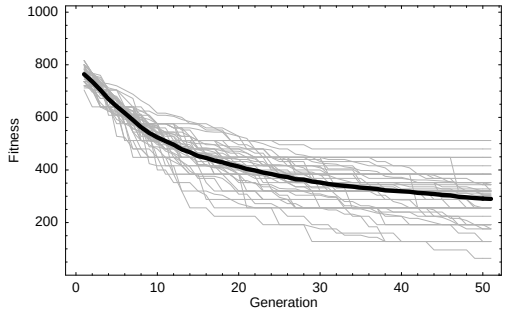


Figure 6: Generation vs. Fitness, `UNIFORM-true`, 11-Boolean Multiplexer Domain

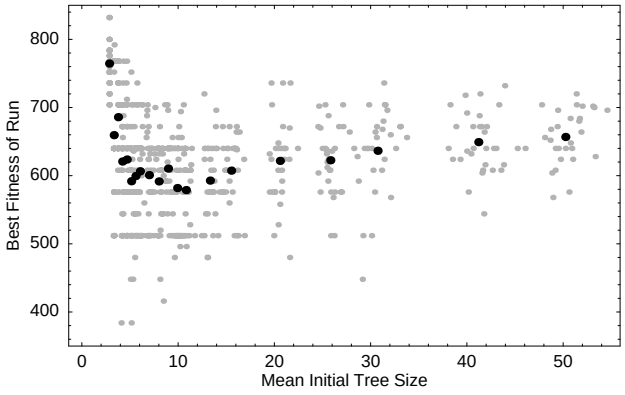


Figure 7: Mean Initial Tree Size vs. Fitness at Generation 8, 11-Boolean Multiplexer Domain

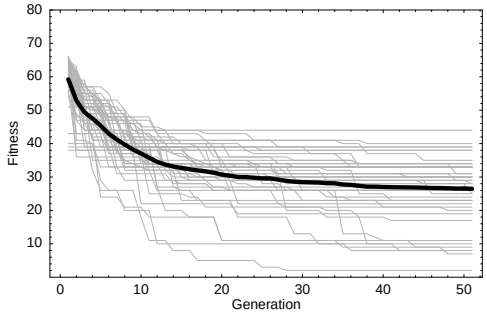


Figure 8: Generation vs. Fitness, `RAMPED HALF-AND-HALF`, Artificial Ant Domain

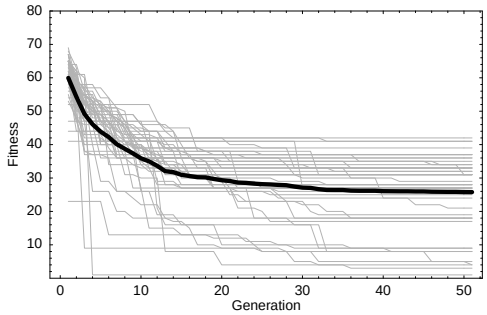


Figure 9: Generation vs. Fitness, `PTC1`, Artificial Ant Domain

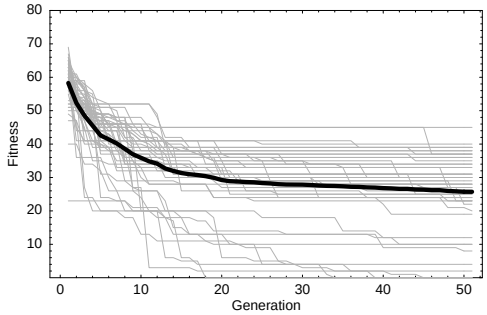


Figure 10: Generation vs. Fitness, `PTC2`, Artificial Ant Domain

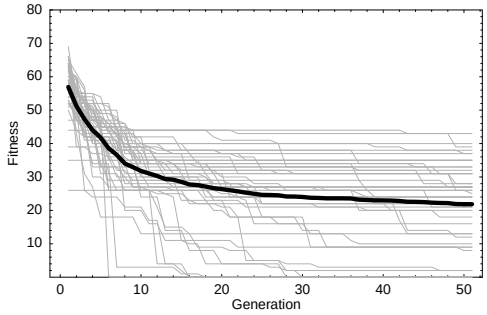


Figure 11: Generation vs. Fitness, `RANDOMBRANCH`, Artificial Ant Domain

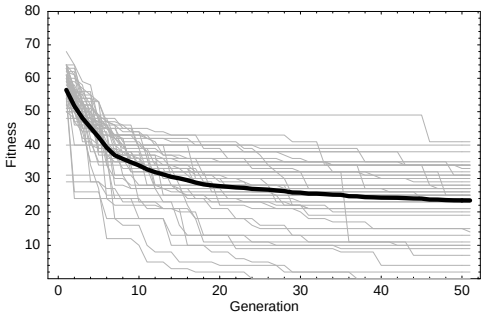


Figure 12: Generation vs. Fitness, `UNIFORM-even`, Artificial Ant Domain

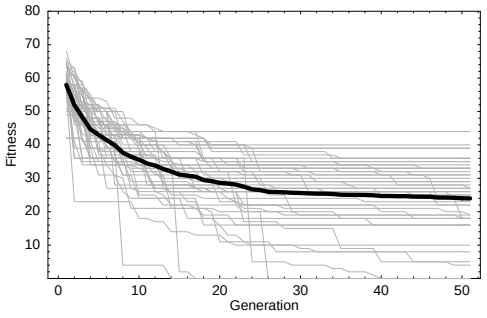


Figure 13: Generation vs. Fitness, `UNIFORM-true`, Artificial Ant Domain

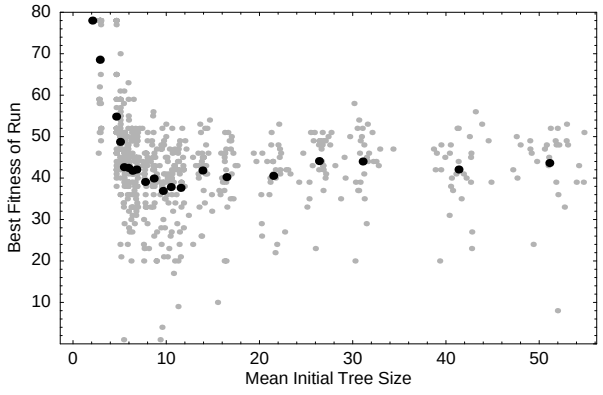


Figure 14: Mean Initial Tree Size vs. Fitness at Generation 8, Artificial Ant Domain

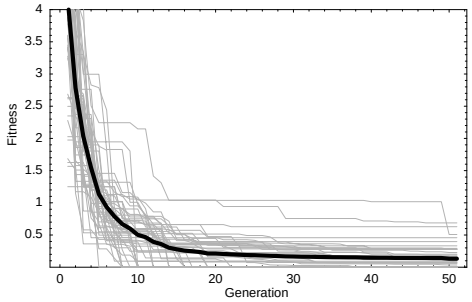


Figure 15: Generation vs. Fitness, `RAMPED HALF-AND-HALF`, Symbolic Regression Domain

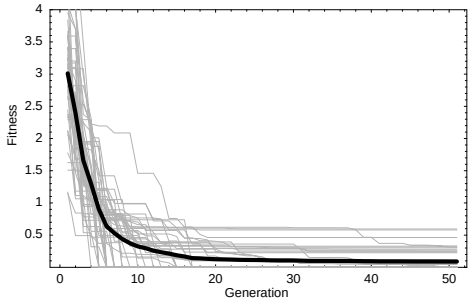


Figure 16: Generation vs. Fitness, `PTC1`, Symbolic Regression Domain

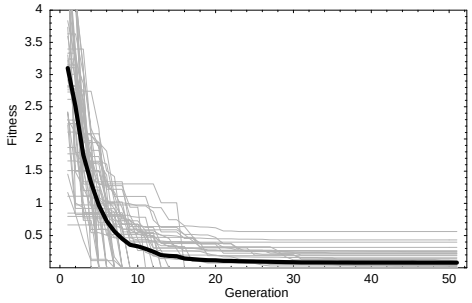


Figure 17: Generation vs. Fitness, `PTC2`, Symbolic Regression Domain

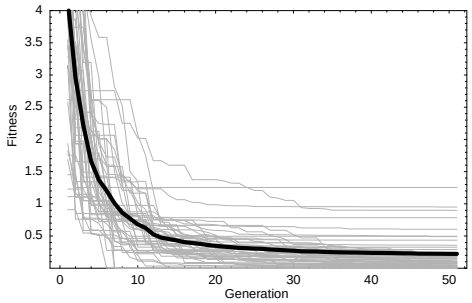


Figure 18: Generation vs. Fitness, `RANDOMBRANCH`, Symbolic Regression Domain

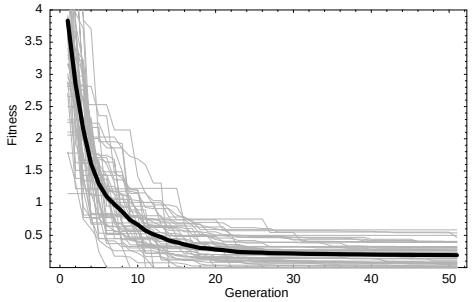


Figure 19: Generation vs. Fitness, `UNIFORM-even`, Symbolic Regression Domain

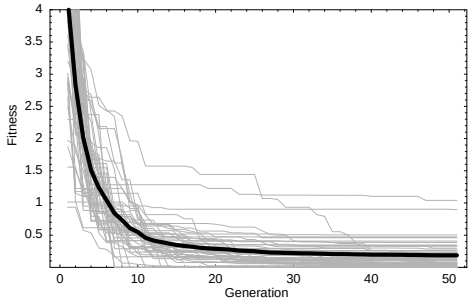


Figure 20: Generation vs. Fitness, `UNIFORM-true`, Symbolic Regression Domain

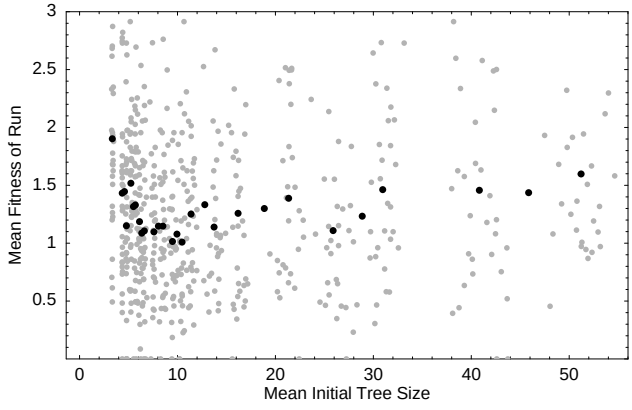


Figure 21: Mean Initial Tree Size vs. Fitness at Generation 8, Symbolic Regression Domain

Genetic Programming using Chebishev Polynomials

Nikolay Nikolaev
Dept. of Math. and Computing Sciences
Goldsmiths College, University of London
London SE14 6NW
UnitedKingdom
nikolaev@mcs.gold.ac.uk

Hitoshi Iba
Dept. of Inf. and Comm. Engineering
School of Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo
113-8656 Japan
iba@miv.t.u-tokyo.ac.jp

Abstract

This paper proposes a tree-structured representation for genetic programming (GP) using Chebishev polynomials as building blocks. They are incorporated in the leaves of tree-structured polynomial models. These trees are used in a version of the GP system STROGANOFF to avoid overfitting with the data when searching for polynomials. Search control is organized with a statistical fitness function that favours accurate, predictive, and parsimonious polynomials. The improvement of the evolutionary search performance is studied by principal component analysis of the error variations of the elite individuals in the population. Empirical results show that the novel version outperforms STROGANOFF, and the traditional Koza-style GP on processing benchmark and real-world time series.

1 INTRODUCTION

Polynomials are often preferred for function modeling due to their reliable approximation properties. Successful results with evolutionary computation systems that search for polynomials have been reported. They consider polynomials made as fixed-length structures (Kargupta and Smith, 1991), (Nissen and Koivisto, 1996), (Gomez-Ramirez *et al.*, 1999), (Sheta and Abel-Wahab, 1999), and variable-length tree-like structures (Iba *et al.*, 1996), (Rodriguez-Vazquez *et al.*, 1997), or sigma-pi neural networks (Zhang *et al.*, 1997). Important design issues for such systems are: 1) elaboration of search control mechanisms that may help to achieve convergence to optimal models; and, 2) elaboration of flexible functional model representations that may enable finding of predictive solutions.

These issues are addressed here with enhancement of the representation of the GP system STROGANOFF (Iba *et al.*, 1996), (Nikolaev and Iba, 2001) that learns polynomials. STROGANOFF manipulates tree-like models of basis polynomials in their leaves. It uses the GP paradigm to learn the model structure from the data, that is to discover which basis polynomials are components of the unknown function. One problem of these tree-like polynomials is that they tend to overfit the data as their parent GMDH networks (Ivakhnenko, 1971). Overfitting occurs mainly because the models contain very high order terms that exhibit low residual errors. One approach to combat evolving models with very low fitting errors is to use statistical fitness functions that estimate not only the residual error, but also the coefficients amplitudes and the model complexity.

Another improvement of GP for overfitting avoidance is proposed here using Chebishev polynomials as building blocks for tree-structured polynomials. The development of a *Chebishev polynomial* GP (*cp*GP) system has four objectives: 1) to encapsulate structural information in the polynomials so that they become more sparse, compared to the same polynomials without building blocks, for increasing of the generalization; 2) to decrease the search space size due to the decrease of the tree size; 3) to describe better oscillating properties of the data and to make the polynomials especially suitable for time-series modeling; and, 4) to accelerate the search convergence to good solutions. Since the basic idea is to capture common information in the data, this idea is similar to the automatically defined functions (ADF) of (Koza, 1994), the modules (MA) of (Angeline, 1994), and the adaptive representations (AR) of (Rosca and Balard, 1995).

The evolutionary search performance is studied by principal component analysis (PCA) of the error variance of the elite polynomials in the population. More precisely, applying PCA allows to observe the error trajectory during the generations by plotting it in

three dimensions. Using such error trajectory plots we demonstrate that the Chebishev building blocks contribute to improve the search and to discover polynomials with better generalization, compared to STROGANOFF using the same fitness function. In this sense, the fitness function alone is not sufficient to guarantee finding good polynomials that avoid overfitting the data. This claim is confirmed after experiments on time series prediction using two benchmark and one financial exchange rates series. The results indicate that *cp*GP outperforms STROGANOFF and the traditional GP (Koza, 1992) on these tasks.

This paper outlines the tree-structured representation using Chebishev polynomials for function approximation in section two. Section three offers the regularized fitness function and the *cp*GP mechanisms. The performance studies using PCA are in section four. Section five provides experimental results. Finally a discussion is made and conclusions are derived.

2 POLYNOMIAL APPROXIMATION

The function approximation problem is: given a series $D = \{(x_i, y_i)\}_{i=1}^{N}$ of points $x_i \in \mathcal{R}$, and corresponding values $y_i \in \mathcal{R}$, find the best function $y = f(x)$, $f \in L_2$. Our preferred functions are the high-order multivariate polynomials, called *Kolmogorov-Gabor polynomials*:

$$P(\mathbf{x}) = a_0 + \sum_{i=1}^{M} a_i \prod_{j=1}^{s} \varphi_j(\mathbf{x})^{r_j} \qquad (1)$$

where a_i are term coefficients, i iterates over the terms M: $i \leq M$, $\mathbf{x}$ is the independent variable vector of dimension s, $\varphi_j(\mathbf{x})$ are simple functions of first, second, third, etc. order (degree), and $r_j = 0, 1, ...$ are the powers of the j-th function $\varphi_j(\mathbf{x})$ in the i-th term.

The Kolmogorov-Gabor polynomials are universal modelling functions with which any continuous mapping may be approximated up to an arbitrary precision, if there are sufficiently large number of terms.

2.1 TREE-STRUCTURED POLYNOMIALS

The GP system STROGANOFF (Iba *et al.*, 1996) pioneered the employment of binary tree structures for representing polynomials. The terminal leaves in the tree provide the independent variables. In each internal functional tree node there are allocated basis polynomials whose outputs are fed in the basis polynomials at next layer higher in the tree as variables. Thus, high-order models are composed hierarchically leading to power series (1) at the tree root.

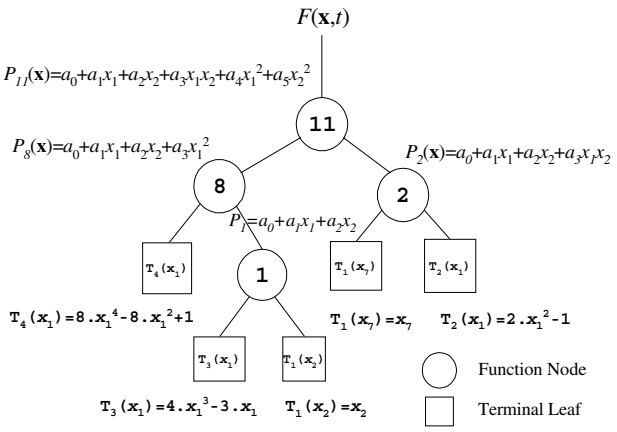


Figure 1. A tree-structured polynomial used in *cp*GP

This tree-like polynomial construction, however, adds very high-order terms to the model since the hierarchy rapidly increases the model order. The terms of very high-order are not necessarily well structurally related to the information in the data.

One remedy for such difficulties are the ready to use model components that capture common information in the data known as building blocks. The assumption is that the unknown function is resolvable in building block components, and we may learn such components by evolutionary search. A reasonable choice of such components for approximation tasks are the Chebishev polynomials which give minimax fit of the data.

2.2 CHEBISHEV TERMINALS

Chebishev polynomials may be considered as building blocks for genetic programming with GMDH-like polynomials. The idea is to take Chebishev polynomials in order to capture the essential partial information in the data. Thus, ready partial building blocks of the unknown true function may be identified and propagated during the search process.

We propose to pass Chebishev polynomials as terminals to enter the tree-structured models (Figure 1):

$$\varphi_j(\mathbf{x}) \equiv T_k(x) \qquad (2)$$

where $\mathbf{x} = (x_1, x_2, ..., x_s)$ is the input variable vector, and $T_k(x)$ are Chebishev polynomials applied with some $x \in \mathbf{x}$. An important requirement for practical application of the Chebishev polynomials $T_k(x)$ is to transform in advance the values of the input vectors: $-1 \leq x_i \leq 1$, for each x_i, $1 \leq i \leq s$, that is to scale all the input values in the interval $[-1, 1]$.

The Chebishev polynomials are derived with the recurrent formula [Lanczos, 1957]:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad (3)$$

where k is the polynomial order, and the starting polynomial is: $T_1(x) = x$.

Using Chebishev polynomials implies that *cp*GP will feature the following characteristics: 1) the polynomials become more sparse due to the use of building blocks, compared to the same models without them. The sparseness implies that the polynomials may be expected to overfit less the data; 2) the tree-structures become smaller and, thus, the search space size decreases. The effect of this is a possible acceleration of the convergence to good solutions; 3) oscillating terms are injected into the model which helps to describe better the frequency relationships between the data. Empirical evidence for achieving these characteristics is provided in subsections 5.1, 5.2 and 5.3 below.

3 MECHANISMS OF *cp*GP

The developed *cp*GP system uses *fitness proportional selection* with stochastic universal sampling, and performs *steady-state reproduction* of the population. A statistical fitness function is offered, and two genetic learning operators: crossover and mutation.

3.1 STATISTICAL FITNESS FUNCTION

The fitness function should control the evolutionary search so as to identify polynomials that are *accurate*, *predictive*, and of *short size*. We design a *statistical fitness function* with three ingredients that together counteract the overfitting with the data: 1) a mean-squared-error measurement that favors highly fit models; 2) a regularization factor that tolerates smoother mappings with higher generalization; and, 3) a complexity penalty that prefers short size polynomials.

3.1.1 Regularized Average Error

The fitting of the data is evaluated with a *regularized average error* (*RAE*) (Nikolaev and Iba, 2001):

$$RAE = \frac{1}{N}\left(\sum_{t=1}^{N}(y_t - P(\mathbf{x}_t))^2 + k\sum_{j=1}^{A} a_j^2\right) \quad (4)$$

where k is a regularization parameter, A is the number of all coefficients a_j in the whole model $P(\mathbf{x})$ (1), and N is the number of the data. The first term shows the improvement in mean square error sense. The second term is a regularizer that tolerates models with coefficients having small magnitudes.

Transfer Polynomials
$P_1(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2$
$P_2(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2$
$P_3(\mathbf{x}) = a_0 + a_1x_1 + a_2x_1x_2$
$P_4(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_2^2$
$P_5(\mathbf{x}) = a_0 + a_1x_1^2 + a_2x_2^2$
$P_6(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2$
$P_7(\mathbf{x}) = a_0 + a_1x_1 + a_2x_1x_2 + a_3x_1^2$
$P_8(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_1^2$
$P_9(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2^2$
$P_{10}(\mathbf{x}) = a_0 + a_1x_1x_2$
$P_{11}(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2$

Table 1. The set of transfer polynomials

3.1.2 Coefficients Estimation

The *cp*GP system uses a small set $\{p_i\}_{i=1}^{11}$ of *complete* and *incomplete* bivariate polynomials (Table 1). Their terms are derived with the functions: $h_0(\mathbf{x}) = 1$, $h_1(\mathbf{x}) = x_1$, $h_2(\mathbf{x}) = x_2$, $h_3(\mathbf{x}) = x_1x_2$, $h_4(\mathbf{x}) = x_1^2$, and $h_5(\mathbf{x}) = x_2^2$. The coefficients a_i are estimated by *regularized ordinary least squares* (ROLS) fitting:

$$\mathbf{a} = (\mathbf{H}^T\mathbf{H} + k\mathbf{I})^{-1}\mathbf{H}^T\mathbf{y} \quad (5)$$

where $\mathbf{a}$ is $(s+1) \times 1$ vector of coefficients, $\mathbf{H}$ is $N \times (s+1)$ design matrix of row vectors $\mathbf{h}(\mathbf{x}_i) = (h_0(\mathbf{x}_i), h_1(\mathbf{x}_i), ..., h_s(\mathbf{x}_i))$, $i = 1..N$, $\mathbf{y}$ is the $N \times 1$ output vector, and k is a regularization parameter.

3.1.3 Complexity Penalty

A *statistical fitness function* that measures the final prediction error (FPE) is synthesized to favor short size polynomials (Akaike, 1969):

$$FPE = \frac{(N+A)}{(N-A)}RAE \quad (6)$$

where RAE is the regularized error (4), A are coefficients, and N are the examples.

3.2 GENETIC OPERATORS

The *crossover* operator chooses randomly a cut point node in each tree, and swaps the subtrees rooted in the cut-point nodes. The *mutation* operator selects randomly a tree node, and performs one of the following tree transformations: 1) insertion of a randomly chosen node before the selected one, so that the selected becomes an immediate child of the new one, and the other child is a random terminal; 2) deletion of the selected node, and replacing it by one of its children nodes; and 3) replacement of the selected node by another randomly chosen node.

4 PERFORMANCE STUDIES BY PCA

We carry out a principal component analysis (Jolliffe, 1986) to examine the error variations of the elite polynomials in the population. This allows to plot the *error trajectory* which provides an illustration of the search problems encountered during evolutionary learning.

The PCA application may be explained as follows. The mean square errors e_g of the elite models are recorded at each generation g, and error vectors are formed: $\mathbf{e}_g = (e_g^1, e_g^2, ..., e_g^n)$, where e_g^n is the error of the n-th model and n is the size of the population elite. Usually elite are the best 25%. The PCA is taken to project the error changes in three dimensions, that is to enable plotting of the error changes in three dimensions in order to investigate the evolutionary search difficulties. Because these errors reflect the degree of accurate learning of the model coefficients.

Let each elite error $\mathbf{e}$ be a point in the n-dimensional error space. Therefore, we may write: $\mathbf{e} = \sum_{i=1}^{n} e_i \mathbf{u}_i$, where $\mathbf{u}_i$ are unit orthonormal basis vectors such that: $\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$, and δ_{ij} is the Kroneker delta. The individual model errors are: $e_i = \mathbf{u}_i^T \mathbf{e}$. The PCA helps to change the coordinate system and to project these points on the dimensions in which they exhibit largest variance. The basis vectors $\mathbf{u}_i$ are changed with new basis vectors $\mathbf{v}_i$ so that in the new coordinate system: $\mathbf{e} = \sum_{i=1}^{n} z_i \mathbf{v}_i$. This can be made by extracting $\mathbf{v}_i$ as eigenvectors of the covariance matrix $\mathbf{\Sigma}$ of the error trajectory recorded during a number of generations G:

$$\mathbf{\Sigma} \mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{7}$$

where λ_i is the i-th eigenvalue of the covariance matrix $\mathbf{\Sigma}$ defined as follows:

$$\mathbf{\Sigma} = \sum_{g=1}^{G} (\mathbf{e}_g - \bar{\mathbf{e}})^T (\mathbf{e}_g - \bar{\mathbf{e}}) \tag{8}$$

and the mean error is $\bar{\mathbf{e}} = \frac{1}{G} \sum_{g=1}^{G} \mathbf{e}_g$.

The theoretical studies suggest that the first two principal components (PCs) capture the most essential variations in the errors. The extent to which the i-th principal component captures the error variance can be measured as follows: $E_{pc} = \lambda_i^2 / \sum_i \lambda_i^2$.

We relate the first and the second PCs of the errors $\mathbf{pc} = \sum_{i=1}^{2} z_i \mathbf{v}_i$, $\mathbf{pc} = (pc_1, pc_2)$, to the average mean square error (MSE) of the population elite in order to visualize the GP performance. These MSE trajectory plots against the first two principal components pc_1 and pc_2 may be considered pictures of the coefficients learning process during evolutionary search.

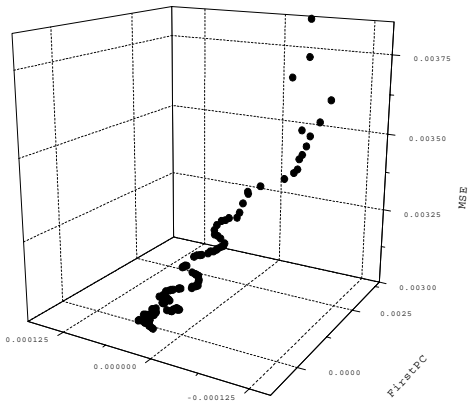


Figure 2. Error trajectory of the 40 elite polynomials (from a population of size 100) evolved with the GP system STROGANOFF applied to the *Sunspots data* with the FPE (6) statistical fitness function using $k = 0.0015$.

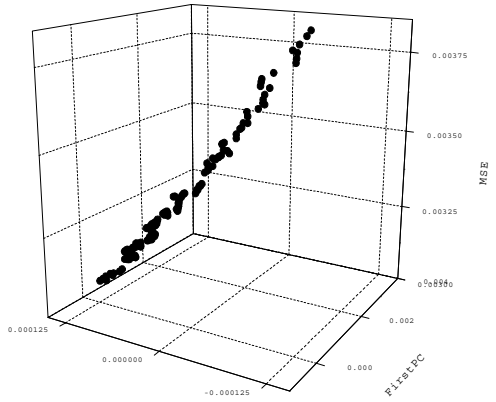


Figure 3. Error trajectory of the 40 elite polynomials (from a population of size 100) evolved with *cp*GP applied to the *Sunspots data* with the FPE (6) statistical fitness function using $k = 0.0015$.

Figures 2 and 3 depict the error trajectories computed after runs of STROGANOFF and *cp*GP on the Sunspot data series (Weigend et *al.*, 1992). These are the representative runs that achieved the best results (given in Table 2). One can see in Figure 2 that the error trajectory of STROGANOFF does not go down smoothly. The variation of the elite population error slopes down with a zig-zag movement which can be seen from the slightly changing error directions after $MSE = 0.0033$ ($pc_1 = 0.000271$, and $pc_2 = -0.0000462$), $MSE = 0.00325$ ($pc_1 = 0.000138$, and $pc_2 = -0.0000235$), and $MSE = 0.0032$ ($pc_1 = 0.00011$, and $pc_2 = -0.0000164$). This means that the population elite faces search difficulties and can not orient precisely on the search landscape toward the optimal solution. We are inclined to think that

the landscape of STROGANOFF is more rugged, and more difficult to search. That is why, the population evolved by STROGANOFF moves in curved directions on the search landscape and in some sense jumps from one basin to another basin of attraction without careful exploration of the landscape neighborhood.

The *cp*GP error trajectory in Figure 3 shows that the evolutionary search progresses directly, following almost a straight line direction of error decrease, toward its best result. In this sense, its population exploits meticulously the local search neighborhood and orients well on the search landscape. Since the two GP are controlled by the same FPE fitness function, it seems that the search improvement can be due mainly to the use of Chebishev polynomials as building blocks.

The plots in Figures 2 and 3 are meaningful because these PCs capture respectively: pc_1 99.315% and pc_2 0.685% of the variance of all elite errors, and therefore they make us certain about the search behaviour.

5 TIME SERIES MODELING

Three GP systems were implemented and tested on time series prediction problems: the *original* STROGANOFF (Iba *et al.*, 1996), (Nikolaev and Iba, 2001), the *cp*GP system, and a traditional Koza-style GP (Koza, 1992). All the systems use the FPE fitness function (6), and parameters: $PopulationSize = 100$, and $MaxNumberOfGenerations = 250$. The regularization parameter is determined in advance for each task by a statistical technique (Myers, 1990). The *cp*GP system uses five Chebishev polynomials: $T_1(x)$,$T_2(x_{t-1})$,$T_3(x_{t-1})$,$T_4(x_{t-1})$ and $T_5(x_{t-1})$. Thus, ten variables are passed as terminals: $\mathbf{x} = (x_{t-1}, x_{t-2}, ..., x_{t-5}, x_{t-6}, T_2, ..., T_5)$. The Koza-style GP is made using *sin* and *cos* in order to produce functions with similar representation power. The question that we rise is whether or not the *cp*GP system can outperform STROGANOFF and traditional GP?

5.1 PROCESSING THE SUNSPOTS DATA

The Sunspots series (Weigend et *al.*, 1992) contains 280 data points divided into one training and two testing subsets.

Table 2 demonstrates that using Chebishev polynomials helps to achieve improved results compared to the case without such building blocks. One can see in Table 2 that the models learned by STROGANOFF and the novel version *cp*GP exhibit higher accuracy on the training series as well as higher generalization on the testing series than traditional GP.

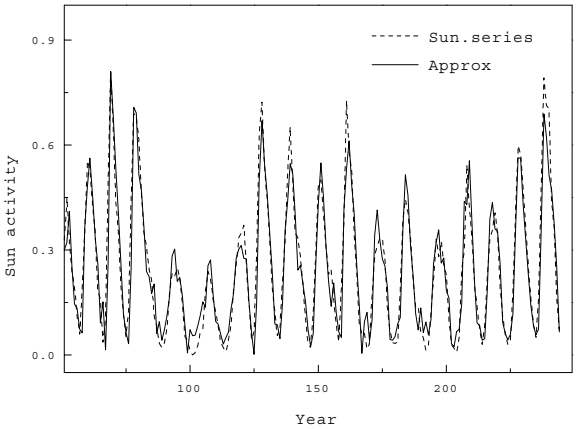


Figure 4. Approximated segment from the *Sunspots* curve by the best polynomial harmonic network evolved with *cp*GP in 50 runs using $k = 0.0015$.

Table 2. Results on the *Sunspots series* obtained in 50 runs with each GP using: $MaxTreeDepth = 4$ in STROGANOFF and *cp*GP, $MaxTreeDepth = 10$ in traditional Koza-style GP, and parameter $k = 0.0015$.

	Accuracy (ARV)	*Generalization* (ARV)	
	1700-1920	1700-1955	1700-1979
GP	0.128476	0.129685	0.132557
STROGA	0.114726	0.118257	0.129731
*cp*GP	0.103754	0.099159	0.104265

The *cp*GP system outperforms all the other systems showing a better accuracy $ARV_{1700-1920} = 0.103754$, better short forecasting: $ARV_{1700-1955} = 0.099159$ in the future period $1700 - 1955$, and better long term forecasting $ARV_{1700-1979} = 0.104265$ in $1700 - 1979$. The important observation in Table 2 is that the *cp*GP polynomial features a considerably improved generalization especially in the two future periods. Therefore, the use of oscillating building blocks really can increase the predictability of the acquired results. It should be noted that the $MaxTreeDepth$ parameter is used in order to constrain the maximal model degree for fair comparisons. The complexities of the best results found by the systems are: 28 coefficients in STROGANOFF, 25 coefficients in *cp*GP.

An approximated segment of the Sunspots series by the best learned network from *cp*GP is plotted in Figure 4. The acquired numerical results in Table 2 confirm the theoretical expectation that using oscillating building block components in the representation can help to model well spikes in the series as these in Figure 4. It is likely that when the time series contains spikes, a superior GP performance may be expected using the novel representation.

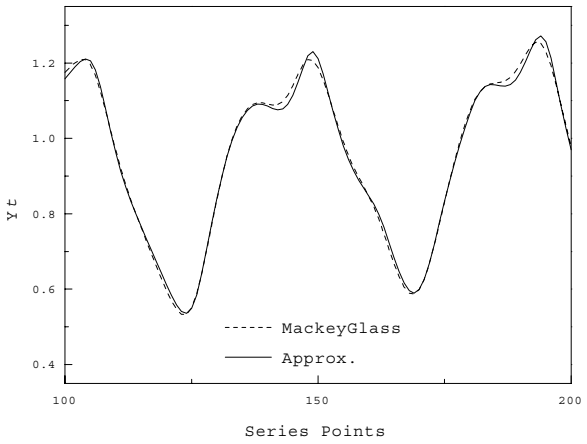


Figure 5. Approximated segment from the *Mackey-Glass* curve by the best polynomial harmonic network evolved with *cp*GP in 50 runs using $k = 0.0001$.

Table 3. Results on the *Mackey-Glass* series generated with: $a = 0.2, b = 0.1$, $\Delta = 17$, obtained in 50 runs using: $MaxTreeDepth = 3$ in STROGANOFF and *cp*GP, $MaxTreeDepth = 8$ in traditional GP, and $k = 0.0001$.

	Accuracy (*ARV*)	*Generalization* (*ARV*)	
	0-100	0-200	0-400
GP	0.005429	0.003691	0.002794
STROGA	0.004751	0.003503	0.002591
*cp*GP	0.003390	0.002952	0.002483

5.2 PROCESSING THE MACKEY-GLASS SERIES

A trajectory of 400 points from the benchmark Mackey-Glass series (Mackey and Glass, 1977) is derived. The first 100 points are used for training, and the remaining for testing. Again the first five Chebishev polynomials are considered: $\mathbf{x} = (x_{t-1}, x_{t-2}, ..., x_{t-5}, x_{t-6}, T_2, ..., T_5)$. The systems are tuned to evolve models of up to a predefined maximal degree to make fair comparisons. The complexities of the best results are: 25 coefficients in STROGANOFF, and 22 coefficients in *cp*GP. The *cp*GP system locates slightly more parsimonious models not only because the fitness function favours simpler models, since this fitness is also used by the other GP, but also because the Chebishev building blocks contribute directly nonlinearities to the representation.

Several observations can be made from the results in Table 3: 1) the STROGANOFF and *cp*GP systems outperform the traditional GP on this task; 2) the novel *cp*GP is best on accuracy $(0 - 100)$ with $ARV_{0-100} = 0.003390$, excellent on short term $(0 - 200)$ prediction with $ARV_{0-200} = 0.002952$, and also best on long term $(0 - 400)$ prediction with $ARV_{0-400} = 0.002483$. The slight differences in the results given in Table 3 are due to the fact that the smooth curvature of the Mackey-Glass series is approximated by models of relatively high degree.

5.3 PROCESSING FINANCIAL DATA

Experiments with GP are performed attempting to identify non-linear trends in currency exchange rates taken from the financial market. We report results derived with a real financial series of 14,000 data relating the changes between the dollar (USD) and the Japanese yen (JPY) obtained on demand by a financial company during a certain period of time.

The given financial data series is pre-processed by a differential technique in order to eliminate obscuring information in the data, and to emphasize the rates of directional changes in the series as follows (Iba and Nikolaev, 2000):

$$x_d = x_t - x_{t-1} \tag{9}$$

where x_t is the data point at time t. Thus, delay vectors are formed: $\mathbf{x} = (x_{d-1}, ..., x_{d-6}, T_2, ..., T_5)$ and passed for the GP systems to learn the regularities among them. The tree limit parameters of the studied GP systems are: $MaxTreeDepth = 25$ in STROGANOFF and *cp*GP, and $MaxTreeDepth = 50$ in traditional GP. An approximated segment by the best result from *cp*GP is plotted in Figure 6.

The characteristics of the best evolved results are measured with the mean square error (MSE) and with the hit percentage estimate (Table 4). The *hit percentage* (HIT) shows how accurately the trend directions have been tracked by the model [Iba and Nikolaev, 2000]:

$$HIT = \frac{N_{up\_up} + N_{down\_down}}{N} \tag{10}$$

where N_{up_up} means number of times when the model outcome and the given outcome exhibit both upward raising tendency, and N_{down_down} means number of times when the model outcome and the given outcome exhibit both falling tendency.

One can see in Table 4 that STROGANOFF is not better than traditional Koza-style GP in the sense of economic $HITs$ achievements. The good result from traditional GP can be explained with its high MSE which means that it does not overfit the data. It has been already studied that STROGANOFF tends to evolve overfitting polynomials which have always to be controlled by applying the regularization technique (Nikolaev and Iba, 2001).

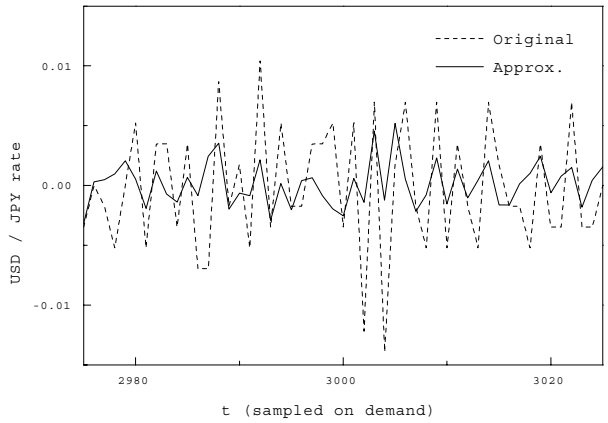


Figure 6. Approximated segment from the *financial exchange rates series* curve by the best polynomial harmonic network from *cp*GP in 50 runs using $k = 0.001$.

Table 4. Estimates of the best polynomials learned from the *financial series* in 50 runs with each GP using: $MaxTreeDepth$= 25 in STROGANOFF and *cp*GP, $MaxTreeDepth$= 50 in traditional GP, $k = 0.0001$.

	Accuracy (MSE)	*Prediction* ($HITs$)
	(training)	(testing)
GP	0.0011329	50.43%
STROGA	0.0007521	49.02%
*cp*GP	0.0000234	78.26%

The *cp*GP system shows lowest mean square error $MSE = 0.0000234$ on the training series, and demonstrates superior predictability $HITs = 78.26\%$ on this task. Despite exhibiting lowest error *cp*GP does not seem to overfit the training data. The derived best polynomial describes well the directional changes in the series up or down (Figure 6), which is a promising feature for the practical application of *cp*GP.

6 DISCUSSION

Oscillating Building Blocks. The employment of Chebishev polynomials for introducing ready nonlinear building blocks in function representations, used in GP systems breeding polynomials, showed successful results on several time series prediction tasks. The benefit from such building blocks is likely to be the discovery of polynomial models with improved generalization on future unseen data. Our findings concern explicitly the case when the search control of GP is made with fitness functions that contain both a size dependent component and a coefficients amplitude dependent component. If some of these two components are missing in the fitness function the effect from the novel representation may not be the same.

Other alternatives for including nonlinear oscillating components in the polynomial representation are also possible. For example, currently under investigation is a technique with harmonic components with non-multiple frequencies derived analytically using the discrete Fourier transform.

The Error Trajectory. The presented plots of the elite error trajectory suggest that although keeping of the binary tree structures, the employment of Chebishev polynomials as building blocks causes the *cp*GP to flow on different search landscapes than STROGANOFF. The oscillatory building blocks impact the landscape characteristics, i.e. make it more or less difficult to search, through the fitness function. The novel polynomials feature different fitnesses because the incorporated Chebishev terminals contribute different nonlinearities in the model, and, thus, the Chebishev terminals imply different errors of fit. The developed *cp*GP representation seems to make the fitness landscape easier to search despite the use of the same fitness function in both GPs. This can be seen from the trajectory plots in Figures 2 and 3.

A close methodology using PCA to examine the coefficients/weight changes has been proposed for neural network learning (Gallagher and Downs, 1997). The presented here PCA of the elite population error is more general as by explaining the error variance it explains the coefficients and term learning processes. This is because the polynomial error measurements actually reflect the accuracy of identification of the model coefficients and the identification of proper model terms. Moreover, in GP the coefficients can not be considered directly for PCA since the evolved polynomials have different number of coefficients.

It is not very clear yet whether *cp*GP is considerably better on periodic series, on aperiodic series or on both, for example on the Sunspots series *cp*GP shows close performance to this of the Koza-style GP but on the financial data series *cp*GP is considerably better.

7 CONCLUSION

This paper contributes to the research into increasing the expressive power of the tree-structured GP representations especially for function approximation tasks. Initial results from the development of a GP system using polynomials in the functional nodes and Chebishev polynomials passed as terminals have been reported. The Chebishev polynomials serve as oscillatory building blocks which capture well the nonlinear properties of the given training data, and there is a need to search for these building blocks that should enter the model as

their descriptive significance is not known in advance. It was shown that this tree-structured polynomial representation has enabled to discover superior results on several benchmark and real-world time-series prediction problems.

We suppose that the novel polynomial representation scheme could be of practical importance and it can be used successfully for addressing nonparametric approximation tasks because of the following advantages: 1) it generates explicit analytical models in the form of multivariate high-order polynomial functions amenable to human understanding; and 2) it makes the polynomials well-conditioned, thus computationally stable and suitable for practical purposes.

References

H. Akaike (1969). "Power Spectrum Estimation through Autoregression Model Fitting". *Annals Inst. Stat. Math.* **21**:407-419.

P.J. Angeline (1994). "Genetic Programming and Emerging Intelligence". In E.Kinnear Jr. (Ed.), *Advances in Genetic Programming*. Cambridge, MA: The MIT Press, pp.75-98.

M. Gallagher and T. Downs (1997). "Weight Space Learning Trajectory Vizualization". In M.Dale (Ed.), *Proc. Eighth Australian Conference on Neural Networks, ACNN-98,* pp.55-59.

E. Gomez-Ramirez, A.Poznyak, A.Gonzalez-Yunes and M. Avila-Alvarez (1999). "Adaptive Archictecture of Polynomial Artificial Neural Network to Forecast Nonlinear Time Series". In *Proc. of 1999 Congress on Evolutionary Computation, CEC-1999*. IEEE Press, vol.1, pp.317-324.

H. Iba, H. deGaris, and T. Sato (1996). "Numerical Approach to Genetic Programming for System Identification". *Evolutionary Computation* **3**(4).

H. Iba and N. Nikolaev (2000). "Genetic Programming Polynomial Models of Financial Data Series". In *Proc. of 2000 Congress on Evolutionary Computation, CEC-2000*. IEEE Press, pp.1459-1466.

A.G. Ivakhnenko (1971). "Polynomial Theory of Complex Systems", *IEEE Trans. on Systems, Man, and Cybernetics* **1**(4):364-378.

I.T. Jolliffe (1986). *Principal Component Analysis.* New York, NY: Springer-Verlag.

H. Kargupta, and R.E. Smith (1991). "System Identification with Evolving Polynomial Networks. In R.K.Belew and L.B.Booker (Eds.), *Proc. 4th Int. Conf. Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, pp.370-376.

J.R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

J.R. Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: The MIT Press.

C. Lanczos (1957). *Applied Analysis*. London, UK: Prentice-Hall.

R.H. Myers (1990). *Classical and Modern Regression with Applications*. Cambridge, CA: PWS-KENT Publ., Duxbury Press.

A.S. Nissen and H. Koivisto (1996). "Identification of Multivariate Volterra Series using Genetic Algorithm", In J.Alander (Ed.), *Proc. Second Nordic Workshop on Genetic Algorithms and their Applications*. Finland: University of Vaasa Press, pp.151-161.

M.C. Mackey and L. Glass (1977). "Oscillation and Chaos in Physiological Control Systems". *Science* **197**:287-289.

N. Nikolaev and H. Iba (2001). "Regularization Approach to Inductive Genetic Programming". *IEEE Trans. on Evolutionary Computation* (in press).

K. Rodriguez- Vazquez, C.M. Fonseca and P.J. Fleming (1997). "An Evolutionary Approach to Non-Linear Polynomial System Identification". In *Proc. 11th IFAC Symposium on System Identification,* pp.2395-2400.

J. Rosca and D.H. Ballard (1995). "Discovery of Subroutines in Genetic Programming", In P.Angeline and K.Kinnear Jr. (Eds.), *Advances in Genetic Programming II*. Cambridge, MA: The MIT Press, pp.177-202.

A.F. Sheta and A.H. Abel-Wahab (1999). In *Proc. of 1999 Congress on Evolutionary Computation, CEC-1999*. IEEE Press, vol.1, pp.229-235.

A.S. Weigend and N.A. Gershenfeld (Eds.) (1994). *Time Series Prediction*. Reading, MA: Addison-Wesley.

B.-T. Zhang, P. Ohm, and H. Mühlenbein (1997) "Evolutionary Induction of Sparse Neural Trees", *Evolutionary Computation* **5**(2):213-236.

Grammar Defined Introns: An Investigation Into Grammars, Introns, and Bias in Grammatical Evolution.

Michael O'Neill **Conor Ryan** **Miguel Nicolau**
Dept. Of Computer Science & Information Systems
University of Limerick
Ireland.
{Michael.ONeill|Conor.Ryan|Miguel.Nicolau}@ul.ie

Abstract

We describe an investigation into the design of different grammars on Grammatical Evolution. As part of this investigation we introduce introns using the grammar as a mechanism by which they may be incorporated into Grammatical Evolution. We establish that a bias exists towards certain production rules for each non-terminal in the grammar, and propose alternative mechanisms by which this bias may be altered either through the use of introns, or by changing the degeneracy of the genetic code. The benefits of introns for Grammatical Evolution are demonstrated experimentally.

1 Introduction

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve code in any language, using linear genomes [O'Neill & Ryan 2001] [Ryan C., Collins J.J. & O'Neill M. 1998]. We have previously presented results relating to an analysis of some of GE's distinctive features, such as its degenerate genetic code, wrapping operator and crossover [O'Neill & Ryan 1999b] [O'Neill & Ryan 1999a]. We now present the first results from an investigation into the role of the grammar in GE. Specifically, we introduce a mechanism by which introns can be incorporated into the genotypic representation through the grammar, and conduct an analysis on the effects of these grammar defined introns on the performance of GE. We also establish the existence of a bias towards the use of certain production rules for each non-terminal, dependent upon their ordering in the grammar, and propose a mechanism by which this bias can be altered as desired through the use of grammar defined introns.

We begin with a brief overview of GE, for a more complete description we refer the reader to [O'Neill & Ryan 2001]. Grammar defined introns are then introduced, followed by a description of the experimental approach adopted to test the effects of introns, before a discussion on bias and introns.

2 Grammatical Evolution

Unlike standard GP [Koza 1992], GE uses a variable length binary string to represent programs. Each individual contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to either elements of the set of terminals, or to elements of the set of non-terminals, according to the production rules. An excerpt from a BNF grammar is given below. These productions state that S can be replaced with any one of `expr`, `if-stmt`, or `loop`.

```
S ::= expr       (0)
    | if-stmt    (1)
    | loop       (2)
```

In order to select a rule in GE, the next codon value on the genome is generated and placed in the following formula:

$$Rule = \text{“}Codon\ Integer\ Value\text{”}$$

$$MOD$$

$$\text{“}Number\ of\ Rules\ for\ this\ non-terminal\text{”}$$

If the next codon integer value was 4, given that we have 3 rules to select from as in the above example, we get $4\ MOD\ 3\ =\ 1$. S will therefore be replaced with the non-terminal `if-stmt`.

Beginning from the left hand side of the genome, codon integer values are generated and used to select rules from the BNF grammar, until one of the following situations arise:

1. A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar.
2. The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue, unless an upper threshold representing the maximum number of wrapping events has occurred during this individual's mapping process.
3. In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual is assigned the lowest possible fitness value.

GE uses a steady state replacement mechanism, such that two parents produce two children, the best of which replaces the worst individual in the current population if the child has a greater fitness. The standard genetic operators of point mutation, and crossover (one point) are adopted. It also employs a duplication operator that duplicates a random number of codons and inserts these into the penultimate codon position on the genome. A full description of GE can be found in [O'Neill & Ryan 2001].

3 Grammar Defined Introns

The benefit, or otherwise, of introns in evolutionary computation have been hotly debated for some time [Levenick 1991] [Altenberg 1994] [Angeline 1994] [Nordin & Banzhaf 1995] [Nordin, Francone & Banzhaf 1995] [Wu & Lindsay 1995] [Andre & Teller 1996] [Wineberg & Oppacher 1996] [Haynes 1996] [Wu & Lindsay 1996] [Lobo et al. 1998] [Smith & Harries 1998] [Luke 2000]. In the standard implementation of GE, introns can only occur at the end of a chromosome due to the nature of the mapping process. The role of an intron in the preservation of building blocks due to destructive crossover events is therefore minimised in GE. We wish to investigate the effects introns might have on the performance of GE and, as such, have devised a mechanism by which they may be incorporated into the system. We call this mechanism Grammar Defined Introns, whereby the grammar is used to incorporate introns into the genome. This is achieved by allowing codons to be skipped over during the mapping process, by using `introns` as a choice(s) for non-terminals.

For example, the following non-terminal uses an intron as a rule:

```
<line> :: =  <if-statement>      (A)
           |<op>                 (B)
           |intron               (C)
```

When a codon evaluates to the `intron` rule being selected we simply skip over this codon, and the code undergoing the mapping is unchanged. In this case the non-terminal `<line>` would remain as `<line>` if the `intron` rule is selected, and the next codon is read.

4 Bias in Grammatical Evolution

When choosing a production rule to be applied to a non-terminal during the mapping process, there is a bias towards certain choices. The amount of bias depends on the number of choices that are to be made, and on the number of genetic codes that are used to represent each choice. Taking the example of the non-terminal `<op>`:

```
<op> :: =    left()              (A)
           | right()             (B)
           | move()              (C)
```

there are 3 possible mappings for `<op>` that can be made in this case. Given a 2-bit codon, there are 4 possible genetic codes representing these choices. This results in a strong bias towards the first choice with a probability of selection of 0.5 as opposed to 0.25 for both of the other rules, see Table 1.

Genetic Code	Choice
00	A
01	B
10	C
11	A

Choice	Probability
A	2/4
B	1/4
C	1/4

Table 1: Probabilities of selecting a production rule using 2-bit codons.

However, given an 3-bit codon the bias due to the probability of using any one rule is reduced, see Table 2.

Taking the case of an 8-bit codon as adopted in the standard GE implementation this bias is minimised even further, see Table 3.

In the case of there being two choices as in

```
(1) <code> :: =  <line>              (A)
               |<code><line>          (B)
```

there is no bias to either choice no matter how many codes exist.

One approach to alleviate the problem of bias was that used by [Paterson & Livesley], who duplicated certain rules. Unfortunately, that system was difficult to control, and not very

Genetic Code	Choice
000	A
001	B
010	C
011	A
100	B
101	C
110	A
111	B

Choice	Probability
A	3/8 (.375)
B	3/8 (.375)
C	2/8 (.25)

Table 2: Probabilities of selecting a production rule using 3-bit codons.

Choice	Probability
A	86/256 (.336)
B	85/256 (.332)
C	85/256 (.332)

Table 3: Probabilities of selecting a production rule using 8-bit codons.

successful at removing the bias. Another approach that GE can employ is to minimise the bias towards any one rule by increasing the size of the codon.

This paper will consider both the possibility of introducing and removing bias through the incorporation of introns.

5 Experimental Approach

The aim of this paper is to examine bias in the grammar and see if using introns and increasing codon size can be used to alter any bias effects that might be observed. We also wish to establish if introns may be useful to GE.

We conduct our experimentation on the Santa Fe ant trail problem. A tableau describing this problem and parameters can be seen in Table 4. The default grammar used for this problem is outlined below.

$$N = \{code, line, if - statement, op\}$$

$$T = \{left(), right(), move(), food\_ahead(), else, if, \{, \}, (,)\}$$

$$S =< code >$$

And P can be represented as:

```
(A) <code> :: =  <line>                (0)
               |<code><line>           (1)

(B) <line> :: =  <if-statement>        (0)
               |<op>                   (1)

(C) <if-statement> :: = if(food_ahead()){
                           <line>
                        }
                        else{ <line> }

(D) <op> :: =    left()                (0)
               | right()               (1)
               | move()                (2)
```

To determine the effect of introns on the performance of GE, grammar defined introns were placed at various points in the grammar, and the cumulative frequency of success measured on the target problem.

For example, 100 runs were conducted where an intron was placed at position zero of Rule (A) as follows:

```
(A) <code> :: = intron                 (0)
               |<line>                  (1)
               |<code><line>            (2)
```

100 runs were then conducted with the intron placed at the other two remaining positions:

```
(A) <code> :: = <line>                 (0)
               |intron                  (1)
               |<code><line>            (2)
```

and,

```
(A) <code> :: = <line>                 (0)
               |<code><line>            (1)
               |intron                  (2)
```

The same approach was taken for the other two non-terminals involving a choice (i.e. Rules B and D).

To take into account the bias that might result from using a smaller codon size, we repeat the above experiments using a 2-bit codon instead of the 8-bits used normally.

6 Results

Cumulative frequencies of success for each of the experiments outlined in the previous section are given in Figures 1, 2, 3 and 4.

Figure 1 shows results for the insertion of an intron at the various positions of rule A. With the intron in position zero, a success rate superior to standard GE is achieved in the case of both 8-bit and 2-bit codons, with little difference between the 8-bit and 2-bit results. In the cases of positions one and two, it can be seen that the presence of the intron has the similar effect of improving success over standard GE. With the addi-

Objective :	Find a computer program to control an artificial ant so that it can find all 89 pieces of food located on the Santa Fe Trail.
Terminal Operators:	left(), right(), move(), food_ahead()
Fitness cases	One fitness case
Raw Fitness	Number of pieces of food before the ant times out with 615 operations.
Standardised Fitness	Total number of pieces of food less the raw fitness.
Hits	Same as raw fitness.
Wrapper	Standard productions to generate C functions
Parameters	$Population$ = 500, $Generations$ = 50 $pmut$ = 0.01, $pcross$ = 0.9

Table 4: Grammatical Evolution Tableau for the Santa Fe Trail

tion of an intron to Rule A, we change the number of choices from two to three, thus biasing the rule in position zero.

In the case that the intron is in position zero and therefore biased towards (stronger bias in the case of a 2-bit codon) we see a superior performance to standard GE, particularly in the case of a 2-bit codon.

These results would suggest that by inserting bias towards the choice of an intron we achieve an improved performance, comparing to what would otherwise be an unbiased rule choice. When 8-bit codons are adopted (reduction in bias towards the rule at position zero), the improvement in performance by placing an intron at position zero is less evident than in the case of 2-bit codons.

In the case of inserting an intron in positions 1 or 2, we are creating a bias towards the rule in position 0, i.e. $< code >::=< line >$. This also gave us superior performance comparing to standard GE. This seems to suggest by forcibly inserting a bias towards certain rules, we can guide the system to make its choices, thus improving the overall performance.

Similar results are observed for rule B, see Figure 2. The presence of introns generally enhances the performance over standard GE, with positive effects due to the insertion of bias either towards introns, or towards existing rules.

Insertion of an intron into rule D has the opposite effect to insertion into rules A and B, i.e. change from an uneven number of choices (3) to an even number (4), see Figure 3 and 4. With the addition of an intron, the bias towards any one of the production rules is removed. The results demonstrate that with the intron placed at all the positions other than position zero, a reduction in performance over standard GE with 8-bit codons is observed. The change in success rate when placed in position zero appears to be less evident in the case of 8-bit codons, but much larger for 2-bit codons.

6.1 Discussion

These results suggest that it is quite possible for a grammar to implicitly contain bias. This, in turn, can have severe implications for the type and quality of individuals explored by the system.

Previous results [O'Neill & Ryan 1999a] have shown that when degeneracy was removed from the system, the performance dropped dramatically. Indeed, Figures 2 to 4 illustrate just how poorly the 2 bit representation (minimal degeneracy) fares.

While it wasn't clear from earlier work exactly why a degenerate encoding was better, these results suggest that degeneracy acts to remove bias from the search. The performance of the 2 bit representation with bias removed approaches that of the 8 bit representation, but on no occasion does it outperform the 8 bit with bias removed. This suggests that degeneracy is doing more than counteracting bias.

Finally, it is clear from the results that sometimes the removal of bias towards a grammar production rule will not improve performance. This in turn suggests that bias in grammars can guide the system to better choices, thus improving the search for a solution.

These findings are, however, limited to the problem domain examined, and as such, further investigations will be required to determine their generality.

7 Conclusions & Future Work

A technique called Grammar Defined Introns is introduced to incorporate introns into GE. Following a discussion on the bias that exists towards certain production rules of the BNF grammar, we demonstrate that the creation of bias has positive effects in the case of the problem domain and grammar examined here. In particular, bias towards introns has been shown to have beneficial effects, thus suggesting that introns

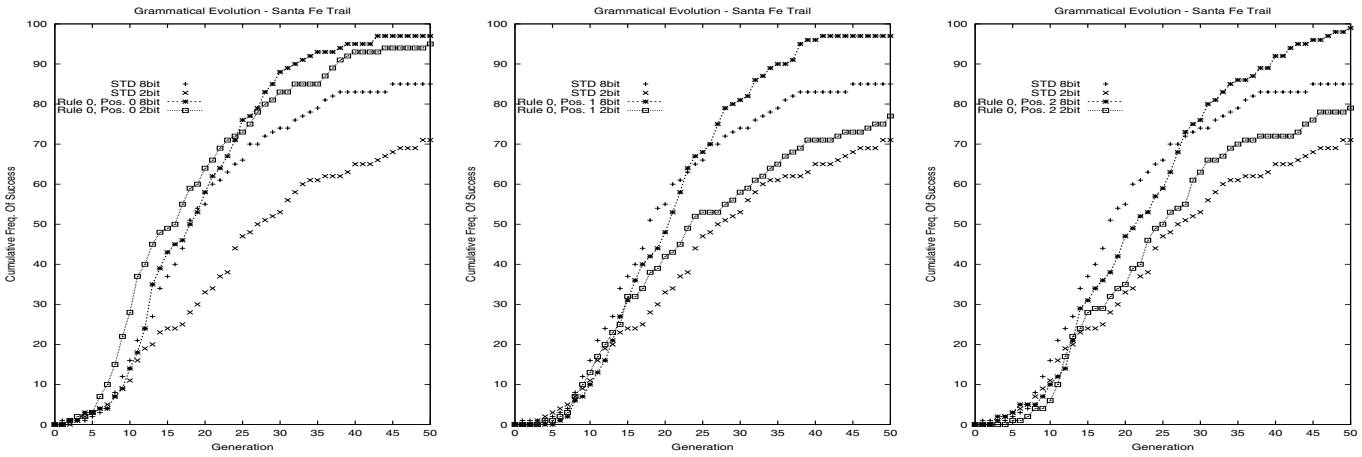


Figure 1: The effects of inserting introns for each choice on the first non-terminal `code`

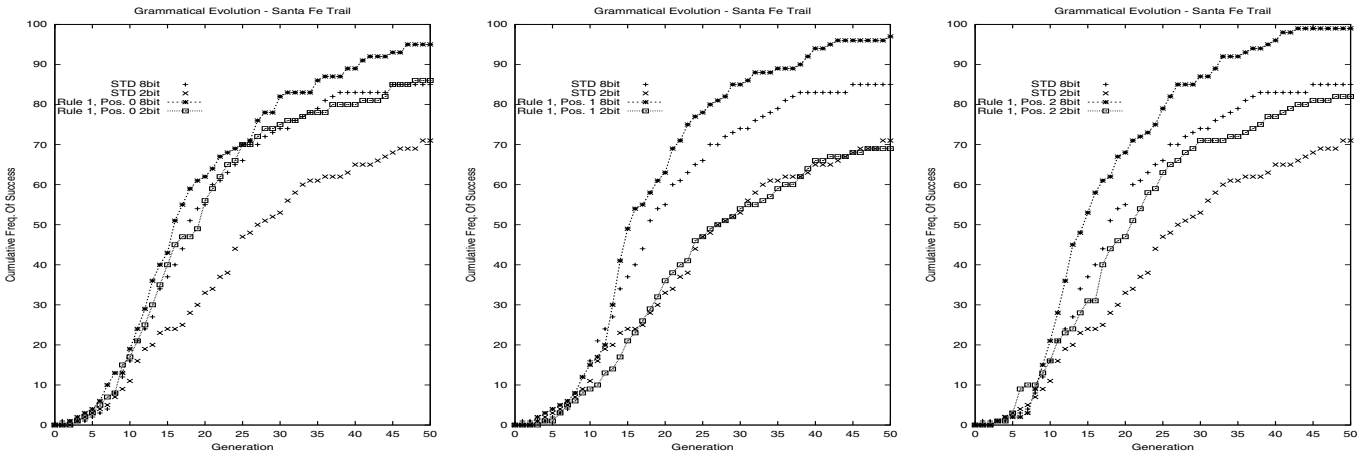


Figure 2: The effects of inserting introns for each choice on the second non-terminal `line`

have a useful role to play in their own right, i.e. in addition to their ability to alter bias towards other production rules.

We show that degeneracy can remove the effect of bias, and that, in many cases, using a degenerate code can outperform a tweaked insertion of introns. In certain cases, a combination of Grammar Defined Introns and degenerate code produces the best performance.

The effect of counteracting bias can be dramatic, and this suggests that much care should be taken in the design of a grammar. Future work will consider the possibility of ideal numbers of productions, and also examine the effects of removing/introducing bias on other problems.

Acknowledgment

The authors wish to thank Maarten Keijzer and Mike Cattolico for the many conversations that helped to form the foundations of this work.

References

[Altenberg 1994] Altenberg L. 1994. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., Ed., *Advances in Genetic Programming*. MIT Press, 1994.

[Andre & Teller 1996] Andre D., Teller A. 1996. A Study in Program Response and the Negative Effects of Introns in Genetic Programming. In *Proceedings of Genetic Programming 1996: Proceedings of the First Annual Conference*, John R. Koza, David E. Goldberg, David B. Fogel, & Rick L. Riolo, Eds. Stanford, USA 1996, pp 12-20.

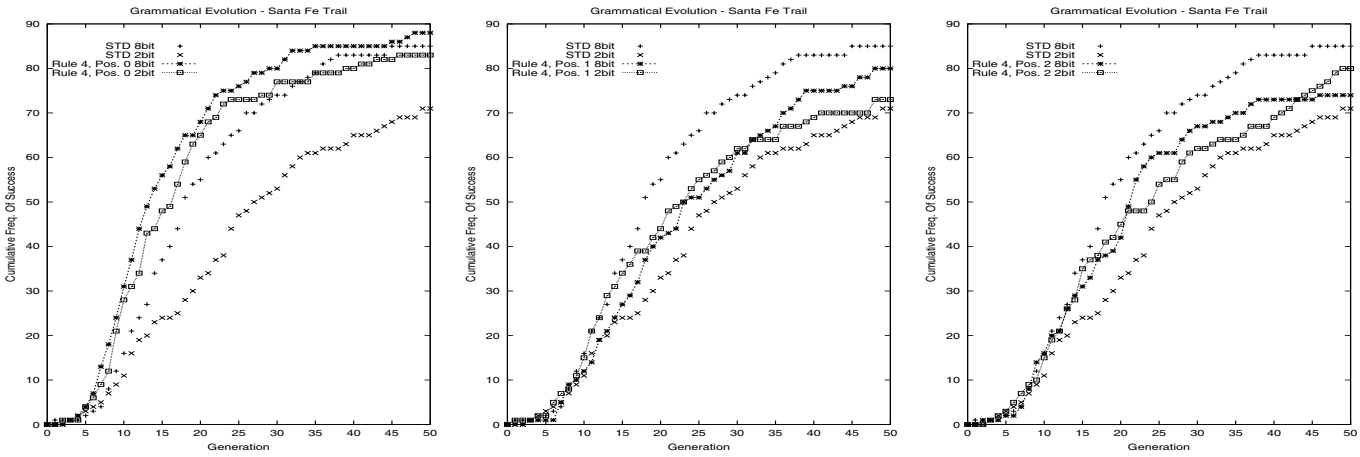


Figure 3: The effects of inserting introns for the first three choices on the fourth non-terminal op

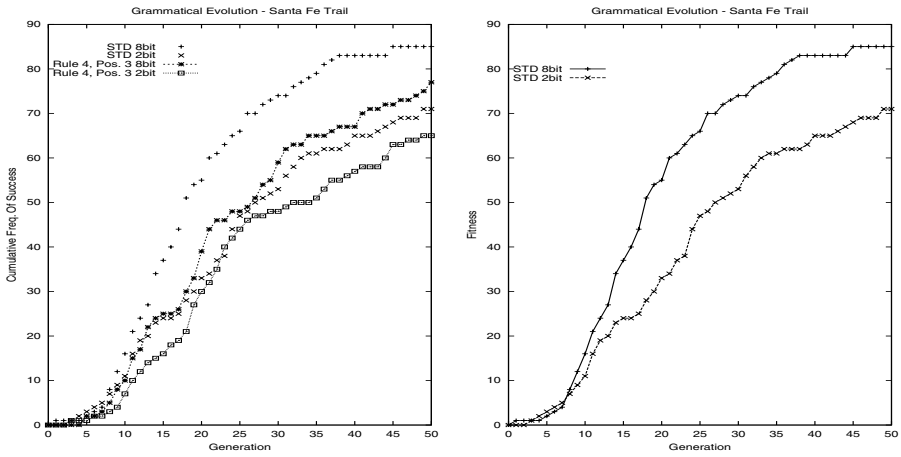


Figure 4: (Left) The effects of inserting introns for the fourth choice on the fourth non-terminal op (Right) Results for 2-bit and 8-bit codons using the standard grammar

[Angeline 1994] Angeline P.J. 1994. Genetic Programming and Emergent Intelligence. In Kenneth E. Kinnear, Jr., Ed., *Advances in Genetic Programming*, MIT Press, pp 75-98.

[Goldberg 1989] Goldberg, David E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley.

[Koza 1992] Koza, J. 1992. *Genetic Programming*. MIT Press.

[Haynes 1996] Haynes T. 1996. Duplication of Coding Segments in Genetic Programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, pp 344-349.

[Levenick 1991] Levenick J. R. 1991. Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology. In *Proceedings of the 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker Eds. San Diego, CA 1991, pp 123-127.

[Lobo et al. 1998] Lobo F.G., Deb K., Goldberg D.E., Harik G., Wang L. 1998. Compressed Introns in a Linkage Learning Genetic Algorithm. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Madison, Wisconsin, pp 551-558.

[Luke 2000] Luke S. 2000. Code Growth Is Not Caused by Introns. In *GECCO'2000*, Las Vegas, pp

[Nordin & Banzhaf 1995] Nordin P. and Banzhaf W. 1995. Complexity compression and evolution. In *Proceedings of the 6th International Conference on Genetic Algorithms* (ICGA-95), Pittsburgh, L. Eshelman (ed.), Morgan Kaufmann, San Francisco, 1995, pp. 310 - 317.

[Nordin, Francone & Banzhaf 1995] Nordin P., Francone F., and Banzhaf W. 1995. Explicitly defined introns and destructive crossover in genetic programming. In Kenneth

E. Kinnear, Jr. and Peter J. Angeline Eds., *Advances in Genetic Programming 2*. MIT Press.

[O'Neill & Ryan 2001] O'Neill M., Ryan C. Grammatical Evolution. *IEEE Trans. Evolutionary Computation*. 2001.

[O'Neill & Ryan 2000] O'Neill M., Ryan C. 2000. Crossover in Grammatical Evolution: A Smooth Operator? *Lecture Notes in Computer Science 1802, Proceedings of the European Conference on Genetic Programming*, pages 149-162. Springer-Verlag.

[O'Neill & Ryan 1999a] O'Neill M., Ryan C. 1999. Genetic Code Degeneracy: Implications for Grammatical Evolution and Beyond. In *Proceedings of the Fifth European Conference on Artificial Life*.

[O'Neill & Ryan 1999b] O'Neill M., Ryan C. 1999. Under the Hood of Grammatical Evolution. In *Proceedings of the Genetic & Evolutionary Computation Conference 1999*.

[O'Neill & Ryan 1999c] O'Neill M., Ryan C. 1999. Evolving Multi-line Compilable C Programs. *Lecture Notes in Computer Science 1598, Proceedings of the Second European Workshop on Genetic Programming*, pages 83-92. Springer-Verlag.

[Paterson & Livesley] Paterson N., Livesley M. Evolving Caching Algorithms in C by Genetic Programming. In *GP'97: Proceedings of the Second Annual Conference*, pages 262-267.

[Ryan C., Collins J.J. & O'Neill M. 1998] Ryan C., Collins J.J., O'Neill M. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, pages 83-95. Springer-Verlag.

[Smith & Harries 1998] Smith P.W.H., and Harries K. 1998. Code Growth, Explicitly Defined Introns, and Alternative Selection Schemes. *Evolutionary Computation* **6**:4, pp 339-360.

[ICGA Workshop 1997] Workshop on Exploring Noncoding Segments and Genetics-based Encodings, International Conference on Genetic Algorithms 1997, MI, USA.

[Wineberg & Oppacher 1996] Wineberg M. and Oppacher F. 1996. The Benefits of Computing with Introns, In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo Eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, pages 410-415.

[Wu & Lindsay 1995] Wu A. S. and Lindsay R. K. 1995. Empirical studies of the genetic algorithm with noncoding segments. *Evolutionary Computation* **3**, pp 121-48.

[Wu & Lindsay 1996] Wu A. S and Lindsay R. K. 1996. A survey of intron research in genetics, in *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, Berlin, Germany, September 1996.

Exact Schema Theory for GP and Variable-length GAs with Homologous Crossover

Riccardo Poli
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
R.Poli@cs.bham.ac.uk

Nicholas Freitag McPhee
Division of Science and Mathematics
University of Minnesota, Morris
Morris, MN, USA
mcphee@mrs.umn.edu

Abstract

In this paper we present a new exact schema theory for genetic programming and variable-length genetic algorithms which is applicable to the general class of homologous crossovers. These are a group of operators, including GP one-point crossover and GP uniform crossover, where the offspring are created preserving the position of the genetic material taken from the parents. The theory is based on the concepts of GP crossover masks and GP recombination distributions (both introduced here for the first time), as well as the notions of hyperschema and node reference systems introduced in other recent research. This theory generalises and refines previous work in GP and GA theory.

1 Introduction

Genetic programming theory has had a difficult childhood. After some excellent early efforts leading to different approximate schema theorems [1, 2, 3, 4, 5, 6, 7], only very recently have schema theories become available which give exact formulations (rather than lower bounds) for the expected number of instances of a schema at the next generation. These exact theories are applicable to GP with one-point crossover [8, 9, 10], standard crossover and other subtree-swapping crossovers [11, 12, 13], and different types of subtree mutation and headless chicken crossover [14, 15].

Here we extend this work by presenting a new exact schema theory for genetic programming which is applicable to a very important and general class of operators which we call homologous crossovers. This group of operators generalises most common GA crossovers and includes GP one-point crossover and GP uniform crossover [16]. These operators differ from the standard subtree swapping crossover [1] in that they require that the offspring being created preserve the position of the genetic material taken from the parents.

The paper is organised as follows. Firstly, we provide a review of earlier relevant work on GP schemata and cover the key definitions and terms in Section 2. Then, in Section 3 we show how these ideas can be used to define the class of homologous crossover operators and build probabilistic models for them. In Section 4 we use these to derive schema theory results and an exact definition of effective fitness for GP with homologous crossover. In Section 5 we give an example that shows how the theory can be applied. Some conclusions are drawn in Section 6.

2 Background

Schemata are sets of points of the search space sharing some syntactic feature. For example, in the context of GAs operating on binary strings, the syntactic representation of a schema is usually a string of symbols from the alphabet $\{0,1,*\}$, where the character * is interpreted as a "don't care" symbol. Typically schema theorems are descriptions of how the number of members of the population belonging to a schema vary over time. Let $\alpha(H,t)$ denote the probability at time t that a newly created individual samples (or matches) the schema H, which we term the *total transmission probability* of H. Then an exact schema theorem for a generational system is simply [17]

$$E[m(H,t+1)] = M\alpha(H,t), \tag{1}$$

where M is the population size, $m(H,t+1)$ is the number of individuals sampling H at generation $t+1$ and $E[\cdot]$ is the expectation operator. Holland's [18] and other worst-case-scenario schema theories normally provide a lower bound for $\alpha(H,t)$ or, equivalently, for $E[m(H,t+1)]$.

One of the difficulties in obtaining theoretical results on GP using the idea of schema is that finding a workable definition of a schema is much less straightforward than for GAs. Several alternative definitions have been proposed in

the literature [1, 2, 3, 4, 6, 7, 5]. For brevity here we will describe only the definition introduced in [6, 7], since this is what is used in the rest of this paper. We will refer to this kind of schemata as *fixed-size-and-shape schemata*.

Syntactically a GP fixed-size-and-shape schema is a tree composed of functions from the set $\mathcal{F} \cup \{=\}$ and terminals from the set $\mathcal{T} \cup \{=\}$, where $\mathcal{F}$ and $\mathcal{T}$ are the function and terminal sets used in a GP run. The primitive $=$ is a "don't care" symbol which stands for a *single* terminal or function. A schema H represents the set of all programs having the same shape as H and the same labels for the non-$=$ nodes. For example, if $\mathcal{F}$=`{+, *}` and $\mathcal{T}$=`{x, y}` the schema `(+ x (= y =))` represents the four programs `(+ x (+ y x))`, `(+ x (+ y y))`, `(+ x (* y x))` and `(+ x (* y y))`.

In [6, 7] a worst-case-scenario schema theorem was derived for GP with point mutation and one-point crossover; as discussed in [8], this theorem is a generalisation of the version of Holland's schema theorem [18] presented in [19] to variable size structures. One-point crossover works by using the same crossover point in both parent programs, and then swapping the corresponding subtrees like standard crossover. To account for the possible structural diversity of the two parents, the selection of the crossover point is restricted to the *common region*, the largest rooted region where the two parent trees have the same topology. The common region will be defined formally in Section 3.

One-point crossover can be considered to be an instance of a much broader class of operators that can be defined through the notion of the common region. For example, in [16] we defined and studied a GP operator, called *uniform crossover* (based on uniform crossover in GAs), in which the offspring is created by independently swapping the nodes in the common region with a uniform probability. If a node belongs to the boundary of the common region and is a function then also the nodes below it are swapped, otherwise only the node label is swapped. Many other operators of this kind are possible. We will call them *homologous crossovers*, noting that our definition is more restrictive than that in [20]. A formal description of these operators will be given in Section 3.

The approximate schema theorem in [6, 7] was improved in [9, 10], where an exact schema theory for GP with one-point crossover was derived which was based on the notion of hyperschema. A *GP hyperschema* is a rooted tree composed of internal nodes from $\mathcal{F} \cup \{=\}$ and leaves from $\mathcal{T} \cup \{=, \#\}$. Again, `=` is a "don't care" symbols which stands for exactly one node, while `#` stands for any valid subtree. For example, the hyperschema `(* # (= x =))` represents all the programs with the following characteristics: a) the root node is a product, b) the first argument of the root node is any valid subtree, c) the second argument of the root node is any function of arity two, d) the first argument of this function is the variable `x`, e) the second argument of the function is any valid node in the terminal set. One of the results obtained in [10] is

$$\alpha(H,t) = (1 - p_{xo})p(H,t) + p_{xo}\alpha_{xo}(H,t) \qquad (2)$$

where

$$\alpha_{xo}(H,t) = \sum_{k,l} \frac{1}{\mathbf{NC}(G_k, G_l)} \qquad (3)$$
$$\times \sum_{i \in C(G_k,G_l)} p(U(H,i) \cap G_k, t)p(L(H,i) \cap G_l, t)$$

and: p_{xo} is the crossover probability; $p(H,t)$ is the selection probability of the schema H;[1] G_1, G_2, $\cdots$ are an enumeration of all the possible program shapes, i.e. all the possible fixed-size-and-shape schemata containing = signs only; $\mathbf{NC}(G_k, G_l)$ is the number of nodes in the common region between shape G_k and shape G_l; $C(G_k, G_l)$ is the set of indices of the crossover points in such a common region; $L(H,i)$ is the hyperschema obtained by replacing all the nodes on the path between crossover point i and the root node with = nodes, and all the subtrees connected to those nodes with # nodes; $U(H,i)$ is the hyperschema obtained by replacing the subtree below crossover point i with a # node; if a crossover point i is in the common region between two programs but it is outside the schema H, then $L(H,i)$ and $U(H,i)$ are defined to be the empty set. The hyperschemata $L(H,i)$ and $U(H,i)$ are important because, if one crosses over at point i *any* individual in $L(H,i)$ with *any* individual in $U(H,i)$, the resulting offspring is always an instance of H. The steps involved in the construction of $L(H,i)$ and $U(H,i)$ for the schema H = `(* = (+ x =))` are illustrated in Figure 1.

As discussed in [8], it is possible to show that, in the absence of mutation, Equations 2 and 3 generalise and refine not only the GP schema theorem in [6, 7] but also the version of Holland's schema theorem [18] presented in [19], as well as more recent GA schema theory [21, 22].

Very recently, this work has been extended in [11] where a general, exact schema theory for genetic programming with subtree swapping crossover was presented. The theory is based on a generalisation of the notion of hyperschema and on a Cartesian node reference system which makes it possible to describe programs as functions over the space $\mathbb{N}^2$.

The Cartesian reference system is obtained by considering the ideal infinite tree consisting entirely of nodes of some fixed maximum arity a_{max}. This maximal tree would include 1 node of arity a_{max} at depth 0, a_{max} nodes of arity a_{max} at depth 1, $(a_{max})^2$ nodes of arity a_{max} at depth 2, and

[1] In fitness proportionate selection $p(H,t) = m(H,t)f(H,t)/(M\bar{f}(t))$, where $m(H,t)$ is the number of trees in the schema H at time t, $f(H,t)$ is their mean fitness, and $\bar{f}(t)$ is the mean fitness of the trees in the population.

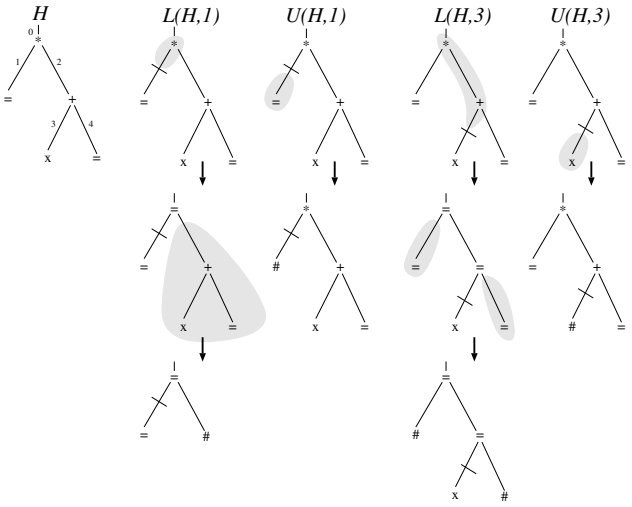


Figure 1: Example of a schema and some of its potential hyperschema building blocks. The crossover points in H are numbered as shown in the top left.

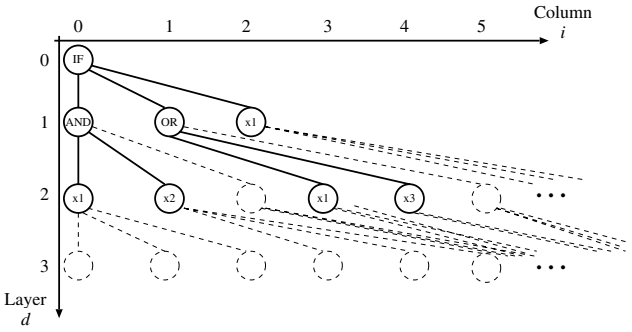


Figure 2: Syntax tree for the program `(IF (AND x1 x2) (OR x1 x3) x1)` represented in a tree-independent Cartesian node reference system for nodes with maximum arity 3. Unused nodes and links of the maximal tree are drawn with dashed lines. Only four layers and six columns are shown.

generally $(a_{max})^d$ nodes at depth d. Then one could imagine organising the nodes in the tree into layers of increasing depth (see Figure 2) and assigning an index to each node in a layer. The layer number d and the index i can then be used to define a Cartesian coordinate system. Clearly, one could also use this reference system to locate the nodes of non-maximal trees. This is possible because a non-maximal tree can always be described using a subset of the nodes and links in the maximal tree. This is illustrated for the program `(IF (AND x1 x2) (OR x1 x3) x1)` in Figure 2. So, for example, the `IF` node would have coordinates (0,0), the `AND` would have coordinates (1,0), and the `x3` node would have coordinates (2,4). In this reference system it is always possible to find the route to the root node from any valid coordinate. Also, if one chooses a_{max} to be the maximum arity of the functions in the function set, it is possible to use this reference system to represent the structure of any program that can be constructed with that function set.

The theory in [11] is also applicable to standard GP crossover [1] with and without uniform selection of the crossover points, one-point crossover [6, 7], size-fair crossover [20], strongly-typed GP crossover [23], context-preserving crossover [24], and many others. The theory has also been recently extended to subtree mutation and headless chicken crossover [14, 15]. It does not, however, currently cover the class of homologous operators and the goal of this paper is to fill that theoretical gap.

3 Modelling Homologous Crossovers

Given a node reference system it is possible to define functions over it. An example of such functions is the *arity function* $A(d, i, h)$ which returns the arity of the node at coordinates (d, i) in h. For example, for the tree in Figure 2, $A(0, 0, h) = 3$, $A(1, 0, h) = 2$ and $A(2, 1, h) = 0$. Similarly, it is possible to define the *common region membership function* $\mathcal{C}(d, i, h_1, h_2)$ which returns **true** when (d, i) is part of the common region of h_1 and h_2. Formally, $\mathcal{C}(d, i, h_1, h_2) = \mathbf{true}$ when either $(d, i) = (0, 0)$ or

$$A\left(d-1, i', h_1\right) = A\left(d-1, i', h_2\right) \neq 0$$
$$\textbf{and} \quad \mathcal{C}\left(d-1, i', h_1, h_2\right) = \mathbf{true},$$

where $i' = \lfloor i/a_{max} \rfloor$ and $\lfloor \cdot \rfloor$ is the integer-part function. This allows us to formalise the notion of *common region*:

$$C(h_1, h_2) = \{(d, i) \mid \mathcal{C}(d, i, h_1, h_2) = \mathbf{true}\}. \quad (4)$$

This is the notion of common region used in the schema theorem for one-point crossover in Equation 2. As indicated before, one-point crossover selects the same crossover point in both parents by randomly choosing a node in the common region. An alternative way to interpret the action of one-point crossover is to imagine that the subset of nodes in $C(h_1, h_2)$ below such a crossover point are transferred from parent h_2 into an empty coordinate system, while all the remaining nodes in $C(h_1, h_2)$ are taken from parent h_1. Clearly, nodes representing the leaves of the common region should be transferred together with their subtrees, if any. Other homologous crossovers can simply be defined by selecting subsets of nodes in the common region differently.

A good way to describe and model the class of homologous crossovers is to extend the notions of crossover masks and recombination distributions used in genetics [25] and in the GA literature [26, 27, 28]. In a GA operating on fixed-length strings a crossover mask is simply a binary string. When crossover is executed, the bits of the offspring corresponding to the 1's in the mask will be taken from one parent, those corresponding to 0's from the other parent. For example, if the parents are the strings `aaaaaa` and `bbbbbb` and the crossover mask is `110100`, one offspring would be `aababb`. For operators returning two offspring it

is easy to show that the second offspring can be obtained by simply complementing, bit by bit, the crossover mask. For example, the complement of the mask `110100`, `001011`, gives the offspring `bbabaa`. If the GA operates on strings of length N, then 2^N different crossover masks are possible. If, for each mask i, one defines a probability, p_i, that the mask is selected for crossover, then it is easy to see how different crossover operators can simply be interpreted as different ways of choosing the probability distribution p_i. For example, for strings of length $N = 4$ the probability distribution for one-point crossover would be $p_i = 1/3$ for the crossover masks $i = 1000, 1100, 1110$ and $p_i = 0$ otherwise, while for uniform crossover $p_i = 1/16$ for all 16 i's. The probability distribution p_i is called a *recombination distribution*.

Let us now extend the notion of recombination distributions to genetic programming with homologous crossover. For any given shape and size of the common region we can define a set of *GP crossover masks* which correspond to all possible ways in which a recombination event can take place within the given common region. Because the nodes in the common region are always arranged so as to form a tree, it is possible to represent the common region as a tree or an equivalent S-expression. So, GP crossover masks can be thought of as trees constructed using 0's and 1's that have the same size and shape as the common region. So, for example, if the common region is represented by the set of node coordinates {(0,0),(1,0),(1,1)}, then there are eight valid GP crossover masks: `(0 0 0)`, `(0 0 1)`, `(0 1 0)`, `(0 1 1)`, `(1 0 0)`, `(1 0 1)`, `(1 1 0)` and `(1 1 1)`. The complement of a GP crossover mask is an obvious extension, where the complement $\bar{i}$ has the same structure as mask i but with the 0's and 1's swapped. In the following we will use χ_c to denote the set of the $2^{N(c)}$ crossover masks associated with the common region c, where $N(c)$ is the number of nodes in c. Since we are typically interested in the common region defined by two trees, we'll use $\chi(h_1, h_2)$ as a shorthand for $\chi_{C(h_1,h_2)}$.

Once χ_c is defined we can define a *fixed-size-and-shape recombination distribution* p_i^c which gives the probability that crossover mask $i \in \chi_c$ will be chosen for crossover between individuals having common region c. Then the set $\{p_i^c \mid \forall c\}$, which we call a *GP recombination distribution*, completely defines the behaviour of a GP homologous crossover operator, different operators being characterised by different assignments for the p_i^c. For example, the GP recombination distribution for uniform GP crossover with 50% probability of exchanging nodes is $p_i^c = (0.5)^{N(c)}$.

GP crossover masks and GP recombination distributions generalise the corresponding GA notions. Indeed, as also discussed in [8], GAs operating on fixed-length strings are simply a special case of GP with homologous crossover. This can be shown by considering the case of function sets including only unary functions and initialising the population with programs of the same length. Since in a linear GP system with fixed length programs every individual has exactly the same size and (linear) shape, only one common region c is possible. Therefore, only one fixed-size-and-shape recombination distribution p_i^c is required to characterise crossover. In variable length GAs and GP, multiple fixed-size-and-shape recombination distributions are necessary, one for every possible common region c.

4 Exact GP Schema Theory for Homologous Crossovers

Using hyperschemata and GP recombination distributions for homologous crossover, we obtain the following:

Theorem 1. *The total transmission probability for a fixed-size-and-shape GP schema H under homologous crossover is given by Equation 2 with*

$$\alpha_{xo}(H,t) = \sum_{h_1}\sum_{h_2} p(h_1,t)p(h_2,t) \sum_{i\in\chi(h_1,h_2)} p_i^{C(h_1,h_2)} \times \delta(h_1 \in \Gamma(H,i))\delta(h_2 \in \Gamma(H,\bar{i})) \quad (5)$$

where: the first two summations are over all the individuals in the population; $C(h_1, h_2)$ is the common region between program h_1 and program h_2; $\chi(h_1, h_2)$ is the set of crossover masks associated with $C(h_1, h_2)$; $\delta(x)$ is a function which returns 1 if x is true, 0 otherwise; $\Gamma(H, i)$ is defined below; $\bar{i}$ is the complement of crossover mask i.

$\Gamma(H, i)$ is defined to be the empty set if i contains any node not in H. Otherwise it is the hyperschema obtained by replacing certain nodes in H with either = or # nodes:

- If a node in H corresponds to (i.e., has the same coordinates as) a non-leaf node in i that is labelled with a 0, then that node in H is replaced with a =.
- If a node in H corresponds to a leaf node in i that is labelled with a 0, then it is replaced with a #.
- All other nodes in H are left unchanged.

If, for example, $H =$ `(* = (+ x =))`, as indicated in Figure 3(a), then $\Gamma(H, (0\ 1\ 0))$ is obtained by first replacing the root node with a = symbol (because the crossover mask has a function node 0 at coordinates (0,0)) and then replacing the subtree rooted at coordinates (1,1) with a # symbol (because the crossover mask has a terminal node 0 at coordinates (1,1)) obtaining `(= = #)`. The schema $\Gamma(H, (1\ 0\ 1))$, which forms a complementary pair with the previous one, is instead obtained by replacing the subtree rooted at coordinates (1,0) with a # symbol obtaining `(* # (+ x =))`, as illustrated in Figure 3(b).

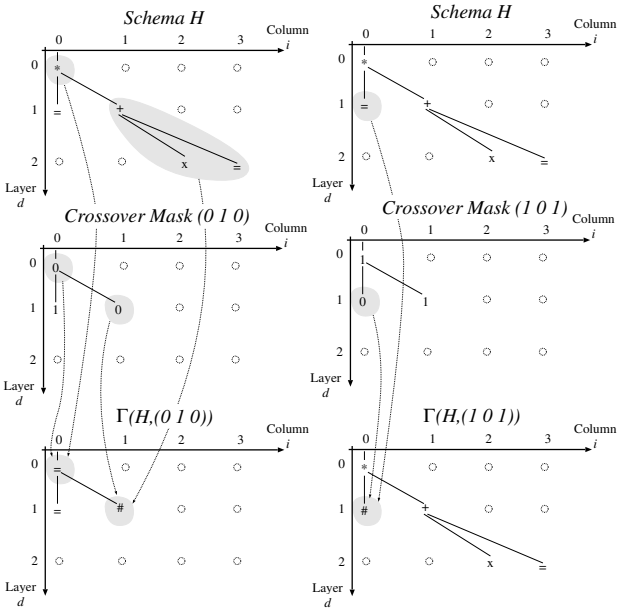


Figure 3: A complementary pair of hyperschemata $\Gamma(H,i)$ for the schema $H = (* \ = \ (+ \ x \ =))$.

The hyperschemata $\Gamma(H,i)$ and $\Gamma(H,\bar{i})$ are generalisations of the schemata $L(H,i)$ and $U(H,i)$ used in Equation 2 (compare Figures 1 and 3). In general if one crosses over using crossover mask i *any* individual in $\Gamma(H,i)$ with *any* individual in $\Gamma(H,\bar{i})$, the resulting offspring is always an instance of H.

Once the concept of $\Gamma(H,i)$ is available, the theorem can easily be proven.

Proof. Let $p(h_1,h_2,i,t)$ be the probability that, at generation t, the selection-crossover process will choose parents h_1 and h_2 and crossover mask i. Then, let us consider the function

$$g(h_1,h_2,i,H) = \delta(h_1 \in \Gamma(H,i))\delta(h_2 \in \Gamma(H,\bar{i})).$$

Given two parent programs, h_1 and h_2, and a schema of interest H, this function returns the value 1 if crossing over h_1 and h_2 with crossover mask i yields an offspring in H. It returns 0 otherwise. This function can be considered as a measurement function (see [27]) that we want to apply to the probability distribution of parents and crossover masks at time t, $p(h_1,h_2,i,t)$. If h_1, h_2 and i are stochastic variables with joint probability distribution $p(h_1,h_2,i,t)$, the function $g(h_1,h_2,i,H)$ can be used to define a stochastic variable $\gamma = g(h_1,h_2,i,H)$. The expected value of γ is:

$$E[\gamma] = \sum_{h_1}\sum_{h_2}\sum_{i} g(h_1,h_2,i,H)p(h_1,h_2,i,t). \quad (6)$$

Since γ is a binary stochastic variable, its expected value also represents the proportion of times it takes the value 1. This corresponds to the proportion of times the offspring of h_1 and h_2 are in H.

We can write

$$p(h_1,h_2,i,t) = p(i|h_1,h_2)p(h_1,t)p(h_2,t),$$

where $p(i|h_1,h_2)$ is the conditional probability that crossover mask i will be selected when the parents are h_1 and h_2, while $p(h_1,t)$ and $p(h_2,t)$ are the selection probabilities for the parents. In homologous crossover $p(i|h_1,h_2) = p_i^{C(h_1,h_2)}\delta(i \in \chi(h_1,h_2))$, so

$$\begin{aligned} & p(h_1,h_2,i,t) \\ &= \ p(h_1,t)p(h_2,t)p_i^{C(h_1,h_2)}\delta(i \in \chi(h_1,h_2)). \end{aligned}$$

Substituting this into Equation 6 with minor simplifications leads to the expression of α_{xo} in Equation 5. □

Equations 2 and 5 allow one to compute the exact total transmission probability of a GP schema in terms of microscopic quantities. It is possible, however, to transform this model into the following exact macroscopic model of schema propagation

Theorem 2. *The total transmission probability for a fixed-size-and-shape GP schema H under homologous crossover is given by Equation 2 with*

$$\alpha_{xo}(H,t) = \sum_j \sum_k \sum_{i \in \chi(G_j,G_k)} p_i^{C(G_j,G_k)} \times \quad (7)$$
$$p(\Gamma(H,i) \cap G_j, t)p(\Gamma(H,\bar{i}) \cap G_k, t).$$

Proof. Let us start by considering all the possible program shapes G_1, G_2, $\cdots$. These schemata represent disjoint sets of programs. Their union represents the whole search space, so

$$\sum_j \delta(h_1 \in G_j) = 1.$$

We insert the l.h.s. of this expression and of an analogous expression for $\delta(h_2 \in G_k)$ in Equation 5 and reorder the terms obtaining:[2]

$$\begin{aligned} & \alpha_{xo}(H,t) \\ &= \sum_j \sum_k \sum_{h_1} \sum_{h_2} p(h_1,t)p(h_2,t) \\ &\quad \sum_{i \in \chi(h_1,h_2)} p_i^{C(h_1,h_2)}\delta(h_1 \in \Gamma(H,i))\delta(h_1 \in G_j) \\ &\quad \delta(h_2 \in \Gamma(H,\bar{i}))\delta(h_2 \in G_k) \\ &= \sum_j \sum_k \sum_{h_1 \in G_j} \sum_{h_2 \in G_k} p(h_1,t)p(h_2,t) \\ &\quad \sum_{i \in \chi(h_1,h_2)} p_i^{C(h_1,h_2)}\delta(h_1 \in \Gamma(H,i))\delta(h_2 \in \Gamma(H,\bar{i})) \end{aligned}$$

[2] Note that $h_1 \in G_j \wedge h_2 \in G_k \Rightarrow C(h_1,h_2) = C(G_j,G_k)$.

$$= \sum_j \sum_k \sum_{h_1 \in G_j} \sum_{h_2 \in G_k} p(h_1,t)p(h_2,t) \sum_{i \in \chi(G_j,G_k)} p_i^{C(G_j,G_k)} \delta(h_1 \in \Gamma(H,i))\delta(h_2 \in \Gamma(H,\bar{i}))$$

$$= \sum_j \sum_k \sum_{i \in \chi(G_j,G_k)} p_i^{C(G_j,G_k)} \sum_{h_1 \in G_j} p(h_1,t) \delta(h_1 \in \Gamma(H,i)) \sum_{h_2 \in G_k} p(h_2,t)\delta(h_2 \in \Gamma(H,\bar{i})).$$

Since $\sum_{h_1 \in G_j} p(h_1,t)\delta(h_1 \in \Gamma(H,i)) = p(\Gamma(H,i) \cap G_j,t)$ (and similarly for $p(\Gamma(H,\bar{i}) \cap G_k,t)$), this equation completes the proof of the theorem. □

This theorem is a generalisation of Equations 2 and 3. These, as indicated in Section 2, are a generalisation of a recent GA schema theorem for one-point crossover [21, 22] and a refinement (in the absence of mutation) of both the GP schema theorem in [6] and Goldberg's version [19] of Holland's schema theory [18]. The schema theorems in this paper also generalise other GA results (such as those summarised in [29]), as well as the result in [27, appendix], since they can be applied to linear schemata and even fixed-length binary strings. So, in the absence of mutation, *the schema theory in this paper generalises and refines not only earlier GP schema theorems but also old and modern GA schema theories for one- and multi-point crossover, uniform crossover and all other homologous crossovers*.

Once the value of $\alpha(H,t)$ is available, it is trivial to extend (as we did in [10, 11]) the notion of effective fitness provided in [21, 22] obtaining the following:

Corollary 3. *The effective fitness of a fixed-size-and-shape GP schema H under homologous crossover is*

$$f_{\text{eff}}(H,t) = \frac{\alpha(H,t)}{p(H,t)} f(H,t) = f(H,t)\left[1 - p_{xo}\left(1 - \sum_{j,k} \sum_{i \in \chi(G_j,G_k)} p_i^{C(G_j,G_k)} \times \frac{p(\Gamma(H,i) \cap G_j,t)p(\Gamma(H,\bar{i}) \cap G_k,t)}{p(H,t)}\right)\right]. \quad (8)$$

5 Example

Since the calculations involved in applying exact GP schema theorems can become quite lengthy, we will limit ourselves here to one extremely simple example. For applications of this and related schema theories see [12, 13, 14, 15, 30]. To make clearer the relationship between this work and our theory for one-point crossover, we will use the same example as in [10], this time using general homologous crossover operators instead of just one-point crossover.

Let us imagine that we have a function set $\{A_f, B_f, C_f, D_f, E_f\}$ including only unary functions, and the terminal set $\{A_t, B_t, C_t, D_t, E_t\}$. Since, all functions are unary, we can unambiguously represent expressions without parenthesis. In addition, since the only terminal in each expression is the rightmost node, we can remove the subscripts without generating any ambiguity. Thus, every member of the search space can be seen as a variable-length string over the alphabet $\{A, B, C, D, E\}$, and GP with homologous crossover is really a non-binary variable-length GA.

Let us now consider the schema AB=. We want to measure its total transmission probability (with $p_{xo} = 1$) under fitness proportionate selection and an arbitrary homologous crossover operator for the following population:

Population	Fitness
AB	2
BCD	2
ABC	4
ABCD	6

In order to apply Equation 7 we first need to number all the possible program shapes G_1, G_2, etc.. Let G_1 be =, G_2 be ==, G_3 be === and G_4 be ====. We do not need to consider other, larger shapes because the population does not contain any larger programs. We then need to evaluate the shape of the common regions to determine $\chi(G_j,G_k)$ for all valid values of j and k. In this case the common regions can be naturally represented using integers which represent the length of the common region. Since the length of the common region is the length of the shorter parent, we know $C(G_j,G_k) = \min(j,k)$. Then, for each common region c we need to identify the hyperschemata $\Gamma(\texttt{AB=},i)$ for all the meaningful crossover masks $i \in \chi_c$ and calculate $\Gamma(\texttt{AB=},i) \cap G_j$ for all meaningful values of j. These calculations are shown in Table 1. Using this table we can apply Equation 7, obtaining, after simplification and omitting t and the superscript c from p_i^c for brevity,

$$\begin{aligned}
\alpha(\texttt{AB=}) &= \alpha_{xo}(\texttt{AB=}) \\
&= \sum_{j,k=1}^{4} \sum_{i \in \{0,1\}^{\min(j,k)}} p_i p(\Gamma(H,i) \cap G_j)p(\Gamma(H,\bar{i}) \cap G_k) \\
&= (p_0 + p_1)p(AB=)p(=) + \\
&\quad (p_{00} + p_{11})p(AB=)p(==) + \\
&\quad (p_{01} + p_{10})p(=B=)p(A=) + \\
&\quad (p_{000} + p_{111})p(AB=)(p(===) + p(====)) + \\
&\quad (p_{001} + p_{110})p(===)(p(AB=) + p(AB==)) + \\
&\quad (p_{010} + p_{101})p(A==)(p(=B=) + p(=B==)) + \\
&\quad (p_{011} + p_{100})p(=B=)(p(A==) + p(A===)).
\end{aligned}$$

This equation is valid for any homologous crossover operator, each of which is defined by the set of p_i. It is easy to specialises it for one-point crossover by using the

Mask i	$\Gamma(\texttt{AB=}, i)$	$\Gamma(\texttt{AB=}, i) \cap G_j$			
		$j=1$	$j=2$	$j=3$	$j=4$
0	#	=	==	===	====
1	AB=	$\emptyset$	$\emptyset$	AB=	$\emptyset$
00	=#	$\emptyset$	==	===	====
01	=B=	$\emptyset$	$\emptyset$	=B=	$\emptyset$
10	A#	$\emptyset$	A=	A==	A===
11	AB=	$\emptyset$	$\emptyset$	AB=	$\emptyset$
000	==#	$\emptyset$	$\emptyset$	===	====
001	===	$\emptyset$	$\emptyset$	===	$\emptyset$
010	=B#	$\emptyset$	$\emptyset$	=B=	=B==
011	=B=	$\emptyset$	$\emptyset$	=B=	$\emptyset$
100	A=#	$\emptyset$	$\emptyset$	A==	A===
101	A==	$\emptyset$	$\emptyset$	A==	$\emptyset$
110	AB#	$\emptyset$	$\emptyset$	AB=	AB==
111	AB=	$\emptyset$	$\emptyset$	AB=	$\emptyset$
0000	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Crossover masks and schemata necessary to calculate $\alpha_{xo}(\texttt{AB} =)$.

recombination distribution $p_0 = 1$, $p_{00} = p_{10} = 1/2$, $p_{000} = p_{100} = p_{110} = 1/3$ and $p_i = 0$ for all other crossover masks. This leads to the same result as in [10].

It is also easy to specialise the previous equation to uniform crossover by using the recombination distribution $p_i = (0.5)^{N(i)}$, where $N(i)$ is the length of crossover mask i. Doing so in this case yields $\alpha(\texttt{AB=}, t) \approx 0.2806$. For the same example, in [10] we obtained $\alpha(\texttt{AB=}, t) \approx 0.2925$ for one-point crossover, which indicates that uniform crossover is slightly less "friendly" towards the schema. We can also use Equation 8 to compute the effective fitness for the schema `AB=` for both uniform and one-point crossover, obtaining values of approximately 3.9 and 4.1, respectively. These values are very close to the actual average fitness of the schema in the current population, 4, suggesting that in this case disruption and creation effects tend to balance out. This is not always the case, however, as is shown in [10].

6 Conclusions

Unlike GA theory, which has made considerable progress in the last ten years or so, GP theory has typically been scarce, approximate and, as a rule, not terribly useful. This is not surprising given the youth of GP and the complexities of building theories for variable size structures. In the last year or so, however, significant breakthroughs have changed this situation radically. Today not only do we have exact schema theorems for GP with a variety of operators including subtree mutation, headless chicken crossover, standard crossover, one-point crossover, and all other subtree swapping crossovers, but this GP theory also generalises and refines a broad spectrum of GA theory, as indicated in Section 2.

We believe that this paper extends this series of breakthroughs. Here we have presented a new schema theory applicable to genetic programming and both variable- and fixed-length genetic algorithms with homologous crossover. The theory is based on the concepts of GP crossover masks and GP recombination distributions, both introduced here for the first time. As discussed in Section 4, this theory also generalises and refines a broad spectrum of previous work in GP and GA theory.

Clearly this paper is only a first step. We have not yet made any attempt to use our new schema evolution equations to understand the dynamics of GP or variable-length GAs with homologous crossover or to design competent GP/GA systems. In other recent work, however, we have specialised and applied the theory for other operators to understand phenomena such as operator biases and the evolution of size in variable length GAs [12, 13, 14, 15]. In the future we hope to be able to do the same and produce exciting new results with the theory presented here.

Acknowledgements

The authors would like to thank the members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group at Birmingham, for useful discussions and comments. Nic thanks to The University of Birmingham School of Computer Science for graciously hosting him during his sabbatical, and various offices and individuals at the University of Minnesota, Morris, for making that sabbatical possible.

References

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

[2] L. Altenberg, "Emergent phenomena in genetic programming," in *Evolutionary Programming — Proceedings of the Third Annual Conference* (A. V. Sebald and L. J. Fogel, eds.), pp. 233–241, World Scientific Publishing, 1994.

[3] U.-M. O'Reilly and F. Oppacher, "The troubling aspects of a building block hypothesis for genetic programming," in *Foundations of Genetic Algorithms 3* (L. D. Whitley and M. D. Vose, eds.), (Estes Park, Colorado, USA), pp. 73–88, Morgan Kaufmann, 31 July–2 Aug. 1994 1995.

[4] P. A. Whigham, "A schema theorem for context-free grammars," in *1995 IEEE Conference on Evolutionary Computation*, vol. 1, (Perth, Australia), pp. 178–181, IEEE Press, 29 Nov. - 1 Dec. 1995.

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

[6] R. Poli and W. B. Langdon, "A new schema theory for genetic programming with one-point crossover and point mutation," in *Genetic Programming 1997: Proceedings of the Second Annual Conference* (J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, eds.), (Stanford University, CA, USA), pp. 278–285, Morgan Kaufmann, 13-16 July 1997.

[7] R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," *Evolutionary Computation*, vol. 6, no. 3, pp. 231–252, 1998.

[8] R. Poli, "Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover," *Genetic Programming and Evolvable Machines*, vol. 2, no. 2, 2001. Forthcoming.

[9] R. Poli, "Hyperschema theory for GP with one-point crossover, building blocks, and some new results in GA theory," in *Genetic Programming, Proceedings of EuroGP 2000* (R. Poli, W. Banzhaf, and *et al.*, eds.), Springer-Verlag, 15-16 Apr. 2000.

[10] R. Poli, "Exact schema theorem and effective fitness for GP with one-point crossover," in *Proceedings of the Genetic and Evolutionary Computation Conference* (D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, eds.), (Las Vegas), pp. 469–476, Morgan Kaufmann, July 2000.

[11] R. Poli, "General schema theory for genetic programming with subtree-swapping crossover," in *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, (Milan), Springer-Verlag, 18-20 Apr. 2001.

[12] R. Poli and N. F. McPhee, "Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size," in *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, (Milan), Springer-Verlag, 18-20 Apr. 2001.

[13] N. F. McPhee and R. Poli, "A schema theory analysis of the evolution of size in genetic programming with linear representations," in *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, (Milan), Springer-Verlag, 18-20 Apr. 2001.

[14] R. Poli and N. F. McPhee, "Exact GP schema theory for headless chicken crossover and subtree mutation," in *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, (Seoul, Korea), May 2001.

[15] N. F. McPhee, R. Poli, and J. E. Rowe, "A schema theory analysis of mutation size biases in genetic programming with linear representations," in *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, (Seoul, Korea), May 2001.

[16] R. Poli and W. B. Langdon, "On the search properties of different crossover operators in genetic programming," in *Genetic Programming 1998: Proceedings of the Third Annual Conference* (J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, eds.), (University of Wisconsin, Madison, Wisconsin, USA), pp. 293–301, Morgan Kaufmann, 22-25 July 1998.

[17] R. Poli, W. B. Langdon, and U.-M. O'Reilly, "Analysis of schema variance and short term extinction likelihoods," in *Genetic Programming 1998: Proceedings of the Third Annual Conference* (J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, eds.), (University of Wisconsin, Madison, Wisconsin, USA), pp. 284–292, Morgan Kaufmann, 22-25 July 1998.

[18] J. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, USA: University of Michigan Press, 1975.

[19] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 1989.

[20] W. B. Langdon, "Size fair and homologous tree genetic programming crossovers," *Genetic Programming And Evolvable Machines*, vol. 1, pp. 95–119, Apr. 2000.

[21] C. R. Stephens and H. Waelbroeck, "Effective degrees of freedom in genetic algorithms and the block hypothesis," in *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)* (T. Bäck, ed.), (East Lansing), pp. 34–40, Morgan Kaufmann, 1997.

[22] C. R. Stephens and H. Waelbroeck, "Schemata evolution and building blocks," *Evolutionary Computation*, vol. 7, no. 2, pp. 109–124, 1999.

[23] D. J. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[24] P. D'haeseleer, "Context preserving crossover in genetic programming," in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, vol. 1, (Orlando, Florida, USA), pp. 256–261, IEEE Press, 27-29 June 1994.

[25] H. Geiringer, "On the probability theory of linkage in Mendelian heredity," *Annals of Mathematical Statistics*, vol. 15, pp. 25–57, March 1944.

[26] L. B. Booker, "Recombination distributions for genetic algorithms," in *FOGA-92, Foundations of Genetic Algorithms*, (Vail, Colorado), 24–29 July 1992. Email: booker@mitre.org.

[27] L. Altenberg, "The Schema Theorem and Price's Theorem," in *Foundations of Genetic Algorithms 3* (L. D. Whitley and M. D. Vose, eds.), (Estes Park, Colorado, USA), pp. 23–49, Morgan Kaufmann, 31 July–2 Aug. 1994 1995.

[28] W. M. Spears, "Limiting distributions for mutation and recombination," in *Proceedings of the Foundations of Genetic Algorithms Workshop (FOGA 6)* (W. M. Spears and W. Martin, eds.), (Charlottesville, VA, USA), July 2000. In press.

[29] D. Whitley, "A genetic algorithm tutorial," Tech. Rep. CS-93-103, Department of Computer Science, Colorado State University, Aug. 1993.

[30] R. Poli, J. E. Rowe, and N. F. McPhee, "Markov chain models for GP and variable-length GAs with homologous crossover," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, (San Francisco, California, USA), Morgan Kaufmann, 7-11 July 2001.

Markov Chain Models for GP and Variable-length GAs with Homologous Crossover

Riccardo Poli
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
R.Poli@cs.bham.ac.uk

Jonathan E. Rowe
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
J.E.Rowe@cs.bham.ac.uk

Nicholas Freitag McPhee
Division of Science and Mathematics
University of Minnesota, Morris
Morris, MN, USA
mcphee@mrs.umn.edu

Abstract

In this paper we present a Markov chain model for GP and variable-length GAs with homologous crossover: a set of GP operators where the offspring are created preserving the position of the genetic material taken from the parents. We obtain this result by using the core of Vose's model for GAs in conjunction with a specialisation of recent GP schema theory for such operators. The model is then specialised for the case of GP operating on 0/1 trees: a tree-like generalisation of the concept of binary string. For these symmetries exist that can be exploited to obtain further simplifications. In the absence of mutation, the theory presented here generalises Vose's GA model to GP and variable-length GAs.

1 Introduction

After a strong initial interest in schemata [1, 2], the interest of GA theorists has shifted in the last decade towards microscopic Markov chain models, such as Vose's model, possibly with aggregated states [3, 4, 5, 6, 7, 8, 9, 10, 11].

In the last year or so the theory of schemata has made considerable progress, both for GAs and GP. This includes several new schema theorems which give exact formulations (rather than the lower bounds previously presented in the literature [12, 13, 14, 15, 16, 17, 18]) for the expected number of instances of a schema at the next generation. These exact theories model GP with one-point crossover [19, 20, 21], standard and other subtree-swapping crossovers [22, 23, 24], homologous crossover [25], and different types of subtree mutation and headless chicken crossover [26, 27]. While considerable progress has been made in GP schema theory, no Markov chain model for GP and variable-length GAs has ever been proposed.

In this paper we start filling this theoretical gap and present a Vose-like Markov-chain model for genetic programming with homologous crossover [25]: a set of operators, including GP one-point crossover [16] and GP uniform crossover [28], where the offspring are created preserving the position of the genetic material taken from the parents. We obtain this result by using the core of Vose's theory in conjunction with a specialisation of the schema theory for such operators. This formally links GP schema theory and Markov chain models, two worlds believed by many people to be quite separate.

The paper is organised as follows. Given the complexity of the GP mechanics, exact GP schema theories, such as the exact schema theory for homologous crossover in [25], tend to be relatively complicated. Similarly, Vose's model for GAs [3] presents significant complexities. In the following section, we will summarise these theories providing as much detail as reasonable, occasionally referring to [3] and [25] for more details. Then, in Section 3 we present the extensions to both theories which allow the construction of a Markov chain model for GP and variable-length GAs with homologous crossover. In Section 4 we indicate how the theory can be simplified thanks to symmetries which exist when we restrict ourselves to 0/1 trees: a tree-like generalisation of the concept of binary string. In Section 5 we give an example. Some conclusions are drawn in Section 6.

2 Background

2.1 Nix and Vose's Markov Chain Model of GAs

The description provided here is largely based on [3, 29] and [4]. See [30] for a gentler introduction to this topic.

Let Ω be the set of all possible strings of length l, i.e. $\Omega = \{0,1\}^l$. Let $r = |\Omega| = 2^l$ be the number of elements of such a space. Let P be a population represented as a multiset of elements from Ω, let $n = |P|$ be the population size, and let N be the number of possible populations;

in [3] it was shown that

$$N = \left(\begin{array}{c} n+r-1 \\ r-1 \end{array} \right).$$

Let Z be an $r \times N$ matrix whose columns represent the possible populations of size n. The ith column $\Phi_i = \langle z_{0,i}, \ldots, z_{r-1,i} \rangle^T$ of Z is the incidence vector for the ith population P_i. That is $z_{y,i}$ is the number of occurrences of string y in P_i (where y is unambiguously interpreted as an integer or as its binary representation depending on the context).

Once this state representation is available, one can model a GA with a Markov chain in which the N columns of Z represent the states of the model. The transition matrix for the model, Q, is an $N \times N$ matrix where the entry Q_{ij} represents the conditional probability that the next generation will be P_j assuming that the current generation is P_i.

In order to determine the values Q_{ij} let us assume that we know the probability $p_i(y)$ of producing individual y in the next generation given that the current generation is P_i. To produce population P_j we need to get exactly $z_{y,j}$ copies of string y for $y = 0, \ldots, r-1$. The probability of this joint event is given by a multinomial distribution with success probabilities $p_i(y)$ for $y = 0, \ldots, r-1$, so [31]

$$Q_{i,j} = \frac{n!}{z_{0,j}! z_{1,j}! \ldots z_{r-1,j}!} \prod_{y=0}^{r-1} (p_i(y))^{z_{y,j}}. \quad (1)$$

The calculations necessary to compute the probabilities $p_i(y)$ depend crucially on the representation and the operators chosen. In [4] results for various GA crossover operators were reported. As noted in [3], it is possible to decompose the calculations using ideas firstly introduced in [29] as follows.

Assuming that the current generation is P_i, we can write

$$p_i(y) = \sum_{m,n=0}^{r-1} s_{m,i} s_{n,i} r_{m,n}(y) \quad (2)$$

where $r_{m,n}(y)$ is the probability that crossing over strings m and n yields string y and $s_{x,i}$ is the probability of selecting x from P_i. Assuming fitness proportionate selection,

$$s_{x,i} = \frac{z_{x,i} f(x)}{\sum_{j=0}^{r-1} z_{j,i} f(j)}, \quad (3)$$

where $f(x)$ is the fitness of string x.

We can map these results into a more recent formulation of Vose's model [4] by making use of matrices and operators. We start by treating the fitness function as a vector f of components $f_k = f(k)$. Then, if x is the incidence vector representing a particular population, we define an operator $\mathcal{F}$, called the *selection scheme*,[1] which computes the selection probabilities $s_{x,i}$ for all the members of Ω. For proportional selection

$$\mathcal{F}(x) = \mathrm{diag}(f)x / f^T x.$$

Then we organise the probabilities $r_{m,n}(y)$ into r arrays M_y of size $r \times r$, called *mixing matrices*, the elements of which are $(M_y)_{m,n} = r_{m,n}(y)$. We finally define an operator $\mathcal{M}$, called the *mixing scheme*,

$$\mathcal{M}(x) = \langle x^T M_0 x, x^T M_1 x, \ldots, x^T M_{r-1} x \rangle$$

which returns a vector whose components are the expected proportion of individuals of each type assuming that individuals are selected from the population x randomly (with replacement) and crossed over.

Finally we introduce the operator $\mathcal{G} = \mathcal{M} \circ \mathcal{F}$, which provides a compact way of expressing the probabilities $p_i(y)$ since (for fitness proportionate selection)

$$p_i(y) = \{\mathcal{G}(\Phi_i)\}_y = \left\{ \mathcal{M} \left(\frac{\mathrm{diag}(f)\Phi_i}{f^T \Phi_i} \right) \right\}_y$$

where the notation $\{\cdot\}_y$ is used to represent the yth component of a vector. So, the entries of the transition matrix for the Markov chain model of a GA can concisely be written as

$$Q_{i,j} = n! \prod_{y=0}^{r-1} \frac{(\{\mathcal{G}(\Phi_i)\}_y)^{z_{y,j}}}{z_{y,j}!}. \quad (4)$$

In [29, 3, 4] it is shown how, for fixed-length binary GAs, the operator $\mathcal{M}$ can be calculated as a function of the mixing matrix M_0 only. This is done by using a set of permutation operators which permute the components of any generic vector $x \in \mathbb{R}^r$:

$$\sigma_j \langle x_0, \ldots, x_{r-1} \rangle^T = \langle x_{j \oplus 0}, \ldots, x_{j \oplus r-1} \rangle^T, \quad (5)$$

where $\oplus$ is a bitwise XOR.[2] Then one can write

$$\mathcal{M}(x) = \langle (\sigma_0 x)^T M_0 \sigma_0 x, \ldots, (\sigma_{r-1} x)^T M_0 \sigma_{r-1} x \rangle^T. \quad (6)$$

2.2 Exact GP Schema Theory for Homologous Crossover

In [25] the following exact schema theorem for GP with homologous crossover was reported:

[1] In this paper we have chosen to use the symbol $\mathcal{F}$ to represent both the selection scheme of a GA and the function set used in GP, since this is the standard notation for both. This produces no ambiguity since the selection scheme is not used outside this section, and the function set is not referred to inside it.

[2] The operators σ_j can also be interpreted as permutation matrices.

$$\alpha(H,t) = (1-p_{xo})p(H,t) + p_{xo}\sum_j\sum_k\sum_{l\in\chi_{C(G_j,G_k)}} p_l^{C(G_j,G_k)} p(\Gamma(H,l)\cap G_j,t)p(\Gamma(H,\bar{l})\cap G_k,t) \quad (7)$$

where

- H is a GP schema, i.e. a tree composed of functions from the set $\mathcal{F}\cup\{=\}$ and terminals from the set $\mathcal{T}\cup\{=\}$, where $\mathcal{F}$ and $\mathcal{T}$ are the function and terminals sets used in our GP system and the primitive $=$ is a "don't care" symbol which stands for a *single* terminal or function.
- $\alpha(H,t)$ is the probability that a newly created individual matches the schema H.
- p_{xo} is the crossover probability.
- $p(H,t)$ is the selection probability of the schema H.
- G_1, G_2, $\cdots$ are all the possible program shapes, i.e. all the possible schemata containing = signs only.
- $C(G_j,G_k)$ is the common region between programs of shape G_j and programs of shape G_k. The common region between two generic trees h_1 and h_2 is the set

  $$C(h_1,h_2) = \{(d,i)|\mathcal{C}(d,i,h_1,h_2)\},$$

  where (d,i) is a pair of coordinates in a Cartesian node reference system (see [22, 25] for more details on the reference system used). The predicate $\mathcal{C}(d,i,h_1,h_2)$ is true if $(d,i) = (0,0)$. It also true if $A(d-1,i',h_1) = A(d-1,i',h_2) \neq 0$ and $\mathcal{C}(d-1,i',h_1,h_2)$ is true, where $A(d,i,h)$ returns the arity of the node at coordinates (d,i) in h, $i' = \lfloor i/a_{\max}\rfloor$ and $\lfloor\cdot\rfloor$ is the integer-part function. The predicate is false otherwise.
- For any given common region c we can define a set of *GP crossover masks*, χ_c, which contains all different trees with the same size and shape as the common region which can be built with nodes labelled 0 and 1.
- The *GP recombination distribution* p_l^c gives the probability that, for a given common region c, crossover mask l will be chosen from the set χ_c.
- A *GP hyperschema* is a rooted tree composed of internal nodes from $\mathcal{F}\cup\{=\}$ and leaves from $\mathcal{T}\cup\{=,\#\}$. Again, = is a "don't care" symbols which stands for exactly one node, while # stands for any valid subtree.
- $\Gamma(H,l)$ is defined to be the empty set if l contains any node not in H. Otherwise it is the hyperschema obtained by replacing certain nodes in H with either = or # nodes:
  - If a node in H corresponds to (i.e., has the same coordinates as) a non-leaf node in l that is labelled with a 0, then that node in H is replaced with a =.
  - If a node in H corresponds to a leaf node in l that is labelled with a 0, then it is replaced with a #.
  - All other nodes in H are left unchanged.
- $\bar{l}$ is the complement of the GP crossover mask l. The complement of a mask is a tree with the same structure but with the 0's and 1's swapped.

3 Markov Chain Model for GP

In order to extend Vose's model to GP and variable-length GAs with homologous crossover we define Ω to be an indexed set of all possible trees of maximum depth ℓ that can be constructed with a given function set $\mathcal{F}$ and a given terminal set $\mathcal{T}$. Assuming that the initialisation algorithm selects programs in Ω, GP with homologous crossover cannot produce programs outside Ω, and Ω is therefore a finite search space. Again, $r = |\Omega|$ is the number of elements in the search space; this time, however, r is not 2^l. All other quantities defined in Section 2.1 can be redefined by simply replacing the word "string" with the word "program", provided that the elements of Ω are indexed appropriately. With these extensions, all the equations in that section are also valid for GP, except Equations 5 and 6.

These are all minor changes. A major change is instead required to compute the probabilities $p_i(y)$ of generating the yth program in Ω when the population is P_i. Fortunately, these probabilities can be computed by applying the schema theory developed in [25] and summarised in Section 2.2. Since schema equations are applicable to schemata as well as to individual programs, it is clear that:

$$p_i(y) = \alpha(y,t) \quad (8)$$

where α is calculated for population P_i. This can be done by specialising Equation 7. Doing this allows one to instantiate the transition matrix for the model using Equation 1. However, it is possible to express $p_i(y)$ in terms of more primitive quantities as follows.

Let us specialise Equation 7 for the yth program in Ω:

$$p_i(y) = (1-p_{xo})p(y,t) + p_{xo}\sum_j\sum_k\sum_{l\in\chi_{C(G_j,G_k)}} p_l^{C(G_j,G_k)} \times p(\Gamma(y,l)\cap G_j,t)p(\Gamma(y,\bar{l})\cap G_k,t)$$

$$
\begin{aligned}
= \quad & (1-p_{xo}) \sum_{h_1 \in \Omega} \delta(h_1 = y) p(h_1, t) \times \underbrace{\sum_{h_2 \in \Omega} p(h_2, t)}_{=1} \\
& + p_{xo} \sum_j \sum_k \sum_{l \in \chi_{(G_j, G_k)}} p_l^{C(G_j, G_k)} \times \\
& \sum_{h_1 \in \Omega} p(h_1, t) \delta(h_1 \in \Gamma(y, l)) \delta(h_1 \in G_j) \times \\
& \sum_{h_2 \in \Omega} p(h_2, t) \delta(h_2 \in \Gamma(y, \bar{l})) \delta(h_2 \in G_k) \\
= \quad & \sum_{h_1, h_2 \in \Omega} p(h_1, t) p(h_2, t) \times \\
& \Big[(1-p_{xo}) \delta(h_1 = y) + p_{xo} \sum_{l \in \chi_{C(h_1, h_2)}} p_l^{C(h_1, h_2)} \times \\
& \delta(h_1 \in \Gamma(y, l)) \delta(h_2 \in \Gamma(y, \bar{l})) \Big],
\end{aligned}
$$

where we used the fact that $\sum_w \delta(x \in G_w) = 1$.

Assuming the current population is P_i, we have that $p(h,t) = s_h(t)$. So, the last equation can be rewritten in the same form as Equation 2 provided we set

$$
\begin{aligned}
r_{m,n}(y) = \Big[& (1-p_{xo}) \delta(m = y) + \\
& p_{xo} \sum_{l \in \chi_{C(m,n)}} p_l^{C(m,n)} \delta(m \in \Gamma(y, l)) \delta(n \in \Gamma(y, \bar{l})) \Big].
\end{aligned} \tag{9}
$$

Note that this equation could have been obtained by direct calculation, rather than through the specialisation of a schema theorem. However, this would still have required the definition and use of the hyperschema-returning function Γ and of the concepts of GP crossover masks and GP recombination distributions. Also, notice that the set of GP crossover masks also include masks containing all ones. These correspond to cloning the first parent. Therefore, by suitable readjustement of the probabilities $p_l^{C(m,n)}$, we can rewrite Equation 9 as

$$
r_{m,n}(y) = \sum_{l \in \chi_{C(m,n)}} p_l^{C(m,n)} \delta(m \in \Gamma(y, l)) \delta(n \in \Gamma(y, \bar{l})). \tag{10}
$$

This formula is analogous to the case of crossover defined by masks for fixed-length binary strings [4].

4 Mixing Matrices for 0/1 Trees

As has already been stated in Section 2.1, for the case of fixed-length binary strings, the mixing operator $\mathcal{M}$ can be written in terms of a single mixing matrix M_0 and a group of permutation matrices. This works because the permutation matrices are a representation of a group that acts transitively on the search space. This group action describes the symmetries that are inherent in the definition of crossover for fixed-length strings [4]. This idea can be generalised to other finite search spaces (see [32] for the detailed theory). However, in the case of GP, where the search space is a set of trees (up to some depth), the amount of symmetry is more limited and does not seem to give rise to a single mixing matrix.

In this section we will look at what symmetry does exist and the simplifications of the mixing operator it produces when we restrict ourselves to the space of *0/1 trees*. These are trees constructed using primitives from a terminal set $\mathcal{T} = \{0_0, 1_0\}$ and from a function set $\mathcal{F} = \bigcup_{i \in \iota} \mathcal{F}_i$ where $\mathcal{F}_i = \{0_i, 1_i\}$, ι is a finite subset of $\mathbb{N}$, and the subscripts 0 and i represent the arity of a 0/1 primitive.[3] It should be noted that the semantics of the primitives in 0/1 trees is unimportant for the theory, and that 0/1 trees are a generalisation of the notion of binary strings.[4]

Let Ω be the set of 0/1 trees of depth at most ℓ (where a program containing only a terminal has depth 1). Let $L(\Omega)$ be the set of full trees of exactly depth ℓ obtained by using the primitive set $\mathcal{T} \cup \mathcal{F}_{i_m}$ where i_m is the maximum element in ι. We term *node-wise XOR* the operation which, given two trees a and b in $L(\Omega)$, returns the 0/1 tree whose nodes are labelled with the result of the addition (modulo 2) of the binary labels of the nodes in a and b having corresponding coordinates; this operator is denoted $a \oplus b$.

For example, if we represent 0/1 trees in prefix notation, (1 (1 0 1) (0 0 1)) $\oplus$ (0 (1 0 0) (0 1 1)) = (1 (0 0 1) (0 1 0)). $L(\Omega)$ is a group under node-wise XOR. Notice that the definition of $\oplus$ extends naturally to pairs of trees with identical size and shape.

For each tree $k \in \Omega$ we define a truncation function

$$
\pi_k : L(\Omega) \longrightarrow \Omega
$$

as follows. Given any tree $a \in L(\Omega)$ we match up the nodes in k with the nodes in a, recursively:

1. The root nodes are matched.
2. The children of a matched node in k are matched to children of the corresponding node in a from the left. Recall that each node in a has the maximum possible arity, and that a has the maximum possible depth. Note that the arity of nodes in a will be reduced (if necessary) to that of the matching nodes in k.

This procedure corresponds to matching by co-ordinates. The effect of the operator π_k on a tree $a \in L(\Omega)$ is to throw away all nodes that are not matched against nodes in

[3] Subscripts will be dropped whenever it is possible to infer the arity of a primitive from the context.

[4] The space of 0/1 trees obtained when $\mathcal{F} = \mathcal{F}_1$ is isomorphic to the space of binary strings of arbitrary length.

k. The remaining tree $\pi_k(a)$ will then be of the same size and shape as k.

For example, suppose the maximum depth is $\ell = 3$ and the maximum arity is also 3. Let $a \in L(\Omega)$ be the tree (1 (0 1 1 0) (1 0 1 1) (1 1 1 0)) and let $k =$ (0 (1 1 0) (0 1)). Then matching nodes and truncating a produces $\pi_k(a) =$ (1 (0 1 1) (1 0)).

The group $L(\Omega)$ acts on the elements of Ω as follows. Let $a \in L(\Omega)$ and $k \in \Omega$. Then define

$$a(k) = \pi_k(a) \oplus k$$

which means we apply addition modulo 2 on each matched pair of nodes. We have used the extended definition of $\oplus$ since $\pi_k(a)$ and k are guaranteed to have the same size and shape. In our previous example we would have $a(k) =$ (1 (1 0 1) (1 1)).

We can extend the definition of $\oplus$ further by setting

$$a \oplus k = a(k)$$

for any $k \in \Omega$ and $a \in L(\Omega)$. The effect of this is essentially a relabelling of the nodes of the tree k in accordance with the pattern of ones found in a.

For each $a \in L(\Omega)$ we define a corresponding $r \times r$ permutation matrix σ_a with

$$(\sigma_a)_{i,j} = \delta((a \oplus i) = j)$$

Lemma 1. *Let $m, n, y \in \Omega$ and let $a \in L(\Omega)$. Then for homologous crossover*

$$r_{m,n}(y) = r_{a\oplus m, a\oplus n}(a \oplus y)$$

Proof: Interpreting Equation 9 for 0/1 trees m, n and y, the following hold:

$$a \oplus m = a \oplus y \Longleftrightarrow m = y$$

$$C(a \oplus m, a \oplus n) = C(m, n)$$

$$(a \oplus m) \in \Gamma(a \oplus y, l) \Longleftrightarrow m \in \Gamma(y, l)$$

and the result follows. The third assertion follows from the fact that we are relabelling the nodes in tree m according to the pattern of ones in a, and we relabel the nodes in the hyperschema $\Gamma(y, l)$ according to exactly the same pattern. □

Let us consider the GP schema G consisting only of "=" nodes representing the shape of some of the programs in Ω. We denote with 0^G the element of Ω obtained by replacing the = nodes in G with 0 nodes.

Theorem 2. *On the space of 0/1 trees with depth at most ℓ homologous crossover gives rise to a mixing operator*

$$\mathcal{M}(x) = \langle x^T M_0 x, x^T M_1 x, \ldots \rangle$$

(where we are indexing vectors by the elements of Ω). Then for each fixed shape G of depth not bigger than ℓ there exists a mixing matrix

$$M = M_{0^G}$$

such that if $y \in \Omega$ is of shape G then

$$M_y = \sigma_a^T M \sigma_a$$

for some $a \in L(\Omega)$.

Proof: Let $y \in \Omega$ be of shape G as required. Construct a maximal full tree a of depth not bigger than ℓ by appending a sufficient number of 0 nodes to the tree y so that each internal node in a has i_m children.[5]

Now suppose $m, n \in \Omega$ are trees which cross together to form y with probability $r_{m,n}(y)$. Because crossover is assumed to be homologous, the set of the coordinates on the nodes in m must be a superset of the set of node coordinates of G. Likewise for n.

The m, nth component of $\sigma_a^T M \sigma_a$ is

$$\begin{aligned}
(\sigma_a^T M \sigma_a)_{m,n} &= \sum_v (\sigma_a^T M)_{m,v} (\sigma_a)_{v,n} \\
&= \sum_v \sum_w (\sigma_a)_{w,m} M_{w,v} (\sigma_a)_{v,n} \\
&= M_{a^{-1}\oplus m, a^{-1}\oplus n} \\
&= r_{a^{-1}\oplus m, a^{-1}\oplus n}(0^G) \\
&= r_{m,n}(a \oplus 0^G) \\
&= r_{m,n}(y \oplus 0^G) \\
&= r_{m,n}(y) \\
&= (M_y)_{m,n}
\end{aligned}$$

where we have used the lemma to show

$$r_{a^{-1}\oplus m, a^{-1}\oplus n}(0^G) = r_{m,n}(a \oplus 0^G)$$

and a^{-1} is the inverse of the group element a. For 0/1 trees $a^{-1} = a$ since $a \oplus a = 0^{G_m}$, where G_m is the schema representing the shape of the trees in $L(\Omega)$. □

5 A Linear Example

In this section we will demonstrate the application of this theory to an example. To keep the presentation of the calculations manageable in the space available this example must perforce be quite simple, but should still be sufficient to illustrate the key concepts.

For this example we will assume that the function set contains only unary functions, with the possible labels for both

[5] For example, if $\ell = 3$, $i_m = 3$, G is (= = (= = =)) and $y =$ (1 1 (1 1 1)), then $a =$ (1 (1 0 0 0) (1 1 1 0) (0 0 0 0)).

functions and terminals being 0 and 1 (i.e., $\mathcal{F} = \mathcal{F}_1 = \mathcal{T} = \{0, 1\}$). As a result we can think of our structures as being variable length binary strings. We will let $\ell = 2$ (i.e., we restrict ourselves to strings of length 1 or 2), which means that $r = 6$ and

$$\Omega = \{0, 1, 00, 01, 10, 11\}.$$

We will also limit ourselves here to the mixing matrices for GP one-point crossover and GP uniform crossover; we could however readily extend this to any other homologous crossover operator.

5.1 GP one-point crossover

The key to applying this theory is to compute $r_{m,n}(y)$ as described in Equation 9. In other words, for each $y \in \Omega$ we need to construct a matrix $M_y = r_{m,n}(y)$ that contains the probabilities that GP one-point crossover with parents m and n will yield y. Since $r = |\Omega| = 6$, this will yield six 6×6 matrices. In the (fixed-length) GA case it would only be necessary to specify one mixing matrix, since symmetries would allow us to derive the others through permutations of the indices. As indicated in the previous section, the symmetries in 0/1 trees case are more complex, and one can not reduce the situation down to just one case. In particular we find, as mentioned above, that the set of mixing matrices for our variable-length GA case splits into two different subsets, one for y of length 1, and one for y of length 2, and the necessary permutations are generated by the group $L(\Omega) = \{00, 01, 10, 11\}$.

To make this more concrete, let us consider M_0 and M_1, each of which has exactly one non-zero column:[6]

$$M_0 = \begin{array}{c|cccc} & 0 & 1 & 00 & \cdots \\ \hline 0 & 1 & 0 & 0 & \cdots \\ 1 & 1 & 0 & 0 & \cdots \\ 00 & 1/2 & 0 & 0 & \cdots \\ 01 & 1/2 & 0 & 0 & \cdots \\ 10 & 1/2 & 0 & 0 & \cdots \\ 11 & 1/2 & 0 & 0 & \cdots \end{array}$$

$$M_1 = \begin{array}{c|cccc} & 0 & 1 & 00 & \cdots \\ \hline 0 & 0 & 1 & 0 & \cdots \\ 1 & 0 & 1 & 0 & \cdots \\ 00 & 0 & 1/2 & 0 & \cdots \\ 01 & 0 & 1/2 & 0 & \cdots \\ 10 & 0 & 1/2 & 0 & \cdots \\ 11 & 0 & 1/2 & 0 & \cdots \end{array}$$

[6] Since these matrices are indexed by variable length binary strings instead of natural numbers, we have indicated the indices (0, 1, 00, 01, 10 and 11) along the top and left-hand side of each matrix. In M_0, for example, the value in position (1, 0) is 1 and (01, 0) is $1/2$.

Clearly M_1 is very similar to M_0. Indeed, Theorem 2 shows that M_1 can be obtained by applying a permutation matrix to M_0:

$$M_1 = \sigma_{10}^T M_0 \sigma_{10},$$

where

$$\sigma_{10}^T = \begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 00 & 0 & 0 & 0 & 0 & 1 & 0 \\ 01 & 0 & 0 & 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 1 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}.$$

The situation is more interesting for the mixing matrices for y of length 2:

$$M_{00} = \begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 00 & 0 & 0 & 1 & 0 & 1/2 & 0 \\ 01 & 0 & 0 & 1 & 0 & 1/2 & 0 \\ 10 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 11 & 0 & 0 & 1/2 & 0 & 0 & 0 \end{array}$$

$$M_{01} = \begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 00 & 0 & 0 & 0 & 1 & 0 & 1/2 \\ 01 & 0 & 0 & 0 & 1 & 0 & 1/2 \\ 10 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 11 & 0 & 0 & 0 & 1/2 & 0 & 0 \end{array}$$

$$M_{10} = \begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 00 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 01 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 10 & 0 & 0 & 1/2 & 0 & 1 & 0 \\ 11 & 0 & 0 & 1/2 & 0 & 1 & 0 \end{array}$$

$$M_{11} = \begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 00 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 01 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 10 & 0 & 0 & 0 & 1/2 & 0 & 1 \\ 11 & 0 & 0 & 0 & 1/2 & 0 & 1 \end{array}$$

Here again we can write these mixing matrices as permutations of M_{00}, i.e.,

$$M_s = \sigma_s^T M_{00} \sigma_s$$

for $s \in \{00, 01, 10, 11\}$. M_{01}, for example, can be written as

$$M_{01} = \sigma_{01}^T M_{00} \sigma_{01}$$

where σ_{01} is as above.

5.2 GP uniform crossover

Here will just show the mixing matrices M_0 and M_{00} since, as we have seen, the other four matrices can be readily obtained from these using the permutation matrices σ_s:

$$M_0 = \left[\begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 1 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 \\ 1 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 00 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 01 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 10 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 11 & 1/2 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$M_{00} = \left[\begin{array}{c|cccccc} & 0 & 1 & 00 & 01 & 10 & 11 \\ \hline 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 00 & 1/2 & 1/2 & 1 & 1/2 & 1/2 & 1/4 \\ 01 & 0 & 0 & 1/2 & 0 & 1/4 & 0 \\ 10 & 0 & 0 & 1/2 & 1/4 & 0 & 0 \\ 11 & 0 & 0 & 1/4 & 0 & 0 & 0 \end{array}\right]$$

Comparing these matrices to those obtained for one-point crossover one can see that these are symmetric, where those for one-point crossover were not, pointing out that uniform crossover is symmetric with respect to the parents, where one-point crossover is not. The matrices for uniform crossover also have considerably more non-zero entries than those for one-point crossover, highlighting the fact that uniform crossover provides more ways to construct any given string.

6 Conclusions

In this paper we have presented the first ever Markov chain model of GP and variable-length GAs. Obtaining this model has been possible thanks to very recent developments in the GP schema theory, which have given us exact formulas for computing the probability that reproduction and recombination will create any specific program in the search space. Our GP Markov chain model is then easily obtained by plugging this ingredient into a minor extension of Vose's model of GAs. This theoretical approach provides an excellent framework for studying the dynamics of evolutionary algorithms (in terms of transient and long-term behaviour). It also makes explicit the relationship between the local action of genetic operators on individuals and the global behaviour of the population.

The theory is applicable to GP and variable-length GAs with homologous crossover [25]: a set of operators where the offspring are created preserving the position of the genetic material taken from the parents. If one uses only unary functions and the population is initialised with programs having a fixed common length, a GP system using these operators is entirely equivalent to a GA acting on fixed-length strings. For this reason, in the absence of mutation, our GP Markov chain model is a proper generalisation Vose's model of GAs. This is an indication that perhaps in the future it will be possible to completely unify the theoretical models of GAs and GP.

In the paper we analysed in detail the case of 0/1 trees (which include variable length binary strings), where symmetries can be exploited to obtain further simplifications in the model. The similarity with Vose's GA model is very clear in this case.

This paper is only a first step. In future research we intend to analyse in more depth the general case of tree-like structures to try to identify symmetries in the mixing matrices similar to those found for 0/1 trees. Also, we intend to study the characteristics of the transition matrices for the GP model, to gain insights into the dynamics of GP.

Acknowledgements

The authors would like to thank the members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group at Birmingham, for useful discussions and comments. Nic would like to extend special thanks to The University of Birmingham School of Computer Science for graciously hosting him during his sabbatical, and various offices and individuals at the University of Minnesota, Morris, for making that sabbatical possible.

References

[1] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, USA, 1975.

[2] Nicholas J. Radcliffe, "Schema processing", in *Handbook of Evolutionary Computation*, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds., pp. B2.5–1–10. Oxford University Press, 1997.

[3] Allen E. Nix and Michael D. Vose, "Modeling genetic algorithms with Markov chains", *Annals of Mathematics and Artificial Intelligence*, vol. 5, pp. 79–88, 1992.

[4] Michael D. Vose, *The simple genetic algorithm: Foundations and theory*, MIT Press, Cambridge, MA, 1999.

[5] Thomas E. Davis and Jose C. Principe, "A Markov chain framework for the simple genetic algorithm", *Evolutionary Computation*, vol. 1, no. 3, pp. 269–288, 1993.

[6] Günter Rudolph, "Stochastic processes", in *Handbook of Evolutionary Computation*, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds., pp. B2.2–1–8. Oxford University Press, 1997.

[7] Günter Rudolph, "Genetic algorithms", in *Handbook of Evolutionary Computation*, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds., pp. B2.4–20–27. Oxford University Press, 1997.

[8] Günter Rudolph, "Convergence analysis of canonical genetic algorithm", *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 96–101, 1994.

[9] Günter Rudolph, "Models of stochastic convergence", in *Handbook of Evolutionary Computation*, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds., pp. B2.3–1–3. Oxford University Press, 1997.

[10] Jonathan E. Rowe, "Population fixed-points for functions of unitation", in *Foundations of Genetic Algorithms 5*, Wolfgang Banzhaf and Colin Reeves, Eds. 1999, pp. 69–84, Morgan Kaufmann.

[11] William M. Spears, "Aggregating models of evolutionary algorithms", in *Proceedings of the Congress on Evolutionary Computation*, Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, Eds., Mayflower Hotel, Washington D.C., USA, 6-9 July 1999, vol. 1, pp. 631–638, IEEE Press.

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

[13] Lee Altenberg, "Emergent phenomena in genetic programming", in *Evolutionary Programming — Proceedings of the Third Annual Conference*, A. V. Sebald and L. J. Fogel, Eds. 1994, pp. 233–241, World Scientific Publishing.

[14] Una-May O'Reilly and Franz Oppacher, "The troubling aspects of a building block hypothesis for genetic programming", in *Foundations of Genetic Algorithms 3*, L. Darrell Whitley and Michael D. Vose, Eds., Estes Park, Colorado, USA, 31 July–2 Aug. 1994 1995, pp. 73–88, Morgan Kaufmann.

[15] P. A. Whigham, "A schema theorem for context-free grammars", in *1995 IEEE Conference on Evolutionary Computation*, Perth, Australia, 29 Nov. - 1 Dec. 1995, vol. 1, pp. 178–181, IEEE Press.

[16] Riccardo Poli and W. B. Langdon, "A new schema theory for genetic programming with one-point crossover and point mutation", in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, Eds., Stanford University, CA, USA, 13-16 July 1997, pp. 278–285, Morgan Kaufmann.

[17] Justinian P. Rosca, "Analysis of complexity drift in genetic programming", in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, Eds., Stanford University, CA, USA, 13-16 July 1997, pp. 286–294, Morgan Kaufmann.

[18] Riccardo Poli and William B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation", *Evolutionary Computation*, vol. 6, no. 3, pp. 231–252, 1998.

[19] R. Poli, "Hyperschema theory for GP with one-point crossover, building blocks, and some new results in GA theory", in *Genetic Programming, Proceedings of EuroGP 2000*, Riccardo Poli, Wolfgang Banzhaf, and *et al.*, Eds. 15-16 Apr. 2000, Springer-Verlag.

[20] Riccardo Poli, "Exact schema theorem and effective fitness for GP with one-point crossover", in *Proceedings of the Genetic and Evolutionary Computation Conference*, D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, Eds., Las Vegas, July 2000, pp. 469–476, Morgan Kaufmann.

[21] Riccardo Poli, "Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover", *Genetic Programming and Evolvable Machines*, vol. 2, no. 2, 2001, Forthcoming.

[22] Riccardo Poli, "General schema theory for genetic programming with subtree-swapping crossover", in *Genetic Programming, Proceedings of EuroGP 2001*, Milan, 18-20 Apr. 2001, LNCS, Springer-Verlag.

[23] Riccardo Poli and Nicholas F. McPhee, "Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size", in *Genetic Programming, Proceedings of EuroGP 2001*, Milan, 18-20 Apr. 2001, LNCS, Springer-Verlag.

[24] Nicholas F. McPhee and Riccardo Poli, "A schema theory analysis of the evolution of size in genetic programming with linear representations", in *Genetic Programming, Proceedings of EuroGP 2001*, Milan, 18-20 Apr. 2001, LNCS, Springer-Verlag.

[25] Riccardo Poli and Nicholas F. McPhee, "Exact schema theory for GP and variable-length GAs with homologous crossover", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, 7-11 July 2001, Morgan Kaufmann.

[26] Riccardo Poli and Nicholas Freitag McPhee, "Exact GP schema theory for headless chicken crossover and subtree mutation", in *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul, Korea, May 2001.

[27] Nicholas F. McPhee, Riccardo Poli, and Jon E. Rowe, "A schema theory analysis of mutation size biases in genetic programming with linear representations", in *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul, Korea, May 2001.

[28] Riccardo Poli and William B. Langdon, "On the search properties of different crossover operators in genetic programming", in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, Eds., University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998, pp. 293–301, Morgan Kaufmann.

[29] Michael D. Vose and Gunar E. Liepins, "Punctuated equilibria in genetic search", *Complex Systems*, vol. 5, no. 1, pp. 31, 1991.

[30] Melanie Mitchell, *An introduction to genetic algorithms*, Cambridge MA: MIT Press, 1996.

[31] Murray R. Spiegel, *Probability and Statistics*, McGraw-Hill, New York, 1975.

[32] Jonathan E. Rowe, Michael D. Vose, and Alden H. Wright, "Group properties of crossover and mutation", Manuscript submitted for publication, 2001.

The Evaluation of a Stochastic Regular Motif Language for Protein Sequences

Brian J. Ross
Department of Computer Science
Brock University
St. Catharines, Ontario
Canada L2S 3A1
bross@cosc.brocku.ca

Abstract

A probabilistic regular motif language for protein sequences is evaluated. SRE-DNA is a stochastic regular expression language that combines characteristics of regular expressions and stochastic representations such as Hidden Markov Models. To evaluate its expressive merits, genetic programming is used to evolve SRE-DNA motifs for aligned sets of protein sequences. Different constrained grammatical forms of SRE-DNA expressions are applied to aligned protein sequences from the PROSITE database. Some sequences patterns were precisely determined, while others resulted in good solutions having considerably different features from the PROSITE equivalents. This research establishes the viability of SRE-DNA as a new representation language for protein sequence identification. The practicality of using grammatical genetic programming in stochastic biosequence expression classification is also demonstrated.

1 INTRODUCTION

The rate of biological sequence acquisition is accelerating considerably, and this data is freely accessible from biosequence databases such as PROSITE (Hofmann *et al.* 1999). Research in bioinformatics is investigating more effective technology for classifying and analysing this wealth of new data. One important problem in this regard is the automated discovery of sequence patterns (Brazma *et al.* 1998*a*). A sequence pattern, also known as a *motif* or *consensus pattern*, encodes the common characteristics of a set of biosequences. From one point of view, a sequence pattern is a signature identifying a set of related biosequences, and hence can be used as a means of database query. Alternatively, and perhaps more importantly, a motif can also characterizes the salient biological and evolutionary characteristics common to a family of sequences. The use of computational tools which automatically determine biologically meaningful patterns from sets of sequences is of obvious practical importance to the field.

The contributions of this research are two-fold. Firstly, the viability of SRE-DNA, a new motif language, is investigated. SRE-DNA shares characteristics of deterministic regular expressions and stochastic representations such as Hidden Markov Models (Krogh *et al.* 1994). Since full SRE-DNA is likely too unwieldy to be practical, this research investigates what restrictions to the language are practical for biosequence classification. To do this, genetic programming (GP) is used to evolve SRE-DNA motifs for aligned sequences. SRE-DNA's probabilistic basis can be exploited during fitness evaluation in GP evolution.

A second goal of this research is to test the practicality of logic grammar-based genetic programming in an application of bioinformatics. The system used is DCTG-GP, a logic grammar-based GP system based on definite clause translation grammars (DCTG) (Ross 2001*a*). With DCTG-GP, a variety of constrained grammatical variations of SRE-DNA are straight-forwardly defined and applied towards motif discovery.

Generally speaking, motif discovery for aligned sequences is a simpler problem than for unaligned sequences. With aligned sequences, the basic problem of determining the common subsequences amongst a set of sequences has been already determined. Nevertheless, a number of fundamental issues regarding the viability of SRE-DNA are more clearly addressable if aligned data is studied initially. In the course of these experiments, it was discovered that motif discovery for some families of aligned data is very challenging. This

justifies studying aligned sequences before commencing on unaligned data.

Section 2 gives an overview of biosequence identification, stochastic regular expressions and DCTG-GP. Section 3 discusses experiment design and preparation. Results are reported in Section 4. A discussion concludes the paper in section 5.

2 BACKGROUND

2.1 Biosequence Identification

DNA molecules are double-stranded sequences of the four base nucleic acids adenine (A), thymine (T), cytosine (C) and guanine (G) (Alberts *et al.* 1994). The A and T bases bond together, as do the C and G. Other molecular forces will cause the strand to bend and convolute, creating a 3-dimensional double-bonded structure essentially unique to the molecule, and critical to various organic functions. In terms of sequence characterization, one of the strands of bases is adequate for identification purposes, since the other strand of bonded base pairs is complementary. A complete molecule, or a portion of it denoting a particular structure of interest, is denoted by a sequence of A, T, C and G bases. A higher level of representation is often used, in which the 20 unique amino acids created from triples of nucleic acids are represented. This results in smaller sequences using a larger alphabet.

The representation and automatic identification of subsequences in organic molecules has attracted much research effort over the years, and has resulted in a number of practical applications. New sequences can be searched for instances of known subsequences ("aligned"), which can indicate organic properties of interest, and hence identify their genetic functionality. Families of sequences can be classified by their distinguishing common sequence patterns. Sequence patterns are natural interfaces for biosequence database access. Sequences are also conducive to mathematical and computational analyses, which makes them natural candidates for automated synthesis and search algorithms.

A variety of representation languages have been used for biosequence identification, including regular languages (Arikawa *et al.* 1993, Brazma *et al.* 1998*b*), context–free and other languages (Searls 1993, Searls 1995), and probabilistic representations (Krogh *et al.* 1994, Sakakibara *et al.* 1994, Karplus *et al.* 1997). Although languages higher in the Chomsky hierarchy are more discriminating than lower-level representations, they may be less efficiently parsed or synthesized than than lower–level languages. In many cases, simple languages such as regular languages are the most practical representation for biosequence identification and database access. The PROSITE database, for example, uses a constrained regular expression language.

Much work has been done on machine learning techniques for families of biosequences using regular languages as a representation language (Brazma *et al.* 1998*a*, Baldi and Brunak 1998). GP has been used successfully to evolve regular motifs for unaligned sequences (Hu 1998, Koza *et al.* 1999).

2.2 Stochastic Regular Expressions

Stochastic Regular Expressions (SRE) is a probabilistic regular expression language (Ross 2000). It is essentially a conventional regular expression language (Hopcroft and Ullman 1979), embellished with probability fields. It is similar to a stochastic regular language proposed by (Garg *et al.* 1996), where a number of mathematical properties of the language are proven.

Let E range over SRE, α range over atomic actions, n range over integers ($n \geq 1$), and p range over probabilities ($0 < p < 1$). SRE syntax is:

$$\begin{aligned} E \;::=\; & \alpha \mid E:E \mid E^{*p} \mid E^{+p} \\ & \mid E_1(n_1) + ... + E_k(n_k) \end{aligned}$$

The terms denote atomic actions, concatenation, iteration (Kleene closure and '+' iteration), and choice. Plus iteration, E^{+p}, is equivalent to $E : E^{*p}$. The probability fields work as follows. With choice, each term $E_i(n_i)$ is chosen with a probability equivalent to $n_i/\Sigma_j(n_j)$. With Kleene closure, each iteration of E occurs with a probability p, and the termination of E occurs with a probability $1-p$. Probabilities between terms propagate in an intuitive way. For example, with concatenation, the probability of $E:F$ is the probability of E multiplied by the probability of F. With choice, the aforementioned probability of a selected term is multiplied by the probability of its chosen expression E_i. Each iteration of Kleene iteration also includes the probability of the iterated expression E. The overall effect of this probability scheme is the definition of a probability distribution of the regular language denoted by an expression. Each string $s \in L(E)$ has an associated probability, while any $s \notin L(E)$ has a probability of 0. It can be shown that SRE defines a well-formed probability function (the sum of all the probabilities for all $s \in L(E)$ is 1).

An example SRE expression is $(a : b^{*0.7})(2) + c^{*0.1}(3)$. It recognizes string c with $Pr = 0.054$ (the term with c can be chosen with $Pr = \frac{3}{2+3} = 0.6$; then that term

iterates once with $Pr = 0.1$; finally the iteration terminates with $Pr = 1 - 0.1 = 0.9$, giving an overall probability of $0.6 \times 0.1 \times 0.9 = 0.054$). The string bb is not recognized; its probability is 0.

An SRE interpreter is implemented and available for GP fitness functions. To test whether a string s is a member of an SRE expression E, the interpreter attempts to consume s with E. If successful, a probability $p > 0$ is produced. Unsuccessful matches will result in probabilities of 0. The SRE-DNA interpreter only succeeds if an entire SRE-DNA expression is successfully interpreted. For example, in $E_1 : E_2$, if E_1 consumes part of a string, but E_2 does not, then the interpretation fails and yields a probability of 0.

As with conventional regular expressions (Hopcroft and Ullman 1979), string recognition for SRE expressions is of polynomial time complexity. Note, however, that the interpretation of regular expressions can be exponentially complex with respect to overall expression size. For example, in $((a+b)^*)^*$, even though the expression's language is equivalent to that for $(a+b)^*$, there is a combinatorial explosion in the number of ways the nested iterations can be interpreted with respect to one other: a string of size k can be interpreted 2^k different ways.

SRE-DNA, a variant of SRE, is used in this paper. A number of embellishments and constraints are used, which are practical for biosequence identification. Details are given in Section 3.1.

2.3 DCTG-GP

```
expr ::= guardedexpr^^A, expr^^B
<:>
(construct( E:F ) ::- A^^construct(E),
                      B^^construct(F)),
(recognize(S, S2, PrSoFar, Pr) ::-
          check_prob(PrSoFar),
          A^^recognize(S, S3, PrSoFar, Pr1),
          check_prob(Pr1),
          B^^recognize(S3, S2, Pr1, Pr)).
```

Figure 1: DCTG rule for SRE-DNA concatenation

DCTG-GP is a grammatical genetic programming system (Ross 2001*a*). It is inspired by other work in grammatical GP (Whigham 1995, Geyer-Shulz 1997, Ryan *et al.* 1998), and in particular, the LOGENPRO system (Wong and Leung 1997). Like LOGENPRO, DCTG-GP uses logical grammars for defining the target language for evolved programs. The logic grammar formalism used is the definite clause translation grammar (DCTG) (Abramson and Dahl 1989). A DCTG is a logical version of a context-free attribute grammar, and it permits the complete syntax and semantics of a language to be defined in one unified framework. DCTG-GP is implemented in Sicstus Prolog 3.8.5 (SICS 1995).

In a DCTG-GP application, the syntax and semantics of a target language are defined together. Each DCTG rule contains a syntax field and one or more semantic fields. The syntax field is the grammatical definition of a language component, while the semantic fields encode interpretation code, tests, and other language and problem specific constraints. The general form of a rule is:

$$H \ ::= \ B_1, \ B_2, \ ..., \ B_j$$
$$<:>$$
$$S \ ::- \ G_1, \ G_2, \ ..., \ G_k.$$

The rule labeled with nonterminal H is a grammar rule. Each term B_i is a reference to a terminal or nonterminal of the grammar. Embedded Prolog goals may also be listed among the B_i's. These grammar rules are used to denote programs in the population, which are in turn implemented as derivation trees. Hence DCTG-GP is a tree-based GP system. The rule labeled S is a semantic rule associated with nonterminal H. Its goals G_i may refer to semantic rules associated with the nonterminal references B_i, or calls to Prolog predicates.

Figure 1 shows the DCTG-GP rule for SRE-DNA's concatenation operator. The grammatical rule states that concatenation consists of a guarded expression followed by an expression. The `A` and `B` variables are used for referencing parts of the grammar tree for these nonterminals within the semantic rules. The first semantic rule `construct` builds a text form for the rule, for printing purposes. The ":" operator denotes concatenation. The second semantic rule `recognize` is used during SRE-DNA expression interpretation. The argument `S` is a string to be consumed, and `S2` is the remainder of the string after consumption. The value `PrSoFar` is the overall probability thus far in the interpretation, and `Pr` is the probability after this expression's interpretation is completed. The references to `recognize` in the semantic rule are recursive calls which permit the two terms in the concatenation to recognize portions of the string. Finally, `check_prob` determines if the current running probability is larger than the minimal required for interpretation to continue.

3 EXPERIMENT DETAILS

3.1 SRE-DNA Variations

1. $expr ::= guard \mid choice \mid guard{:}expr \mid expr^{*p} \mid expr^{+p}$
   $choice ::= guard(n) + guard(n) \mid guard(n) + choice$
   $guard ::= mask \mid mask{:}skip$
   $skip ::= x^{*p} \mid x^{+p}$

2. $expr ::= guard \mid guard{:}expr \mid expr^{+p}$
   $guard ::= mask \mid mask{:}skip$
   $skip ::= x^{+p}$

3. $expr ::= guard \mid guard{:}expr \mid expr{:}guard \mid expr^{*p} \mid expr^{+p}$
   $guard ::= mask \mid mask{:}skip$
   $skip ::= x^{*p} \mid x^{+p}$

4. $expr ::= guard \mid choice \mid expr{:}expr \mid expr^{*p} \mid expr^{+p}$
   $choice ::= guard(n) + guard(n) \mid guard(n) + choice$
   $guard ::= mask \mid mask{:}skip$
   $skip ::= x^{*p} \mid x^{+p}$

Figure 2: SRE-DNA Variations

A goal of this research is to explore how language constraints affect the quality of motif solutions. To this end, four different grammatical variations of SRE-DNA are defined in Figure 2. SRE-DNA embellishes SRE as follows. Firstly, masks are introduced. The mask $[\alpha_1 ... \alpha_k]$ denotes a choice of atoms α_i each with a probability $1/k$. This is equivalent to $\alpha_1(1)+...+\alpha_k(1)$ in SRE. Secondly, skip terms are defined. A skip term x^{*p} is a Kleene closure over the wild-card element x, which substitutes for any atom. The skip expression x^{+p} is equivalent to $x{:}x^{*p}$.

A summary of the SRE-DNA variants in Figure 2 is as follows. Grammar 1 uses constrained concatenation and choice expressions, in which *guards* are used. A guard is a term borrowed from concurrent programming, and specifies a constrained action. Guards promote efficient interpretation, because expressions are forced to consume string elements whenever a guard is encountered. It also reduces the appearance of iteration and choice in concatenation expressions, which helps reduce the scope of the target expressions. An intention for doing this is to try to make SRE-DNA have similar characteristics to conventional motif languages such as PROSITE's. In addition, the grammar prohibits nested iteration. This prevents some of the efficiency problems discussed in Section 2.2. Three minor variations of grammar 1 are used, each having different maximum iteration ranges ("i"): 1a (i=0.5); 1b (i=0.1); and 1c (i=0.2).

Grammar 2 is the closest to the PROSITE language. Choice is not used, and all skip and iterations use "+" iteration. It is also the only grammar that permits nested iteration. Grammar 3 is a minor relaxation of grammar 1, in which guards can be the first or second term in a concatenation. Nested iteration is prohibited. Finally, Grammar 4 is the least restrictive grammar, where concatenation uses general SRE-DNA expressions in both terms. Choice expressions still use guards, however, and nested iteration is prohibited.

It should be mentioned that a full version of SRE-DNA without guards or nested iteration constraints was initially attempted. Expression interpretation was very inefficient in that language, due to the preponderance of nested "*"–iterations, as well as iterations within choice and concatenation terms. The above constrained grammars are more efficient to interpret, and do not suffer any practical loss of expressiveness, at least with respect to the problem of motif recognition tackled here.

3.2 Fitness Evaluation

Fitness evaluation tests an expression's ability to recognize positive training examples, and to reject negative examples. Positive examples comprise a set of N aligned protein sequences. Negative examples are N randomly generated sequences, each having approximately the same length as the positive sequences.

Consider the formula:

$$Fitness = N + NegFit - PosFit$$

where *NegFit* and *PosFit* are the negative and positive training scores respectively. A fitness of 0 is the ideal "perfect" score. It is not attainable in practice, because the probabilities incorporated into *PosFit* are typically small.

Positive example scoring is calculated as:

$$PosFit = \sum_{e_i \in Pos} maximum(Fit(e_i'))$$

where Pos is the set of positive training examples, and e_i' is a suffix of example e_i (ie. $e_i = se_i', |s| \geq 0$). For each example in Pos, a positive test fitness Fit is found for all its suffixes, and the maximum of these values is used for the entire example. Fitness evaluation incorporates two distinct measurements: the probability of

recognizing an example, and the amount of the example recognized in terms of its length:

$$Fit(e) = \frac{1}{2}\left(Pr(s_{max}) + \frac{|s_{max}|}{|e|}\right)$$

Here, s_{max} is the longest recognized prefix of e, $|s_{max}|$ is its length, and $Pr(s_{max})$ is its probability of recognition. The first term accounts for the probability obtained when recognizing substring s_{max}, and the second term scores the size of the covered substring relative to the entire example. The fitness pressure obtained with *Fit* is to recognize an entire example string with a high probability. In early generations, the sequence cover term dominates the score, which forces fitness to favour expressions that recognize large portions of examples. The probability field comes into consideration as well, however, and is especially pertinent in later generations when expressions recognize a large proportion of the example set. At that time, the probability fitness measure favours expressions that yield high probabilities.

Negative fitness scoring is calculated as:

$$NegFit = maximum(Fit(n_i)) * N$$

where $n_i \in Neg$ (negative examples). The highest obtained fitness value for any recognized negative example suffix is used for the score. A discriminating expression will not normally recognize negative examples, however, and so $Fit(n_i) = 0$ for most n_i.

3.3 GP Parameters

Table 1 lists parameters used for GP runs. Although most parameters are self–explanatory, some require explanation. The initial population is oversampled, and culled at the beginning of a run. Reproduction may fail, for example, due to tree size limitations, and so a maximum of 3 reproduction attempts are undertaken before the reproduction is discarded. The terminals are a subset of amino acid codons, determined by the alphabet used in the positive training examples.

Crossover and mutation use the methods commonly applied by grammatical GP systems that denote programs with derivation trees. For example, when a subtree node of nonterminal type t is selected in one parent, then a similar node of type t will be selected in the other parent, and the two selected subtrees are swapped. Some SRE specific crossover and mutation operators are used. SRE crossover permits mask elements in two parents to be merged together. SRE mutation implements a number of numeric and mask mutations. The SRE mutation range parameter specifies that a numeric field is perturbed ±10% of its original value. Mask mutations include adding, removing, or changing a single item from a mask.

Table 1: GP Parameters

Parameter	Value
GA type	generational
Functions	SRE-DNA variants
Terminals	amino acid codons, integers, probabilities
Population size (initial)	2000
Population size (culled)	1000
Unique population	yes
Maximum generations	150
Maximum runs	10
Tournament size	7
Elite migration size	10
Retries for reproduction	3
Prob. crossover	0.90
Prob. mutation	0.10
Prob. internal crossover	0.90
Prob. terminal mutation	0.75
Prob. SRE crossover	0.25
Prob. SRE mutation	0.30
SRE mutation range	0.1
Max. depth initial popn.	12
Max. depth offspring	24
Min. grammar prob.	10^{-12}
Max. mask size	5

The minimum grammar probability value specifies the minimal probability used by the SRE evaluator before an expression interpretation is preempted. This improves the efficiency of expression evaluation by pruning interpretation paths with negligibly small probabilities.

4 RESULTS

The initial test case is the amino acid oxidase family of sequences. It is completely defined by a relatively small example set (8 unique sequences in the PROSITE database as of November, 2000). Table 2 shows the training results for the SRE-DNA grammars in Figure 2. (Having only 8 examples precluded the ability to perform testing on the results). ΣPr is the sum of recognized probabilities for all the positive examples. The best fitness and ΣPr fields are given for the top solution in the 10 runs for each case, while the average ΣPr is an average of all the solutions from the 10 runs. In the 60 solutions obtained in all these runs, only one expression was unable to recognize the entire

PROSITE ⇒	$[ilmv](2) : h : [ahn] : y : g : x : [ags](2) : x : g : x(5) : g : x : a$
Grammar	
1a	$[iglv] : x^{+.12} : h : x^{+.12} : y : (g : x^{+.45} : g : x^{+.47} : [ghqs] : x^{+.47} : (g : x^{+.47}(947) + [fghqs] : x^{+.14}(101) + [chmvy] : x^{+.14}(842)))^{+.12}$
1b	$[ilv] : x^{+.1} : h : x^{+.1} : y : g : x^{+.1} : g : x^{+.1} : [gq] : x^{+.1} : [ghis] : x^{+.1} : g : x^{+.1} : ([aqst](325) + [afsw] : x^{*.1}(210) + [fhnqs](223))^{*.1}$
1c	$[ilv] : x^{+.1} : h : x^{+.1} : y : x^{*.19} : g : x^{+.19} : (g : x^{+.19} : [gtq] : x^{+.19} : [ghs] : x^{+.1} : g : x+^{.1} : a)^{+.1}$
2	$[ilv] : x^{+.1} : h : x^{+.1} : y : x^{+.1} : [afh] : x^{+.1} : [gs] : x^{+.1} : g : x^{+.19} : [smqt] : x^{+.19} : [wy] : g : x^{+.1} : (a^{+.11})^{+.1}$
3	$[ilv] : x^{+.11} : h : x^{+.1} : y : g : x^{+.19} : [sg] : x^{+.19} : g : x^{+.1} : [aqst] : x^{+.1} : [ghs] : x^{+.13} : g : x^{+.1} : a$
4	$([ligv] : x^{+.14} : h : x^{+.11} : y : g : x^{+.17} : [gs] : x^{+.18} : g : x^{+.17})^{+.11} : [astqi] : x^{+.15} : ([hgs] : x^{+.11} : g : x^{+.19}(567) + ([ihswl] : x^{+.15} : ([hi] : x^{+.11})^{+.15} : ([ligv] : x^{+.11} : h : x^{+.11} : (y : g : x^{+.19})^{+.12} : [gs] : x^{+.17}(567) + (h : x^{+.14})^{+.15}(4)) : g : x^{+.17})^{+.11} : (((y : g : x^{+.19})^{+.12} : [gs] : x^{+.18} : g : x^{+.19})^{+.12} : [gs] : x^{+.17}(567) + g : x^{+.1})^{+.15}(4)) : g : x^{+.17} : g : x^{+.17}(4))$

Figure 3: Best solutions for various grammars: amino acid oxidase

Table 3: Solution statistics for other families (grammar 2)

	Training			Testing (best soln)		
Family	Set size	Seq size	100% solns	Set size	True pos (%)	False neg (%)
a) Aspartic acid	44	12	10	452	100	0.2
b) Zinc finger, C2H2 type	29	23	9	678	93	1
c) Zinc finger, C3HC4 type	21	10	10	168	100	0
d) Sugar transport 1	18	18	0	190	88	1
e) Sugar transport 2	18	26	2	178	100	12
f) Snake toxin	18	21	10	127	51	0
g) Kazal inhibitor	20	23	10	125	93	0

Table 2: Solution statistics (training) for SRE-DNA variations: amino acid oxidase. Grammars 1a, 1b, and 1c use maximum iteration limits of 0.5, 0.1, and 0.2 respectively.

Grammar	Best Fitness	Best Σ Pr	Avg Σ Pr
1a	3.999611	0.00078	0.000140
1b	3.999977	0.00005	0.000009
1c	3.999044	0.00191	0.000356
2	3.998157	0.00369	0.000588
3	3.992940	0.01412	0.002502
4	3.999396	0.00121	0.000272

training set. Clearly, version 3 of SRE-DNA (unrestricted, but no choice operator) yielded the strongest solutions.

Figure 3 shows the best solutions obtained for the runs in Table 2, along with the PROSITE expression used to obtain the training set. Note that PROSITE motifs are typically made manually by scientists, and are error-prone. While similarities are often seen between the GP solutions and PROSITE expression, there are also differences in the way consensus patterns are handled between them. Note how E^{+p}, S^{*p}, and x^{*p} are nonexistent in the best overall solution (grammar 3). It seems to contradict conventional GP wisdom that this richer grammar containing these superfluous operators performs better than grammar 2, which omits these operators in the first place. One hypothesis for this is that the iterative terms in grammar 3 help conserve and transport useful genetic material from early generations, but disappears later.

The solution motif that least matches the others is the one from grammar 4 (unrestricted with choice). This expression suffers from bloat, in which intron material is attached to low-probability choice terms. Even though such intron material may not contribute to language membership, it definitely has a negative impact

a) Aspartic	$P: c:x:[dn]:x(4):[fy]:x:c:x:c$ $S: c:x^{+.19}:[dn]:x^{+.19}:[fy]:x^{+.1}:c:x^{+.1}:c$
b) Zinc C2H2	$P: c:x(2,4):c:x(3):[cfilmvwy]:x(8):h:x(3,5):h$ $S: c:x^{+.19}:c:x^{+.19}:[afkr]:x^{+.19}:[fhqrs]:x^{+.19}:[ahlrs]:x^{+.19}:[hlnt]:x^{+.19}$ $:[hikrv]:x^{+.19}$
c) Zinc C3HC4	$P: c:x:h:x:[filmvy]:c:x(2):c:[ailmvy]$ $S: c:x^{+.1}:h:x^{+.19}:c:x^{+.19}:c:x^{+.1}$
d) Sugar 1	$P: [agilmstv]:[afgilmsv]:x(2):[ailmsv]:[de]:x:[afilmvwy]:g:r$ $:[kr]:x(4,6):[agst]$ $S: [agilm]:x^{+.32}:[dilr]:x^{+.32}:g:r:x^{+.32}:[gilmv]:x^{+.32}$
e) Sugar 2	$P: [filmv]:x:g:[afilmv]:x(2):g:x(8):[fily]:x(2):[eq]:x(6):[kr]$ $S: [filmv]:x^{+.19}:(g:x^{+.48}:g:x^{+.48}:[fgily]:x^{+.48}:[ailtv]:(x)^{+.48})^{+.21}$
f) Snake	$P: g:c:x(1,3):c:p:x(8,10):c:c:x(2):[denp]$ $S: g:c:x^{+.12}:c:x^{+.49}:[gkrv]:x^{+.48}:[gl]:x^{+.48}:c:c:x^{+.12}:[kt]:x^{+.1}$
g) Kazal	$P: c:x(7):c:x(6):y:x(3):c:x(2,3):c$ $S: c:x^{+.39}:[cp]:x^{+.39}:[acdgs]:x^{+.39}:y:x^{+.11}:[nsy]:x^{+.1}:c:x^{+.38}:c^{+.11}$

Figure 4: PROSITE (P) and best solutions (S) for other families (grammar 2)

on the overall probability distribution of a motif.

The solutions generated from a single experiment can often vary considerably. Consider this alternate solution from the grammar 1c runs ($\Sigma Pr = 0.00007$):

$$[ilv]:[iv]:h:x^{+.1}:y:x^{+.19}:[ghs]:x^{+.19}:g \\ :x^{+.19}:[ghst]:x^{+.19}:g:x^{+.19}$$

Comparing it with the solution for 1c in Figure 3, it more precisely discriminates the beginning of the sequence.

Experiments using other families of sequences were undertaken using grammar 2. Training and testing results are shown in Table 3. The maximum iteration limit was changed for different families, in an attempt to address the relative range of skipping allowed in the corresponding PROSITE expressions. "100% solns" indicate the number of solutions from the 10 runs that recognize the entire set of training examples, "True pos" is the proportion of true positives (positive examples correctly identified from the testing set), and "False neg" is the proportion of the false negatives (negative examples falsely identified as being member sequences). The positive and negative testing sets are the same size.

The testing results suggest that nearly all of the experiments found acceptable solutions. One exception is the snake toxin case, whose positive testing results are poor. This is probably due to over-training on an inadequately small training set. The sugar transport examples (d and e) were also challenging. Experiment (d) yielded no expressions which completely recognized the entire training set. Considering the results of Table 2, better results might have arisen if grammar 3 had been used instead of grammar 2. Also note that a strong overall probability score does not necessarily directly correlate with a high testing score. This is because a motif might recognize a lower-proportion of true positives, but with high probabilities. A good solution will balance the probability distribution and positive example recognition.

The motif expressions for the best solutions in Table 3 are given in Figure 4. In the aspartic and zinc C3HC4 experiments (a, c), all the runs generated the identical expression. In the aspartic case, the solution is nearly a direct match to the PROSITE expression, except that SRE-DNA's probabilistic skipping is used. In the solution for experiment (c), evolution chose skip expressions instead of the PROSITE $[filmvy]$ and $[ailmvy]$ terms. The preference of skip terms instead of masks was not always the case, as is seen in other solutions in Figure 4.

An interesting characteristic of many of the evolved motifs using grammar 2 is that the + iteration operator usually evolved out of final expressions. In the 80 grammar 2 motifs evolved for all the protein families studied, only 28 motifs used the iteration operator. In three families (aspartic acid, zinc finger 2, and snake toxin), none of the solution motifs used iteration. When iteration arose, it was often highly nested, indicating that it was being used as intron code. Even though iteration is not an important operator for expressing these motifs, it does seem to be beneficial for evolution performance, as was seen earlier in Table 2.

Regular expressions are coarse representations of the 3D structure relevant to a protein's organic functionality. Nevertheless, it is interesting to consider whether

any of the evolved motifs have captured the essential biological feature of the given protein. In some cases, the important features were indeed found. For example, in the snake toxin example, the four c's evident in both the PROSITE and SRE-DNA motifs are involved in disulfide bonds. In the aspartic acid motif, the hydroxylation site at the d or n codon is correctly identified. In the sugar 1 example, part of a strong sub-motif "$g : r : [kr]$" in the PROSITE source is seen in the SRE-DNA motif (the "$g : r$" term was found).

5 CONCLUSION

This research establishes that SRE-DNA is a viable motif language for protein sequences. SRE-DNA expressions were successfully evolved using grammatical GP, as implemented with the DCTG-GP system. A number of families were tested, and acceptable results were usually obtained. Like other regular motif languages, SRE-DNA is most practical for small- to medium-sized sequences, since larger sequences require correspondingly large expressions that generate relatively miniscule probabilities. Variations of SRE-DNA were tried, and preliminary results show that the most successful variation is one with unrestricted non-nested iteration, guards, and no choice operator. The choice operator is definitely detrimental, as it increases the frequency of intron material. Although the iteration operator was not important in final solutions, using it enhances evolution performance. One hypothesis for this is that iteration acts as a transporter of genetic material in early generations. Further testing on more families of sequences should confirm these results.

The style of motifs obtained is highly dependent upon grammatical constraints. Besides the kinds of grammar restrictions tested in the experiments, such factors as minimum and maximum iteration limits and maximum mask sizes are also critical factors in the character of realized motifs. Mask usage can be increased by reducing the maximum skip iteration limit, thereby increasing the likelihood of more guarded terms, and hence masks. Increasing the maximum mask size, however, does not result in better solutions. Larger masks tend to generate less discriminating motifs (higher false negative rates), and also are less efficiently interpreted. If the maximum iteration limit is set too large, evolved expressions tend to take the form:

$$(unique\ \ prefix) : (x)^{+.9} : (unique\ \ suffix)$$

In other words, evolution tends to find an expression that has two discriminating components for the beginnings and ends of sequences, while it skips the majority of the sequence in between. By reducing the iteration limit, more interesting motifs are obtained.

Multiple runs often find varying solutions that identify different consensus patterns within sequences. It is worth considering whether there is some means by which different solutions might be reconciled or "merged" together. Of course, the best way to judge a consensus pattern is to allow a biologist to examine it, in order to determine whether the identified patterns are biologically meaningful. It is worth remembering that grammatical motifs are crude approximations of the *real* relevant biological factor - the 3D shape of the protein molecule.

One automatic optimization that is easily applied to evolved motifs is to simplify mask terms by removing extraneous elements. This has two effects. First, it increases the probability performance of expressions, because smaller masks have proportionally larger probabilities for selected elements. Secondly, smaller masks make expressions more discriminating. This is easy to see, since a mask of one element is the most discriminating, while a skip term is the least (it is akin to a mask of all elements).

Recently, SRE-DNA has been applied successfully in synthesizing motifs for unaligned sequences (Ross 2001*b*). The results in this paper have been indispensable for this new work, since it is now known which versions of SRE-DNA are apt to be most successful. The knowledge that the choice operator is impractical and should be ignored is very helpful.

This research is similar in spirit to that by Hu, in which PROSITE-style motifs were for unaligned protein sequences (Hu 1998). Hu used demes and local optimization during evolution, unlike this work, which used a single population and no local optimization. Hu also seeds the initial population with terms generated from the example proteins. (Koza *et al.* 1999) have used GP to evolve regular motifs for proteins. One solution performed better than the established motif created by experts. Their use of ADF's was advantageous for the proteins analyzed, given the many instances of repeated patterns.

Acknowledgments

Thanks to Bob McKay for suggesting a means for improving the performance of DCTG-GP, and to anonymous referees for their constructive advice. This work is supported though NSERC Operating Grant 138467-1998.

References

Abramson, H. and V. Dahl (1989). *Logic grammars*. Springer-Verlag.

Alberts, B., D. Bray, J. Lewis, M. Raff, K. Roberts and J.D. Watson (1994). *Molecular Biology of the Cell*. 3 ed.. Garland Publishing.

Arikawa, S., S. Miyano, A. Shinohara, S. Kuhara, Y. Mukouchi and T. Shinohara (1993). A Machine Discovery from Amino Acid Sequences by Decision Trees over Regular Patterns. *New Generation Computing* **11**, 361–375.

Baldi, P. and S. Brunak (1998). *Bioinformatics: the Machine Learning Approach*. MIT Press.

Brazma, A., I. Jonassen, I. Eidhammer and D. Gilbert (1998*a*). Approaches to the Automatic Discovery of Patterns in Biosequences. *Journal of Computational Biology* **5**(2), 279–305.

Brazma, A., I. Jonassen, J. Vilo and E. Ukkonen (1998*b*). Pattern Discovery in Biosequences. Springer Verlag. pp. 255–270. LNAI 1433.

Garg, V.K., R. Kumar and S.I Marcus (1996). Probabilistic Language Framework for Stochastic Discrete Event Systems. Technical Report 96-18. Institute for Systems Research, University of Maryland. http://www.isr.umc.edu/.

Geyer-Shulz, A. (1997). The Next 700 Programming Languages for Genetic Programming. In: *Proc. Genetic Programming 1997* (John R. Koza *et al*, Ed.). Morgan Kaufmann. Stanford University, CA, USA. pp. 128–136.

Hofmann, K., P. Bucher, L. Falquet and A. Bairoch (1999). The PROSITE database, its status in 1999. *Nucleic Acids Research* **27**(1), 215–219.

Hopcroft, J.E. and J.D. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley.

Hu, Y.-J. (1998). Biopattern Discovery by Genetic Programming. In: *Proceedings Genetic Programming 1998* (J.R. Koza *et al*, Ed.). Morgan Kaufmann. pp. 152–157.

Karplus, K., K. Sjolander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm and C. Sander (1997). Predicting protein structure using hidden Markov models. *Proteins: Structure, Function, and Genetics* pp. 134–139. supplement 1.

Koza, J.R., F.H. Bennett, D. Andre and M.A. Keane (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann.

Krogh, A., M. Brown, I.S. Mian, K. Sjolander and D. Haussler (1994). Hidden Markov Models in Computational Biology. *Journal of Molecular Biology* **235**, 1501–1531.

Ross, B.J. (2000). Probabilistic Pattern Matching and the Evolution of Stochastic Regular Expressions. *International Journal of Applied Intelligence* **13**(3), 285–300.

Ross, B.J. (2001*a*). Logic-based Genetic Programming with Definite Clause Translation Grammars. *New Generation Computing*. In press.

Ross, B.J. (2001*b*). The Evolution of Stochastic Regular Motifs for Protein Sequences. Submitted for publication.

Ryan, C., J.J. Collins and M. O'Neill (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. In: *Proc. First European Workshop in Genetic Programming (EuroGP-98)* (W. Banzhaf *et al.*, Ed.). Springer-Verlag. pp. 83–96.

Sakakibara, Y., M. Brown, R. Hughey, I.S. Mian, K. Sjolander, R.C. Underwood and D. Haussler (1994). Stochastic Context-Free Grammars for tRNA Modeling. *Nucleic Acids Research* **22**(23), 5112–5120.

Searls, D.B. (1993). The Computational Linguistics of Biological Sequences. In: *Artificial Intelligence and Molecular Biology* (L. Hunter, Ed.). pp. 47–120. AAAI Press.

Searls, D.B. (1995). String Variable Grammar: a Logic Grammar Formalism for the Biological Language of DNA. *Journal of Logic Programming*.

SICS (1995). *SICStus Prolog V.3 User's Manual*. http://www.sics.se/isl/sicstus.html.

Whigham, P.A. (1995). Grammatically-based Genetic Programming. In: *Proceedings Workshop on Genetic Programming: From Theory to Real-World Applications* (J.P. Rosca, Ed.). pp. 31–41.

Wong, M.L. and K.S. Leung (1997). Evolutionary Program Induction Directed by Logic Grammars. *Evolutionary Computation* **5**(2), 143–180.

Priorities in Multi-Objective Optimization for Genetic Programming

Frank Schmiedle **Nicole Drechsler** **Daniel Große** **Rolf Drechsler**
Chair of Computer Architecture (Prof. Bernd Becker)
Institute of Computer Science, University of Freiburg i.Br., Germany
e-mail: {schmiedl,ndrechsl,grosse,drechsle}@informatik.uni-freiburg.de

Abstract

A new technique for multi-objective optimization is presented that allows to include priorities. But in contrast to previous techniques they can be included very easily and do not require much user interaction. The new approach is studied from a theoretical and practical point of view. The main differences to existing methods, like relation dominate and favor, are discussed. An experimental study of applying priorities in heuristics learning based on Genetic Programming (GP) is described. The experiments confirm the advantages presented in comparison to several other techniques.

1 Introduction

When applying optimization techniques, it should be taken into account that many problems in real-world applications depend on several independent components. This is one of the reasons why several approaches to multi-objective optimization have been proposed in the past (see e.g. [13]). They mainly differ in the way the elements are compared and in the granularity of the ranking. One major drawback of most of these methods is that a lot of user interaction is required. (For a more detailed description of the different models and a discussion of the main advantages and disadvantages see Section 3).

With applications becoming more and more complex, the user often does not have enough information and insight to guide the tool. In [5], a new relation has been introduced that allows to rank elements with a finer granularity than [8], keeping the main advantages of the model. Experimental studies have shown that this model is superior to the "classical" approach of relation dominate [9]. Even though, originally developed for *Evolutionary Algorithms* (EAs), recently it has also been applied in the field of *Genetic Programming* (GP) [10]. One of the major drawbacks of the model of [5] is that the handling of priorities is not covered.

In this paper we present an extension of [5] that allows to work with priorities and at the same time keeps all the advantages of the original model. Experimental results in the field of GP-based heuristics learning for minimization of *Binary Decision Diagrams (BDDs)* [2] show that the approach obtains the best result in comparison to previously published methods.

In the next section we first briefly review the application of GP-based heuristics learning. Multi-objective optimization is discussed in detail in Section 3, where we put special emphasis on handling of priorities. In Section 4 the experimental results are described and discussed. Finally, the paper is summarized.

2 Basic Concepts

We assume that the reader is familiar with *GA* and *GP* concepts and refer to [4, 10] for details. A model for heuristics learning with *GAs* has been proposed in [7]. Known basic problem solving strategies are used as heuristics, and ordering and frequency of the strategies is optimized by evolution. Recently, a generalization to the *GP* domain has been reported [6]. The multi-objective optimization approach that is presented in this paper has been used during heuristics learning for BDD minimization and first experiments were given. Therefore, we give a brief review of *GP*-based heuristics learning and the resulting BDD minimization method to make the paper self-contained.

2.1 Heuristics Learning

For learning heuristics in order to find good solutions to an optimization problem, it is necessary that several (non-optimal) strategies solving the problem can be provided. Typically, for different classes of problem instances there are also different strategies that perform best. A strategy that behaves well on one problem class may return poor results when being applied to another problem class. Thus, it is promising to learn how to combine the strategies to heuristics that can be applied successfully to most or even all classes of problems.

The learning process by *GP* and for a better understanding, some fundamental terms are introduced by the following

Definition 1 *Given an optimization problem P and a non-empty set of different non-optimal strategies $B = \{b_1, \ldots, b_{max}\}$ to solve the problem. Then the elements of B are called* BOMs (Basic Optimization Modules).

Moreover, a heuristics *for P is an algorithm that generates a sequence of BOMs.*

During evolution, the strategies are combined to generate heuristics that are the individuals in the population. The fitness of an individual can be evaluated by application of the heuristics to a training set of problem instances. The target is to find heuristics that perform well according to some given optimization criteria. Additionally, a good generalization is important, i.e. a heuristics that returns good results for the training set examples should also performs well on unknown instances. Note that for this, the handling of the different criteria, i.e. the special multi-objective optimization approach, plays a critical role for the success of the evolutionary process.

2.2 BDD Minimization by Genetic Programming

Binary Decision Diagrams (BDDs) [2] are a state-of-the-art data structure often used in VLSI CAD for efficient representation and manipulation of Boolean functions. BDDs suffer from their size being strongly dependent on the variable ordering used. In fact, BDD sizes may vary from linear to exponential for different orderings. Optimization of variable orderings for BDDs is difficult, but nevertheless, successful strategies for BDD minimization that are based on dynamic variable ordering have been reported, see e.g. *sifting* [11].

For heuristics learning, the strategies sifting (SIFT), group sifting (GROUP), symmetric sifting (SYMM), window permutation of size 3 and 4, respectively, (WIN3, WIN4) are used as BOMs. For all these techniques there is an additional BOM that iterates the method until convergence is reached, and the "empty" operator NOOP completes the set of BOMs B.

The individuals of the *GP* approach for BDD minimization consist of trees with leaf nodes labeled with BOMs and inner operator nodes that belong to different types. During evaluation of the heuristics, the tree is traversed by a *depth-first-search*-based method in order to generate a flat sequence. The types of the inner nodes decide if

- both subtrees are evaluated subsequently (CONCAT),

- according to the truth value of a given condition either the left or the right son is considered (IF) or
- evaluation of the sons is iterated until a truncation condition is fulfilled (WHILE).

For recombination, two crossover operators are provided. While CAT concatenates the two parents, the more sophisticated MERGE does the same for subtrees of the parents and by that, bloating can be prevented. In addition to this, there are four mutation operators that exchange BOMs (BMUT), node types (CIMUT, CWMUT) and modify conditions of IF-nodes (IFMUT), respectively. A probability table determines the frequencies for using the different operators. (For more details see [6].)

3 Multi-Objective Optimization with Priorities

In this section, the multi-objective aspect for solving optimization problems is analyzed. Without loss of generality we consider only *minimization* problems.

For n optimization criteria, an objective vector $(c_1, \ldots, c_n) \in \mathbb{R}_+^n$ of values for these criteria completely characterizes a solution belonging to the search space Π. Thus, solutions can be identified with objective vectors and as a consequence, $\Pi \subset \mathbb{R}_+^n$.

In most cases some or all of the c_i's are mutually dependent, and often conflicts occur, i.e. an improvement in one objective c_i leads to a deterioration of c_j for some $j \neq i$. This must be taken into account during the optimization process. If priorities have to be considered, a good handling of multi-objective optimization becomes even more complex.

3.1 Previous Work

In the past, several techniques for ranking solutions according to multiple optimization criteria have been developed. Some approaches define a *fitness* function $f : \mathbb{R}_+^n \mapsto \mathbb{R}_+$ that maps solutions c to one scalar value $f(c)$. The most commonly used method is linear combination by WEIGHTED SUM*.

Values for the c_i's $(1 \leqslant i \leqslant n)$ are weighted by constant coefficients W_i, and $f(c)$ is given by

$$f(c) = \sum_{i=1}^{n} W_i \cdot c_i.$$

The fitness value is used for comparison with the fitness of other solutions. Obviously, criteria with large weights have more influence on the fitness than those with small coefficients.

There are other methods that compare solutions based on one of the relations which are introduced by

Definition 2 *Let* $c = (c_1, \ldots, c_n)$ *and* $d = (d_1, \ldots, d_n) \in \Pi$ *be two solutions. The relations* $\prec_d$ (dominate) *and* $\prec_f$ (favor) $\subset \Pi \times \Pi$ *are defined by*

$$\begin{aligned} c \prec_d d \quad &:\Leftrightarrow \quad \exists i : c_i < d_i \ \wedge \\ &\qquad\quad \forall i : c_i \leqslant d_i \quad (1 \leqslant i \leqslant n) \\ c \prec_f d \quad &:\Leftrightarrow \quad |\{c_i < d_i | 1 \leqslant i \leqslant n\}| > \\ &\qquad\quad |\{c_i > d_i | 1 \leqslant i \leqslant n\}| \end{aligned}$$

We say that c dominates d *if* $c \prec_d d$ *and* $c \prec_f d$ *means that* c *is* favored *to* d.

$\prec_d$ is a partial ordering on any solution set $S \subset \Pi$ and the set $P \subset S$ that contains all non-dominated solutions in S is called *pareto-set*. In [9], the DOMINATE approach that approximates pareto-sets has been proposed.

An interactive technique for multi-objective optimization that divides Π into three subsets containing solutions of different *satisfiability classes* has been reported in [8]. It was generalized to the use of a variable number of satisfiability classes in [5]. The classes can be represented by the strongly connected components in the relation graph of $\prec_f$ and hence they can be computed by known graph algorithms. By this, it becomes possible to classify

*This is also the name of the method.

solutions $c \in \Pi$. We refer to this technique (introduced in [5]) as PREFERRED in the following. If priorities have to be handled, lexicographic sorting is used instead of $\prec_f$. This method will be called LEXICOGRAPHIC in further sections.

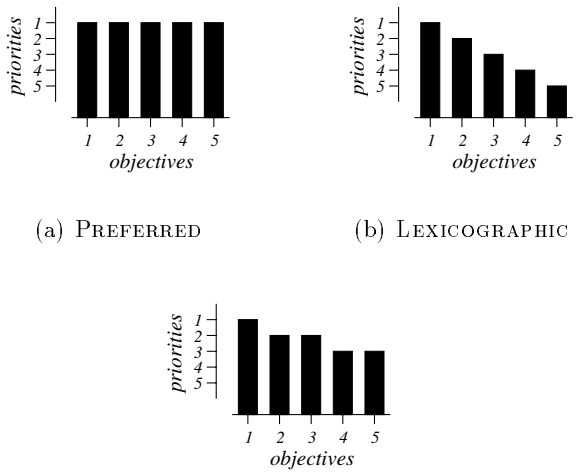
3.2 Drawbacks of Existing Approaches

The WEIGHTED SUM method is most popular for multi-objective optimization since it is easy to implement and allows to scale objectives. However, there are two major drawbacks:

1. Priorities cannot be handled directly but only by huge penalty weights. If there are many different priorities, the fitness function becomes very complex by that.
2. For adjusting the weights, problem specific knowledge is necessary. Usually, good settings for the weights are not known in advance and for finding and tuning them in experiments much effort has to be spent.

The approach proposed in [8] does not use weights that have to be adjusted, but it is interactive and therefore additional effort by the user is required, too. Moreover, the granularity of the method is very coarse since the solutions are divided in three different classes only. PREFERRED is a generalization of that technique that overcomes this drawback, i.e. an arbitrary number of satisfiability classes can be handled. By that, objectives with nearly the same importance can be optimized in parallel conveniently. However, priorities can not be considered by PREFERRED and in the approach presented in [5], LEXICOGRAPHIC is applied instead of PREFERRED if different priorities occur. By that, the following disadvantages are implied:

1. Instead of the relation $\prec_f$, the less powerful lexicographic sorting is applied for comparison of solutions and hence the results that can be expected are not as good as if $\prec_f$ was used.
2. Lexicographic sorting does not permit assigning the same priority to more than one optimization criteria. Thus if there are two objectives with a similar impact on the overall quality of solutions, one of them has to be preferred during LEXICOGRAPHIC in comparison to the other one.



(a) PREFERRED (b) LEXICOGRAPHIC

(c) PRIORITY

Figure 1: Priority schemes for different optimization methods.

Figure 1 (a) and (b) illustrate the priority handling of PREFERRED and LEXICOGRAPHIC, respectively. None of the existing methods can deal with priority schemes like described in Figure 1 (c), where the same priority is assigned to some objectives while some other criteria have lower or higher priorities. In the next section, an approach is presented that fulfills this requirement.

3.3 Multi-Objective Optimization with Priority Handling

The PRIORITY multi-objective optimization method introduced in this section combines properties of PREFERRED and LEXICOGRAPHIC. Thus it is more powerful and arbitrary priority numbers can be assigned to each objective. Without loss of generality

we assume that the priorities $1, 2, \ldots, m$ are used in non-descending order for the objectives $1, \ldots, n$[†].

Definition 3 *Given an optimization problem with search space $\Pi \subset \mathbb{R}^n_+$ and a priority vector $p = (p_1, \ldots, p_m) \in \mathbb{N}^m_+$ such that p_i determines for how many objectives the priority i occurs. According to this, the priority of an objective can be calculated by the function*

$$pr : \{1, \ldots, n\} \mapsto \{1, \ldots, m\},$$
$$pr(i) = k \quad where \sum_{j=1}^{k-1} p_j \leqslant i < \sum_{j=1}^{k} p_j$$

The projection *of $c \in \Pi$ on a priority i is given by*

$$c|_i \in \mathbb{R}^{p_i}, \quad c|_i = (c_l, \ldots, c_h)$$
$$where\ l = \sum_{j=1}^{i-1} p_j + 1 \ \wedge \ h = \sum_{j=1}^{i} p_j$$

Finally, for $c, d \in \Pi$ the relation $\prec_{pf} \subset \Pi \times \Pi$ (priority favor) *is defined by*

$$c \prec_{pf} d \ :\Leftrightarrow \ \exists j \in \{1, \ldots, m\} : c|_j \prec_f d|_j \ \wedge \ (\forall k < j : c|_k \not\prec_f d|_k \wedge d|_k \not\prec_f c|_k)$$

"c is priority-favored to d" also means $c \prec_{pf} d$.

The priority favor relation is used to compare solutions, but a complete ranking cannot be generated by $\prec_{pf}$ as can be seen in the following

Example 1 *For a problem with $n = 4$ optimization objectives and $m = 2$ different priorities, the search space and the priority vector are given as follows:*

$$\begin{aligned} \Pi_{ex} &= \{(2,8,8,8), (5,6,0,8), (5,7,5,1), \\ &\quad (2,6,0,8), (5,2,7,5), (2,7,8,9)\}, \\ p &= (1,3) \end{aligned}$$

[†]Otherwise the objectives have to be re-ordered.

The relation graph for $\prec_{pf}$ is illustrated in Figure 2. There are three solutions with value 2 and as well three solutions with value 5 for the objective with priority 1. Obviously the solutions with the lower value are priority-favoured to the other ones due to the value of objective 1 and regardless of the values of the other objectives. Among these priority-favored solutions, (2,6,0,8) is priority-favored to the others:

$$(6,0,8) \prec_f (8,8,8) \Rightarrow (2,6,0,8) \prec_{pf} (2,8,8,8)$$
$$(6,0,8) \prec_f (7,8,9) \Rightarrow (2,6,0,8) \prec_{pf} (2,7,8,9)$$

(2,8,8,8) and (2,7,8,9) can not be compared by $\prec_{pf}$ and for the remaining solutions, ranking is not possible since the graph for $\prec_{pf}$ contains a cycle.

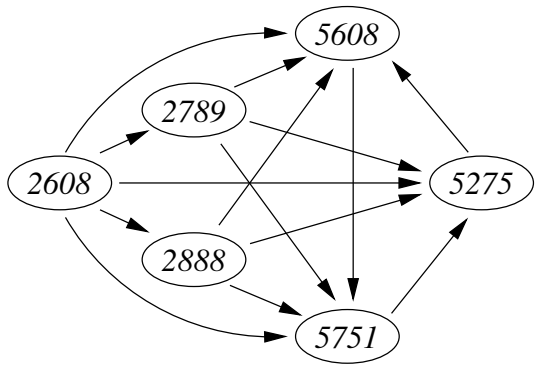


Figure 2: Relation graph $G = (\Pi_{ex}, \prec_{pf})$

The reason why cycles can occur is —as can easily be seen— that $\prec_{pf}$ is not transitive. This is not surprising since $\prec_{pf}$ is based on $\prec_f$ that is not transitive either [5]. To overcome this problem, analogously to the PREFERRED approach, solutions that belong to a cycle in the relation graph $G = (\Pi, \prec_{pf})$ are considered to be equal and merged to one single meta-node. This is done by generation of a meta-graph $G_{m,\Pi}$ by a linear time graph algorithm [3] that finds the set of strongly connected components SCC in G. We have

$$G_{m,\Pi} = (SCC, E) \ where$$
$$E = \{(q_1, q_2) \in \prec_{pf} | SCC(q_1) \neq SCC(q_2)\}$$

Since $G_{m,\Pi}$ by construction is free of cycles, there has to be at least one root node with

indegree 0 and by that, it is possible to rank the set of solutions according to

Definition 4 *Given a set of solutions* $\Pi \subset \mathbb{R}_+^n$*, the relation graph* $G = (\Pi, \prec_{pf})$*, the meta-graph* $G_{m,\Pi} = (SCC, E)$ *and the set of its root nodes* $G_0 = \{q | indeg(q) = 0\}$.

Then the satisfiability class *or* fitness $f(c)$ *of a solution* $c \in \Pi$ *can be determined by*

$$
\begin{aligned}
&f : \Pi \mapsto \mathbb{N}_+, \\
&f(c) = max\{r | \exists (q_1, \ldots, q_r) \in SCC^r : \\
&\qquad q_1 \in G_0 \ \wedge \ c \in q_r \ \wedge \\
&\qquad \forall 1 \leqslant i < r : (q_i, q_{i+1}) \in E\}
\end{aligned}
$$

The solutions $c \in \Pi$ can now be ranked according to their fitness that by Definition 4 is the increment of the length of the longest path in $G_{t,\Pi}$ from a root node to c . For computation of the ranking, well-known graph algorithms are used.

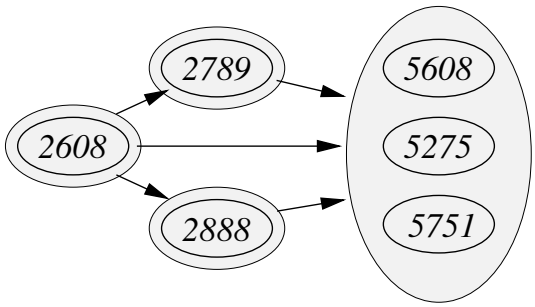


Figure 3: Meta-graph $G_{m,\Pi_{ex}}$

Example 2 *Consider again* Π_{ex} *from Example 1. Figure 3 includes the meta-graph* $G_{m,\Pi_{ex}}$ *with* $G_0 = \{(2,6,0,8)\}$ *with nodes representing SCCs in G. The fitness values for the solutions can easily be derived from* $G_{m,\Pi_{ex}}$*, e.g.* $f((2,8,8,8)) = 2$ *and* $f(5,6,0,8) = 3$.

4 Experimental Results

We implemented the PRIORITY multi-objective optimization approach described in Section 3 in the programming language C++ and embedded it in the software for BDD minimization by *GP*-based heuristics learning (see Section 2). In our experiments[‡], examples of Boolean functions that are taken from LGSYNTH91 [12] are used. The corresponding BDDs are minimized by WEIGHTED SUM as well as by relation based methods, i.e. the techniques PREFERRED, LEXICOGRAPHIC and PRIORITY. The objectives are the reduced BDD sizes for the single benchmarks. Notice that the discussion of the experiments in our approach can also be seen in the context of *design of experiments* (for more details see [1]).

For setting the weights in WEIGHTED SUM, several different approaches have been tried and we report two of them here. In EQUAL, weights are adjusted according to the initial BDD sizes of the benchmarks in a way that each example has the same impact on the fitness function. The idea behind this method is to favor the generalization ability of the generated heuristics to their optimized performance on the training set. In other words, by using EQUAL intuitively heuristics can be expected that perform better on unknown examples while slightly weaker results on the training set are tolerated.

For the technique REDR, the reduction rates that are obtained when applying the strategy SIFT to the single benchmarks are calculated. Weights are chosen indirectly proportional to the reduction rates, i.e. large weights are assigned to examples for which a large reduction is observed. Here, the intuition is that learning is focussed on benchmarks with a large potential for reduction in order to generate heuristics that exploit this potential well on unknown functions.

It is obvious that for weight setting, much effort has to be spent on experiments and computation of e.g. the reduction rates for the training set. In comparison to this, PREFERRED needs no preprocessing at all while for LEXICOGRAPHIC, only the objectives have

[‡]All experiments have been carried out on SUN ULTRA 1 workstations.

Table 1: Results for application on the training set

Name of circuit	I/O		Weighted Sum		Relation Based		
	in	out	Equal	RedR	Pref	Lexic	Prior
bc0	26	11	522	522	522	522	522
ex7	16	5	71	71	71	71	71
frg1	28	3	80	82	80	80	80
ibm	48	17	206	207	206	206	206
in2	19	10	233	233	233	233	233
s1196	32	32	597	597	597	597	597
ts10	22	16	145	145	145	145	145
x6dn	39	5	239	237	239	239	239
average	–	–	261.6	262.0	261.6	261.6	261.6

to be ordered (in our experiments according to initial BDD sizes). The same is done for Priority — the only difference is that the same priority is assigned to benchmarks with a similar initial BDD size.

For the *GP*, the same settings as in [6] are used. The population consists of 20 individuals and in each generation 10 offsprings are generated. The evolutionary process is terminated after 100 generations. For more details about the experimental setup like e.g. the method for generating the initial population, we refer to [6]. In the final population, one of the individuals with the best fitness value is chosen. The results for minimization of the training set examples are given in Table 1.

In the first three columns, the names and the input and output sizes, respectively, of the benchmarks are given. Columns 4 and 5 include the results for the Weighted Sum methods Equal and RedR while in the last three columns, final BDD sizes of the heuristics generated by the methods Lexicographic, Preferred and Priority are given. It can be seen that nearly all methods perform identically with respect to the behavior of the best individuals on the training set examples. Only The RedR approach slightly differs for three benchmarks.

The situation changes when the heuristics are applied to unknown benchmarks. The results are given in Table 2.

Except for *chkn* where RedR performs slightly better, Priority achieves the best results for all benchmarks. It clearly outperforms the other relation based methods as well as Equal on average while being still slightly better than RedR. As a result, it can be seen that setting weights for a fitness function by intuition is not always successful. Although the ideas for both approaches Equal and RedR sound sensible, only the latter achieves good results. Thus many experiments have to be conducted for tuning weights if Weighted Sum is used while this is not needed when applying Priority.

5 Conclusions

A new technique for handling priorities in multi-objective optimization has been presented. Application in GP-based heuristics learning has clearly demonstrated that the new approach outperforms existing methods, while at the same time the user interaction is reduced.

It is focus of current work to further study the relation between GA-based and GP-based heuristics learning using multi-objective optimization techniques.

Table 2: Application to new benchmarks

Name of circuit	Weighted Sum		Relation Based		
	Equal	RedR	Pref	Lexic	Prior
apex2	601	349	601	601	320
apex7	291	288	291	291	288
bcd	568	573	568	568	568
chkn	266	261	266	264	264
cps	975	975	975	975	970
in7	76	78	76	76	76
pdc	793	792	793	792	792
s1494	386	386	386	386	386
t1	112	112	112	113	112
vg2	79	79	79	79	79
average	414.7	389.3	414.7	414.5	385.5

References

[1] F. Brglez and R. Drechsler. Design of experiments in CAD: Context and new data sets for ISCAS'99. In *Int'l Symp. Circ. and Systems*, pages VI:424–VI:427, 1999.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] T.H. Cormen, C.E. Leierson, and R.C. Rivest. *Introduction to Algorithms.* MIT Press, McGraw-Hill Book Company, 1990.

[4] L. Davis. *Handbook of Genetic Algorithms.* van Nostrand Reinhold, New York, 1991.

[5] N. Drechsler, R. Drechsler, and B. Becker. A new model for multi-objective optimization in evolutionary algorithms, LNCS 1625. In *Int'l Conference on Computational Intelligence (Fuzzy Days)*, pages 108–117, 1999.

[6] N. Drechsler, F. Schmiedle, D. Große, and R. Drechsler. Heuristic learning based on genetic programming. In *EuroGP*, 2001.

[7] R. Drechsler and B. Becker. Learning heuristics by genetic algorithms. In *ASP Design Automation Conf.*, pages 349–352, 1995.

[8] H. Esbensen and E.S. Kuh. EXPLORER: an interactive floorplaner for design space exploration. In *European Design Automation Conf.*, pages 356–361, 1996.

[9] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addision-Wesley Publisher Company, Inc., 1989.

[10] J. Koza. *Genetic Programming - On the Programming of Computers by means of Natural Selection.* MIT Press, 1992.

[11] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[12] S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1/95, Microelectronic Center of North Carolina, 1991.

[13] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Trans. on Evolutionary Comp.*, 3(4):257–271, 1999.

Automated Discovery of Numerical Approximation Formulae Via Genetic Programming

Matthew Streeter

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Lee A. Becker

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Abstract

This paper describes the use of genetic programming to automate the discovery of numerical approximation formulae. The authors present results involving rediscovery of known approximations for Harmonic numbers and discovery of rational polynomial approximations for functions of one or more variables, the latter of which are compared to Padé approximations obtained through a symbolic mathematics package. For functions of a single variable, it is shown that evolved solutions can be considered superior to Padé approximations, which represent a powerful technique from numerical analysis, given certain tradeoffs between approximation cost and accuracy, while for functions of more than one variable, we are able to evolve rational polynomial approximations where no Padé approximation can be computed. Further, it is shown that evolved approximations can be refined through the evolution of approximations to their error function. Based on these results, we consider genetic programming to be a powerful and effective technique for the automated discovery of numerical approximation formulae.

1 INTRODUCTION

1.1 MOTIVATIONS

Numerical approximation formulae are useful in two primary areas: firstly, approximation formulae are used in industrial applications in a wide variety of domains to reduce the amount of time required to compute a function to a certain degree of accuracy (Burden and Faires 1997), and secondly, approximations are used to facilitate the simplification and transformation of expressions in formal mathematics. The discovery of approximations used for the latter purpose generally requires human intuition and insight, while approximations used for the former purpose tend to be polynomials or rational polynomials obtained by a technique from numerical analysis such as Padé approximants (Baker 1975; Bender and Orszag 1978) or Taylor series. Genetic programming (Koza 1992) provides a unified approach to the discovery of approximation formulae which, in addition to having the obvious benefit of automation, provides a power and flexibility that potentially allows for the evolution of approximations superior to those obtained using existing techniques from numerical analysis.

1.2 EVALUATING APPROXIMATIONS

In formal mathematics, the utility or value of a particular approximation formula is difficult to analytically define, and depends perhaps on its syntactic simplicity, as well as the commonality or importance of the function it approximates. In industrial applications, in contrast, the value of an approximation is uniquely a function of the computational cost involved in calculating the approximation and the approximation's associated error. In the context of a specific domain, one can imagine a utility function which assigns value to an approximation based on its error and cost. We define a reasonable utility function to be one which always assigns lower (better) scores to an approximation a_1 which is *unequivocally superior* to an approximation a_2, where a_1 is defined to be unequivocally superior to a_2 iff. neither its cost nor error is greater than that of a_2, and at least one of these two quantities is lower than the corresponding quantity of a_2. Given a set of approximations for a given function (obtained through any number of approximation techniques), one is potentially interested in any approximation which is not *unequivocally inferior* (defined in the natural way) to any other approximation in the set. In the terminology of multi-objective optimization, this subset is referred to as a Pareto front (Goldberg 1989). Thus, the Pareto front contains the set of approximations which could be considered to be the most valuable under some reasonable utility function.

1.3 RELATED WORK

The problem of function approximation is closely related to the problem of function identification or symbolic regression, which has been extensively studied by

numerous sources including (Koza 1992; Andre and Koza 1996; Chellapilla 1997; Luke and Spector 1997; Nordin 1997; Ryan, Collins, and O'Neill 1998). Approximation of specific functions has been performed by Keane, Koza, and Rice (1993), who use genetic programming to find an approximation to the impulse response function for a linear time-invariant system, and by Blickle and Thiele (1995), who derive three analytic approximation formulae for functions concerning performance of various selection schemes in genetic programming. Regarding general techniques for the approximation of arbitrary functions, Moustafa, De Jong, and Wegman (1999) use a genetic algorithm to evolve locations of mesh points for Lagrange interpolating polynomials.

2 EVOLVING NUMERICAL APPROXIMATION FORMULAE USING GENETIC PROGRAMMING

All experiments reported in this paper make use of the standard genetic programming paradigm as described by Koza (1992). Our task is to take a function in symbolic form (presented to the system as a set of training points) and return a (possibly singleton) set of expressions in symbolic form which approximate the function to various degrees of accuracy. The authors see two essential methods of applying genetic programming to this task: either by limiting the available function set in such a way that the search space contains only approximations to the target function, rather than exact solutions, or by in some way incorporating the computational cost of an expression into the fitness function, so that the evolutionary process is guided toward simpler expressions which presumably will only be able to approximate the data. Only the former approach is considered here.

The system used for the experiments described in this paper was designed to be functionally equivalent to that described by Koza (1992) with a few minor modifications. Firstly, the evolution of approximation formulae requires the cost of each approximation to be computed. We accomplish this by assigning a raw cost to each function in the function set, and taking the cost of an approximation to be the sum of the functional costs for each function node in its expression tree whose set of descendent nodes contains at least one input variable. For all experiments reported in this paper, the function costs were somewhat arbitrarily set to 1 for the functions `/`, `*`, and `RCP` (the reciprocal function), 0.1 for the functions `+` and `-`, and 10 for any more complex function such as `EXP`, `COS`, or `RLOG`.

Secondly, this system uses a slightly modified version of the standard adjusted fitness formula 1/(1+[error]) which attempts to maintain selection pressure when error values are small. We note that although an approximation which attains an error of 0.1 is twice as accurate as one with an error of 0.2, the standard formula will assign it an adjusted fitness which is just over 9% greater. We attempt to avoid this problem by introducing an *error multiplier*, so that the adjusted fitness formula becomes 1/(1+[error multiplier][error]). For one experiment described in this paper, the error multiplier was set to 1000. In the given example, this causes the approximation with an accuracy of 0.1 to have a fitness which is nearly twice (~1.99 times) that of the approximation whose accuracy is 0.2, which is more appropriate.

Finally, rather than simply reporting the best (i.e. most accurate) approximation evolved in each of a number of runs, we report the Pareto front for the union of the population histories of each independent run, computed iteratively and updated at every generation. Thus, this system returns the subset of approximations which are potentially best (under some reasonable utility function) from the set of all approximations evolved in the course of all independent runs.

The integrity of the system used in these experiments, which was written by the authors in C++, was verified by reproducing the experiment for symbolic regression of $f(x) = x\text{^}4 + x\text{^}3 + x\text{^}2 + x$ as reported by Koza (1992).

3 REDISCOVERY OF HARMONIC NUMBER APPROXIMATIONS

One commonly used quantity in mathematics is the Harmonic number, defined as:

$$H_n \equiv \sum_{i=1}^{n} 1/i$$

This series can be approximated using the asymptotic expansion (Gonnet 1984):

$$H_n = \gamma + \ln(n) + 1/(2n) - 1/(12n^2) + 1/(120n^4) - \ldots$$

where γ is Euler's constant ($\gamma \approx 0.57722$).

Using the system described in the previous section, and the function set {`+`, `*`, `RCP`, `RLOG`, `SQRT`, `COS`}, the authors attempted to rediscover some of the terms of this asymptotic expansion. Here `RLOG` is the protected logarithm function (which returns 0 for a non-positive argument) and `RCP` is a protected reciprocal function which returns the reciprocal of its argument if the argument is non-zero, or 0 otherwise. `SQRT` and `COS` are included as extraneous functions.

All parameter settings used in this experiment are the same as those presented as the defaults in John Koza's first genetic programming book (Koza 1992), including a population size of 500 and generation limit of 51. The first 50 Harmonic numbers (i.e. H_n for 1<=n<=50) were used as training data. 50 independent runs were executed, producing a single set of candidate approximations. Error was calculated as the sum of absolute error for each training instance. The error multiplier set to 1 for this experiment (e.g. effectively not used).

The set of evolved approximations returned by the genetic programming system (which represent the Pareto front for

the population histories of all independent runs) is given in Table 1. For the purpose of analysis, each approximation was simplified using the Maple symbolic mathematics package; for the sake of brevity, only the simplified expressions (rather than the full LISP expressions) are given in this table.[1]

Table 1. Evolved Harmonic Number Approximations.

SIMPLIFIED EXPRESSION			
ERROR	**COST**	**RUN**	**GENERATION**
1. ln(*x*)+.5766598187+1/(sqrt(ln(*x*)+.5766598187 +1/(1/*x*+2**x*+.6426220121)+*x*^2)+*x*)			
0.0215204	39.1	22	32
2. ln(*x*)+.5766598187+1/(2**x*+1/(1.219281831 + ln(1/(ln(*x*)+.5766598187))+*x*))			
0.0229032	35.8	22	35
3. ln(*x*)+.5766598187+1/(2**x*+1/(1.285244024 + ln(1.734124639+2**x*)))			
0.0264468	26.9	22	37
4. ln(*x*)+.5766598187+1/(2**x*+1/(2.584025920 + ln(*x*)+1/(3.007188263+*x*)))			
0.0278816	25.9	22	49
5. ln(*x*)+.5766598187+1/(2**x*+1/(.5766598187 + 1/*x*+*x*))			
0.0286254	15.7	22	36
6. ln(*x*)+.5766598187+1/(2**x*+.3592711879)			
0.0293595	13.4	22	37
7. ln(*x*)+.5766598187+1/(2**x*+.3497550998)			
0.0297425	11.4	22	42
8. ln(*x*+.5022291180)+.5779513609			
0.0546846	10.3	40	28
9. ln(*x*+.4890238595)+.5779513609			
0.0653603	10.2	40	21
10. 0.5965804779+ln(*x*)			
1.44089	10.1	49	49
11. 3.953265289-4.348430001/*x*			
20.2786	2.2	3	1
12. 3.815981083			
31.0297	0	10	4

[1] Note that since the cost and error values given in Table 1 were calculated by the genetic programming system (using the unsimplified versions of the approximations), the cost values are not necessarily the same as those which would be obtained by manually evaluating the simplified Maple expressions.

An analysis of this set of candidate solutions follows. For comparison, Table 2 presents the error values associated with the asymptotic expansion when carried to between 1 and 4 terms.

Table 2. Accuracy of Asymptotic Expansion

TERMS	EXPRESSION	ERROR
1	0.57722	150.559
2	0.57722 + ln(*n*)	2.12094
3	0.57722 + ln(*n*) + 1/(2*n*)	0.128663
4	0.57722 + ln(*n*) + 1/(2*n*) - 1/(12*n*^2)	0.00683926

Candidate approximation 12, the cheapest approximation in the set, is simply a constant, while candidate approximation 11 is a simple rational polynomial. Candidate approximation 10 represents a variation on the first two terms of the asymptotic expansion, with a slightly perturbed version of Euler's constant which gives greater accuracy on the 50 supplied training instances. Candidate solutions 8 and 9 represent slightly more costly variations on the first two terms of the asymptotic expansion which provide increased accuracy over the training data. Similarly, candidate solutions 6 and 7 are slight variations on the first three terms of the asymptotic expansion, tweaked as it were to give greater accuracy on the 50 training points. Candidate solutions 2-5 can be regarded as more complicated variations on the first three terms of the asymptotic expansion, each giving a slight increase in accuracy at the cost of a slightly more complex computation. Candidate solution 1 represents a unique and unexpected approximation which has the greatest accuracy of all evolved approximations, though it is unequivocally inferior to the first four terms of the asymptotic expansion has presented in Table 2.

Candidate approximations 1-7 all make use of the constant 0.5766598187 as an approximation to Euler's constant, which was evolved using the LISP expression:

```
(RCP(SQRT(* 4.67956 RLOG(1.90146))))
```

This approximation is accurate to two decimal places. Candidate approximations 8 and 9 make use of the slightly less accurate approximation of 0.5779513609, evolved using the LISP expression:

```
(COS(LN 2.59758))
```

Note that in this experiment, pure error-driven evolution has produced a rich set of candidate approximations exhibiting various trade-offs between accuracy and cost. Also note that with the exception of the first candidate approximation, which uses the `SQRT` function, the `SQRT` and `COS` functions were used only in the creation of constants, so that these extraneous functions did not provide a significant obstacle to the evolution of the desired approximations. Thus, this experiment represents

a partial rediscovery of the first three terms of the asymptotic expansion for H_n.

4 DISCOVERY OF RATIONAL POLYNOMIAL APPROXIMATIONS FOR KNOWN FUNCTIONS

4.1 INTRODUCTION

By limiting the set of available functions to the arithmetic function set {*, +, /, -}, it is possible to evolve rational polynomial approximations to functions, where a rational polynomial is defined as the ratio of two polynomial expressions. Since approximations evolved with the specified function set use only arithmetic operators, they can easily be converted to rational polynomial form by hand, or by using a symbolic mathematics package such as Maple. Approximations evolved in this manner can be compared to approximations obtained through other techniques such as Padé approximations by comparing their Pareto fronts. In the section, we present the results of such a comparison for three common mathematical functions: the natural logarithm ln(x), the square root sqrt(x), and the hyperbolic arcsine arcsinh(x), approximated over the intervals [1,100], [0,100], and [0,100], respectively. The functions were selected to be common, aperiodic functions whose calculation was sufficiently complex to warrant the use of approximation. The intervals were chosen to be relatively large due to the fact that Padé approximations are weaker over larger intervals, and we wished to construct examples for which the genetic technique might be most applicable.

4.2 COMPARISON WITH PADÉ APPROXIMATIONS

The Padé approximation technique is parameterized by the value about which the approximation is centered, the degree of the numerator in the rational polynomial approximation, and the degree of the denominator. Using the Maple symbolic mathematics package, we calculated all Padé approximations whose numerator and denominator had a degree of 20 or less, determined their associated error and cost, and calculated their (collective) Pareto front for each of the three functions being approximated. The center of approximation was taken as the leftmost point on the interval for all functions except the square root, whose center was taken as x=1 since the necessary derivatives of sqrt(x) are not defined for x=0. Error was calculated using a Riemann integral with 1000 points. For simplicity, the cost of Padé approximations was taken only as the minimum number of multiplications/divisions required to compute the rational polynomial, as calculated by a separate Maple procedure.

The Maple procedure written to compute the cost of an approximation operated by first putting the approximation in continued-fraction form (known to minimize the number of necessary multiplications/divisions), counting the number of multiplications/divisions required to compute the approximation in this form, and then subtracting for redundant multiplications. As an example of a redundant multiplication, the function $f(x)=x^2+x^3$ when computed literally requires 3 multiplications (1 for x^2, 2 for x^3), but need be computed using only 2, since in the course of computing x^3 one naturally computes x^2.

For consistency, the candidate approximations evolved through the genetic programming technique were also evaluated (subsequent to evolution) using the Reimann integral and Maple cost procedure, and the Pareto front for this set of approximations was recomputed using the new cost and error values. Finally, it should be noted that a Padé approximation with denominator of degree zero is identical to the Taylor series whose degree is that of the numerator, so that the Pareto fronts reported here effectively represent the potentially best (under some reasonable utility function) members of a set of 20 Taylor series and 380 uniquely Padé approximations.

4.3 RESULTS

All experiments involving rational polynomial approximations were performed using the same settings as described in the previous section, but with a generation limit of 101 (we have found that accurate rational polynomial approximations take a while to evolve). The / function listed in the function set was defined to be a protected division operator which returns the value 10^6 if division by zero is attempted. In analyzing evolved approximations via Maple, any approximation which performed division by zero was discarded. To reduce the execution time of these experiments, we employed the technique suggested as a possible optimization by Koza (1990) of using only a subset of the available training instances to evaluate individuals at each generation. In our experiments, the subset is chosen at random for the initial generation, and selected as the subset of examples on which the previous best-of-generation individual performed the worst for all subsequent generations. The subset is assigned a fixed size for all generations; for all experiments reported in this section, the subset size was 25. Training data consisted of 100 points, uniformly spaced over the interval of approximation. Each of the three experiments reported was completed in approximately 4-5 hours on a 600 MHz Pentium III system.

Figures 1-3 present the Pareto fronts for Padé approximations and for genetically evolved approximations of the functions ln(x), sqrt(x), and arcsinh(x), respectively, evaluated over the intervals [1,100], [0,100], and [0,100], respectively. In each of these three figures, the dashed line connects points corresponding to Padé approximations, while the solid line connects points corresponding to genetically evolved approximations. All Padé approximations not accounted for in computing the Pareto front represented by the dashed line (i.e. all Padé approximation whose numerator or denominator has a degree larger than 20) must involve

at least 20 multiplications/divisions, if only to compute the various powers of x: x, x^2, x^3, . . . x^{21}. For this reason, a dashed horizontal line at *cost*=20 is drawn in each figure, so that the horizontal line, combined with the dashed lines representing the Pareto front for Padé approximations with numerator and denominator of degree at most 20, represents the best case Pareto front for all Padé approximations of any degree.

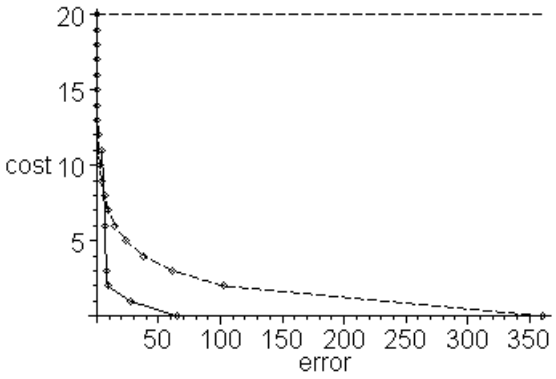


Figure 1: Pareto Fronts for Approximations of ln(x).

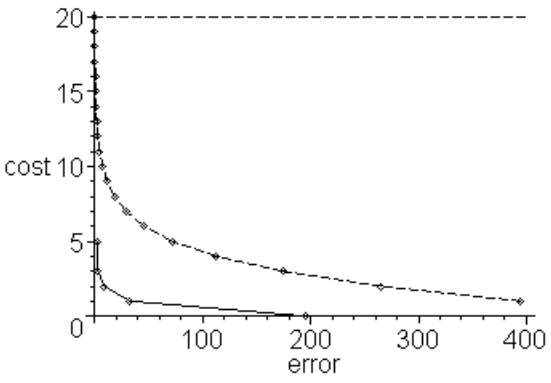


Figure 2: Pareto Fronts for Approximations of sqrt(x).

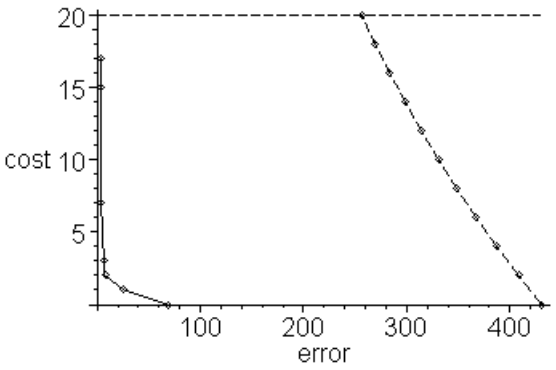


Figure 3: Pareto Fronts for Approximations of arcsinh(x).

For each experiment, we are interested in the genetically evolved approximations which lie to the interior of the Pareto fronts for Padé approximations, and thus are superior to Padé approximations given certain trade-offs between error and cost. Tables 3-5 list all such approximations for ln(x), sqrt(x), and arcsinh(x), respectively, along with their associated cost and error as calculated by the Maple cost procedure and by a Riemann integral, respectively. For ln(x), we are able to obtain 5 approximations which lie to the interior of the Pareto front for Padé approximations, for sqrt(x) we are also able to obtain 5 such approximations, and for arcsinh(x) we are able to obtain 7 approximations, all exhibiting various trade-offs between error and cost. As can be seen from Figure 3, arcsinh(x) proved to be a particularly difficult function for Padé approximations to model over the given interval.

Table 3: Evolved Approximations for ln(x).

EXPRESSION	COST	ERROR
(.0682089-2**x*-*x*/(-.1218591501**x*-3.842080570))/(-.385143144**x*-4.6585)	6	6.798897089
1.426990291**x*/(4.132660+.2760372098**x*)	3	7.436110884
(4.205966**x*-6.601615)/(*x*+12.85128201)+.694754	2	8.743267301
4.70397-29.12598131/(*x*+2.82952)	1	26.93968611
3.91812	0	64.55919780

Table 4: Evolved Approximations for sqrt(x).

EXPRESSION	COST	ERROR
x/(x/(4.78576+x/(9.17981+x/(15.39292+.04005697704*x)))+1.48335)	5	2.591348148
(x+.06288503787)/((x-9.04049)/(.05822627334*x+8.30072)+4.32524)+.795465	3	3.123452980
x/(5.5426193+.06559635887*x)+1.48335	2	8.935605674
.07262106112*x+3.172308452	1	32.95322345
7.011926	0	195.5193204

Table 5: Evolved Approximations for arcsinh(x).

EXPRESSION	COST	ERROR
1.86636*(1.277853316*x/((.3868816181*(-2.90216-x)/(-4.88586-x)+1.02145)*(-1.122792357-.3868816181*x))-.03522759767*(-1.122792357-.3868816181*x)*(x+4.86602)*(x-.269326)/(.0840785+x)+4.83551*x)/(9.684284+2.08151*x)	17	3.361399200
1.86636*(.07017092454*x^2/((2*x+4.86602)*(3.111694208+4.83551*x))-.03539134480*(.2502505059-.3868816181*x)*(x+4.86602)*(x-.269326)/(.0840785+x)+4.83551*x)/(9.684284+2.08151*x)	15	3.533969225
1.86636*(.0840785-.03522759767*(-1.122792357-.3868816181*x)*(x-.269326)+4.83551*x)/(9.684284+2.08151*x)	7	3.804858563
2.46147/(.4180284579-4.28068*1/(-2.299172064-.7261005920*x))	3	6.596080331
4.466119361*x/(18.01575130+x)+1.32282	2	7.581253733
3.30409+.02369172723*x	1	25.83927515
4.600931145	0	68.51916981

5 APPROXIMATING FUNCTIONS OF MORE THAN ONE VARIABLE

For some functions of more than one variable, it is possible to obtain a polynomial or rational polynomial approximations using techniques designed to approximate functions of a single variable; this can be done by nesting and combining approximations. For example, to obtain a rational polynomial approximation for the function $f(x,y)$=ln(x)*sin(y), one could compute a Padé approximation for ln(x) and a Padé approximation for cos(x) and multiply the two together. To compute a rational polynomial approximation for a more complex function such as $f(x,y)$=cos(ln(x)*sin(y)), one could again compute two Padé approximations and multiply them together, assign the result to an intermediate variable z, and compute a Padé approximation for cos(z). However, for any function of more than one variable that involves a non-arithmetic, non-unary operator whose set of operands contains at least two variables, there is no way to compute a polynomial or rational polynomial approximation using techniques designed to compute approximations for functions of a single variable. For the function $f(x)=x^y$, for example, there is no way to use Padé approximations or Taylor series to obtain an approximation, since the variables x and y are inextricably entwined by the exponentiation operator. In contrast, the genetic programming approach can be used on any function for which data points can be generated. To test the ability of genetic programming to evolve rational polynomial approximations for the type of function just described, an experiment was conducted to evolve approximations of the function $f(x)=x^y$ over the area $0<=x<=1$, $0<=y<=1$. Parameter settings were the same as described in the section on Harmonic numbers, including the generation limit of 51. Training data consisted of 100 (three dimensional) points chosen at random from the given rectangle. As in the previous section, a subset of 25 examples was used to evaluate the individuals of each generation.

The approximations returned by the genetic programming system were further evaluated through Maple. As in the previous section, a Maple procedure was used to calculate the minimum number of multiplications/divisions necessary to compute the approximation, while the error was evaluated using a double Riemann integral with 10000 points. The Pareto front for this set of approximations was then recomputed using the new cost and error values. The results of this evaluation are presented in Table 6.

Table 6: Evolved Approximations for x^y.

EXPRESSION	COST	ERROR
$x/(y\text{^}2+x-x*y\text{^}3)$	4	.03643611691
$x/(y\text{^}2+x-x*y\text{^}2)$	3	.04650160477
$x/(y+x-x*y)$	2	.04745973920
$x*y-y+.989868$	1	.05509570980
$x+.13336555$	0	.1401316648

The most accurate approximation evolved as a result of this experiment was $x/(y^2+x-xy^3)$. Figures 4 and 5 present graphs for the target surface $f(x)=x^y$ and for this approximation, respectively. Visually, the evolved surface is quite similar to the target function.

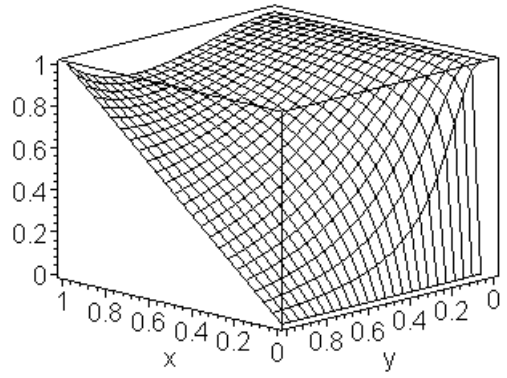


Figure 4: $f(x)=x^y$.

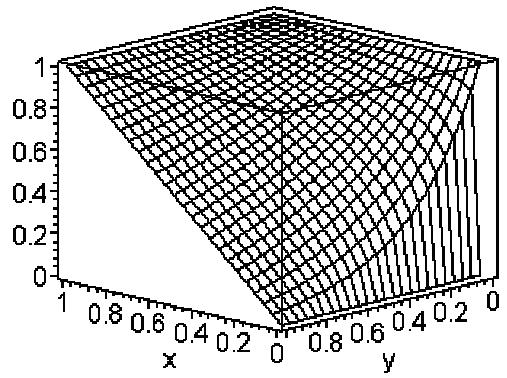


Figure 5: $x/(y^2+x-xy^3)$.

6 REFINING APPROXIMATIONS

It is possible to refine an approximation $a(x)$ by evolving an approximation ($a'(x)$) to its error function, then taking the refined approximation as $a(x)+a'(x)$. To test the practicality of this idea, we performed refinement of several evolved approximations to the function $\sin(x)$ over the interval $[0,\pi/2]$. Available space prohibits us from presenting the full results of this experiment. We note, however, that we are able to obtain 4 approximations in this manner which improve upon the Pareto front for our original experiment (prior to refinement), which contains a total of 7 approximations. The experiment was conducted using the same settings as in sections 4 and 5, but with an error multiplier of 1000. Refinement in this manner could be applied iteratively, to produce successively more accurate approximations. We have not investigated this possibility in any detail, but it is clear from our preliminary findings that the technique of refining approximations in this manner is indeed capable of producing significantly more accurate approximations.

In addition to refining evolved approximations using genetic programming, it is also possible to refine approximations generated through some other technique (such as Padé approximations) through genetic programming, or to refine approximations evolved via genetic programming through a technique from numerical analysis. Were the latter approach to prove effective, it could be incorporated on-the-fly in the evalution of individual approximations; one can imagine a rather different approach to the problem in which all evolving approximations are refined to a certain degree of accuracy by adding terms based on Padé approximations or Taylor series, and fitness is taken simply as the cost of the resulting expression. This provides for an interesting possible extension of the work reported in this paper.

7 FUTURE WORK

The work presented in this paper suggests a number of possible extensions. First, by adding if-then functions and appropriate relational operators such as less-than and greater-than to the function set, one could evolve piecewise rather than unconditional approximations to functions. Second, as suggested in the previous section, several extensions to this work based on the refinement of approximations could be attempted. Third, little attempt was made in this work to optimize parameters for the problem of finding rational polynomial approximations in general, and no parameter optimizations were made for specific functions being approximated, so that alteration of parameter settings represents a significant potential for improvement on the results presented in this paper. These results could also presumably be improved by using additional computational power and memory, and by employing a genetic programming system which allows for automatically defined functions (Koza 1994).

Perhaps the ideal application of this technique would be to perform the equivalent of conducting the Harmonic number experiment prior to 1734, the year that Leonhard Euler established the limiting relation

$$\lim n\to\infty H_n-\ln(n) \equiv \gamma$$

which defines Euler's constant (Eulero 1734). Such a result would represent "discovery" of an approximation formula in the truest sense, and would be a striking and exciting application of genetic programming.

8 CONCLUSIONS

This paper has shown that genetic programming is capable of rediscovering approximation formulae for Harmonic numbers, and of evolving rational polynomial approximations to functions which, under some reasonable utility functions, are superior to Padé approximations. For common mathematical functions of a single variable approximated over a relatively large interval, it has been shown that genetic programming can provide a set of rational polynomial approximations whose Pareto front lies in part to the interior of the Pareto front for Padé approximations to the same function. Though it has not been demonstrated explicitly in this paper, one would expect that genetic programming would also be able to expand upon the Pareto front for approximations to functions of more than one variable obtained by combining and nesting Padé approximations. Furthermore, for at least one function of more than one variable, genetic programming has been shown to provide a way to evolve rational polynomial approximations where the Padé approximation technique cannot be applied. Finally, we have presented results involving evolutionary refinement of evolved approximations. Based upon these results, the authors regard the genetic programming approach described in this paper as a powerful, flexible, and effective technique for the automated discovery of approximations to functions.

Acknowledgments

The authors wish to thank Prof. Micha Hofri of Worcester Polytechnic Institute for valuable advice and feedback received during the course of this project.

References

D. Andre and J. R. Koza (1996). Parallel genetic programming: A scalable implementation using the transputer network architecture. In P. J. Angeline and K. E. Kinnear, Jr. (eds.), *Advances in Genetic Programming 2*, 317-338. Cambridge, MA: MIT Press.

G. A Baker (1975). *Essentials of Padé Approximants*. New York: Academic Press.

C. M. Bender and S. A. Orszag (1978). *Advanced Mathematical Methods for Scientists and Engineers*. New York: McGraw-Hill.

T. Blickle and L. Thiele (1995). A comparison of selection schemes used in genetic algorithms. TIK-Report 11, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Networks Laboratory, ETH, Swiss Federal Institute of Technology.

R. L. Burden and J. D. Faires (1997). *Numerical Analysis*. Pacific Grove, CA: Brooks/Cole Publishing Company.

K. Chellapilla (1997). Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation* **1**(3):209-216.

L. Eulero (1734). De progressionibus harmonicus observationes. In *Comentarii academiæ scientarum imperialis Petropolitanæ* **7**(1734):150-161.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

G. H. Gonnet (1984). *Handbook of Algorithms and Data Structures*. London: Addison-Wesley.

M. A. Keane, J. R. Koza, and J. P. Rice (1993). Finding an impulse response function using genetic programming. In *Proceedings of the 1993 American Control Conference*, 3:2345-2350.

J. R. Koza (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Stanford University Computer Science Department technical report STAN-CS-90-1314.

J. R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

J. R. Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press.

S. Luke and L. Spector (1997). A comparison of crossover and mutation in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 240-248. San Mateo, CA: Morgan Kaufmann.

R. E. Moustafa, K. A. De Jong, and E. J. Wegman (1999). Using genetic algorithms for adaptive function approximation and mesh generation. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference*, 1:798. San Mateo, CA: Morgan Kaufmann.

P. Nordin (1997). *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, der Universitat Dortmund am Fachereich Informatik.

C. Ryan, J. J. Collins, and M. O'Neill (1998). Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (eds.), *Proceedings of the First European Workshop on Genetic Programming*, 1391:83-95. New York: Springer-Verlag.

Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values

Alexander Topchy
Computer Science Dept.
Michigan State University
East Lansing, MI 48824
topchyal@cse.msu.edu

W. F. Punch
Computer Science Dept.
Michigan State University
East Lansing, MI 48824
punch@cse.msu.edu

Abstract

We examine the effectiveness of gradient search optimization of numeric leaf values for Genetic Programming. Genetic search for tree-like programs at the population level is complemented by the optimization of terminal values at the individual level. Local adaptation of individuals is made easier by algorithmic differentiation. We show how conventional random constants are tuned by gradient descent with minimal overhead. Several experiments with symbolic regression problems are performed to demonstrate the approach's effectiveness. Effects of local learning are clearly manifest in both improved approximation accuracy and selection changes when periods of local and global search are interleaved. Special attention is paid to the low overhead of the local gradient descent. Finally, the inductive bias of local learning is quantified.

1 INTRODUCTION

The quest for more efficient Genetic Programming (GP) is an important research problem. This is due to the fact that a high computational complexity of GP is among its distinctive features (Poli & Page, 2000). Especially now, when variants of GP are being used on very ambitious projects (Thompson, 1998; Koza et al., 1997), the speed and efficiency of evolution are very crucial for such problems.

Numerous modifications of the basic GP paradigm (Koza, 1992) are currently known, e.g. see (Langdon, 1998) for a review. Among them, several researchers have considered GP augmentation by hill climbing, simulated annealing and other stochastic techniques. In (O'Reilly & Oppacher, 1996) crossover and mutation are used as move operators of hill climbing, while Esparcia-Alcazar & Sharman (1997) considered optimization of extra parameters (node gains) using simulated annealing. Terminal search was employed in (Watson & Parmee, 1996), but due to the associated computational expense it was limited to 2-4% of individuals. The presence of stochasticity in local learning makes it relatively slow, even though some hybrid algorithms yield overall improvement. For example, Iba and Nikolaev (2000) and Rodriguez-Vazquez (2000) considered least squares coefficients fitting limited to linear models. Apparently, the full potential of local search optimization is yet to be realized.

The focus of this paper is on a local adaptation of individual programs during the GP process. We rely on gradient descent for improved generation of GP individuals. This adaptation can be performed repeatedly during the lifetime of an individual. The results of local learning may or may not be coded back into the genotype (reverse transcription) based on the modified behavior, which is reported in the literature as Lamarckian and Baldwinian learning, respectively (Hinton & Nowlan, 1987; Whitley et al., 1994). The resulting new fitness values affect the selection process in both cases, which in turn changes the global optimization performance of a GP. Such an interaction between local learning, evolution and associated phenomena without reverse transcription are also generally referred to as the Baldwin effect.

We were motivated by a number of successful applications of hybridization to neural networks (Belew et al., 1991; Zhang & Mühlenbein, 1993; Nolfi et al., 1994). Both neural networks and GP trees perform input-output mapping with a number of adjustable parameters. In this respect, terminal values (leaf coefficients) in a GP perform a similar function as weights in neural network. A form of gradient descent is usually used to adjust weights in a neural net architecture. In contrast, various terminal constants are typically random within GP trees and are rarely adjusted by gradient methods. The reasons for this are twofold: the unavailability of gradients/derivatives in some GP problems and the computational expense that is assumed to exist in computing those gradients. However, the complexity of computing derivatives is largely overestimated. In order to differentiate programs explicitly, algorithmic differentiation (Griewank, 2000) may be adopted. Algorithmic (computational) differentiation is a technique that accurately determines values of derivatives with

essentially the same time complexity as found in the execution of the evaluation function itself. In fact, gradients may often be computed as part of the function evaluation. This is especially true for trees and at least potentially true for arbitrary non-tree programs. Generalization of the method for any program is possible, given that the generated program computes numeric values, even in presence of loops, branches and intermediate variables. The main requirement is that the function be piecewise differentiable. While not always true, this is the case for a great majority of engineering design applications. Moreover, it is also known, that directional derivatives can be computed with many non-smooth functions (Griewank, 2000). Knowledge of only gradient direction, not its value, is often enough to optimize the values of parameters.

In this paper we empirically compare conventional GP with a GP coupled with terminal constant learning. The effectiveness of the approach is demonstrated on several symbolic regression problems. Arithmetic operations have been chosen as the primitives set in our GP implementation for simplicity sake. While such functions make differentiation easy, again these techniques can be adapted to more difficult problems.

Our results indicate that inexpensive differentiation along with the Baldwin effect leads to a very fast form of GP. Significant improvement in accuracy was also achieved beyond that which could be achieved by either local search or more generations of GP.

The Baldwin effect is known to change the inductive bias of the algorithm (Turney, 1996). In the case of GP, where functional complexity is highly variable, it is expected that such a change of bias can be properly quantified. Two manifestations of the learning bias were observed in our experiments. Firstly, the selection process is affected by local learning since the fitness of many individuals dramatically improves during their lifetime. Secondly, changes in the functional complexity of individuals were observed in the experiments. Both the length (number of nodes) of the best evolved programs and the number of leaf coefficients were higher using local learning as opposed to regular GP.

2 LAMARCKIAN VS BALDWIN STRATEGY IN GP

Evolution rarely proceeds without phenotypic changes. As we are interested in digital evolution, two dominating strategies have been proposed which allow environmental fitness to affect genetic features. Lamarckian evolution, an alternative proposition to Darwinian approaches of the time, claimed that traits acquired from individual experience could be directly encoded into the genotype and inherited by offspring. In contrast, Baldwin claimed that Lamarckian effects could be observed where no direct transfer of phenotypic characteristic to the genotype occurred, in keeping with Darwinism. Rather, Baldwin claimed that "innate" behaviors could be

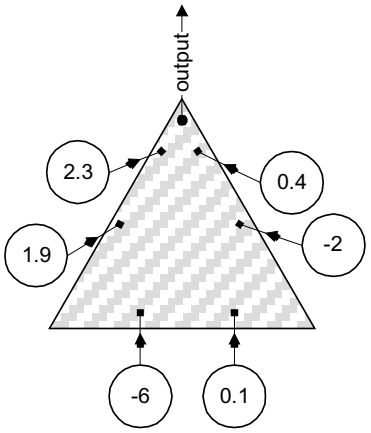


Figure 1: Sample tree with a set of random constants. In hybrid GP all these leaf coefficients are subjected to training

selected for (in a Darwinian sense) which the individual originally had to learn. In Lamarckian evolution learning affects fitness distribution as well as the underlying genotypic values, while the Baldwin effect is mediated via the fitness results only. In our case, the question is whether locally learned constants are copied back into the genotype (Lamarckian) or whether the constants are unmodified while the individual's fitness value reflects the fitness resulting from learning (Baldwin).

Real algorithmic implementations of evolution coupled with local learning are much richer than two original strategies. The researcher, usually guided by the total computational expense, may arbitrarily decide both the amount of and scheduling of learning or local adaptation of solutions. Moreover, since local learning comes with a price, it must be wisely traded off with genetic search costs. Several questions must be answered:

- What aspect of the solution should be learned beyond genetic search, as only a subset of solution parameters may be chosen for adaptation?
- Should learning be performed at every generation or should it be used as a form of fine-tuning when genetic search is converged?
- How many individuals and which of those individuals should have local learning applied to them?
- How many iterations of local learning should be done (really, how much computational cost are we willing to incur)?

Accordingly, there are many ways to introduce local learning into GP. Evolution in GP is both parametric and structural in nature. Two important features are specific to GP:

1. The fitness of the functional structure depends critically on the values of local parameters. Even very fit structures may perform poorly due to inappropriate numeric coefficients.
2. The fitness of the individual is highly context sensitive. Slight changes in structure dramatically influence fitness and may require completely new parameters.

That is why we focus on learning numeric coefficients, so called Ephemeral Random Constants or ERC (Koza, 1992), which are traditionally randomly generated as shown in Figure 1. As explained below, the local learning algorithm -- gradient descent on the error surface in the space of the individual's coefficients, turns out to be a very inexpensive approach, so much so that every individual can do local learning in every generation.

Formally we follow the Lamarckian principle of evolution since we allow the tuned performance of individual to directly affect the genome by modifying numeric constants. At the same time, the choice between Lamarckian and Baldwin strategies in our implementation is not founded on the issue of computational complexity. In both cases the amount of the extra work is approximately the same. The main issue arises when considering the fitness values of the offspring with inherited coefficients vs. offspring with unadjusted terminals. Our experiments indicate that there is little difference between the two fitnesses when crossover is the main operator. Two factors contribute to this:

1. Crossover usually generates individuals with significantly worse fitness than their parents. The coefficients found earlier to be good for the parents are not appropriate for the offspring structures. The subsequent local learning changes fitness dramatically by updating the ERCs to more appropriate values.
2. Newly generated offspring are equally well adjusted starting from any values: earlier trained, not trained or even random.

Hence, inheritance of the coefficients does not much help the performance of the individuals created by crossover. However, if an individual is transferred to a new generation as a part of the elitist pool, i.e. unchanged by crossover or mutation, then its learned coefficients are also transferred. With respect to this structure, the use of the Baldwin strategy would be wasteful, since it requires relearning the same parameters. Thus, even though our implementation formally follows the Lamarckian strategy, we effectively observe the very same phenomena peculiar to the Baldwin effect.

3 HYBRID GP

The organization of the hybrid GP (HGP) is basically the same as that of the standard GP. The only extra activity done by the algorithm is to update the values of numeric coefficients. That is, all individuals in the population are trained using a simple gradient algorithm in every generation of the standard GP. Below we discuss the exact formulation of the corresponding optimization problem.

3.1 PROBLEM STATEMENT

The hybrid GP is intended to solve problems of a numeric nature, which may include regression, recognition, system identification or control. We will assume throughout that there are no non-differentiable nodes, such as Boolean functions. In general, given a set of N input-output pairs $(d,\mathbf{x})_i$ it is required to find a mapping $f(\mathbf{x},\mathbf{c})$ minimizing certain performance criteria, e.g. mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(d_i - f(\mathbf{x}_i,\mathbf{c})\right)^2 \qquad (1)$$

Here, f is scalar function (generalization to multi-trees is trivial), $\mathbf{x}$ is vector of input values, $\mathbf{c}$ is vector of coefficients and the sum is over the training samples. Of course, in GP we are interested in discovering the mapping $f(\mathbf{x},\mathbf{c})$ in the form of a program tree. That is, we seek not only coefficients $\mathbf{c}$, but also the very structure of the mapping f, which is not known in advance. In our approach, finding the coefficients is done by gradient descent during the same time functional structures are evolved. Descriptions of the standard GP approach can be found elsewhere (e.g. Langdon, 1998), instead, we will focus below on details of the local learning algorithm.

3.2 LEARNING LEAF COEFFICIENTS

Minimization of MSE is done by a few iterations of a simple gradient descent. At each generation all numeric coefficients are updated several times using the rule:

$$c_k \to c_k - \alpha\frac{\partial\, MSE\,(\mathbf{c})}{\partial\, c_k}, \qquad (2)$$

where α is the learning rate, and k goes over all the coefficients at our disposal. Three important points must be discussed: how to find the derivatives, what the value of α should be, and how many iterations (steps) of descent should be used.

3.2.1 Differentiation

Using both eq. 1 and 2 we obtain:

$$\frac{\partial\, MSE(\mathbf{c})}{\partial\, c_k} = -\frac{2}{N}\sum_{i=1}^{N}\left(d_i - f(\mathbf{x}_i,\mathbf{c})\right)\frac{\partial f(\mathbf{x}_i,\mathbf{c})}{\partial\, c_k} \qquad (3)$$

Thus, an immediate goal is to differentiate any current program tree with respect to any of its leaves. The chain rule significantly simplifies computing $\partial f/\partial c$. Indeed, if $n_j(\cdot)$ denotes node functions, then:

$$\frac{\partial f(n_1(n_2(n_3(...),...),...))}{\partial c_k} = \frac{\partial f}{\partial n_1}\frac{\partial n_1}{\partial n_2}\frac{\partial n_2}{\partial n_3}\cdots\frac{\partial n_r(c_k,...)}{\partial c_k}$$

Therefore differentiation of the tree simply reduces to the product of the node derivatives on the path which starts at the given leaf and ends at the root. It is clear that each term in the product is a derivative of a node output with respect to its arguments (children). If paths from the different leaves share some common part, then corresponding sub-chains in the derivatives are also shared. Computation of such a product in practice depends on the data structure used for the program tree. In simple cases, differentiation uses single recursive postorder traversal together with the actual function evaluation. Derivatives of the program tree with respect to all its leaves can be obtained simultaneously. As soon as an entire sum in eq. 3 becomes known, i.e. derivatives in all training points obtained, one may need an extra sweep through the tree to update the coefficients. In total, the incurred overhead depends on the complexity of node derivatives and the number of leaves. For instance, in our implementation using only an arithmetic functional set, the cost of differentiation was equal to the cost of function evaluation, making the overall cost twice the standard GP cost for the same problem.

3.2.2 Learning rate and number of steps

In a simple gradient descent algorithm, the proper choice of learning rate is very important. Too large a learning rate may increase error, while too small a rate may require many training iterations. It is also known that the rule in eq. 3 works better in the areas far from the vicinity of local minima (Reklaitis, 1983). Therefore we decided to make the rate as large as possible without sacrificing quality of learning. After a few trials on the test problem of symbolic regression we fixed the learning rate to the value α=0.5. The same learning rate was used for all other test problems. If the algorithm resulted in an increase in the error of an individual, the training was stopped and no update to the individual's fitness was recorded. However, this problem did not have any impact on overall quality of learning since it happened rarely, approximately 1 out of 10 successful individuals. Moreover, those individuals that had this problem showed an error rate that was typically not reduced by any subsequent application of gradient descent.

This simple local learning rule dramatically improved the fitness of individuals. Figure 2 shows the decrease in MSE for typical individuals. It is important to note that the most significant improvements happened after only the first few iterations of local learning. Note that some individuals were improved by as much as 60% or more. We decided that 3 steps of gradient descent was a good trade off between fitness gain and effort overhead. Again, the number of iterations was never altered afterwards and is used in all our experiments.

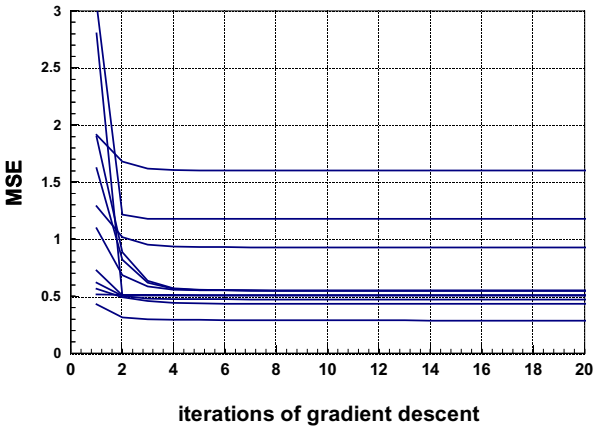


Figure 2: Local learning strongly affects fitness of individuals. Typical learning progress is illustrated using individuals from test problem f_2.

4 EXPERIMENTAL DESIGN

The main goal of the empirical study is to compare the performance of the GP with and without learning. Even though an overall speed-up is very valuable, we are also interested in other effects resulting from local learning. These effects have to be properly quantified to shed light on the internal mechanisms of the interaction between learning and evolution. Three major issues are studied:

- Improvement in search speed
- Changes in fitness distribution and selection
- Changes in the functional structure of the programs

4.1 IMPLEMENTATION DETAILS

The driver GP program included the following major steps:

1. Initialization of the population using the "grow" method. Starting from a set of random roots, more nodes and terminals are aggregated with equal probability until a specified number of nodes are generated. The total number of nodes in the initial population was chosen to be three times greater than the population size, that is three functional nodes per individual on average.
2. Fitness evaluation and training (in HGP) of each individual. Mean squared error over the given training set, as defined by eq. 1, serves as an inverse fitness function since we seek to minimize error. This stage includes parametric training in HGP given that the individual has leaf coefficients.
3. Termination criteria check. The number of function evaluations was the measure of computational effort. For instance, every individual is evaluated only once in every GP generation, but three times in every HGP generation if its parameters are trained for three steps.

4. Tournament selection (tournament size = 2) of parents. Pairs are selected at random with replacement and their number is equal to the population size. The better of the two individuals becomes a parent at the next step.
5. Crossover and reproduction. Standard tree crossover is used. Each pair of parents produces two offspring. Mutation with small probability p_m=0.01 is applied to each node. In addition, elitism was always used and the best 10% of the population survive unchanged.
6. Pruning the trees with the size exceeding predefined threshold value.
7. Continue to step 2.

4.2 TEST PROBLEMS

Five surface fitting problems were chosen as benchmarks.

$$f_1(x,y) = xy + \sin((x-1)(y+1))$$

$$f_2(x,y) = x^4 - x^3 + y^2/2 - y$$

$$f_3(x,y) = 6\sin(x)\cos(y)$$

$$f_4(x,y) = 8(2 + x^2 + y^2)^{-1}$$

$$f_5(x,y) = x^3/5 + y^3/2 - y - x$$

For each problem 20 random training points (fitness cases) were generated in the range [-3...3] along each axis. Figure 3 shows the desired surfaces to be evolved.

5 EXPERIMENTAL RESULTS

To compare the performance we made experiments with both hybrid and regular GP with the same effort of 30,000 function evaluations in each run. All experiments were done with a population size of 100 and the arithmetic operators {+,-,*,%-protected} as the function set with no ADFs. Initial leaf coefficients were randomly generated in the range [-1...1]. Also, the pruning threshold was set to 24 nodes. If the number of nodes in an individual grew beyond this threshold, a sub-tree beginning at some randomly chosen node was cut from the individual. Each experiment was run 10 times and the MSE value was monitored.

Our main results are shown in Figure 3 and also summarized in Table 1. The success of the hybrid GP is quite remarkable. For all the test problems, the average error of the best evolved programs was significantly smaller (1.5 to 25 times) when learning was employed. The first 20 – 30 generations usually brought most of these improvements. The gap in error levels is wide enough to require the regular GP to use hundreds more generations to achieve similar results. Certain improvements were also observed for the average population fitness, but with lesser magnitude. The similarity of each population's average fitness indicates a high diversity and that not all offspring reach small error values after local learning.

Another set of experiments included extra fine-tuning iterations performed only after the regular GP terminates. Again, we run gradient optimization on the population from the last GP generation. Each individual was tuned by applying 100 gradient descent iterations. The results in Table 1 show that this approach is not effective and did not achieve the quality of result found in the HGP. This is a strong argument for Baldwin effect, namely that another factor affecting search speed-up is a change in fitness distribution that directly affects selection outcome. Learning introduces a bias that favors individuals that are more able to adapt to local learning modifications. If we would suppose that the selection bias does not occur, then the hybrid GP would be only a trivial combination of genetic search and fine tuning. However, as we see from the results this is not the case.

We attempted to measure some properties of HGP that would demonstrate this synergy between local learning and evolution.

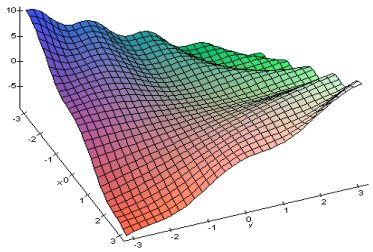
Table 1. Performance Comparison of Hybrid and Regular GP. All data collected after 30000 f.e. and averaged over 10 experiments.

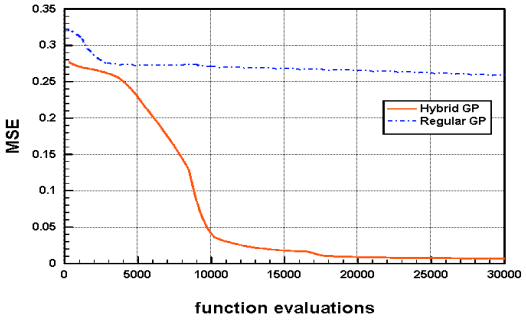
Test problem	Best MSE		Ave. MSE		Best MSE
	HGP	GP	HGP	GP	GP + fine tuning
f_1	0.009	0.26	0.47	0.80	0.233
f_2	0.075	0.761	1.03	2.18	0.31
f_3	2.32	6.22	5.98	6.59	6.21
f_4	0.64	0.76	4.06	4.41	0.76
f_5	0.097	0.36	0.27	0.78	0.30

First of all, a Baldwin effect in selection would mean that the results of some tournament selections are reversed after local learning. Indeed, local learning adaptable individuals win their tournaments due to improved fitness resulting from gradient descent. These individuals would lose the same tournament in regular GP.

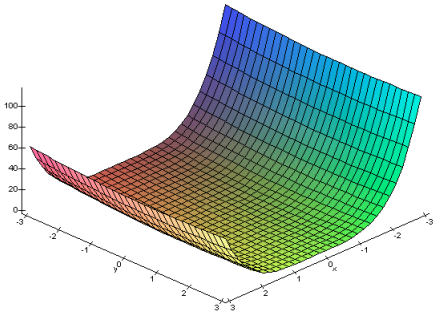
Figure 4 shows both the typical and average percentage of reversed tournaments in the problem f_1. A summary of results for all the test problems is given Table 2.

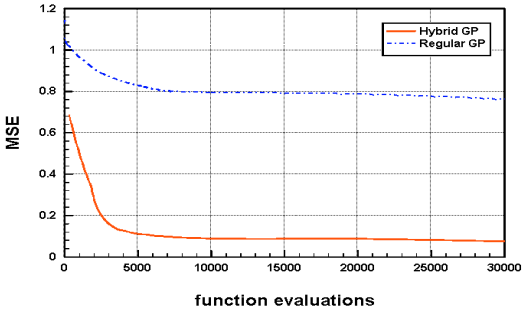
$f_1(x,y) = xy + \sin((x-1)(y+1))$



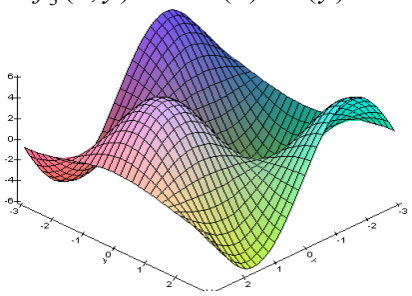


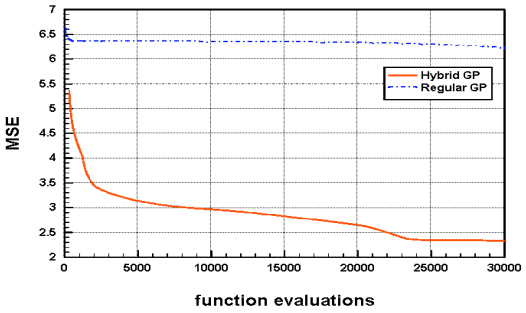
$f_2(x,y) = x^4 - x^3 + y^2/2 - y$



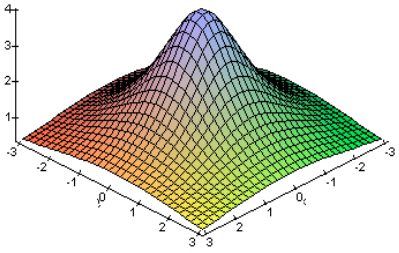


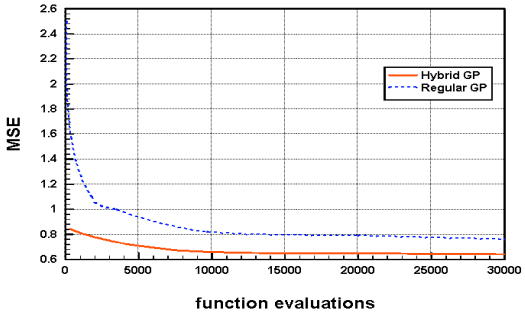
$f_3(x,y) = 6\sin(x)\cos(y)$



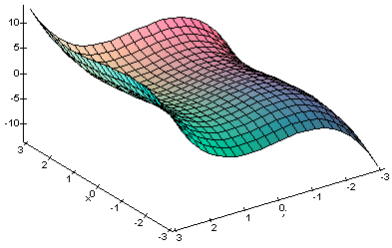


$f_4(x,y) = 8(2 + x^2 + y^2)^{-1}$





$f_5(x,y) = x^3/5 + y^3/2 - y - x$



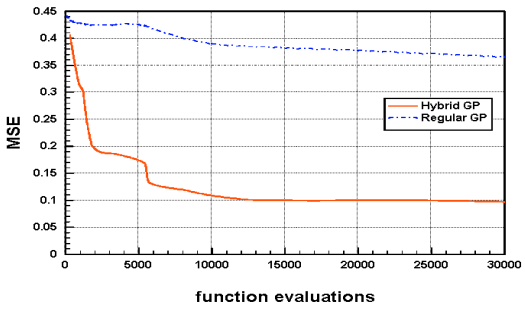


Figure 3: Surface fitting test problems and respective learning curves

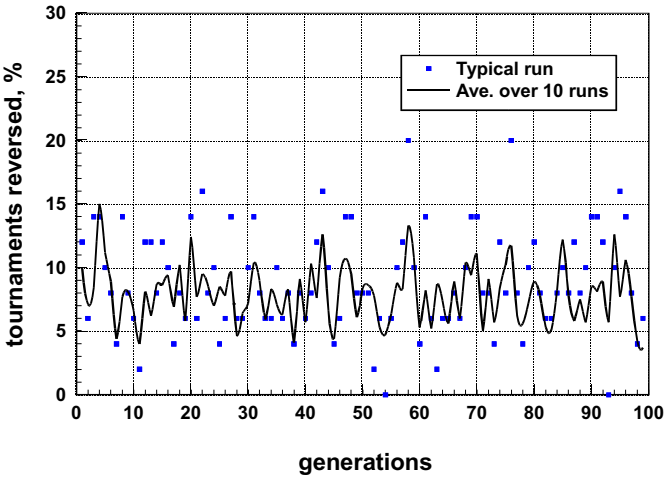


Figure 4: Comparison of Selection Process in HGP and GP. Local learning changes outcome of some tournaments used to select a mating pool.

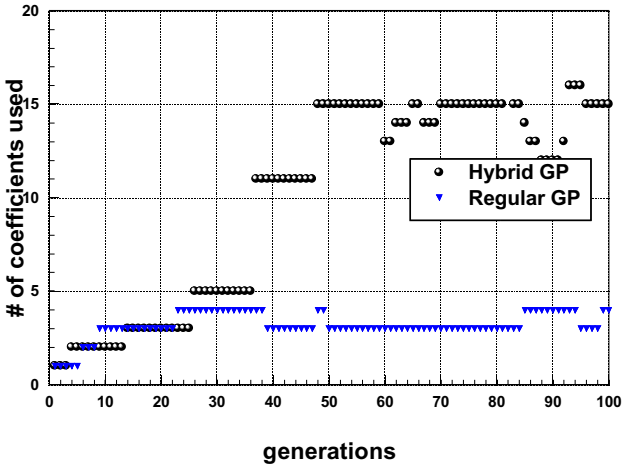


Figure 5: Typical dynamic of number of terminals (numeric coefficients) used by the best program as a function of GP generations (for the test function f1).

It was found that the average percentage of selection changes remains the same during the course of search for all test problems. Such a behavior would be expected if selection pressure pushes offspring that are very adaptable, even when older elite members are almost converged. An empirical measure of this degree of adaptability is provided by the average gain in fitness achieved by newly generated offspring. The values are given in Table 2. We do not include elite members in this statistic to emphasize magnitude of learning from scratch. The average observed drop of MSE is between 12% and 19% on all the test problems.

What exactly makes one program more adaptable than the other? Clearly, it is the functional structure of the program. For example, a program with no numeric leaves cannot learn at all using the gradient local learning method described above. Furthermore, a tree with no terminal arguments (inputs) containing only terminal constants will always produce the same output and will not benefit from local learning. Instead we have tried to understand what characteristics of adaptable programs are unique.

Table 2: Effects of local learning

Test problems	Difference in HGP selection vs. GP in each generation on average, %	Ave. MSE gain for newly generated offspring, %	Complexity of the best programs, #coefficients / #nodes after the same effort (30000 f.e.)	
			HGP	GP
f_1	7.7	16.5	16.0 / 22.4	12.2 / 21.2
f_2	7.1	12.7	16.6 / 23.0	13.7 / 21.8
f_3	8.4	15.1	17.5 / 23.5	11.8 / 20.4
f_4	7.9	18.7	17.3 / 22.9	12.4 / 21.6
f_5	7.4	15.0	17.0 / 23.1	12.9 / 21.8

We have focused on the length (number of nodes) and on the number of coefficients in the best evolved programs (remember, that length had an upper limit too). As Table 2 illustrates both values are noticeably greater for the programs evolved by HGP. This is one illustration of the inductive bias of the hybrid algorithm. More adaptive programs use more coefficients and consequently have lengthier representations. Also, the number of the terminal inputs (x and y) in HGP results is slightly smaller. Figure 5 shows typical changes in the number of coefficients for a "best" individual on a generational scale for both GP and HGP.

6 CONCLUSIONS

This paper has shown a number of important points. First, that local learning in the form of gradient descent can be efficiently included into GP search. Second, that this learning provides a substantial improvement in both final fitness and speed in reaching this fitness. Finally, the use of local learning creates a bias in the structure of the solutions, namely it prefers structures that are more readily adaptable by local learning. We feel that this approach could have significant impact on practical, engineering problems that are addressed by GP.

References

R.K. Belew, J. McInerney, and N.N. Schraudolph (1991). Evolving networks: Using the Genetic Algorithm with connectionist learning. In *Proceedings of the Second Artificial Life Conference*, 511-547, Addison-Wesley.

Anna Esparcia-Alcazar and Ken Sharman (1997). Learning schemes for genetic programming, In Proceedings *Late Breaking Papers at the 1997 Genetic Programming Conference*, 57-65, Stanford University, CAAndreas Griewank (2000). *Evaluating derivatives: Principles and techniques of algorithmic differentiation*, SIAM, Philadelphia.

G. E. Hinton and S. J Nowlan (1987). How learning can guide evolution, *Complex Systems*, **1**, 495-502.

H. Iba and N. Nikolaev (2000). Genetic Programming Polynomial Models of Financial Data Series," in *Proceedings of the Conference on Evolutionary Computation, CEC-2000*, IEEE Press, pp. 1459-1466.

John R. Koza, Forrest H Bennett III, David Andre, Martin A. Keane, and Frank Dunlap (1997). Automated synthesis of analog electrical circuits by means of genetic programming, *IEEE Transactions on Evolutionary Computation*, **1**(2), 109-128, 1997.

John R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

William B. Langdon (1998). *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming*, Kluwer, Boston.

S. Nolfi, J. Elman, and D. Parisi (1994). Learning and evolution in neural networks, *Adaptive Behavior*, **3**(1), 5-28.

Riccardo Poli and Jonathan Page (2000). Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code GP and demes, *Genetic Programming And Evolvable Machines*, **1**(1/2), 37-56.

Una-May O'Reilly and Franz Oppacher (1996). A comparative analysis of GP, In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, ch. 2, 23-44. MIT Press, Cambridge, MA.

G.V. Reklaitis, A. Ravindran and K.M. Ragsdell (1983). *Engineering Optimization: Methods and Applications*, Wiley, New York.

Rodriguez-Vazquez, K. (2000). Identification of Non-Linear MIMO Systems Using Evolutionary Computation, *Late Breaking Papers of Genetic and Evolutionary Computation Conf.*, 411-417.

Adrian Thompson (1998). *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*, Springer-Verlag: London.

P. Turney (1996). How to shift bias: Lessons from the Baldwin effect, *Evolutionary Computation*, **4**(3), 271-295.

B. Zhang and H. Mühlenbein (1993). Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor, *Complex Systems*, **7**(3), 199 -220.

Andrew Watson and Ian Parmee (1996). Systems Identification using Genetic Programming, In *Proceedings of Int. Conf on Adaptive Computing in Engineering Design and Manufacture*, 248 - 255, ACEDC'96, University of Plymouth, UK.

D. Whitley, S. Gordon and K. Mathias (1994) Larmarckian Evolution, The Baldwin Effect and Function Optimization. In *Proceedings Parallel Problem Solving from Nature, PPSN III*, 6-15.