Short Term Memory in Genetic Programming

K Bearpark and A J Keane
School of Engineering Sciences, University of Southampton

Abstract. The recognition of useful information, its retention in memory, and subsequent use plays an important part in the behaviour of many biological species. Information gained by experience in one generation can be propagated to subsequent generations by some form of teaching. Each generation can then supplement its taught learning by its own experience. In this paper we explore the role of memorized information in the performance of a Genetic Programming (GP) system that uses a tree structure as its representation. Memory is implemented in the form of a set of sub-trees derived from successful members of each generation. The memory is used by a genetic operator similar to the mutation operator but with the following difference. In a tree-structured system the mutation operator replaces randomly selected sub-trees by new randomly-generated sub-trees. The memory operator replaces randomly selected sub-trees by sub-trees randomly selected from the memory. To study the memory operator's impact a GP system is used to evolve a well-known expression from classical kinetics using fitness-based selection. The memory operator is used together with the common crossover and mutation operators. It is shown that the addition of a memory operator increases the probability of a successful evolution for this particular problem. At this stage we make no claim for its impact on other problems that have been successfully addressed by Genetic Programming.

1. Introduction

Genetic Programming (GP) is one of the most recently developed fields in the study of Evolutionary Computation (EC). Koza [1] has shown that GP can be successfully applied to a wide range of problems across a number of technical and social disciplines. Included in these problems is Symbolic Regression - finding a function, in symbolic form, that fits a given finite sample of data. This paper is concerned with the application of a short-term memory operator to symbolic regression.

Genetic Programming uses a Genetic Algorithm (GA) to evolve a progressively improving solution to a problem. In a conventional GA the solution is represented in vector form with the simplest GAs operating on a binary string.

The main difference between the pure GA and the GA-based GP approaches lies in the complexity of the representation. A GP representation generally has a hierarchical rather than a linear structure. This hierarchical structure requires some modification to the genetic operators defined for GA systems.

A typical GA operates in parallel on a set or population of solutions. Each member of the population is a potential solution and must be represented in such a way that its suitability, or fitness, as a solution can be measured. An initial population is created by random selection from a pool of components to give the first generation. Successive generations are produced by the application of genetic operators that mirror those responsible for the evolution of biological species. Individuals are selected to participate in the creation of the next generation according to their fitness. Some selected individuals pass into the next generation unchanged while others are subject to a crossover operation in which two parents produce children by exchanging genes. A further operation may occur in which a small amount of genetic material in a child is randomly mutated.

The three operations of selection, crossover and mutation occur in nature and in most GAs. Fitness-based selection ensures that highly fit individuals are well represented in the mating pool for the next generation while individuals with lower fitness tend to disappear. Crossover attempts to produce better individuals by incorporating genes from each parent. Random mutation of some genes in a child introduces new genetic material and extends the search for a solution to different regions of the search space. Many GA and GP applications have shown that all three operators, individually and in combination, are beneficial in improving the average fitness of each generation and evolving better solutions to the problem in hand or, in some cases, the best solution.

Evolution in nature or in EC relies on stochastic processes. The average ability of members of a generation to perform a particular task in a given environment tends to improve as the fitter members survive and reproduce at the expense of the less fit members. This ability may also improve through accumulated experience handed down from one generation to the next, i.e. if this experience is recorded it may be used by future generations to augment their development. In the GP system described here experience is recorded in the form of a set of sub-trees derived from an analysis of the more successful members of each generation. We call this set the 'memory'. The memory is used in a mutation-like process in which replacement material is selected randomly from the memory as a complete sub-tree. This should be contrasted with the conventional mutation operator that produces replacement sub-trees by randomly selecting operators and operands.

The use of a memory operator has some similarity to the encapsulation operator defined by Koza [1], automatically defined functions (ADF), also defined by Koza [2], and module acquisition, described by Angeline and Pollack [3].

Banzhaf *et. al.*[4] point out that these modularization techniques essentially preserve sub-sets of genetic material against the potential disruption of crossover. On the other hand, the memory operator defined here is used after crossover has occurred and as the final process in creating a new generation. It is akin to children learning from their grandparents and earlier generations.

2. The Objective Function

An elementary result from classical kinetics states that the distance travelled by an object in time t subject to constant acceleration a and with an initial velocity u is given by the expression

$$ut + \tfrac{1}{2}at^2.$$

The GP system described here uses symbolic regression to evolve this expression given sets of terminals and arithmetic operators and conventional GP techniques. The system is then used as a vehicle to explore the use of memory to improve its performance.

The fitness of an individual test expression is measured by evaluating the expression for values of t from 1 to 10, two values of u (20 and 200) and a single value of a (980). All units are arbitrary. These values are compared with the corresponding values of the true expression $ut + \tfrac{1}{2}at^2$ to give two error values

$$E20 = \sum_{t=1}^{10} \text{ABS}(\text{test}(t) - \text{true}(t)) \text{ for } u = 20$$

and

$$E200 = \sum_{t=1}^{10} \text{ABS}(\text{test}(t) - \text{true}(t)) \text{ for } u = 200.$$

If both E20 and E200 are zero the evolved expression is a 'hit' and is given an error value of zero. Otherwise the expression is given the error value

$$(E20+E200)/2.$$

3. The GP System

The GP system makes use of largely standard techniques with the exception of the memory operator. Its various components are described in the following sections.

3.1 The function and terminal sets

The functions available to the system are here restricted to the simple arithmetic binary operators resulting in the set

$$\{+,-,*,/\}.$$

The implementation ensures that the division operator is protected against a zero divisor.

The terminals are restricted to the variables u, a and t and integer constants from 1 to 9. The structure of the operand set has some bearing on the success of the evolutionary process. All the results reported here were achieved using the set

$$\{u,1,a,2,t,3,u,4,a,5,t,6,u,7,a,8,t,9\}.$$

These two sets are used in producing the first generation and in selecting replacement sub-strings during mutation.

We have also explored the use of a terminal set from which integer constants are removed thus relying on the emergence of rational numbers (e.g. $x/(x+x)$) during evolution. Our conclusions still hold although the performance of the system is reduced but can be restored by increasing the number of tests.

3.2 Representation

Expressions are represented as Reverse Polish (RP) strings except in the first randomly produced generation where it is simpler to generate conventional (unbracketed) arithmetic expressions and convert them to RP form. The string length is controlled by imposing a limit of 5 functions (and hence a maximum total string length of 11 characters) in the first generation and a maximum length of 19 characters (or 9 functions) in subsequent generations. Modification of the RP strings by genetic operators here makes use of a tree representation.

3.3 Production of the first generation

The first generation is produced by alternately selecting terminals and functions from the respective sets subject to a maximum of 5 functions. Each expression is converted to RP form and evaluated to give an error value as described earlier.

3.4 Fitness-based selection

The fitness of an expression when considering it for entry into the mating pool for the next generation is given by

$$fitness_i = maxerr/(error_i+1)$$

where $fitness_i$ and $error_i$ are respectively the fitness value and the error value of the i^{th} expression and *maxerr* is the maximum error value in the generation.

The mating pool for the next generation is then populated by a conventional roulette-wheel method. Generation $n+1$ is evolved from the generation n mating pool by applying the conventional genetic operators of crossover and mutation and also the memory operator described below.

3.5 Crossover

Crossover is applied according to a probability between 0% (no sexual reproduction) and 100% (full reproduction). Two members are selected at random from the mating pool and crossover points randomly selected in each member. The sub-trees anchored to these points are extracted and exchanged between the two members to provide two children. Iteration of this operation produces an interim generation of the same size as the previous generation which may then be subject to further genetic modification. Crossover may result in an RP string that exceeds the maximum length of 19 characters. If this occurs for a given pair of parents, 3 further attempts are made with alternative parents. If an acceptable string is still not obtained the crossover attempt is abandoned and the final pair of parents copied directly from the mating pool to the interim generation before proceeding to the next pair. The choice of 3 retry attempts is a compromise between allowing crossover to occur and ensuring that the system does not require excessive computing cycles.

3.6 Mutation

Mutation is also governed by a probability parameter. In common with most GP systems, members of the interim generation are selected randomly with a given probability and one randomly selected sub-tree in each selected member is replaced by a sub-tree generated by random selection of functions and terminals from the sets defined in Section 3.1. The length of the replacement and hence the length of the mutated member is controlled so as not to exceed the maximum of 19 characters.

It should be noted that if a mutation probability of *x%* is quoted it means that on average *x%* of the members of a generation are subject to mutation. The limit on RP string length of 19 characters implies a limit of 9 functions and hence 9 sub-trees. On average an individual member has between 4 and 5 sub-trees and a quoted mutation rate of *x%* translates to between *x*/4% and *x*/5% of the genetic material in the generation.

3.7 The memory operator

In a conventional genetic algorithm the quality of a generation is reflected in the next generation through the selection mechanism which ensures that highly fit members are well represented in the mating pool. The best member of a generation may also be copied one or more times into the next generation, following genetic modification, by an elitist strategy. Both operations ensure that good genetic material is available to succeeding generations. The memory mechanism introduced here also ensures that good material is preserved by keeping it in memory and explicitly introducing it into each generation as evolution proceeds.

Fitness-based selection, elitism and the use of memory are similar in that they all contribute to the propagation of successful genetic material through the generations. Selection ensures that highly-fit members of a population survive to reproduction at the expense of less-fit members. Elitism counters the potential disruptive impact of crossover, mutation and, indeed, the memory operator. The use of memory provides a pool of component sub-trees by breaking down good solutions in each generation. The memory is used as a source of replacement sub-trees to improve fitness by the introduction of material previously shown to be successful.

When the best member of generation *n* is an improvement on the best member of generation *n-1* (or when *n=1*) the best member of generation *n*, together with all its sub-trees, is recorded in the memory. The memory is then used as a source of replacement sub-trees during the memory operation by

randomly selecting the replacement from the memory. For example, if the expression *ut + at,* represented by the Reverse Polish string *ut*at*+,* is the best-so-far member of a generation, the strings *ut*at*+*, *ut** and *at** are added to the memory. The note relating to the meaning of a percentage probability in section 3.6 also applies to the memory operator. *x%* use of the memory operator means that *x%* of members have one sub-tree replaced by one from the memory.

3.8 Elitism

An elitist strategy is employed whereby a single copy of the best member of generation *n* replaces a randomly selected member of generation *n+1* after all genetic operations have been performed.

4. Results

In the presentation of results from the GP system, the following parameters are used:

MAXGEN
the maximum number of generations in a run
MAXPOP
the maximum size of the population
MAXRUN
the maximum number of runs in a set
RXOVER
the crossover operator probability (%)
RMUTATE
the mutation operator probability (%)
RMEM
the memory operator probability (%)

4.1 Run parameters

A run consists of *MAXGEN* x *MAXPOP* individual tests. A set consists of *MAXRUN* runs. Results for several combinations of these parameter, each consisting of 1000 runs with 40000 tests per run, are presented. The other important parameters are the probabilities with which the 3 genetic operators are applied and denoted by *RXOVER* for crossover, *RMUTATE* for mutation and *RMEM* for memory.

A set of runs is characterised by these 6 parameters and represented by the notation

{1000,2000,20,95,20,20}

where

1000 = runs in the set
2000 = population size
20 = generations per run
95 = crossover probability (%)
20 = mutation probability (%)
20 = memory probability (%)

and the operator probabilities represent the percentage of the population that, on average, have one sub-tree replaced. Note that the 3 genetic operators are listed in the order in which they are applied.

4.2 Measuring the success of the system

The success of the system is measured by the number of runs in a set in which a hit is achieved. When a hit is achieved the run is terminated and the system re-initialised for the next run. In particular, the memory is emptied, i.e. the runs are all statistically independent.

The number of runs in a set is a trade-off between statistical significance and running time. Table 1 shows 3 sets of 100, 250 and 1000 runs, respectively, with each set repeated 5 times with different random number generator seeds. A single run has 40,000 tests. The spread is defined as *standard deviation/mean* and expressed as a percentage.

Runs		100		250		1000
	Seed	Hits	Seed	Hits	Seed	Hits
	12345	91	12345	214	12345	832
	23456	73	23456	192	23456	782
	34567	84	34567	194	34567	811
	45678	78	45678	206	45678	824
	56789	90	56789	204	56789	816
Mean		81.2		202		813
Spread		7.45%		4.01%		2.10%

Table 1 The effect of different random seeds and numbers of runs

All results in this paper were produced on a 300MHz personal computer capable of 40,000 tests/minute. A set of 1000 runs takes between 60 and 100 minutes depending on the number of hits. Taking these figures into account all results presented here were accumulated with 1000 runs per set.

4.3 Preliminary scan of the parameter space

A series of 1000-run sets was made with a population of 2000, a maximum of 20 generations per run and a fixed mutation rate of 20%. The purpose of this series was to explore the effect of different combinations of crossover and memory rates. The results are presented in Figure 1 which shows a contour map of this region of the search space. The figure plots the number of runs with hits in 1000 runs at different levels of crossover and memory use and a fixed level of mutation. The data shows that the system performs best at high crossover (90%-95%) and medium memory use (30%-40%). The gathering of the data for Figure 1 represents 148 hours of continuous running of the system.

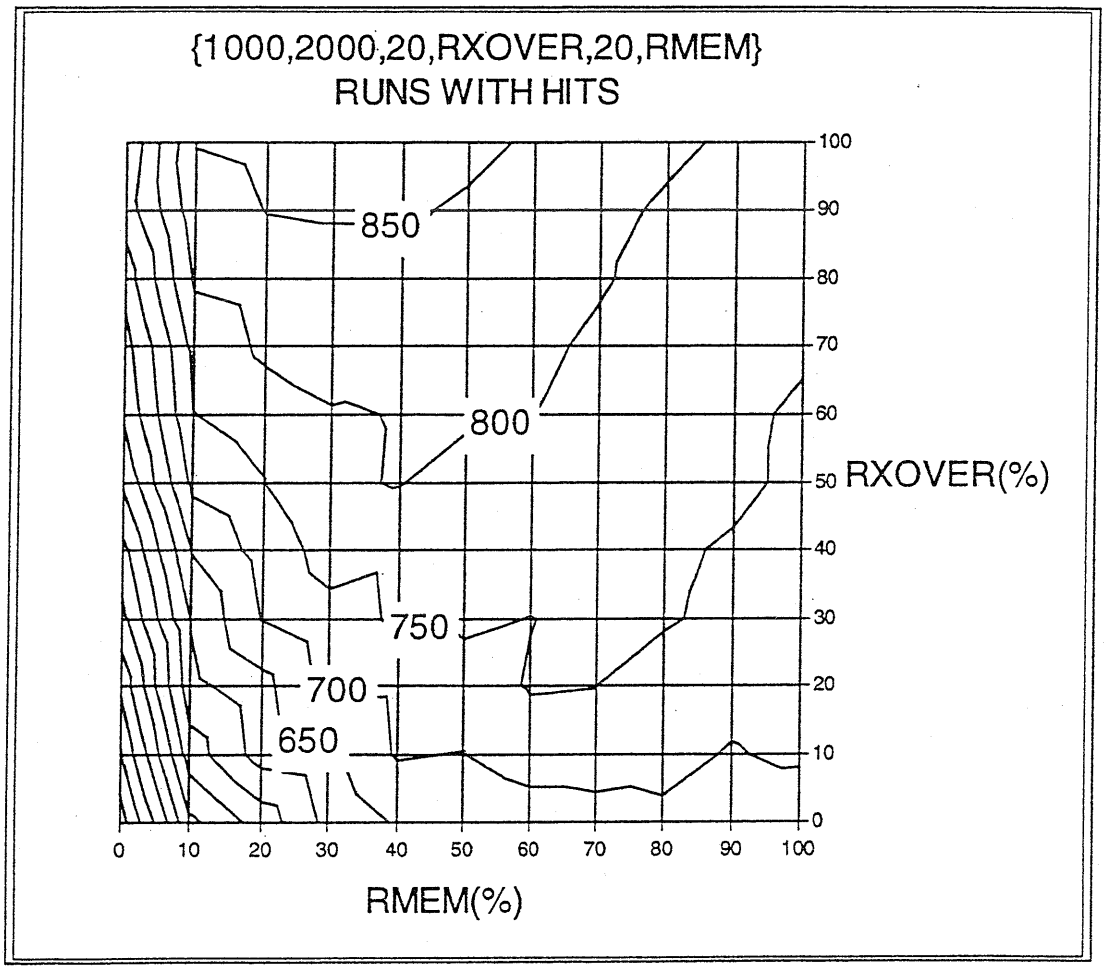

Figure 1 Runs with hits (out of 1000) at different crossover and memory probabilities and fixed mutation probability

It is worth noting here that the parameter set

{1000,80000,1,*,*,*}

i.e. 1000 runs with a population size of 80000 for 1 generation, representing an entirely random search over 80M tests yields no hits. The genetic operator probabilities (shown as '*') are then not relevant since no run proceeds beyond the first generation.

4.4 Variation of the genetic operators

In this section we present the results of applying the crossover, mutation and memory operators, singly or together, for a number of *population x generation* combinations each totalling 40000 tests per run. Table 2 and Figure 2 show the following situations:

100% crossover alone
100% crossover followed by 20% mutation
100% crossover followed by 35% memory
100% crossover followed by 20% mutation followed by 35% memory

for *population x generation* combinations ranging from 500 x 80 to 4000 x 10. Table 2 shows the total number of hits in a set of 1000 runs.

	4000 x 10	2000 x 20	1000 x 40	500 x 80
{100,0,0}	489	721	681	415
{100,20,0}	484	709	807	737
{100,0,35}	824	793	606	471
{100,20,35}	855	894	845	800

Table 2 Runs with hits (out of 1000) for different combinations of operators and different population/generation values

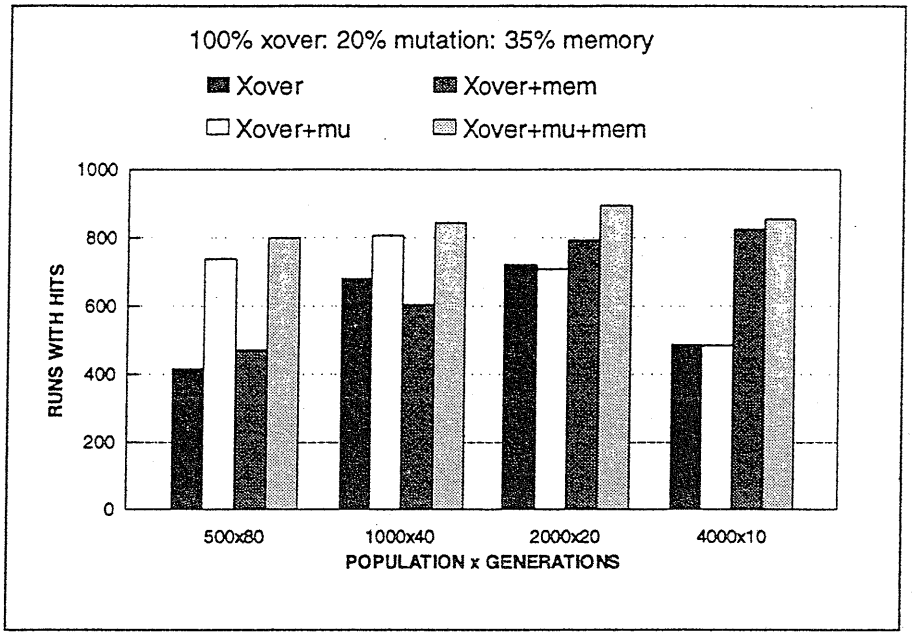

Figure 2

The impact of the different operator combinations may be summarised as follows:

Crossover alone

Reasonably good for population sizes 1000 (681 hits) and 2000 (721 hits).
Falls off at population size 4000 (489 hits) because 10 generations are not sufficient to achieve convergence.
Falls off at population size 500 (415 hits) because of insufficient genetic variety in the initial population with no means of adding to this.

Crossover + mutation

Mutation improves the hit rate at the lower populations, when the initial genetic material lacks variety, but has a slightly destructive effect at the higher population levels where it is not needed.

Crossover + memory

The memory operator is better than crossover alone in 3 of the 4 cases. It is also better than crossover + mutation at the two higher populations which are large enough to ensure that the memory contains good material. At the lower populations, with limited genetic diversity, the memory contents compete unfavourably with the random sub-trees produced during mutation alone.

Crossover + mutation + memory

The 3 operators together achieve high hit rates in all cases with little dependence on the population/generation mix. The reduced performance of the memory operator at lower populations is balanced by the improved performance of the mutation operator. The best result of 894 hits in 1000 runs was achieved with this combination although we believe that detailed tuning of the parameters is capable of a yielding a hit rate in excess of 90%.

These results show that the memory operator provides better performance than conventional mutation if the population size is sufficient to provide good memory material in the early generations. The two operators acting together further improve the hit rate and provide good results in remarkably few generations. For example 855 hits in 1000 runs result from a population size of 4000 evolved for only 10 generations.

5. Summary

We have proposed and investigated the use of a genetic operator based on short-term memory. It is used either as an alternative to mutation or as a supplement to mutation in restoring diversity in a population. When used instead of mutation it improves the hit rate of the GP system provided that the population size is sufficient to provide good genetic material for the memory . When used in combination with mutation the system performance is further improved and is maintained over a range of population sizes.

The objective function $ut + \frac{1}{2}at^2$ used as a vehicle to demonstrate the memory operator is a function of three variables and a non-trivial task for a symbolic regression system. The paper shows that the use of a memory operator, particularly when combined with crossover and mutation, improves the performance of the GP system for this particular problem. We plan to examine the generalization of this approach to other functions.

References

1. Koza J R, 1992. Genetic Programming: On the programming of computers by means of natural selection. MIT Press, Cambridge, MA.
2. Koza J R, 1994. Genetic Programming II: Automatic discovery of reusable programs. MIT Press, Cambridge, MA.
3. Angeline P J and Pollack J B, 1993. Proceedings of the 5th International Conference on Genetic Algorithms pp 264-270
4. Banzhaf W, Nordin P, Keller R E and Francone F D,1998. Genetic Programming - an introduction Morgan Kaufmann, San Francisco, CA.