A STUDY OF CROSSOVER OPERATORS

IN GENETIC PROGRAMMING

William M. Spears
Navy Center for Applied Research in AI
Naval Research Laboratory, Code 5510
Washington, D.C 20375-5000
and
Vic Anand
Massachusetts Institute of Technology
Cambridge, MA 02139

19950510 103

Abstract

Holland's analysis of the sources of power of genetic algorithms has served as guidance for the applications of genetic algorithms for more than 15 years. The technique of applying a recombination operator (crossover) to a population of individuals is a key to that power. Neverless, there have been a number of contradictory results concerning crossover operators with respect to overall performance. Recently, for example, genetic algorithms were used to design neural network modules and their control circuits. In these studies, a genetic algorithm without crossover outperformed a genetic algorithm with crossover. This report re-examines these studies, and concludes that the results were caused by a small population size. New results are presented that illustrate the effectiveness of crossover when the population size is larger. From a performance view, the results indicate that better neural networks can be evolved in a shorter time if the genetic algorithm uses crossover.

1. Introduction

Recently, two heuristic search techniques have generated interest in the artificial intelligence community: genetic algorithms (GAs) and neural networks (NNs). Both GAs and NNs are based on models from nature. A genetic algorithm is modeled on genetics and Darwinian evolution, whereas a neural network is based on models of human cognition. One common application of a genetic algorithm is as a function optimizer. Another common application of a genetic algorithm is to evolve organisms that perform well in a given environment. In either application, the GA is based on the survival-of-the-fittest (natural selection) tenet of

This document has been approved for public release and sale; its distribution is unlimited

DTIC QUALITY INSPECTED 5

Darwinian evolution. Neural networks, on the other hand, appear to be useful as control mechanisms for organisms themselves (e.g., an organism should avoid danger and seek food). These two methods naturally reflect a difference in scale. While a neural network can be used to control a particular organism, a genetic algorithm can be used to evolve a population of organisms (e.g., NNs) that perform well in a given environment. If a neural network is used to encapsulate a particular behaviour, then genetic algorithms can be used to evolve that behaviour, by evolving a population of neural networks.

One particular approach to the evolution of behaviour is described by de Garis [1]. In this approach, a GA is used to evolve a population of neural networks. Each NN has a set of adjustable weights and is used to encapsulate some desired behaviour (e.g., walking). In other words, once good weights have been found, the NN can be used by itself to perform the desired behaviour. However, since a set of good weights is not known in advance, they must be learned. Instead of the more traditional NN learning algorithms (e.g., backpropagation) de Garis uses a genetic algorithm to learn a set of good weights. No learning is being done by the neural network itself. This approach is called *genetic programming* [1].

As mentioned above, GAs evolve a population of individuals according to the process of natural selection. During this process, genetic operators create new (child) individuals from highly fit old (parent) individuals. Recombination (also referred to as *crossover* in this report) is one of the genetic operators and is a key to the power of the genetic algorithm [6]. In his studies of genetic programming, though, de Garis reports that a genetic algorithm without recombination outperforms a genetic algorithm with recombination. These results motivated us to re-examine genetic programming for two reasons. First, from a theoretical standpoint, we sought to explain what appear to be anomalous findings. Second, from a practical standpoint, we wished to use recombination (as theory suggests) to improve de Garis's results, allowing better behaviour to be learned in less time.

2. Genetic Algorithms

The book "Adaptation in Natural and Artificial Systems" [6], lays the groundwork for GAs. A genetic algorithm consists of a population of individuals that reproduce (over many generations) according to their fitness in an environment. Those individuals that are most fit are most likely to survive, mate, and bear children. Children are created by the stochastic application of genetic operators to the (parent) individuals. Individuals of the population, coupled with the genetic operators, combine to perform an efficient domain-independent search strategy that makes few assumptions about the search space.

Acq. Nbr
Codes
Dist | Avail and/or Special
A-1

Each individual in a population is a point in the search space. Traditionally, an individual in a GA is represented as a bit string of some length n. Each individual thus represents one point in a space of size 2^n. Given a bit string representation, a genetic algorithm will produce offspring using the genetic operators crossover (recombination) and mutation. Mutation operates at the bit level by randomly flipping bits within the current population. Mutation rates are low, generally around one bit per thousand.

Crossover operates at the individual level. A *crossover point* is randomly chosen for two randomly selected individuals (parents). This point occurs between two bits and divides each individual into left and right sections. Crossover then swaps the left (or the right) section of the two individuals. As an example of crossover, consider the two parents:

Parent 1: 1010101010
Parent 2: **1000010000**

Suppose the crossover point randomly occurs after the fifth bit. Then each new child receives one half of the parent's bits:

Child 1: 1010110000
Child 2: 1000001010

In genetic programming, a GA is used to evolve a population of increasingly fit neural networks. The fitness of each neural network is some measure of how well a desired behaviour is being performed. In order to use a traditional genetic algorithm for this task, a bit string representation of a neural network is required. Also required is a fitness function that indicates how well a neural network is performing a desired behaviour. These issues are addressed in the next section.

3. Neural Networks and Genetic Programming

The class of neural networks (NNs) is a subclass of parallel distributed processing (PDP) models [7]. PDP models assume that information processing is a result of interactions between simpler processing elements (e.g., neurons). In genetic programming, the number of neurons is set by the user. This number depends on the behaviour to be learned. The network is fully connected (i.e., each neuron has a connection to itself and all others) with real valued weights ranging from -1.0 to 1.0. The net input to each neuron (propagation rule) is simply the sum of the products of its inputs and their weights. Finally, the output of each neuron is simply its activation, and the activation rule uses the traditional sigmoid function [7].

Each genetic programming neural network (called a *GenNet* by de Garis) consists of a set of *input neurons*, *output neurons*, and optional *hidden neurons*. Input neurons are neurons

that receive information from the environment. Output neurons provide actions that affect the environment. Hidden neurons can be used to provide arbitrary transformations of the input stimuli.

Let us now consider an example from [1] that is also used for the experiments described in this report. We will refer to this example as the *Walker*. In this example, a GenNet is evolved to control a set of stick legs. The behaviour to be learned by the genetic algorithm is that of walking. The GenNet considered by de Garis consists of 12 neurons: 8 input neurons and 4 output neurons. The stick legs are composed of 2 legs joined at the hip. Each leg consists of a thigh and a calf, joined at the knee. This is represented as 4 segments (one for each calf and thigh). As the legs are moving, it is possible to sense the angle and angular velocity of each segment. This information is sensed by the 8 input neurons. The 4 output neurons are used to represent the angular acceleration to be sent to each segment (see Figure 1).

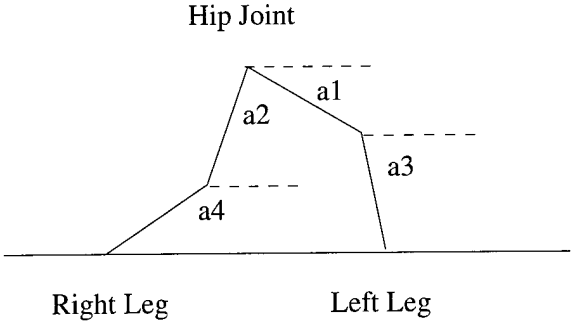

Figure 1. The Stick Legs

In this example, it is possible to use the network as a controller for the stick legs. For any state of the inputs (i.e., angles and angular velocities), the GenNet will output a set of angular accelerations that controls the stick legs. These angular accelerations will change the angles and angular velocities of the segments, providing new inputs to the GenNet. For computational purposes, the GenNet is used as a discrete time simulation in which the stick legs move in small time increments. Given an arbitrary setting of weights in the GenNet, the stick legs will move in a random fashion. However, with the proper weights it is possible for the GenNet to control the stick legs in such a way as to perform the behaviour of walking [1]. The goal is to learn the appropriate weights.

As mentioned earlier, a genetic algorithm is used to learn the weights of the GenNet. This requires a bit string representation of the neural network and a fitness function that quantifies how well the behaviour of walking is achieved. In genetic programming, each real valued weight is represented by a set number of bits b [1]. Since each neuron is connected to all others (including itself), the weights of an n neuron network can be stored in an n by n matrix. Each element of the matrix stores the weight of one connection. Each matrix specifies

a GenNet uniquely and can be represented as a bit string in row major order. Each individual of the GA, then, represents the linearized weight matrix [1]. If there are n neurons in the network, bn^2 bits are required for each individual. For the stick legs, de Garis represents each weight by 7 bits, resulting in individuals that are 1008 bits long, since there are 12 neurons in the network.

The fitness function (used to evaluate each individual described above) should accept each individual in the population and return some quantification of how well the behaviour of walking is being performed by that individual. One straightforward fitness function is to compute the distance traveled by the stick legs [1]. Better individuals travel farther (e.g., walk better) and worse individuals travel less far. The goal is to find a set of weights such that the resulting GenNet will produce good walking behaviour. The following sections outline how well genetic programming succeeds in achieving this goal.

4. Genetic Programming and Crossover Experiments

As mentioned earlier, a genetic algorithm evolves a population of individuals according to some fitness function (environment). Theory indicates that the size of the population is crucial to the performance of the genetic algorithm [3]. Small populations generally find good solutions quickly, but are often stuck on local optima. Larger populations are less likely to be caught by local optima, but generally take longer to find good solutions. Another way to view a population is as a source of statistics about the environment. A larger population allows the space to be sampled more thoroughly, resulting in more accurate statistics. Small populations sample the space less thoroughly, producing results with higher variance.

In his experiment with the Walker problem, de Garis reported that a GA without recombination outperformed a GA with recombination [1]. This is surprising, since an important key to the power of a genetic algorithm is the recombination operator [3, 4, 6]. A genetic algorithm without recombination should amount to little more than random search. However, after examining the program, we noticed that de Garis ran his GA with an extremely small population (20). Our hypothesis, then, was that the small population size produced highly variable results, due to the extremely sparse sampling of the search space. Assuming this hypothesis to be true, we felt that we should be able to explain the reported results. Furthermore, we felt that a larger population size would produce a more accurate sampling of the space, allowing recombination to perform as theory indicates [6]. From a practical standpoint, this should improve the efficiency of the GA search, resulting in improved performance. To test our hypothesis, we ran two experiments. In an attempt to explain the results reported by de Garis, the first was run with a population of size 20. In an attempt to show that a larger population size allows recombination to be more effective, the second was run with a larger

population size (100). Each experiment was run with two GAs, one written by de Garis (referred to as the *GenNet GA*), and one (developed by Spears) that is used as a control (referred to as *GAC*). The purpose of the control is to indicate the generality of the results. Since the two GAs differ in how they choose individuals for survival, similar results indicate generality.

In the earlier example of crossover (Section 2), a single crossover point is selected. This is referred to as *one point crossover* and was the crossover used by de Garis. Since the emphasis of this report is on the effectiveness of crossover, it is appropriate to also consider other forms of crossover. We chose two other common forms: *two point crossover* and *uniform crossover*. With two point crossover, two crossover points are selected. In this case, the parents only swap the bits between the two crossover points. As an example of two point crossover, consider the two parents defined earlier. Suppose the two crossover points randomly occur after the third and sixth bit. Then the children are:

Child 1: 101**001**1010
Child 2: **100**010**0000**

Uniform crossover does not select a set of crossover points. It simply considers each bit position of the two parents, and swaps the two bits with a probability of 50%.† Suppose the first, third, fourth, and ninth bits positions (of the original parents) are swapped. Then the children are:

Child 1: **1000**101000
Child 2: 1**0**1**0**010**0**1**0**

Spears [8] has analyzed the effects of crossover on genetic algorithm performance. In this study it is shown that (with large search spaces) a GA using uniform crossover outperforms a GA using one point crossover, which in turn outperforms a GA using two point crossover. In general, however, theory suggests that a GA with crossover should outperform a GA that does not use crossover (i.e., mutation alone).

5. Experimental Results

There were two experiments, both applied to the Walker problem. A single experiment compared a GA with and without recombination. The mutation operator was always used, and was the only genetic operator used when the GA did not use recombination. There is one graph for each experiment. Each graph shows four curves, one for a GA without

† Spears [8] considers uniform crossover with probabilities other than 50%.

recombination, and one for a GA with each of the three forms of crossover defined above. The horizontal axis of each graph is a measure of the amount of work done by the GA. For genetic programming, the evaluation of one GenNet constitutes the bulk of the computation. Each horizontal axis terminates after 10,000 evaluations. When the GenNet GA was run, a point was plotted for every generation. In this case the number of evaluations is the product of the number of generations and the population size. The vertical axis indicates the fitness of the best individual seen, measured by the fitness function. Since we are maximizing, a higher curve represents better performance. Each curve is an average of 10 independent runs. To indicate statistical significance, we also show error bars for uniform crossover and mutation, at the end of the runs. These error bars are two standard deviations in total height.

The first experiment was an attempt to understand the results reported by de Garis. The original GenNet program was written in C for the Macintosh [2]. For purposes of speed we ported this program to a Sun 4 Workstation. We then modified the program to handle one point, two point, and uniform crossover. Figure 2 indicates the results of running the GenNet GA with a population of size 20.

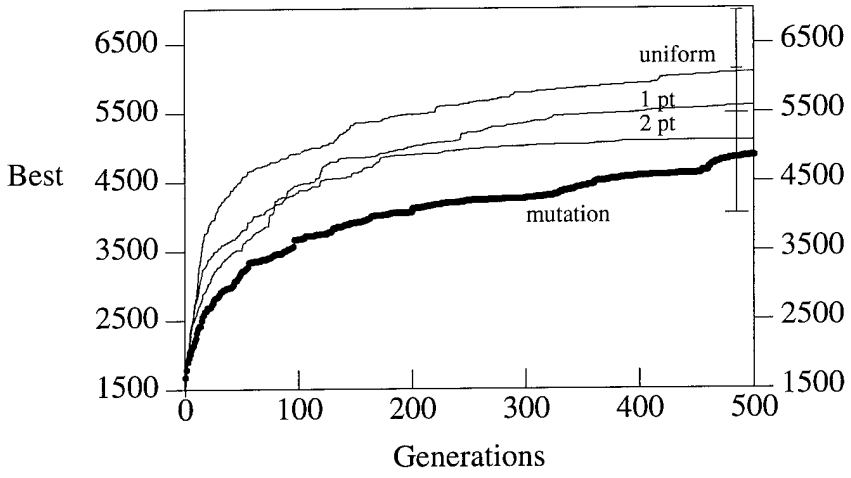

Figure 2. GenNet with a population of size 20

Figure 2 indicates that although all forms of crossover outperform mutation alone (no recombination), the differences are not dramatic. In fact, comparisons of individual runs (not shown here) indicate that a run without crossover (but with mutation) can easily beat a number of runs with crossover. This explains de Garis's results.† The lack of predictable effectiveness is due to the large variance caused by the poor sampling of a small population size. Crossover requires good sampling information in order to be effective [3]. However, despite high variance (note the error bars), when averaged over 10 runs, mutation loses. The results with GAC are similar.

† Due to computational limitations, de Garis was only able to make one run [2].

Apparently, crossover is hampered by a small population size (i.e., poor sampling). A natural way to improve the sampling is with a larger population. If our hypothesis is correct, crossover should become more effective with a larger population size. We reran the previous experiment with a population of size 100. Figure 3 presents the results.

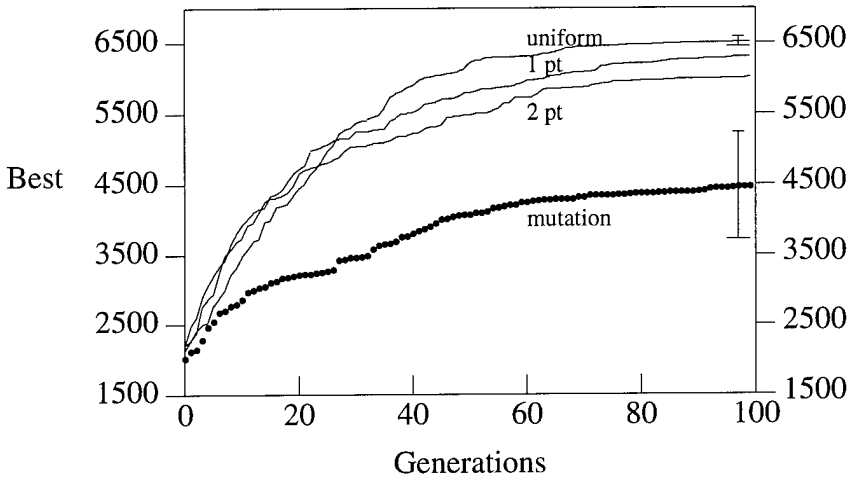

Figure 3. GenNet with a population of size 100

Figure 3 indicates that, with a population of size 100, all forms of crossover dramatically outperform mutation alone. Again, the results with GAC are similar. It is also interesting to note the dramatic reduction in variance (especially for uniform crossover). This is due to the better sampling afforded by a larger population size. This confirms the hypothesis that crossover is more effective with a larger population.

It is also important to note the relative performance of the crossover operators in Figures 2 and 3. In both cases, uniform crossover outperforms both one and two point crossover. This is expected, since uniform crossover more strongly encourages recombination.

In summary, as mentioned earlier, this study was motivated for two reasons. First, we wished to explain why de Garis found that a GA without recombination outperformed a GA with recombination. We hypothesized that the result was caused by a small population size. The first experiment confirmed this hypothesis. The second experiment confirmed the effectiveness of recombination with a larger population.

The second reason for this study was to improve the performance of genetic programming, allowing better neural networks to be evolved in less time. In the context of the Walker problem, this has been done. The experiments indicate that genetic programming with recombination consistently outperforms genetic programming without recombination on large populations. More importantly, despite higher statistical variance, genetic programming with

recombination generally outperforms genetic programming without recombination on small populations. Finally, the greatest performance improvements were obtained by using uniform crossover.

6. Discussion and Conclusions

The interacting roles of population size and crossover are of extreme interest to the genetic algorithm community. Recent theory indicates that crossover is most effective when the sampling is representative of the space being searched [3]. This is achieved in a natural manner by using a larger population. Theory would indicate, then, that recombination is more effective with larger populations. Theory also indicates that uniform crossover is extremely useful in those situations where recombination is important (e.g., when the search space is very large). This study supports these views, although it should be emphasized that the study was limited to one particular genetic programming problem. Future work will extend this analysis to a broader class of problems.

It is also interesting to note that although quick performance improvements occur with a smaller population size, a larger population helps the genetic algorithm find better solutions. This is caused by the slower accumulation of more accurate statistics when using the larger population. Clearly the fast accumulation of accurate statistics would allow the best of both worlds. Future research will focus on such ideas.

Finally, it should be noted that the above observations hold qualitatively for both GAs studied, despite differences between the GAs. This indicates a generality to the observations that could not be assumed otherwise. Recent theory has addressed this issue and indicates that GAs are robust with respect to many implementation details [5]. This report provides some confirmation of this theory.

References

[1] de Garis, H. (1990a). *Genetic Programming: Building Nanobrains with Genetically Programmed Neural Network Modules,* Proceedings of the International Joint Conference on Neural Networks, San Diego, CA, June 1990.

[2] de Garis, H. (1990b). *Personal Communication.*

[3] De Jong, K. A. and William M. Spears (1990). *An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms,* in the International Workshop

Parallel Problem Solving from Nature, University of Dortmund, Oct. 1-3, 1990.

[4] De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems,* Doctoral dissertation, Dept. Computer and Communication Sciences, University of Michigan, Ann Arbor.

[5] Grefenstette, J. (1990). *Conditions for Implicit Parallelism,* Proceedings of the Foundations of Genetic Algorithms Workshop, Bloomington, Indiana, 1990.

[6] Holland, John H. (1975). *Adaptation in Natural and Artificial Systems,* The University of Michigan Press.

[7] McClelland, James L. and David E. Rumelhart (1988). *Explorations in Parallel Distributed Processing,* The MIT Press, Cambridge, MA.

[8] Spears, William M. and K. A. De Jong (1991). *On the Virtues of Uniform Crossover,* to appear in the 4th International Conference on Genetic Algorithms, La Jolla, California, July 1991.