CROSSOVER CONTEXT IN PAGE-BASED LINEAR GENETIC PROGRAMMING

G.C. Wilson, M.I. Heywood
Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 1W5
{gwilson, mheywood}@cs.dal.ca

Abstract

This work explores strategy learning through genetic programming in artificial 'ants' that navigate the San Mateo trail. We investigate several properties of linearly structured (as opposed to typical tree–based) GP including: the significance of simple register based memories, the significance of constraints applied to the crossover operator, and how 'active' the ant are. We also provide a basis for investigating more thoroughly the relation between specific code sequences and fitness by dividing the genome into pages of instructions and introducing an analysis of fitness change and exploration of the trail done by particular parts of a genome. By doing so we are able to present results on how best to find the instructions in an individual's program that contribute positively to the accumulation of effective search strategies.

***Keywords**: Genetic Programming, Strategy Learning*

1. INTRODUCTION

The Artificial Ant problem (wherein the ant navigates a trail of food) is considered to be a challenging GP benchmark problem whose difficulty level is comparable to real problem spaces [1]. The San Mateo Trail version of the problem is the harder of the artificial ant problems [2]. In the traditional form of the San Mateo Trail problem, the trail consists of nine 13 x 13 toroidal grids wherein the trail may have five different types of discontinuities [2]. Koza, to save computer time, altered the problem: the trail is divided into 9 parts (fitness cases) of 13 x 13 grids, the grids are not toroidal, touching the edges of the non-toroidal grids terminates a fitness case, and moving onto a square containing food throws program execution back to the beginning. The ant can spin left or right 90 degrees, or move forward. There are 96 pieces of food altogether on the trail, and an ant's fitness is the number of food pieces eaten. An ant progresses between consecutive parts of the trail (fitness cases) if the ant either touches the electrical fence (it touches the edge), makes a total of 120 right or left turns, or moves 80 times. The ant always begins each 13 x 13 grid in the middle of the top row, facing south. The food eaten in each of the nine sections of the trail is totaled, and the ant's goal is to eat as much food as possible.

In section 2 we describe the specifics of our linear GP implementation of the San Mateo Trail problem. In section 3 we describe the concept of computational effort (E-measure) and compare our E-measure results with those of Koza for this problem. In section 4 we introduce the quantitative concepts of fitness and non-repeating path and examine which parts of the genome contribute to performance gain. This motivates changes that could lead to possible improvement, and the results after these changes were implemented are discussed in section 5. Finally, conclusions of our work are given in section 6.

2. PAGE BASED LINEAR GP

We follow Koza's description of the problem as closely as possible, and we feel a sufficient match between implementations is provided to allow comparison of results. Koza's implementation used tree-structured GP (T-GP) with and without ADFs [2], whereas the solution investigated here utilizes page-based linearly structured GP (L-GP) without ADFs [3]. Moreover, implicit in L-GP is the concept of registers. This leads to the introduction of additional constraints to restrict the number of times that the ant could load information into registers on a particular level of the trail. The limit used here is the maximum number of instructions per individual. This essentially prevents an ant from living forever by developing a program consisting of only memory load instructions (that is, consisting entirely of instructions which start with "0").

Thus, if an ant can die from either turning or moving too many times, it will do that before it will die of too many load instructions. Furthermore, we use a steady state tournament instead of Koza's generational tournament approach. In his tournament, he used a population of 4000 individuals and allowed up to 50 generations. That is, a total of 200 000 individuals are processed per run. In comparison, when using a steady state tournament of size 4, an equivalent number of evaluations per run is given by a tournament limit of 50,000. We use the same terminal set as Koza, that is, RIGHT, LEFT, and MOVE. In an instruction, "00" means RIGHT, "01" means LEFT, and "10" or "11" means MOVE. We accomplish something like Koza's IF-FOOD-AHEAD function implicitly in the program through the interpretation of any instruction that begins with the bit "1" (a fetch-from-memory instruction).

A population of 125 ants is involved in our steady state tournament. In each round of the tournament, the two individuals with the highest fitness are preserved (become the parents) and the two with the lowest fitness are the children, which can be subject to the operators of swapping, mutation, and/or crossover. If there is a tie between two or more individuals regarding whether they will become parents or not, the one(s) that become parents are the one(s) first chosen to compete. Since the 4 ants were chosen to compete by random selection, those that become parents in this situation are thus essentially chosen randomly.

Mutation of a randomly chosen instruction occurs with a probability of 0.5, crossover of a random page between the two children in each tournament occurs with probability 0.9, and the probability of swapping is also 0.9. Mutation is done by bitwise XORing a randomly generated string of instruction length with the chosen instruction. Crossover is done with pages, or groups of adjacent instructions. The size of the pages involved in crossover is determined using dynamic paging, a process described below.

Each individual consists of a 4 or 8 register memory and an instruction set of 12-bit instructions. The instructions sets of the ants are divided into pages to give the crossover operators context. The size of the instruction set is a maximum number of pages specified by the user (16 or 32 pages in our experiments) multiplied by the number of instructions allowed per page. The number of instructions per page has a maximum of 4 but varies throughout the run of the program, as dynamic paging is employed. The dynamic paging prevents plateaus in fitness levels from forming prior to the maximum fitness level of 96 being achieved. This is done by doubling the number of instructions allowed per page (or setting it to 1 when the maximum page size allowed is the current working page size) whenever the maximum fitness remains the same over 10 rounds of the tournament. Dynamic paging essentially results in kick-starting the process of solving the trail if the ants begin to settle at a sub-optimal fitness before eating all 96 pieces of food.

The way the instructions are interpreted depends on the number of registers in the ant's memory, but 12 bits were given per instruction to allow for future flexibility in the interpretation of the ant's instructions. The ants are of two major varieties that affect how they interpret the strings of bits comprising their instructions. One variety is called the "hyper" ants. When these ants encounter an instruction telling them to load information into their memory, they both load it and immediately act on the instruction loaded. The other variety is called "thoughtful." These ants simply load the information in a load instruction into the relevant register, but they do not immediately act on the command loaded. (The motivation behind the two varieties was to see if having the ant take more action would mean solving the problem more quickly.) Upon initialization, the ants were given a number of instructions that was a multiple of 4 (the initial number of instructions per page), with the page limit chosen uniformly over the interval [1, …, MaxPages] (16 or 32). All instructions were initialized to random bits. All registers in the ants' memories were filled with "00," meaning RIGHT.

3. E-MEASURE

The computational effort (called "E-measure") of a GP algorithm can be measured using a formula described by Koza [4],

$$E = T \times i \times \frac{\log(1-z)}{\log((1-C(T,i))}$$

where *T* is the size of the tournament, *i* is the generation where an individual solved the trail, *z* is the probability of success (set to 0.99 in our experiments), and *C(t, i)* is the cumulative probability of having an individual solve the problem in the experiment. Note that when *C(t, i)* is 1.0, that is, when the cumulative probability of having an individual solve the trail is 100%, the formula is not valid. In this special case, the formula becomes simply

$$E = T \times i \times 1$$

Note that the special case will only occur in the last trial of an experiment if and only if all trials (including the last) were successful.

A total of 20 runs were conducted in each experiment, and four experiments for each of the hyper and thoughtful ants (8 experiments in total) were conducted using the implementation details described in section 3. The programs were compiled using the Java 2 version 1.3.1 SDK. The four experiments involved ants with combinations of these characteristics: 4 register memory with 16 page maximum, 4 register memory with 32 page maximum, 8 register memory with 16 page maximum, and 8 register memory with 32 page maximum. Tables 1 and 2 indicate the generation at which the hyper and thoughtful ants, respectively, solved the trail in each trial of the experiments. Table 3 indicates the minimum and average E-measure results based on Tables 1 and 2. We found that thoughtful ants of all varieties completed the trail 100% of the time.

Table 1. Generation at which Hyper Ants Successfully Completed the Trail

	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	1823	2237	1799	915
Mean	6173	13901	9217	11413

Table 2. Generation at which Thoughtful Ants Successfully Completed the Trail

	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	1497	310	282	2555
Mean	7086	6744	5794	10207

Table 3. E-measure (×1000) Results for Hyper Ants, Thoughtful Ants, and Koza's Ants for z = 0.99

Hyper Ants				
	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	174	436	153	167
Mean	280	574	278	276
Thoughtful Ants				
	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	98	74	57	130
Mean.	159	117	148	247
Koza's Ants				
	With ADFs		Without ADFs	
Effort*	136		272	

*Based on Koza's generational tournament version of E-measure [2]

$$E = M \times R(z) \times (i+1)$$

where M = 4000 is population size, i = 16 is the generation, and R(z) is the number of trials to solve a problem by generation i with probability z = 0.99.

We can see from the above table that the best E-measure results for the thoughtful ants, in all cases, outperform Koza's ants with or without ADFs. Also of note in these results is that the thoughtful ants' mean E-measure, in all cases, perform better than Koza's ants that lack ADFs. (Our ants, by design of the L-GP solution, lack ADFs.) The thoughtful 4 register, 32 page limit ants even outperform (based on mean E-measure) Koza's ants that use ADFs. With the exception of the 4 register, 32 page maximum, the hyperactive ants (as a group) are competitive with Koza's ants.

Comparing our ants with one another, we noticed that the thoughtful ants simply demand less computational effort than the hyper ants. Moreover, the thoughtful ant group with the worst performance (using either the minimum or mean E-measure) still does better than the best hyper ant group (considering either the minimum or mean E-measure).

4. THE MOTIVATIONS FOR FURTHER ALTERATION OF THE ALGORITHM

We decided to focus our analysis on the thoughtful ants in light of the results of Section 4. When we decided to take a look at the way the solutions that led to the above results were generated by the thoughtful ants during the execution of the algorithm, we suspected we could further enhance the ants' performance. We performed this analysis of how a particular section of the genome contributed to performance gain by using two measures: fitness and the notion of a non-repeating path. Since context of instructions was insured within pages already, the next step could be to associate higher-level good search strategies (measured by fitness gains and exploration) with broader ranges of instructions in the genome.

4.1 The Notion of Fitness Change

Fitness was incremented for each instruction, iteration pair when the instruction in that pair resulted in a new piece of food being eaten. The trials of the thoughtful ant experiments all exhibited a similar pattern, as seen in the graph for a typical trial shown below.

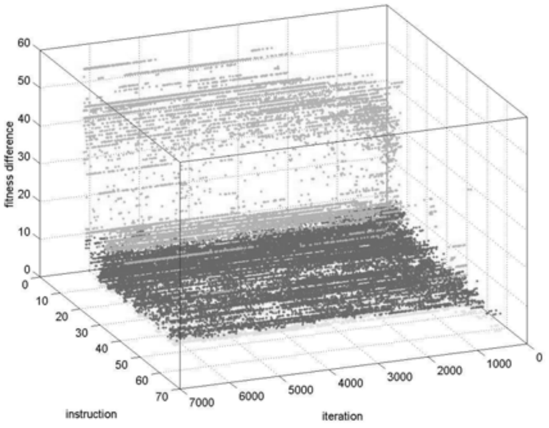

Fig. 1. Fitness change for which each instruction, iteration pair is responsible. The behavior seen above was for thoughtful 4 register ants with a 16 page limit. Fitness changes of 1 are light gray, those <= 10 and > 1 are dark gray, and those > 10 are medium gray.

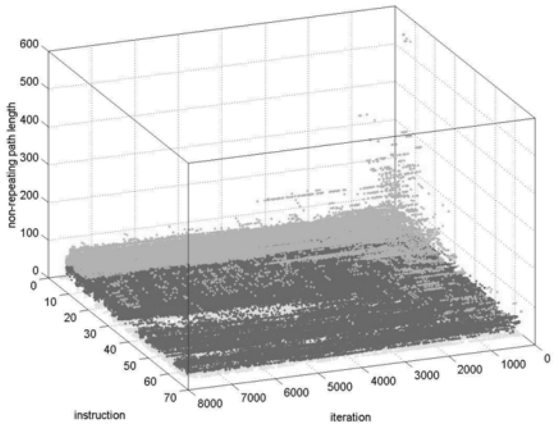

Fig. 2. Increase in non-repeating path corresponding to each instruction, iteration pair. The behavior seen above is for thoughtful 4 register ants with a 16 page limit. Non-repeating paths of length <= 10 are light gray, those > 10 and <= 50 are dark gray, and those > 50 are medium gray.

4.2 The Notion of Non-repeating Path

Non-repeating path is designed to measure whether or not an instruction during a particular iteration is contributing to an ant moving to a new area of the grid that it has not yet explored. An array representing the 13 x 13 level of the trail the ant is currently on has an "x" placed in one of its spaces when the ant visits it. If the ant eats a piece of food, the grid is flushed clean. If the ant visits a square already marked with an "x," then the non-repeating path of the ant is set to 0. If the ant moves and it has not visited that square yet, the length of the non-repeating path is incremented. For instructions where the ant spins left or right, we employ a FIFO stack to determine if the ant is facing a direction it has already faced on a particular square. A typical trial result for non-repeating path change can be seen below. Load instructions in thoughtful ants result in no change in non-repeating path, since no moving or spinning of the ant is actually done.

The trend seen in the fitness graph of Figure 1 was typical of all trials of the thoughtful ants, whereas the trend seen in Figure 2 for non-repeating path was prevalent, but not as typical as the fitness trend. We noted that fitness (and in most cases, non-repeating path) change tended to occur in the first portion of the ants' instruction set. Moreover, the high count of fitness difference early on, in combination with low non-repeating path lengths, appears to indicate that good search strategies are identified.

5. TWEAKING THE GP ALGORITHM

The San Mateo Trail problem is an example of a strategy-learning problem in which instruction sequence is significant. That is to say, if food is not located by the first ten moves, then it is likely that a different search strategy will be more successful than that currently being perused, Figure 1. This might have implications for the manner in which GP search operators are applied. Moreover, in the case of Genetic Algorithms applied to the job-shop scheduling problem using an indirect encoding scheme, significant benefit was observed in utilizing biased crossover and mutation operators [5]. Specifically, the authors reasoned that mutation does little good in later parts of the genome that are just being explored while the earlier part of the genome has converged, so they felt it would increase fitness of the individuals to bias mutation to occur earlier in the genome. The authors also felt that crossover should be biased toward the end of the genome, for crossover done on the earlier, more stable portion of the genome would not provide any change in schemata fitness. Furthermore, crossover near the end of the genome would lead to better exploitation through placing different schema in different contexts.

In order to investigate whether our solutions may also be suffering from premature convergence and inadequate exploitation of crossover in later parts of the genome, we biased mutation to work on earlier parts of the genome and crossover to work on later parts of the genome. The biasing was accomplished by associating a probability with each instruction that was equal to the cumulative fitness change at that instruction divided by total fitness change for all instructions in the genome. If two instructions had the same probability associated with them, mutation would affect the earlier instruction if the given probability were selected. Crossover applied the same methodology; only probabilities for pages were found by totaling probabilities of all instructions in the page. If pages had the same probability and were to be affected by crossover, the later page would be selected. The results can be seen in Table 4 below.

Table 4. Generation at which Thoughtful Ants Biased for Mutation and Crossover Successfully Completed Trail

	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	582	1953	1589	1324
Mean	8795	8937	6606	11929

As we can see by comparing Table 2 and 4, the biasing of both mutation and crossover seemed to mostly result in the better or mediocre solution-finding time of the thoughtful ants to get worse, and the bad times to get better. Believing the time for solution generation to have gotten worse due to the affect of biased mutation on the earlier portion of the genome that caused the beneficial fitness gains in the first place, we attempted to only bias crossover to affect the later part of the genome. These results can be seen in Table 5 below. As can be seen by comparing Table 2 and 5, this change did not cause a solution to appear any faster. (The fastest solution for the thoughtful ants was accomplished after 282 rounds; here the fastest solution appears after 619 rounds.)

Table 5. Generation at which Thoughtful Ants Biased for Crossover Successfully Completed Trail

	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	619	792	2488	2000
Mean	5186	7350	7747	9885

Table 6. E-measure ($\times$1000) Results for Hyper Ants, Thoughtful Ants, and Koza's Ants for z = 0.99

Ants Biased for Mutation & Crossover				
	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	134	116	132	160
Mean	209	264	221	275
Ants Biased for Crossover				
	4 registers		8 registers	
	16 pg	32 pg	16 pg	32 pg
Min.	71	112	111	160
Mean.	148	175	247	272

With respect to Table 6, the best E-measure of the trials did not improve following these changes (with the exception of the 4 register, 32 page limit ants' minimum E-measure, the significance of which is not apparent to the authors). In all types of thoughtful ants, the mean E-measure did not improve.

6. CONCLUSIONS

We present a linear GP implementation of the San Mateo Trail problem using a steady state tournament that outperforms the generational, tree-based GP implementation discussed by Koza with respect to best E-measure. We used two basic types of ants on the trail, and the thoughtful ants outperformed the hyper ants with respect to both how fast the solution to the trail was discovered (minimum and mean) and E-measure (minimum and mean) for the 20 trials in each experiment.

Analyzing the way the ants generated their solutions, we introduced the notions of fitness change and non-repeating path. Examining the graphs, we attempted to further enhance performance of the ants by both biasing the first part of each ant's genome for mutation and biasing the later part for crossover. Noticing no overall improvement, we attempted to only bias the later part of the genome for crossover. Again, no significant performance gains were noted.

While the operator-biased ants were not as successful as our original non-biased ants at yielding faster methods of producing solutions with less effort, the performance gains realized by the thoughtful ants indicate promise. Future work will address whether linear GP with a classical crossover operator provides similar distribution of fitness change and non-repeating path.

ACKNOWLEDGEMENTS

We gratefully acknowledge the funding provided by the Izaak Walton Killam Memorial Scholarship and NSERC-PGSA (Garnett Wilson) and NSERC research grant (Dr. M. Heywood).

REFERENCES

[1] W.B. Langdon and R. Poli, "Why Ants are Hard," *Technical Report: CSRP-98-4*, Univeristy of Birmingham, Birmingham, pp. 1-16, January 1998.

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

[3] Heywood M.I., Zincir-Heywood A.N., "Page-based Linear Genetic Programming" *IEEE Transactions in Systems, Man and Cybernetics – Part B*, 32(3), June 2002.

[4] Koza, J.R., *Genetic Programming: Automatic Discovery of Reusable Programmes*, Cambridge, MA: MIT Press, 1994.

[5] Hsaio-Lan Fang, Peter Ross, and Dave Corne, "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 375-382, 1993.