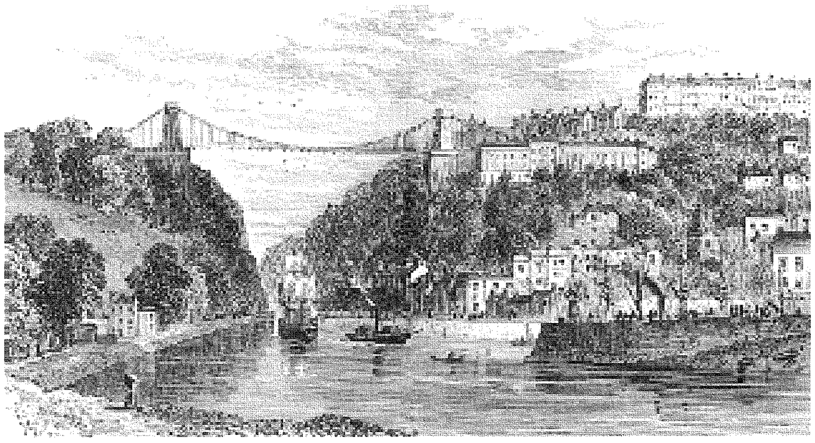
Adaptive Computing in Design and Manufacture 2008

ACDM 2008
APRIL 29th - MAY 1st, 2008
ENGINEERS HOUSE, CLIFTON, BRISTOL, UK



Proceedings of the Eighth International Conference

The integration of evolutionary, stochastic and computational intelligence technologies with design and manufacture processes

Edited by: I.C. Parmee

Canonical Representation in Genetic Programming

K. Bearpark and A.J. Keane
School of Engineering Sciences
University of Southampton, Southampton, SO17 1BJ
Andy.Keane@soton.ac.uk

Abstract

This paper explores the effect of different forms of representation and different forms of genetic operations on the performance of a Genetic Programming (GP) system. The GP system is based on a Genetic Algorithm (GA) that evolves software agents, represented by tree structures, which are then applied to some problems in robotics. In their evolved form, the trees are not well-formed, and many of them include a considerable amount of intronic material that plays no part in the performance of the underlying agent. We introduce an alternative way of representing the agent, which eliminates the intronic material and reflects more clearly the decisions that the agent needs to make in different situations as it attempts to solve problems in spatial awareness. We use the term *canonical* to refer to this alternative tree structure. We then extend the standard GP crossover operator to perform canonical crossover, in which the parents exchange canonical sub-trees.

The paper shows that using canonical evolution alongside conventional evolution can result in substantial improvements in the performance of the GP system and thus the resulting agents, the level of improvement being dependent on the mix of canonical and non-canonical members of the population, and on the rates and types of crossover.

1. Background

Algorithms to model natural evolutionary processes were developed in the early 1970s by Holland and his co-workers [Holland 1975]. Holland originally used the term *adaptive plan* for the process whereby a structure is progressively modified to improve performance within a particular environment. The importance of epistatic sets of alleles was reflected in this early work by Holland's definition of *schemata.* A schema is an interacting co-adapted set of feature values that results in significant gains in performance. The concept of a schema recognises the fact that highly fit structures have common components with particular alleles at particular feature positions in the structure. In Holland's view an important aspect of an adaptive plan is the identification and profitable use of schemata.

Holland also emphasised the need for an adaptive plan to be efficient in the sense that it should produce structures with acceptable fitness in a reasonable time, where 'reasonable' depends on the nature of the problem and on the environment. He uses the term *enumerative plan* to describe an adaptive plan that systematically identifies and tests every structure in the search space. An enumerative plan is out of the question for all but the simplest problems and a major part of any study of the adaptive process must be the discovery of factors that improve efficiency.

The term *adaptive plan* was soon replaced by the term *genetic algorithm* and evolutionary computation was born. The field has developed since that time to the point where genetic algorithms are applied to a wide range of problems in a variety of disciplines [Chambers 1995, Davis 1991].

In the early 1990s, Koza applied genetic algorithms to the evolution of computer programs and founded the discipline of genetic programming [Koza 1992]. The automatic production of computer programs is a long-held goal in computer science that, if it were to be reached, would have a significant impact on the world economy. Genetic programming (GP) has made a small step in the right direction. In our view the most important impact of genetic programming lies in extending the use of genetic algorithms to evolve structures of increased complexity which demand more sophisticated means of representing the structures and more sophisticated adaptive plans to improve their performance.

Koza follows Holland in recognising the importance of epistatic sets of feature values in adaptive systems. He further recognises the need to preserve these sets by protecting them against disruption by genetic operators. One of the main effects of genetic operators is to redistribute alleles within the population – a process that can destroy the benefits of hard-won sets of interactive feature values.

Koza defines an encapsulation operation by taking a component of a structure and defining it as a new indivisible feature that can be protected against disruption [Koza 1992: p.110]. The protected feature also becomes available for use in subsequent stages of evolution. Koza also introduces the concept of *Automatically Defined Functions (ADFs)*. These are functions defined automatically during a run of a GP system and evolved in parallel with the evolution of the main population [Koza 1992: p.534]. These functions are added to the function set of the main population for use in the later stages of evolution. A second work by Koza fully explores and exploits the ADF concept [Koza 1994]. Angeline and Pollack describe *module acquisition* in which a structural component is chosen and a part of the component selected as a module [Angeline 1993]. Those parts of the component below the module are considered arguments to the module. The new module is placed in a library where it can be referenced by individuals in the population. Banzhaf *et al.* review these and other modularization techniques and point out that they have in common the protection of modules of code against the potentially disruptive effect of crossover [Banzhaf 1998]. The present work focuses on a different means of providing such protection which is both effective and intuitively meaningful.

This paper is laid out as follows: in section two we introduce the basic ideas of how GP systems work, noting the traditional representation scheme we start from and the class of problem being tackled. In section three we provide some illustrative results and introduce the concept of the canonical representation. We then provide detailed results of the use of canonical representations in section four and discussion in section five before drawing our conclusions in section six.

2. Introduction

A Genetic Algorithm (GA) evolves data structures that may be applied to solve problems in a number of fields. The GA uses techniques based on the natural evolution of living species to gradually improve the quality of the data structures that it produces until an optimal solution is found, or the run is terminated. Generally, an initial set of possible solutions is chosen randomly to form the first generation in the evolutionary process.

The GA performs genetic operations on the population, to produce another generation, and the process is repeated for a number of generations. The principle genetic operations are those of selection and crossover. The selection process measures the success of each member of the population in performing the allotted task, and selects the better members as candidates for a mating pool, here of the same size as the population. High-scoring members of the population appear multiple times in the mating pool, while low-scoring members are not selected. Consequently the average quality of the population increases with each generation. A new generation is evolved by the crossover operation using the concepts of sexual reproduction. Two parents from the mating pool produce two children by exchanging genetic material extracted from randomly chosen positions in their representative structure. Further genetic operations may be performed, particularly mutation, in which randomly chosen genetic material is removed from a child and randomly generated material inserted in its place.

In a GP system the data structures are computer algorithms, normally represented as trees in which the nodes are function nodes, representing a sub-routine in the algorithm, or terminal nodes, representing constants or variables defined by the algorithm. A simple example is shown in Figure 1.

```
IF (X AND Y) THEN
 IF Z THEN
  CALL XYZ
 ELSE
  CALL XY
 ENDIF
ELSE
 RETURN
ENDIF
```

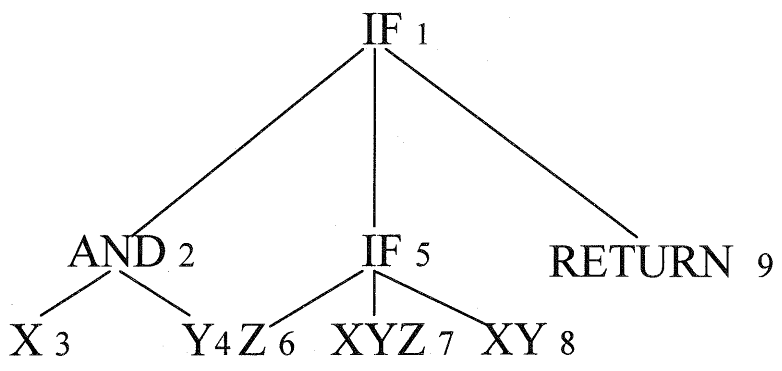


Figure 1 – a simple tree

The algorithm is 'executed' by a depth-first traverse of the tree, starting from the root node, searching for function nodes and their operands. An examination of nodes 2, 3 and 4 yields the logical value *true* or the logical value *false*. A *true* result causes the traverse to continue by examining node 5 and its operands. A *true* value for *Z* calls the sub-routine *XYZ* while a *false* value calls the sub-routine *XY*. In the case where the expression *X AND Y* is *false*, the algorithm relinquishes control.

Our GP system evolves algorithms that enable a software agent to solve some problems in spatial exploration. An example is shown in Figure 2. An enclosed *8* by *8* space has an internal pillar occupying cells *(8,5)*, *(8,6)*, *(9,5)* and *(9,6)*. A software robot is required to circumnavigate the space, visiting all cells marked *x*, starting from any cell and from any of four starting directions, represented by integers 1 (north), 2 (east), 3 (south) and 4 (west). In the example shown in Figure 2, the position of the agent (represented by an arrow) is identified by the triplet (5,6,4) i.e. (row, column, direction) and referred to as a *starting gate*. There are 240 starting gates (60 cells × 4 directions). One point is scored each time a robot visits a cell marked *x*, of which there 30. The maximum score is consequently 7200 (240 ×30).

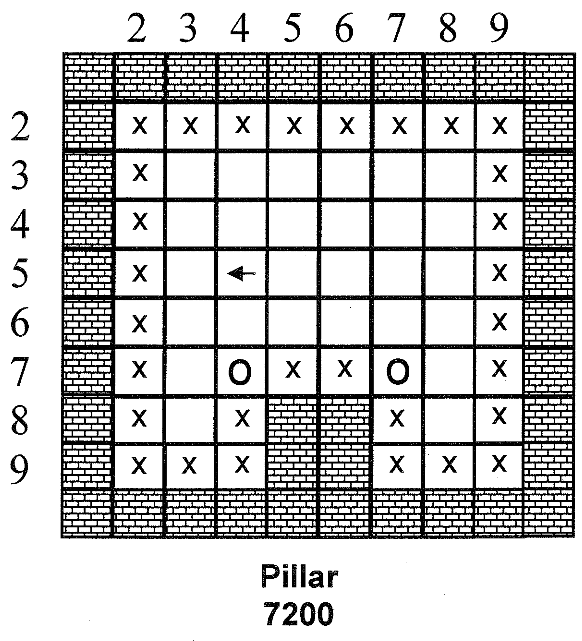
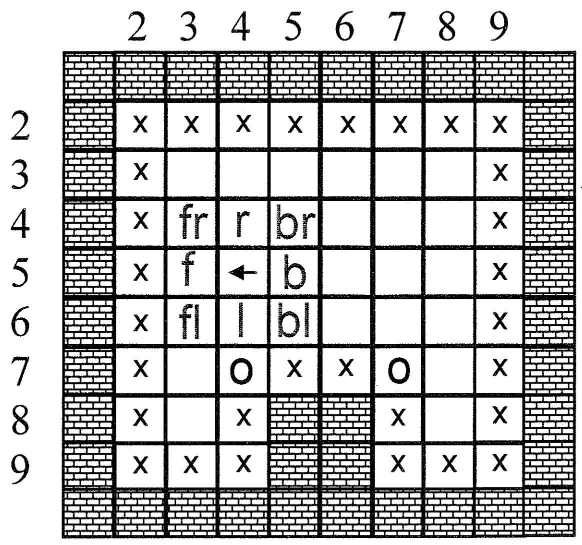


Figure 2 – the *Pillar* problem

The purpose of the GP system used here is to supply each robot in the population with an algorithm to guide it in solving the problem. The fitness of the algorithm is measured by the score achieved by the robot. Selection for the mating pool that will produce the next generation is based on fitness so the better robots survive and may breed with other robots. Each such process produces two children who may or may not be fitter than their parents. The tools that the GP system has at its disposal are the logical operators *IF*, *AND*, *OR* and *NOT*, and eight sensors with which a robot is able to detect an obstacle in an adjacent cell. The eight sensors (front, front-right, right, back-right, back, back-left, left and front-left) are shown in Figure 3 for a robot at (5,4,4). Finally there are four actions *MOVE*, *RIGHT*, *LEFT* and *WAIT*. The *MOVE* action would take the robot in Figures 2 and 3 to (5,3,4), the *RIGHT* action would rotate it to (5,4,1), the *LEFT* action would rotate it to (5,4,3) and *WAIT* would do nothing except use fuel.

We use the concept of fuel to refer to the fact that each robot is limited to a fixed number of actions when it gets its turn on the proving ground. In the simplest version of the GP system, each robot in the population has sole occupancy on the proving ground while its fitness is measured. A different version of the model, not reported here, allows multiple occupancy of the proving ground, and *WAIT* becomes more meaningful since it is then often useful in avoiding other robots.



Pillar
7200

Figure 3 – the sensors

Without the internal pillar, the problem is trivial. The tree

$$\begin{tikzcd}
 & \text{IF} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\text{f} & \text{LEFT} & \text{MOVE}
\end{tikzcd}$$

will cause a robot to move in the direction in which it is facing until it reaches the wall when it will turn left and follow the wall in an anticlockwise direction. The number of actions needed, when the robot starts with its back to a wall, is 38. The robot scores one point by virtue of its starting position, and then makes 7 moves to reach the opposite wall and score a second point. It turns left at the wall and needs four more left turns at the corners. Finally 26 further moves are needed to reach the remaining perimeter cells.

The addition of an internal pillar, as in Figures 2 and 3, significantly increases the difficulty of evolving a successful solution. The cells marked 'O' require special treatment involving a particular combination of two sensors. Figure 4 shows some variants of the pillar problem (the outer walls are not shown except in the *Steps* problem where the outer walls add to the complexity).

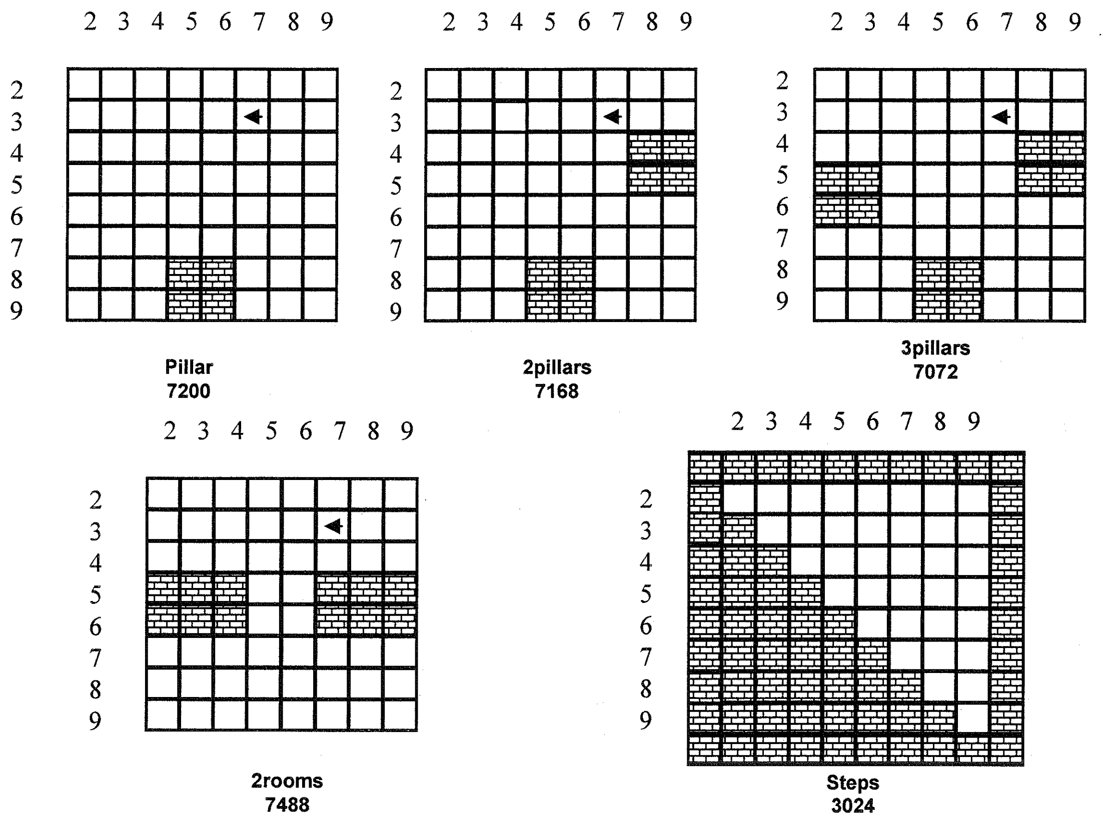


Figure 4 – variants of the internal pillar problem with their maximum scores

3. Some illustrative results – the canonical representation

Each member of the population visits the proving ground once and is limited to a maximum of 80 actions before it runs out of fuel and is removed. The upper tree in Figure 5 was the fittest member in a population of 400 after five generations of the 2pillars problem. The robot scored 3543 points against a maximum score of 7168, largely due to its inability to handle internal corners. The lower tree in the figure is a canonical representation of the upper tree. Both trees are logically identical and yield the same score. The canonical representation consists of a chain of sub-trees each of which results in an action if the associated condition is satisfied. Failing this, the next sub-tree is used. If none of the sub-trees yields an action, the robot waits until its fuel is exhausted. It is apparent from Figure 5 that the canonical representation is much easier to comprehend, and that its form better reflects the abilities of the robot as it tackles the different problems encountered in its exploration. Essentially it is a model of a rule-based system: each rule tackles a problem class faced by the robot.

The evolved tree has 30 nodes and the canonical version has 32 nodes. These are small because after only five generations the growth in the size of data structures associated with evolutionary computation has barely started. After 50 generations the average tree has around 100 nodes. The run that produced the trees in Figure 5 continued until a successful member was evolved, with a maximum score of 7168, in generation 41. The canonical form of this member is shown in Figure 6.

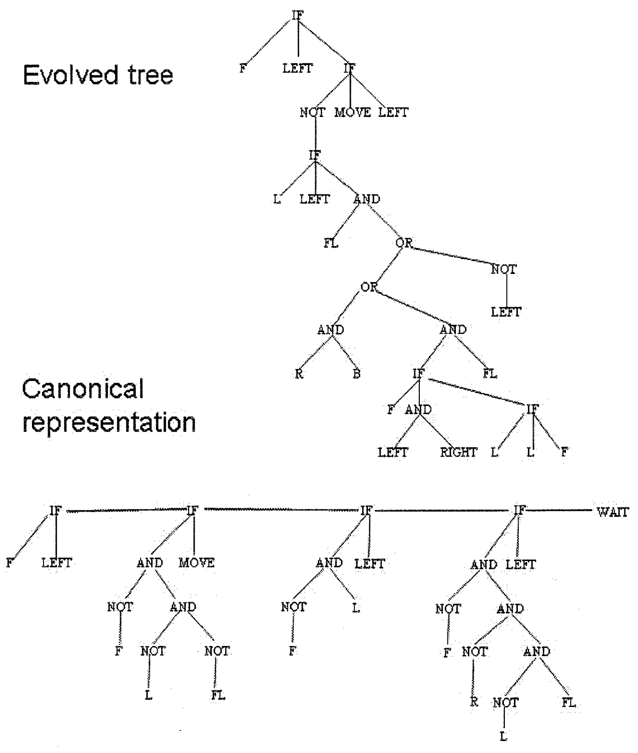
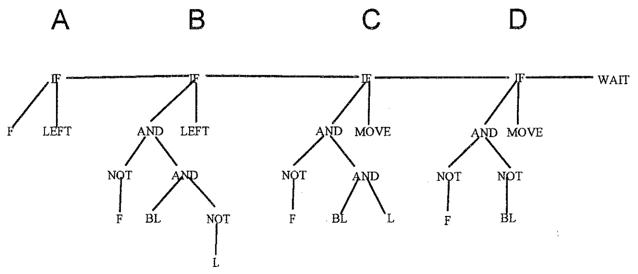


Figure 5 – an evolved tree and its canonical representation



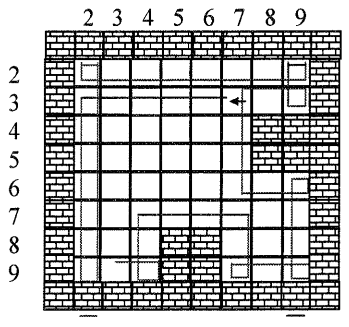


Figure 6 – a hit member

The solution shown in Figure 6 illustrates one of the interesting properties of the GP system. It is able to produce robots that are disabled in some way but manage to overcome the disability. To succeed in dealing with both internal corners and external corners a robot ideally needs to be able to turn both right and left. In the example in Figure 6, the ability to follow the wall at an external corner depends on the (*BL* and *not L*) construct in sub-tree B. To take advantage of this combination of sensors, the robot needs to be moving around the space in a clockwise direction and consequently needs a right turn when it reaches the wall. The control tree does not have the ability to turn right.

The progress of the robot is shown by the line in Figure 6. Starting from gate (3,7,4) it moves to the wall using sub-tree D. At the wall it turns left using sub-tree A. The robot is now going the 'wrong' way round the space and will approach the external corners unable to make use of sub-tree B. On reaching cell (9,2) it reverses direction by using sub-tree A followed by sub-tree B: sub-tree C takes it back to (2,2) where it simulates a right turn by successive use of A, A and B. This same 'right' turn is executed at each subsequent internal corner while sub-tree B takes it round the external corners. This somewhat bizarre behaviour is not an efficient use of fuel. Each robot is allowed 80 units of fuel, where each action consumes one unit. The path traced in Figure 6 consumes 73 units which is close to the available limit. For a robot to achieve the maximum score it needs to be successful from every starting gate, and it may run out of fuel before completing its run.

4. The benefits of canonical representation – detailed results

We have already shown that a canonical representation is beneficial in understanding and describing the way in which a GP system evolves the techniques needed to address problems involving exploration of a complex space. It turns out that the exploitation of canonical representation in the evolutionary process itself, can improve the performance of the system by a significant amount. We make use of the alternative representation in two ways.

As each member of the population is evaluated by sending it out onto the proving ground, the canonical representation of its control tree is built. We have the option of replacing any member of the population with the canonical form of its control tree to produce a mixed mode population. We have also introduced a form of crossover in which complete canonical sub-trees are exchanged between members of the mating pool. Two parameters of the GP system are the rates at which the crossover and mutation operators are applied in forming a new generation. We supplement these parameters by two further rates – the rate at which members of the population are replaced by their canonical forms, and the rate at which canonical crossover is applied. Note that we use the canonical operators in a generation sense, that is either the whole generation is put into canonical form or not, and all cross-over operators within a generation are canonical or not – thus a canonical replacement rate of 75% implies that the whole population is transformed into canonical form in three out of every four generations, while a canonical crossover rate of 25% means that this form of crossover is used for one out of every four generations. It is of course, not possible in our system to apply canonical crossover to non-canonical forms and so the crossover rate is never greater than the replacement rate.

In order to reach statistically significant conclusions we find it necessary to do a set of 100 independent runs with a given set of parameters on a population of 200 designs for 50 generations. We measure the success of each set in two ways – by the average score of the best members from each run, expressed as a percentage of the maximum score for the particular problem under study, or by the number of runs in the set that achieve a maximum score (a hit), i.e., a robot able to solve the complete problem set from any starting square and facing in any direction. Note that the hit rate is indicative of a process able to complete the task set as compared to the average best score which largely measures ability to reaches the easy squares on the proving ground, and is thus a less significant measure of system performance. Table 1 and Figure 7 show our results on the 2pillars problem and begin with the results from a traditional, non-canonical system.

Crossover (%)	*Mutation (%)*	*Canonical Replacement (% generations)*	*Canonical Crossover (% generations)*	*Hits*	*Average/Max (%)*
90	10	0	0	10	64.2
	40			13	66.9
	50			17	68.0
	60			15	68.0
	70			28	73.9
	80			22	72.4
	90			19	69.9
90	10	25	25	41	78.4
		50	50	31	74.7
		75	75	37	76.8
		100	50	33	74.9
		100	100	28	72.2
90	40	25	25	74	90.9
		50	50	79	94.8
		75	75	84	94.2
		100	50	83	94.4
		100	100	83	94.8
90	50	25	25	88	96.2
		50	50	94	98.5
		75	75	89	96.2
		100	50	88	97.0
		100	100	84	95.2
90	60	25	25	84	95.9
		50	50	90	96.7
		75	75	93	98.2
		100	50	95	98.5
		100	100	89	96.8
90	70	25	25	90	96.5
		50	50	95	98.4
		75	75	93	97.7
		100	50	97	98.9
		100	100	94	97.9
90	80	25	25	92	97.3
		50	50	98	99.3
		100	50	99	99.6
		75	75	94	98.4
		100	100	99	99.6
90	90	25	25	91	97.5
		50	50	96	98.9
		75	75	96	98.5
		100	50	100	100.0
		100	100	98	99.4

Table 1 – use of canonical operators on the two pillars problem[†]

[†] The rates quoted for the crossover and mutation operators in Table 1 are maximum values that are rarely achieved, moreover they refer to the % of robots affected, not the amount of genetic material changed. The GP system has a maximum tree size of 400 nodes. If, during a crossover or mutation operator, this limit would be exceeded, the operation is aborted and tried again with changed random choices. Three retries are allowed before the operation is permanently abandoned.

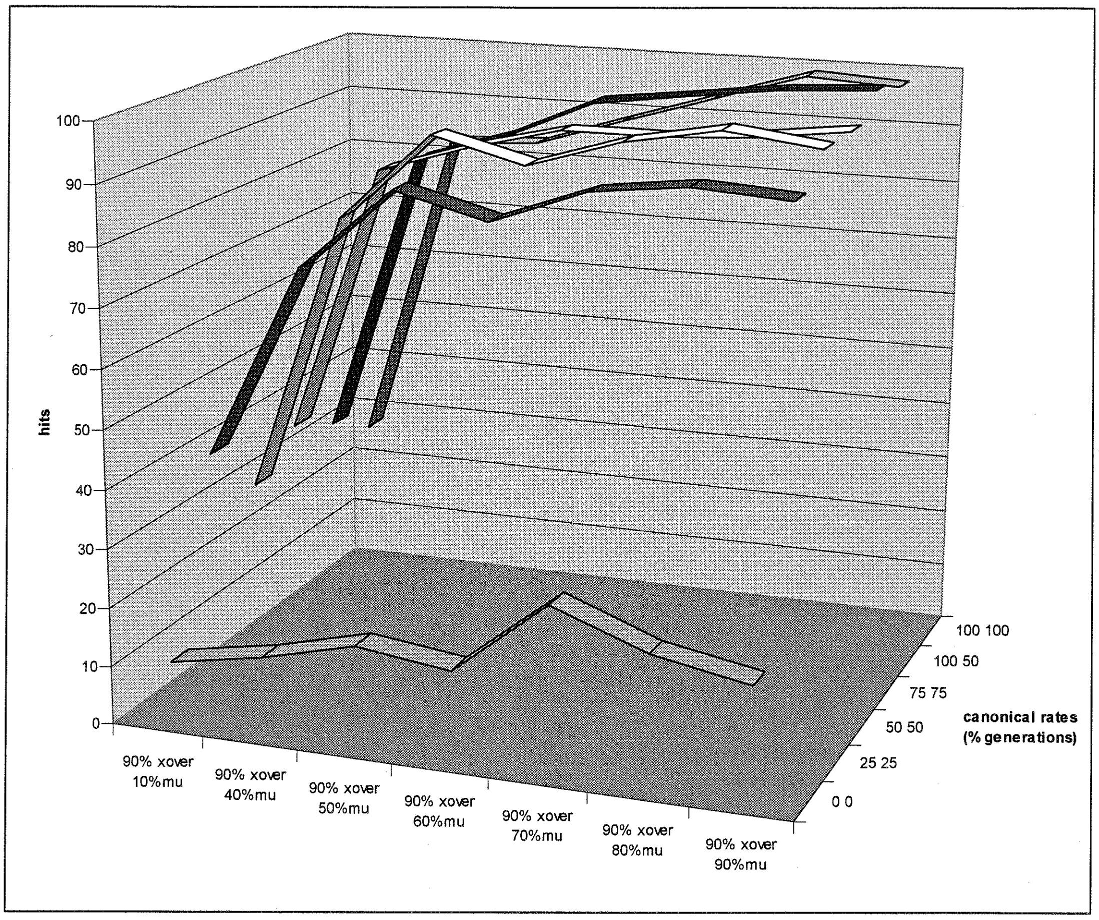


Figure 7a – variation of number of hits with control parameters

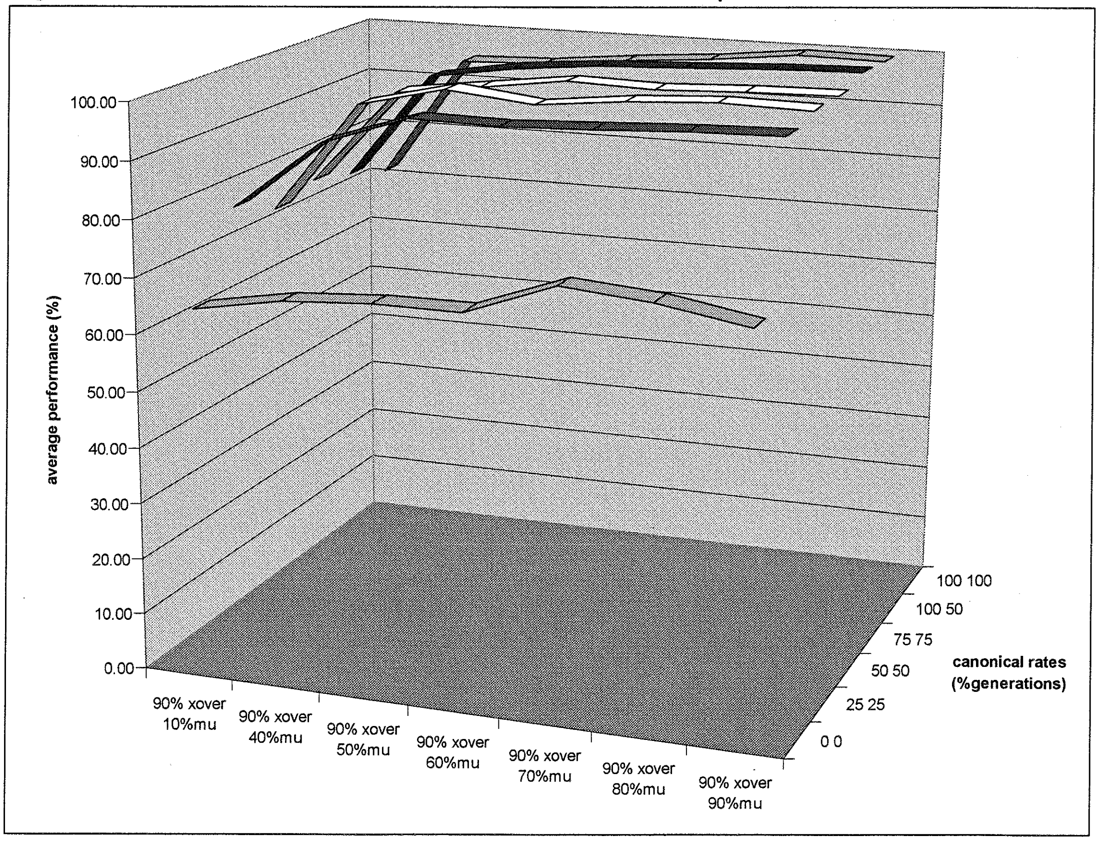


Figure 7b – variation of average performance with control parameters

The first seven rows of Table 1 represent a conventional GP system with various levels of mutation. The best performance is the 28 hits and an average fitness of 73.9% of the maximum with a mutation rate of 70% (note that this mutation rate means that there is a 70% chance that a member of the population is selected for possible mutation – the degree of mutation being a single sub-tree in each case, and then only if the resulting design is not too large – this is nothing like 70% of the available genetic material of course).

The remaining rows of the table result from runs with a fixed crossover rate of 90% and various rates of mutation, from 10% to 90%, combined with increasing use of the canonical operators. The results show that as soon as the canonical operators are introduced a dramatic improvement takes place. The best set of results are those with a mutation rate of 90% where the combination of 100% canonical formation combined with 50% canonical crossover leads to 100 hits in 100 runs. All runs using the canonical operators with mutation of 40% or more have average to maximum ratios over 90% with the worst performer achieving 74 hits, i.e., nearly three times that for the best base system.

We have also examined runs with larger and smaller populations and the results obtained are broadly similar, in that the canonical transformation and crossover process always significantly improves the system. Figure 8 shows the distribution of hit generations for all rates of canonical operators as the population and mutation rates change. It is clear that with a reduced population hits are delayed while with a larger one they arrive earlier. Moreover the impact of increasing mutation is also to cause hits to arrive earlier. With the best performing set-up and a population of 200 members many hits are occurring between generations 15 and 25 with very few taking the full 50 generations to be located, while with a population of only 100 members and moderate mutation many hits are still arriving in the final generations of the search.

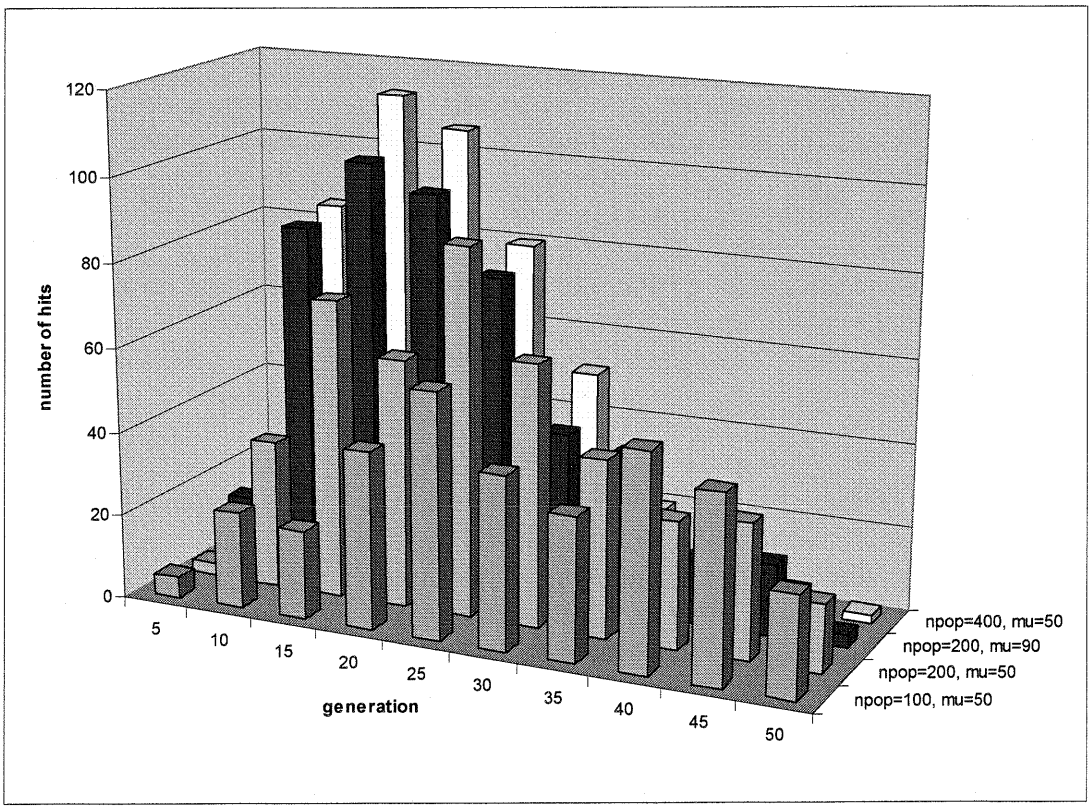


Figure 8 – hits per generation

5. Discussion

When the fitness of a control program is being measured on the proving ground, the agent is faced with a number of situations with which it needs to deal to be successful – internal corners, external corners, getting to the wall and so on. Typically there are between five and eight such situations which are described by the settings of the agent's proximity sensors, which are themselves represented by terminal nodes in the control tree. The control program uses these terminal nodes as arguments to logical functions and identifies the action that the agent should take. The terminal and function nodes that are visited during this process are generally scattered throughout the tree. The purpose of generating a canonical version of the control tree is to gather these nodes into a simple sub-tree with an IF operator at its root. The operands of the IF function are a test of the terminal nodes that yields a true or false value, an action in the case of a true result and a branch to the next IF sub-tree if the test is false. A canonical member consists of a chain of canonical sub-trees each of which deals with a particular situation in the agent's journey. An agent which achieves a hit score has a set of canonical sub-trees which match each of the situations that the agent will meet – in some senses the format is that of a rule based system, with each sub-tree representing a rule that can be used to solve a given problem within the overall task.

The canonical replacement operator replaces a non-canonical tree in the population with its canonical equivalent, thus introducing canonical sub-trees into the gene pool. These valuable building blocks are effectively encapsulated by the canonical crossover operator that ensures that the canonical sub-trees are moved through the population in their entirety. The combination of extracting building blocks and preserving them against corruption by crossover accounts for the significant improvements seen as compared to the standard GP system.

6. Conclusions

The trees evolved by a GP system are often large and difficult to interpret. We have shown that transposing a tree into canonical form, as defined in this paper, makes it easier to associate a path through the evolved tree with a particular feature of the problem under study. These paths are followed during evaluation of the structure and it is relatively simple to record the paths and to express then as canonical sub-trees. The concatenation of these sub-trees is logically identical to the evolved tree. Only paths which lead to an action are recorded as sub-trees, emphasising the fact that an incomplete solution to the problem, as represented by an incomplete canonical sub-tree, is associated with a failure to deal with particular circumstances arising as the robot makes its way round the course.

We have provided canonical sub-trees with protection in the form of canonical cross-over. Canonical sub-trees, each rooted on an IF-node, are exchanged in their entirety and are therefore immune to damage. Such damage may of course occur during mutation. As shown in our results, the conventional, non-canonical, system produces an average of some 18 hits per set of 100 runs. Introducing the canonical operators alongside the conventional GP operators (at 90% cross-over and 10% mutation) immediately increases the average hit rate to 34%. Thereafter ramping up the mutation rate to 90% approaches, and in one set reaches, a perfect score of 100 hits in 100 runs. One of the features of canonical representation is that it removes intronic genes from individual members of the population. The removal of this material increases the average value of the gene pool. High levels of mutation seem to be needed to exploit this added value.

References

Angeline 1993: Angeline and Pollack: Competitive environments evolve better solutions for complex tasks: *Proceedings of the Fifth International Conference on Genetic Algorithms*: Morgan Kaufmann, 1993.

Banzhaf 1998: Banzhaf, Nordin, Keller and Francone: *Genetic Programming - an introduction*: Morgan Kaufmann, 1998.

Chambers 1995: *Practical Handbook of Genetic Algorithms*: CRC Press, 1995.

Davis 1991: (ed.) *Handbook of Genetic Algorithms*: International Thomson Computer Press, 1991.

Holland 1975: *Adaptation in Natural and Artificial Systems*: University of Michigan Press, 1975.

Koza 1992: *Genetic Programming: On the programming of computers by means of natural selection*: MIT Press, 1992.

Koza 1994: *Genetic Programming II: Automatic discovery of reusable programs*: MIT Press, 1994.

López [2004]: López, Poli and Coello Coello, '*Reusing Code in Genetic Programming*', In Una-May O'Reilly, Maarten Keijzer, Terrence Soule et al., editors. Genetic Programming, Proceedings of the 7th European Conference, EuroGP 2004, LNCS, Coimbra, Portugal, April 2004. Springer-Verlag, 2004.

Rosca [1996]: Rosca and Ballard: '*Discovery of Subroutines in Genetic Programming*' in Angeline and Kinnear, editors: Advances in Genetic Programming, Volume 2: MIT Press, 1996