ARTIFICIAL INTELLIGENCE MAKES COMPUTERS LAZY

Simon Kent, Nayna Patel
Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
simon.kent@brunel.ac.uk, nayna.patel@brunel.ac.uk

ABSTRACT

This paper looks at the age-old problem of trying to instil some degree of intelligence in computers. Genetic Algorithms (GA) and Genetic Programming (GP) are techniques that are used to evolve a solution to a problem using processes that mimic natural evolution. This paper reflects on the experience gained while conducting research applying GA and GP to two quite different problems: Medical Diagnosis and Robot Path Planning. The observation is made that when these algorithms are not applied correctly the computer seemingly exhibits lazy behaviour, arriving at a sub-optimal solutions. Using examples, the paper shows how this 'lazy' behaviour can be overcome.

1. INTRODUCTION

Evolutionary Computing (EC) techniques have successfully been applied to many complex problems. These techniques, based on principles borrowed from biological evolution, have been shown to demonstrate emergent intelligence – they can generate new knowledge rather than apply previously known knowledge, as is the case with more traditional Artificial Intelligence techniques.

Experience of applying two particular EC techniques, genetic programming (GP) and genetic algorithms (GA), it has been noted that when solving some problems, the computer seems to exhibit lazy behaviour. Central to both GA and GP is the need for the programmer to provide a fitness function – a means of measuring how well any particular solution is at solving the problem. Rather than evolving complete solutions to the problems being tackled, the computer seems to evolve partial solutions which maximise the fitness whilst expending minimum effort – behaviour sometimes exhibited by university students who seek as many marks as possible for as little effort as possible.

It might almost be considered that if a computer can truly be lazy, then this is almost evidence in itself to suggest that it is exhibiting some degree of intelligence. To investigate this claim further this paper provides an introduction to how Evolutionary Algorithms work, and then describes two example problems where the application of AI techniques seems to have elicited lazy behaviour. It then considers ways in which the lazy behaviour can be discouraged, by applying a carrot and stick, or reward and

punishment approach, and finally concludes as to whether computers are indeed lazy, or whether this lazy behaviour is hiding another problem.

For those new to the field of Evolutionary Computing, there are certain considerations that must be made to ensure that these algorithms are not mis-applied. This paper seeks to provide insight into these considerations

2. EVOLUTIONARY COMPUTING

Evolutionary Computing (EC) is the name given to a branch of Artificial Intelligence in which the processes which drive biological evolution are used within a computer to evolve solutions to problems. Nature uses these processes to evolve solutions to the problem of living within a particular environment; that is it evolves organisms which, as the generations pass, are better and better at living in their environment. The effectiveness with which an individual plant or animal lives within its environment is measured by how likely it is to reproduce, and therefore pass on its genetic material to subsequent generations. The process, which relies on the feedback provided by the fitness value, resembles that of reinforcement learning techniques (Barto, Sutton et al. 1983).

Nature presents us with an enormous number of examples of solutions to complex problems. Every creature is a solution to the problem of thriving within its environment. Each creature is a product of evolution.

2.1. Genotype and Phenotype

In biology, the genetic material that is passed on through subsequent generations is the DNA. Biological evolution operates on the DNA which is an enormous string of

what is effectively a 4-bit code. This code is constructed from pairs of the nucleic. acid bases, adenine, guanine, cytosine and thymine. The representation of a creature in a coded form in DNA is called the genotype, whereas the expression of the DNA in the physical form of a creature with all its particular attributes is known as the phenotype (Johannsen 1911). This is analogous to a program written in a language such as Java, and the resulting running application, which bear no physical resemblance to each other. Evolution is an abstract concept; it cannot understand an algorithm for a fish and then implement it by expressing it using the language of DNA. Instead it efficiently searches all possible combinations of DNA (genotype) to find a representation, which is expressed as a creature (phenotype), which is good at surviving under water. Evolution therefore learns how to produce a solution without being explicitly told how to do it, it is only guided by outside influences.

Evolutionary Algorithms in a computer draw on the principles of evolution, as presented by Charles Darwin (1859). The techniques do not necessarily seek to copy biological evolution, but use Darwin's theory that 'survival of the fittest' is the driving force of evolution, and that a 'good' individual is more likely to reproduce, and consequently pass the genetic information, which makes it 'good', onto subsequent generations. It is not the intention here to describe all EA techniques in detail. GA, GP and related EA techniques are covered in texts including: Banzhaf(1998), Fogel (1966), Fogel (1995), Koza (1992), Holland (1992), Goldberg (1989).

2.2. Computerising the Evolutionary Process

To provide an understanding as to how the evolutionary process is captured in a computer, this section briefly describes Genetic Algorithms (GA) and Genetic

Programming (GP). They are techniques which share an underlying problem solving mechanism which is based on ideas of evolution, and the principle of ‘survival of the fittest' as presented by Darwin (1859).

GA and GP are both based on an iterative process, as shown in Figure 1, which mimics the natural process of evolution. The process begins with the creation of a population of potential solutions to the problem being solved. It is appropriate here to describe the difference between GA and GP.

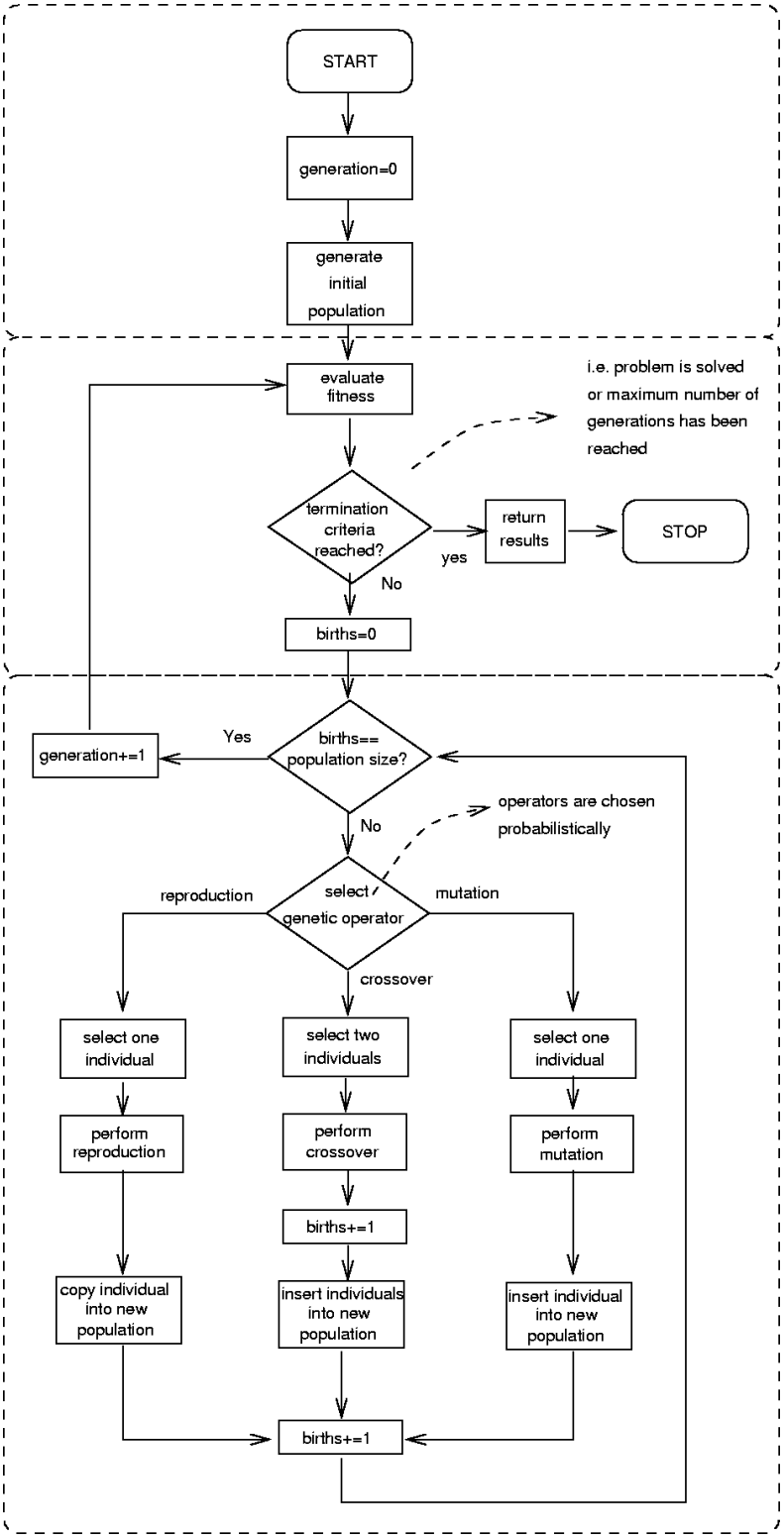

Figure 1: The Computerised Evolutionary Process

In the case of the GA, the data structure is typically a fixed length binary string. A simple example is a problem involving the optimisation of a single integer parameter taking values in range [0..7]. The representation of the solution could be a 3-bit binary string which is capable of denoting eight discrete values. A population of randomly initialised 3-bit binary strings would be created and subjected to the automated evolutionary process to find the optimum value.

The GP stores its representation in a tree data structure, which generally makes it easier to encode solutions that are functions or programs operating on input variables from the problem domain. GP could be used, for example, to evolve a tree that represents a mathematical expression for a curve that approximates a set of training data. A quadratic such as x^3+2x^2-x+5 could be represented in a binary tree form.

So, returning to a description of the iterative process, in GA it commences with the creation of a population of some randomly initialised binary strings, and in GP it starts with the random generation of a population of trees. The size of the populations will depend on the complexity of the problem, and can range from tens of individuals to hundreds of thousands for very difficult problems.

Both GA and GP assign a cost, or fitness value, to each solution in the initial population, and use these costs to select the better individuals for involvement in the next iteration of the evolutionary process. This selection is like natural selection in nature. Individuals with a higher fitness are more likely to be chosen than those with a low fitness. In natural evolution, those animals which are less capable of surviving in their environment die, and those which are stronger survive and their genetic information propagates to subsequent generations.

The individuals in the subsequent generation of this simulated evolution are created either by directly copying them, by making a composite of two individuals, or by making a mutated copy of a selected individual. The genetic operators used mimic asexual reproduction, sexual reproduction and genetic mutation in biology. The process of evaluation, selection and insertion is then repeated until some termination criteria are reached – a solution is found, or a timeout has been reached.

3. LAZINESS IN INTELLIGENT COMPUTERS

When Evolutionary Computing is used to try and solve a problem in a computer, it is not instructed step-by-step what to do as is the usual case when writing a computer program to solve a problem. Instead, the computer is given a framework, in the form of a program that implements an evolutionary algorithm, and some means of measuring how good one of its solutions is at solving the problem at hand. The intention is to make the computer apply an evolutionary process to gradually move towards a perfect, or near-perfect solution to a problem. This section describes two example applications of Evolutionary Computing where the computer is seemingly lazy in its attempt to arrive at a suitable solution.

3.1. Classification Problem

EC techniques can be used to solve classification problems (Elliott, Kent et al. 1997). In this particular example, the goal was to produce a classifier that, given certain lifestyle data of a patient, could accurately classify whether the patient is likely to develop oral cancer. During the evolution of this classifier, a set of real patient data and lifestyles was available to the computer along with corresponding expert

diagnoses of oral cancer based on the results of a biopsy. Each classifier evolved is measured as to how good it is at correctly classifying the likelihood of oral cancer.

Genetic Programming was chosen for this problem because the data structures that it evolves are trees, which could be used to evolve logic expression classifiers.

3.1.1 Patient Data

The data used for the project was derived from data collected from questionnaires distributed to over 2000 dental patients. Most of the data available corresponded to patients who were not likely to develop oral cancer. For each patient, there were 12 true/false values corresponding to 12 facts about the patients. These are shown in Table 1.

1.	Gender is female (i.e. 0=Male, 1=Female)
2.	Age > 50
3.	Age > 65
4.	Age > 80
5.	Age>95
6.	Have been smoking within last 10 years
7.	Currently smoke *more than* 20 cigarettes per day
8.	Have been smoking for *more than* 20 years
9.	Beer and/or wine drinker
10.	Fortified wine and/or spirit drinker
11.	Drink excess units (i.e. Male > 21, Female > 14)
12.	Not visited dentist in last 12months

Table 1: Patient lifestyle data

The data available consisted of diagnoses for over 1000 patients. Only 5% of these diagnoses were negative, and in-fact this was artificially high as the original research to collect the data involved 2000 patients, but some of the data was discarded to balance the data a little because there were so many negative diagnoses (Jullien, Downer et al. 1995).

3.1.2 The lazy Genetic Programming solution

To solve this classification problem a GP system was set up to evolve a logic expression to diagnose the potential of oral cancer given the input lifestyle data. The system was allowed to create expressions from basic logic expressions: AND, OR and NOT. The permitted leaves or arguments of the evolved trees were true/false values corresponding to the 12 pieces of lifestyle data. The resulting solutions could therefore apply the logic operations, to the lifestyle data inputs to produce a resulting true/false diagnosis.

To drive the GP process, a fitness measure was required. The aim is to find a solution to correctly diagnose the patients likelihood to suffer from oral cancer. The fitness measure for this problem – the basis upon which the genetic material is allowed to propagate to subsequent generations – was simply the proportion of correct diagnoses of a given program. An evolved expression correctly diagnosing 75% of the data would therefore have a value of 0.75.

The initial results were disappointing. Because of the high number of negative diagnoses in the input data, a very simple logic expression that always evaluates to false, will accurately diagnose 95% of the data even though such a classifier has no ability to ever make positive diagnoses. Any fragments of logic expression that could

usefully contribute to a classifier that could make positive diagnoses will be lost very quickly in the evolutionary process because their fitness will be very low (around 0.05) compared to the negative classifiers (0.95). So rather than gradually evolve and refine a classifier over many generations, the computer found a quick and easy way to do the maximise its reward for minimum effort.

If anything, for this particular application of artificial intelligence, it would be preferable for a classifier to overzealously make true diagnoses that could be followed up by a biopsy. This apparent laziness of the computer to evolve a classifier that makes negative diagnoses when the patient is at risk of cancer has a real human cost in that all positive diagnoses would be missed.

3.2. Path planning problem

The second example problem where a computer has demonstrated apparent laziness is a path-planning problem. This is a classic example of a problem which humans can solve with considerable ease, but with which computers have great difficulty. In general, the path-planning problem involves generating a sequence of positions for a robot between a start and an end position. The path should prevent the robot from colliding with obstacles in the workspace, and should be optimised, which means that it should take a short, or the shortest route between the two points (Cameron 1994).

3.2.1 Artificial Potential Field Path Planning

There are many techniques which have been developed for path planning (Hwang, Ahuja 1992, Latombe 1991, Yap 1997), but few are simple enough and powerful enough to be practical (Lozano-Perez 1987). One technique which has shown itself to be both simple and powerful is Artificial Potential Field (APF) Path Planning, a

technique attributed to Khatib (1986, 1980). In this experiment, APF Path Planning is optimised using Genetic Algorithms.

The basic idea behind the Artificial Potential Field (APF) is that planning can be divided into two aspects: one of pulling the robot towards the goal from its current position, and the other of repelling it away from obstacles in the workspace. Assuming that the robot has a positive charge, a negative charge can be situated at the goal that results in a force that pulls the robot from its current position in the direction of the goal. Each obstacle in the workspace is also positively charged, the same as the robot. This means that a field that repels the robot away surrounds each obstacle. By combining the fields centred at the goal, and each of the obstacles, a composite potential-field results, which should guide the robot towards the goal, whilst keeping it away from obstacles en-route. Having generated a field, the robot moves from its current position in the direction of the steepest gradient. More simply, it is like dropping a marble on the potential field surface and letting gravity pull it along the natural path to the goal.

An example field for a room containing a single rectangular obstacle is shown in Figure 2. A 3-dimensional surface such as that shown in Figure 2 can be defined by a mathematical expression that contains components defining the repelling and attractive (Latombe 1991, Khatib 1986). The equations used to define the potential-field have a number of controlling parameters that must be precisely set for the planning technique to work effectively. One equation corresponds the goal position, and creates a bowl shape that encourages the robot to move towards the goal. Each obstacle similarly has a corresponding equation which places a ‘hump' around the obstacle to repel the robot should it pass too close.

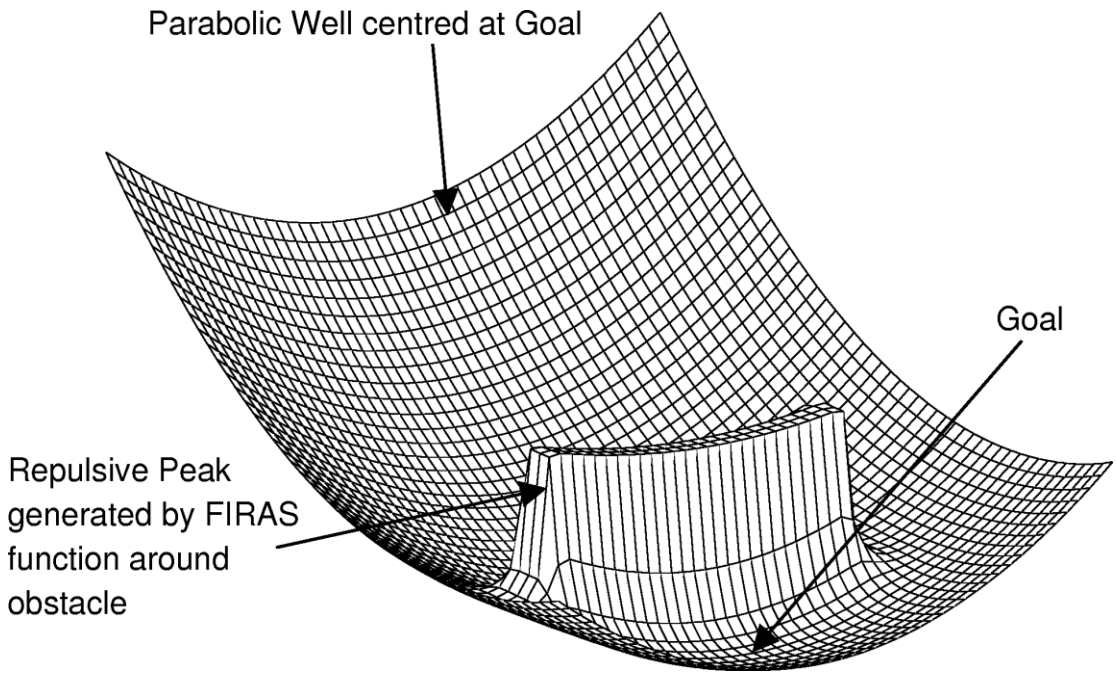

Figure 2: Example of a Potential Field

The trick for successful APF planning is to create a field with a single minimum point situated at the goal position, such that from any starting position, the planner can follow the landscape downhill until the goal is reached. Unfortunately the actual landscapes which arise often suffer from a problem called local minima. This is when a point in the landscape exists which has a lower potential than its surrounding points, but which does not correspond to the goal. This will result in a path which follows the landscape downhill, but which will not arrive at the goal.

3.2.2 Avoiding local-minima

The key to reducing the number of local minima in the workspace is to set the controlling parameters of the equation for the potential fields correctly. For example, how strong should the force be that pulls the robot towards the goal, and how hard should obstacles repel the robot? However, it has been shown by Koditschek (1987) that it is not possible to completely remove these local-minima for all workspaces. In Figure 2, if a marble, representing a robot was placed directly in line with the goal, but on the opposite side of the obstacle, it can be seen that the marble would roll straight towards the goal, but stop at the obstacle because there is no lateral force to pull it round the obstacle. This problem would be less prevalent for elliptical obstacles, but we do not live in an elliptical world.

Therefore the idea of sub-goals was introduced for these experiments. The idea is demonstrated in Figure 3 where a path is achieved using a sub-goal which is positioned such that the robot is first drawn toward it, and then when it is within a certain proximity of the sub-goal, the next goal will be activated.

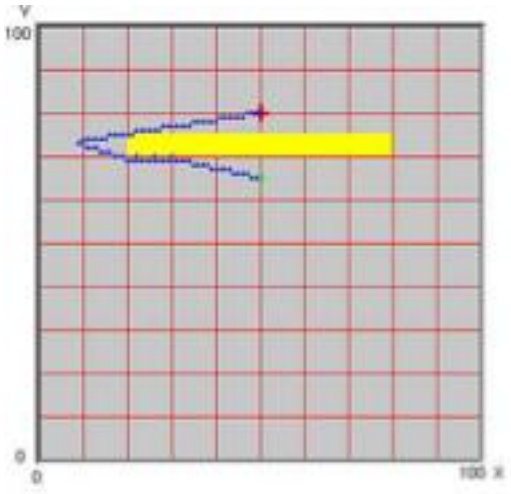

Figure 3: Avoiding local-minima using a sub-goal

3.2.3 The lazy Genetic Algorithms Solution

One approach to setting the controlling parameters of the potential field would be to set them manually using intuition and trial and error. This is time consuming, and not appropriate for automated path planning. A GA can be used to optimally set these parameters to yield suitable APF. The parameters controlling the equations were represented in a simple string of 0s and 1s, which is a binary representation of a floating-point number. These strings, which encoded parameters for potential solutions to the path-planning problem were evolved using Genetic Algorithms.

To drive the evolutionary process, the fitness function was defined in such a way as to evolve a planner which moves the robot as close to the goal as possible in as few moves as possible. As seen with the classifier problem earlier, once again the computer was able to very quickly recognise a way to achieve high reward for little effort. It deduced that by heading in a straight line towards the goal was a very efficient way of minimising the distance travelled while getting fairly close to the goal, therefore achieving a reasonable trade off. An alternative tactic was demonstrated by a planner that failed to move the robot at all, thus minimising the number of moves taken.

As in the GP experiment, the population of parameters being evolved quickly became overwhelmed with solutions that followed this strategy to the extent that genetic diversity in the population was lost, and the potential to evolve a truly successful solution was reduced to a tiny fraction.

4. USING REWARD AND PUNISHMENT

In the two example applications, two related artificial intelligence techniques have demonstrated how a computer has been quite efficient in its laziness to arrive at a significantly sub-standard solution because it has been able to maximise its reward, or its ability to produce solutions that score highly in fitness terms. This section describes steps that were taken in each case to try to overcome the apparent laziness.

4.1. Classifier Problem

4.1.1 Discouraging laziness

In the oral cancer classification problem, a poor solution was quickly evolved because the computer used the GP technique to quickly identify that a simple logic expression would reap a high reward. The driver for the technique is the fitness, and it was recognised that as well as rewarding for correct diagnoses, it was necessary to impose some degree of penalty for incorrect diagnoses. In fact, what is required is some measure that rewards for true, positive diagnoses and punishes for false, negative diagnoses because of the importance of catching the positive diagnoses and the danger of missing them. The measure is defined as:

$$\frac{\frac{TP}{TP+FP}-\frac{FN}{TP+FN}+1.0}{2.0}.$$

This fitness allows the addition of between 0.0 and 1.0 for correctly predicted positive diagnoses, and the subtraction of between 0.0 and 1.0 for incorrectly predicted negative diagnoses, resulting in a total range of -1.0 to +1.0. This is then normalised (scaled between 0.0 and 1.0). The optimum rule, which correctly predicts all positive

diagnoses and does not falsely predict any negative diagnoses is valued at 1.0. The worst case is at 0.0, where the rule incorrectly diagnoses all patients.

4.1.2 Results

Having modified the fitness measure, it was found that effective classifiers could be evolved. The previously successful classifiers were penalised under the new scheme because although they produced many true negative diagnoses, their ability to make true positive diagnoses was extremely poor or non-existent. On the other hand, the individual classifiers in early generations that made some positive diagnoses were able to propagate their genetic material to subsequent generations, such that this material could eventually combine to produce effective classifiers.

The performance of the best evolved rule was measured using a range of metrics used in the evaluation of diagnosis tools. These are defined in Table 2 with results for the best evolved GP rule. For comparison, values are given for a manual dental screener. These results show that although the evolved rule does not perform as well as a manual screener, it could act as a suitable pre-filter which could be embedded in patient management software.

Metric	Description	Definition		
			GP	Manual
Sensitivity	Performance of a test in terms of its ability to accurately predict a positive outcome, given that the outcome was actually positive.	$\frac{TP}{TP+FN}$	73%	74%
Specificity	Performance of a test in terms of its ability to accurately predict a negative	$\frac{TN}{TN+TP}$	65%	99%

	outcome given that the outcome was actually negative			
Positive Predictive Value	Performance of the test in terms of percentage of negative outcomes	$\frac{TP}{TP+FP}$	15%	67%
Negative Predictive Value	Performance of a test in terms of a percentage of negative outcomes	$\frac{TN}{TN+FN}$	96%	99%

Table 2: performance metrics for best evolved diagnostic rule

4.2. Path Planning Problem

4.2.1 Discouraging laziness

In the path planning experiment, the goal for a successful solution is two-fold. On one hand a solution needs to get close to the goal, and on the other hand the number of moves needs to be minimised. It is certainly easy to minimise the number of moves, and indeed the computer using GA very quickly recognised that not moving at all was another way of gaining reasonable reward for no effort at all.

What was required was a means by which to encourage the evolution of solutions that did move away from the start to the extent that they started to receive reward for getting closer to the goal. To this end the reward given for a short was weighted down in proportion to the proximity of the resting point of the robot to the goal. Basically this meant that the reward for a short path would only start to have an affect for planners that did make the effort to plan a path that moved close to the goal.

In fact there were more fitness criteria introduced which further rewarded or penalised the solutions if the robot collided with an object, or if there were objects between the resting point of the robot and the goal. The final fitness function used was:

$$(0.3w(s_n+c)+(1-w)d_n)*l,$$

where the parameters are defined in Table 3.

Symbol	**Description**	**Definition**
w	Weighting factor	0.3 if $s_n = 0$ 0.3*(1.0- d_n) if $s_n > 0$
s_n	Number of Steps	s/s_{max}
d_n	Distance between rest and goal	$d/\sqrt{2}$
c	Boolean style collision indicator	1.0 *if collision occurred* 0.0 *if no collision occurred*
l	Number of intersections on the line between the resting and goal positions, with obstacle boundaries.	

Table 3: Components of the multi-objective GA APF fitness function

4.2.2 Results

The effect of the multi-objective fitness function is demonstrated in Figure 4, which shows series of snapshots of the paths created by planners at various points in the evolutionary process. Initially the planner evolves a planner that is able to successfully navigate the obstacles and get reasonably close to the goal. This is because little reward is derived from a short path, but relatively high reward is gained from getting close to the goal. Eventually a planner is evolved that reaches the goal, albeit by a rather circuitous route.

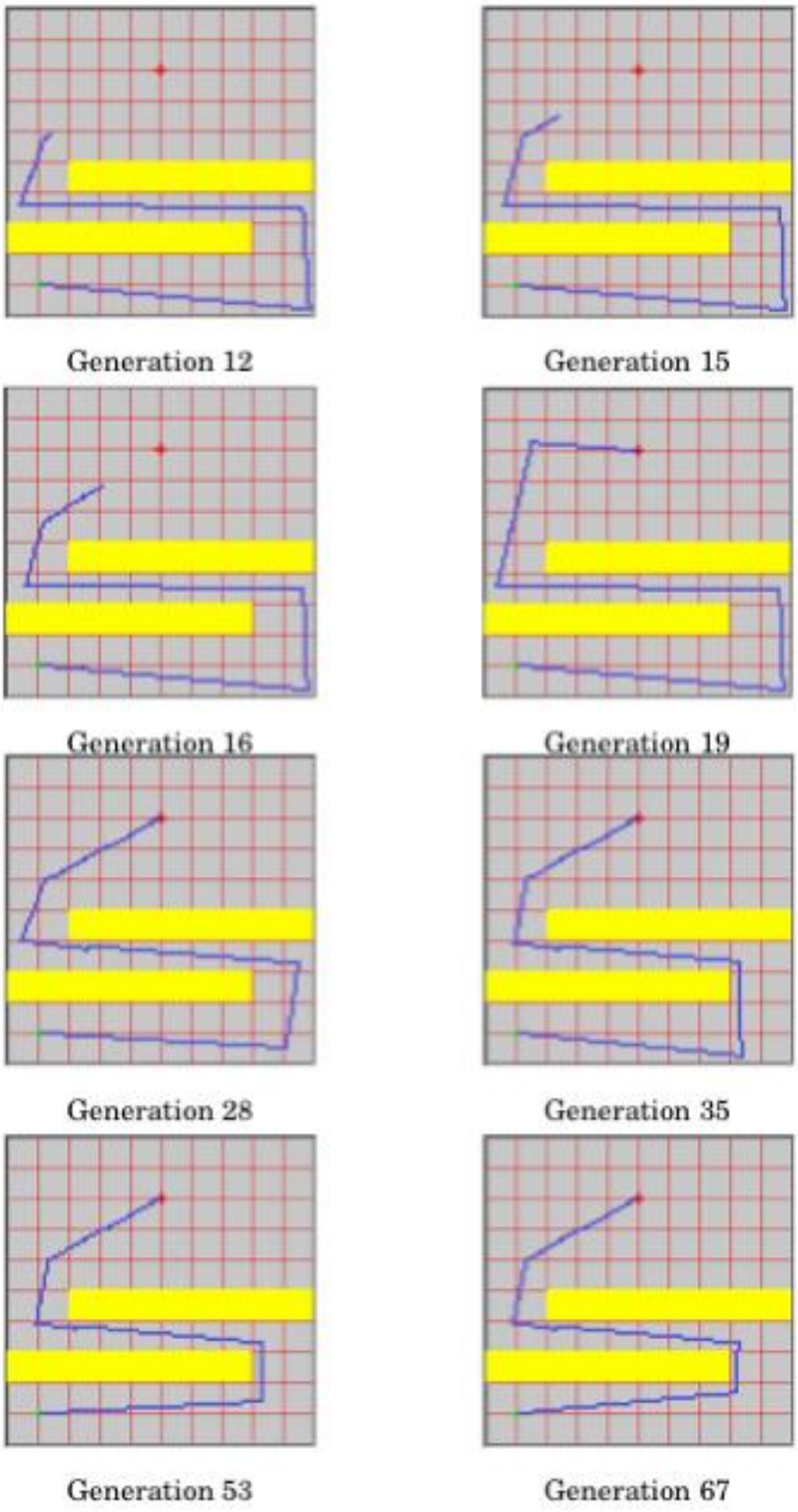

Figure 4: The effect of carrot and stick on evolving a path planner

By this point the weighting will have shifted the balance such that solutions are increasingly rewarded for a smaller number of moves, but because this time there is sufficient genetic diversity the planner does continue to evolve planners which do reach the goal. As the generations pass, the planner generates tighter and tighter path around the obstacles. Thus, by using a combination of reward and punishment, and by dynamically varying this during the evolutionary process, a solution to a complex, multi-objective problem can be produced.

5. CONCLUSION

Having presented two quite different problem scenarios, it has been shown that computers can indeed exhibit apparent lazy behaviour when they are used to apply artificial intelligence techniques. They are able to maximise their reward whilst minimising their effort. However, it was also shown that means could be introduced to reward and punish appropriately, to better guide the computer towards a correct solution.

Genetic Algorithms and Genetic Programming are, in fact, efficient search algorithms. They can be used to elicit knowledge that was previously unknown by searching a vast number of possible solutions for ones that are perfect, or near perfect for the problem at hand. This is analogous to biological evolution searching the enormous number of possible plants and creatures to find ones that fit appropriately with the environment within which they live. The only way that GA and GP *know* what they are looking for is by specifying appropriate search criteria. With these approaches, the search criteria is the fitness function used to measure how good a

solution is at doing its job. The fitness functions for both the classification problem, and the path planning problems presented in Section 3 initially seemed sound, but as shown in Section 4 they could be improved upon. In fact, the means of measuring fitness proposed in the initial experiments in Section 3 only captured half the picture.

For the classification problem the computer was asked to find a classifier that correctly diagnosed patients as often as possible. This is exactly what it did. What was actually required was a classifier that correctly diagnosed patients as much as possible, but which in particular did not falsely make negative diagnoses.

For the path planning problem, the computer was asked to find a planner that optimised two conflicting goals: that of moving close to the goal, and that of moving as little as possible. Once again, the computer searched and found a solution that met the criteria quite well. Only by fully specifying the search criteria was a successful solution found.

From these observations, it can be seen that actually the computer is not lazy; it is the humans that control them. What has really been shown is that the application of Artificial Intelligence is not a silver bullet to be applied arbitrarily to any problem. It is essential that thought is put into understanding what defines a good solution. The term ‘laziness’ is a useful metaphor for reminding those applying EC techniques of the consideration they must make to apply them in an effective manner.

In the field of simulation, Kiviat (1991) has coined the acronym SINSFIT (simulation is no substitute for intelligent thinking). Creating a simulation model does not automatically solve the problems in the application domain that gave rise for the need to carry out simulation modelling. Simulation modeling is simply a tool to aid understanding, but intelligent thought is still required to fully understand the domain.

It is equally appropriate to state that Artificial Intelligence is no substitute for intelligent thinking. The application of Artificial Intelligence in itself does not absolve the researcher of the responsibility of thinking intelligently.

References

BANZHAF, W., 1998. Genetic programming : first European workshop, EuroGP'98, Paris, France, April 14-15, 1998 : proceedings. Berlin: Springer.

BARTO, A.G., SUTTON, R.S. and ANDERSON, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics,* **SMC-13**(5), pp. 834-846.

CAMERON, S., 1994. Obstacle avoidance and path planning. *Industrial Robot,* **21**(5), pp. 9-14.

CHARLES DARWIN, 1859. On the Origin of Spieces. 1st edn. London: John Murray.

ELLIOTT, C., KENT, S., HAMMOND, P., DRACOPOULOS, D., DOWNER, M.C. and SPEIGHT, P.M., 1997. A comparison of artificial intelligence techniques for the identification of people at high risk of oral cancer. *Journal of dental research,* **76**(5), pp. 1053.

FOGEL, D., 1995. Evolutionary Computation. 1st edn. New York: IEEE Press.

FOGEL, L.J., OWENS, A.J. and WALSH, M.J., 1966. Artificial intelligence through simulated evolution. New York: Wiley.

GOLDBERG, D.E., 1989. Genetic algorithms in search, optimization, and machine learning. Reading, Mass.: Addison-Wesley.

HOLLAND, J.H., 1992. Adaption in Natural and Artificial Systems. 2 edn. MIT Press.

HWANG, Y.K. and AHUJA, N., 1992. Gross Motion Planning - A Survey. *ACM Computing Surveys,* **24**(3), pp. 219-291.

JOHANNSEN, W., 1911. The genotype conception of heredity. *The American Naturalist,* **45**, pp. 129-159.

JULLIEN, J., DOWNER, M., ZAKZREWSKA, J. and SPEIGHT, P., 1995. Evaluation of a screening test for the early detection of oral cancer and pre-cancer. *Communication of Dental Health,* **12**(3),.

KHATIB, O., 1986. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *International Journal of Robotics Research,* **5**, pp. 90-98.

KHATIB, O., 1980. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles*, Ecole Nationale Supérieure de L'Aéronautique et de l'Espace, Toulouse.

KIVIAT, P.J., 1991. Simulation, Technology, and the Decision Process. *ACM Transactions on Modelling and Computer Simulation,* **12**(2), pp. 89-98.

KODITSCHEK, D.E., 1987. Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations, *Proceedings of the IEEE International Conference on Robotics and Automation, March 31-April 3, 1987, Radisson Hotel and Raleigh Civic Center, Raleigh, North Carolina, Volume 1*, 1987, pp1-6.

KOZA, J.R., 1992. Genetic programming : on the programming of computers by means of natural selection. Cambridge, Mass.: MIT Press.

LATOMBE, J., 1991. Robot Motion Planning. Kluwer Academic Publishers.

LOZANO-PEREZ, T., 1987. A Simple Motion-Planning Algorithm for General Robot Manipulators. *IEEE Journal of Robotics and Automation,* **RA-3**(3), pp. 224-238.

YAP, C., 1997. Algorithmic Motion Planning. In: T SCHWARTZ and CHEE-KENG YAP, eds, *Algorithmic and Geometric Aspects of Robotics.* 1st edn. Mahwah, NJ, USA: Lawrence Erlbaum Assoc., pp. 95-143.