Genetic Algorithms for the Development of Fuzzy Controllers for Mobile Robots

Donald Leitch and Penelope Probert

Robotics Research Group, University of Oxford, UK

Abstract. In this paper we present a novel genetic algorithm (GA) for designing fuzzy systems for mobile robot control, in which the meaning of a section of chromosome is determined by surrounding genes, much like a language description. We demonstrate that this leads to much improved convergence speed over conventional GA codings, and discuss the mechanisms that lead to this improvement. The algorithm also uses a simple chromosome reordering operator which uses the algorithm to maximise its own efficiency.
Our results show the performance of our algorithm when applied to designing controllers for the commonly used benchmark inverted pendulum problem, as well as problems in mobile robotics. We discuss the application of this method to robotics generally, and some of the difficulties faced when the algorithm is used for more complicated applications, and how these problems could be approached.

1 Introduction

There can be little doubt that almost all of the concepts we use to reason about our physical world are to a certain extent fuzzy. The paradigm of fuzzy systems seeks to formalise this perception for the purpose of intelligent control. To date, one of the criticisms of fuzzy systems has been the lack of any formal method of deciding how to partition the input space of a system into fuzzy sets, and how to associate these sets with outputs. Many algorithms exist for designing fuzzy systems when an input - output data set exists, such as adaptive vector quantisation [11] and the method proposed by Wang and Mendel [18]. Feedforward backpropagation neural networks can also be thought of as fuzzy classifiers, and supervised learning schemes such as gradient descent evolve means of associating overlapping regions of input space with particular outputs to varying extents, which is in essence what a fuzzy system does.

All such schemes however, as well as unsupervised neural network learning algorithms such as differential competitive learning, require input patterns derived from examples of "good" control. The algorithm presented here uses only an objective function and a simulation to assess the performance of a controller, and a genetic algorithm to improve the performance.

Similar approaches have been presented in the work of Cooper and Vidal [5], Lee and Takagi [16] and Karr [10] amongst others. Our algorithm differs significantly in many ways; our coding scheme is novel and improves the performance of the algorithm significantly, we also include an implicit chromosome reordering operator to reduce non-linearity in the algorithm.

In this section we provide some background information on genetic algorithms and fuzzy logic and discuss their compatibility. Section 2 describes our coding scheme and the chromosomal ordering operator. Section 3 presents the results of the algorithm's application to the inverted pendulum problem, and compares these results to some obtained using more traditional GA / Fuzzy combinations. Section 4 is a discussion of some of the issues involved in applying the algorithm to more complex problems in mobile robotics, and section 5 presents the results of such an application. Finally we conclude with a brief discussion of ongoing research.

1.1 Fuzzy systems

A fuzzy system is basically an expert system which uses fuzzy logic for inferring outputs. Fuzzy logic can be regarded as being a extension of classical logic, the central tenet being that a member of a fuzzy set may have a numerical degree of membership anywhere in the interval $[1,0]$ as opposed to either 1 or 0 with no intermediate value allowed [19, 15].

The very nature of a fuzzy set makes it impossible to pin down exactly the 'right' membership function, particularly if the concept expressed by the set is multidimensional. In many cases multi-dimensional fuzzy sets are defined by assuming that set membership functions can be expressed in terms of a set of independent one dimensional membership functions. Although this may produce acceptable controllers in many cases, there is no reason to suppose that optimal performance can be achieved with such simplifications. This representation is the equivalent of assuming that a set can be represented as an ideal point in antecedent space, along with a number of shape parameters, and using a L_∞ distance measure in antecedent space when calculating the degree of membership of a particular antecedent vector. In this paper we use the L_2 metric, as this makes it possible to use a much greater variety of set shapes, that are not constrained to be rectangular with their axes parallel to the antecedent space axes.

Fuzzy systems were originally used in controllers because of their power at representing linguistic concepts, such as HIGH, NEAR etc, and their subsequent ability to model the expertise of trained human process controllers. An alternative view is to regard a fuzzy system as a function or control surface approximation. In this sense, the individual sets cannot meaningfully be given linguistic labels, particularly if they are designed using some optimisation algorithm. Linguistic meaning is only applicable when sets are designed with it in mind, and not when sets are described purely numerically as control surface approximators.

1.2 Genetic Algorithms

Genetic algorithms (GAs) are biologically inspired multi parameter search / optimisation algorithms that have proven to be effective at solving a variety of complex problems other algorithms have difficulty solving. In common with all other search algorithms, a GA performs a search of a multidimensional space containing a hypersurface known as the *fitness surface.* A particular parameter set defines a point on a hyperplane on to which the surface is projected, with the height of the surface above the hyperplane reflecting the relative merit of the problem solution represented by the parameter set.

The basis of a GA is that a population of problem solutions is maintained in the form of *chromosomes*, which are strings encoding problem solutions (in this case fuzzy controllers). Strings can be binary or have many possible alternatives (*genes*) at each position. The strings are converted into problem solutions, which are then evaluated according to an objective scoring function. (Similar to a function used to optimise controller gains in LQP control, for example.) Often it is not possible to exhaustively test all aspects of a solution, and noise may be present on the objective function, so the assigned fitness is an estimate of the true fitness of a chromosome. It is important that this is a good estimate, otherwise the *selective pressure* that favours truly high scoring chromosomes can be lost in the noise caused by poor fitness estimates.

Following fitness evaluation, a new population of chromosomes is generated by applying a set of *genetic operators* to the original population. These are basically random copying and altering of individuals from the original population with the probability of copying of any individual from one generation to the next being proportional to its fitness. During the copying process two operations may be performed - a gene may be erroneously copied (*mutation*), or a new individual may be formed by combining segments from two chromosomes by copying one chromosome up to a specific location on the chromosome, then copying a different chromosome (*crossover*). For example, consider the chromosomes shown in figure 1. The two chromosomes at the bottom are the offspring of the upper two, formed by crossover at the line, and mutation of the gene in the box.

GAs are particularly good at finding maxima where the fitness surface is non-linear, highly convoluted, with many local maxima, and dependent on several parameters simultaneously. This non-linear dependency on several parameters which means that overall fitness cannot be expressed as a function of fitness values due to individual genes is known as *epistasis.*

Although in many cases a GA is unlikely to produce an optimum solution to a problem, it can produce a solution that is within a few percent of the optimum in a time orders of magnitude less than a full solution takes using some other algorithm such as exhaustive search. They have found a vast number of applications, ranging from robotic path planning [6] to gas pipeline pumping optimisation [7]. Beasley et al. give many examples of GA applications in [3].

The power of GAs to solve complex problems can be viewed in terms of the *schema* theorem first proposed by Holland [9]. A schema is a short section of a chromosome, which need not necessarily be contiguous. Holland's theorem

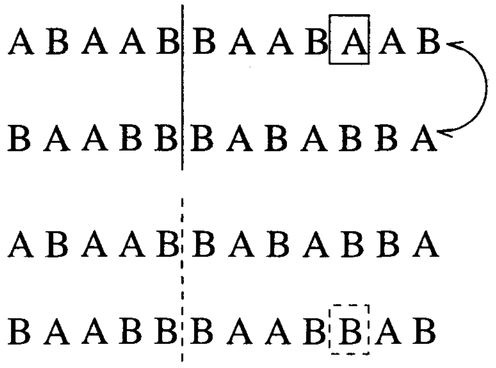

Fig. 1. Genetic operators

basically shows that schema which when part of a chromosome lead to a good score, are likely to increase in frequency throughout the population, thus leading to a gradual improvement in fitness as time passes under certain circumstances such as very large populations.

Although the schema theorem is useful, it ignores the effects of epistasis and makes many other' simplifications and assumptions, and other suggestions have been put forward for explaining the success of genetic algorithms, such as Altenberg's work on the evolution of evolvability [1], and Goldberg's building block hypothesis [7]. Goldberg's book is also a good general introduction to GAs.

Altenberg has produced theoretical analyses of genetic programming (GP), which are particularly useful when analysing the coding scheme described in this paper, as there are many similarities between GAs using the scheme and the paradigm of GP, where tree like logical and arithmetical structures are adapted to produce compound functions and programs. Although the full details are too lengthy to be included here, the essence of the theory in [1] is that a property called *evolvability* arises, which is the ability of particular sections of chromosome not only to contribute towards overall fitness, but also to be good at combining with other sections of chromosome to produce blocks of greater fitness, thus not only is GP good at finding blocks of code which combine well by chance, mechanisms exist by which versatile and highly fit blocks of code arise as a property of the algorithm.

1.3 GAs for the evolution of fuzzy systems

If we imagine a fuzzy rule to represent a patch on a control surface (figure 2), it is easy to visualise why GAs are particularly suited to the design of fuzzy systems.

The output for a given input will generally only be dependent on a single rule. Thus if the parameters on the chromosome which code a particular rule are changed, the input / output function defined by the chromosome's corresponding rule base will only be affected in a small area. If the input falls exclusively

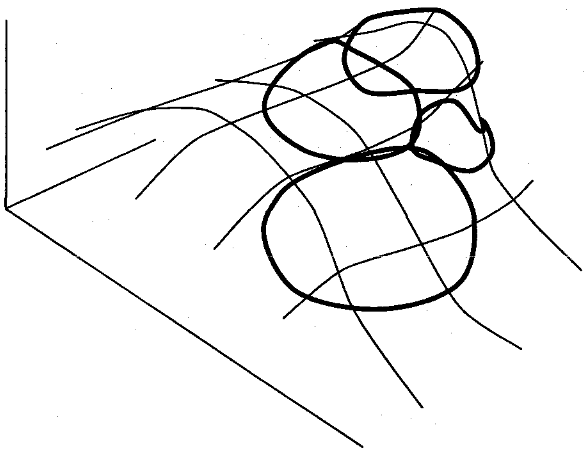

Fig. 2. Fuzzy rules as patches

within a single set, then the effect in that region of changing the chromosome is not affected by any other parameters on the chromosome. Only when an input falls in a region in which two or more sets overlap is there any interdependency between chromosomes, in that the effect on the output of changing one parameter is influenced by the value of another parameter. This is epistasis, and by the very nature of fuzzy systems, is limited. As observed by Davidor [6], GAs are particularly well suited to problems with this kind of limited epistasis.

2 Genetic coding of fuzzy rule bases

The way in which a problem solution, in this case a fuzzy rule base, is represented in the chromosome, is of great importance for the success of the GA [7]. Many schemes suggest themselves for representing a fuzzy rule base using a string of numbers, depending on the amount of freedom we wish to give the GA to alter the rule base. For example, the simplest possible application of a GA to learning a rule base is one in which the input and output sets are predefined by the user, and the GA is simply used to learn associations between them. In this case, the rule base could be coded as a string of integers. This method was tried with some success by Karr [10], but it relies on human expertise to partition input and output spaces, and so slightly defeats the purpose of using the GA in the first place. A more complex method uses a fixed number of parameters to describe set positions and shapes, with the number of rules in a rule base determined in advance. This coding scheme has been widely applied, but still relies on a initial guess as to the number of input space partitions necessary to achieve a good problem solution. Ideally the number of rules as well as their shape could be decided by the GA. For this to occur, however, chromosomal length has to be variable and crossover has to be able to occur at different sites on parent chromosomes.

This can cause problems, as it becomes possible to generate "impossible" chromosomes that cannot be converted into rule bases, unless measures are taken to ensure crossover occurs at legal sites, or that impossible chromosomes are either weeded out or penalised by the objective function. It may also be necessary to constrain the two crossover sites so that chromosomal length does not vary too much. Despite the increase in complexity over other coding methods and the need for support structures to ensure chromosomal viability, methods using variable length codings have been used with success, for example Cooper and Vidal [5] and Lee and Takagi [16]. Examples of more complex coding schemes that do not rely on a string of real numbers as parameters specifying rules can be found in Nomura et. al. [14] who use an interesting binary method where the fixed length chromosome is a direct representation of the input universe of discourse, and features on the chromosome translate directly to features on the universe of discourse, eg set centres.

2.1 Context Dependent Coding

The coding scheme adopted in this paper uses language like strings to encode a rule base, which are interpreted by a parser. The parser has a limited degree of intelligence, and knows what sort of structure the rule base takes. Particular codons in the chromosomal string indicate the context in which sections of string should be interpreted. An example of a general CDC scheme is described, followed by a specific example of CDC applied to the coding of fuzzy rule bases.

General Scheme A general CDC scheme might use four codons, two of which are used to represent instructions, and two of which are used to encode numbers. This way it is possible to represent any instruction by indexing it according to its number which is represented in binary, or code any integer in binary. It is not necessary to use two different codons to represent instructions or numbers; two is just the minimum number that can be used for a general scheme.

The parser operates by scanning a chromosome from left to right, assigning meaning to each codon or group of codons depending on the context in which it finds them. Consider, for example, a system which can be described using four instructions, plus numbers. Each instruction has a number of real values associated with it, and may also have further instructions associated with it. Using a four letter alphabet of codons, assume that A and B are used for instruction coding (*instruction codons*), and 1 and 0 are used for number coding (*numerical codons*). As there are four instructions each one will be coded by a sequence of codons rather than by a single codon. A typical chromosome will be a random sequence of these four codons, of indeterminate length, for example

A1BA0100B10AAB011B00B1A

Initially the parser starts from the left, looking for an instruction, which will be coded as AA, AB, BA or BB, as there are four instructions in the language. The parser ignores the initial two codons, and finds the sequence BA, which is assumed to have some meaning to it in this case. It then may need to look for a numerical sequence, so it will read the sequence of numbers following the BA, terminating it when it finds another instruction. In this case the sequence is 0100B10 before the next instruction , AA, is encountered. The B is ignored and the binary sequence 010010 is converted in to a number by whatever scheme is being used. Following the AA instruction is a B, which is also ignored. The next sequence may be decoded as a number, or ignored, depending on the meaning of instruction AA. The process then continues until the end of the chromosome is reached, at which point any unfinished instructions are either discarded or terminated with a standard sequence. By careful design of the instruction set, any chromosome can be decoded no matter how long, or what sequences of codons appear within it. The simple scheme described above can be varied in numerous ways. For example once an instruction codon has been encountered when looking for an instruction, any numerical codons can be ignored; in the above example the first instruction would be AB instead under this scheme, which is useful if the instruction set is large, as long sequences of letter codons are unusual. Another modification may be that any letter symbol terminates a numerical sequence, this has the effect of increasing the number of codons which are ignored (assigned no meaning) by the parser.

Although the scheme may not be very elegant, it is very flexible and robust, and has several advantages over conventional codings which are enumerated later in this paper.

Coding of a rule base Consider a four input, two output system using sets described by a centre in input space and a radius. Due to the simplicity of the system, it can be described by a string utilising a three letter alphabet, two letters of which (0 and 1) are used to code real valued numbers or binary switches (*binary codons*), and one of which (C) is used as a *context switch* (instruction codon), to indicate to the parser the start and end of particular sections of code. For example, take the section of code shown in figure 3.

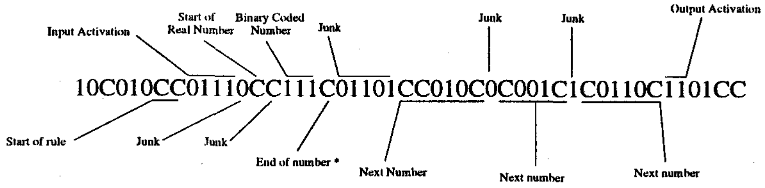

Fig. 3. Section of CDC Chromosome

As the parser scans the chromosome from left to right, initially it is looking for CC to indicate the start of a rule. The parser then knows that for a four input system, it must read the next four binary codons to indicate if each of the four inputs is used in this rule (1) or not (0). In this example, section following the two adjacent context switches is 01110, the first four codons of which indicate that this rule applies to inputs 2, 3 and 4. Had the parser encountered any context switches is the middle of this section, they would have been ignored as they have no meaning in this particular context. The parser then knows it has to look for three coded real values, to indicate the position in input space of the centre of this rule's antecedent set. The start and end of real numbers are marked by single Cs, so the parser ignores the 0 immediately following the four binary codons indicating input activation, and the next C indicates the start of a real number. The adjacent C is ignored, as although a C is used to indicate the end of a number, no binary codons have been found yet. The following string of binary codons, 111, codes the number 1.0000, according to the following scheme. For a string of length n codons (this is obviously variable, as the terminating C is not fixed), the corresponding real number is $\frac{k}{2^n-1}$, where k is the value of the binary number. Due to the variable length, the precision is variable. All numbers are scaled to be in the interval [0,1].

Although the use of Gray coding in binary representations of real numbers is widespread, use of such a scheme is only of benefit when a position dependent coding is used. This is because a GA using a position dependent coding is optimising a fixed length string of parameters, which are always in the same place on the string. In a CDC, the parameters determining fitness are distributed in various different positions on the string and so when two parameters happen to be mixed by crossover, it is almost certain that the new parameter formed will be unrelated in the effect it has on fitness to the original two parameters.

The five binary codons following the terminating C are ignored, as the parser looks for another C to start the next real number, the position in input space of the centre of the antecedent set with respect to the third input to the system (second input to the rule). The number is decoded in the same way as before, and has the value of $\frac{2}{7}$. The final set centre position is $\frac{1}{7}$. The radius of the conical set in input space is decoded as $\frac{6}{15}$. Obviously it would be easy to modify the parser to allow for more set shape parameters, and even different geometries.

The parser has now got all the information describing the input set for this rule, it now looks for information concerning the output part of the rule. First, it looks for another binary sequence to indicate which outputs this rule affects. The next sequence after the section coding the radius is 1101. The first two 1s indicate that this rule affects both outputs. The next two codons are junk, then the next C indicates the start of the real number indicating output 1 if this rule is fired.

It can be seen that a considerable portion of the chromosome does not code any useful information. This may seem inefficient, but these *junk* genes play a useful part in absorbing disruption caused by genetic operators. Consider, for example the effect of a mutation at the codon marked *, say a change to a 1. This results in the first set centre position being decoded as $\frac{461}{512}$, as the following section of junk becomes part of the number. Following that, however, the decoding remains the same. Mutations in other positions have differing effects, but generally the damage is limited. The same applies to crossover; the meaning of the chromosome is disrupted in the region of the break, but it is unusual for more than a single rule to be affected. If some algorithm were implemented to remove the junk every time a chromosome was parsed, genetic operators would become highly disruptive and would be likely to change the meaning of an entire chromosome.

Other advantages of CDC are that the variable length and independent crossover sites on each chromosome allow for much greater genetic variability, as two sections of chromosomes that occupy identical sites on different chromosomes can find themselves on the same chromosome as a result of crossover, which cannot occur for a fixed length position dependent coding.

Furthermore the presence of junk has an effect on efficiency noted by Levenick [13], who introduces *introns* (crossover targets) in to chromosomes and observes a considerable increase in efficiency. Angeline [2] also notes that when using GP redundant sections of code often arise, and deleting these seems to have a negative affect on the performance of the algorithm. It is possible that the presence of junk allows the chromosome to adapt to become robust to disruption by crossover, so that the second order effects predicted by Altenberg [1], namely that not only do good chromosomes arise, but structures within chromosomes that enable them to be more adaptable in the presence of crossover, arise as well.

The greater genetic diversity and unpredictable disruption caused by the genetic operators also helps to reduce premature convergence, as they make it extremely unlikely that all the individuals in a generation become very similar, and even if they are all identical novel chromosomes are still generated from the current population.

The coding is also very flexible; systems far more complicated than rule bases such as recursive networks can easily be described with modifications to the parser, and a general scheme capable of describing any system or program can be realised using four codons, with two used for instruction coding and two for numerical coding.

2.2 Chromosomal Fitness

As mentioned in section 1.3, the interaction between sections of chromosomes representing individual rules is limited as the interaction between rules is limited. In a randomly generated rule base, however, rules that have antecedent set centres close together in input space and are likely to interact are not necessarily close to each other on the chromosome. The result of this is that the crossover operator can be very disruptive, with offspring strings producing behaviour that differs greatly from both of their parents due to large numbers of interactions between the sections of chromosome that came from each parent. This can be advantageous, as it leads to the occasional development of novel behaviours, but it is detrimental to the incremental improvement in behaviour that is due to offspring having the characteristics of already successful parents.

This can be combatted by using chromosomal ordering algorithms, for example [10] describes a heuristic method of aligning strings before crossover to maximise the exchange of similar material, and Hoffmann uses an ordering operator in [8] to reduce the disruptive effect of crossover.

The parent-offspring behavioural continuity is more likely to be maintained if sections of chromosome that represent sets that are likely to interact are as close as possible on the chromosome. Crossover is unlikely to disrupt such clusters, and also unlikely to bring together sections of chromosome that interact and produce unexpected behaviour. What is therefore desired is some ordering operator that manipulates sections of chromosome. If the centres of the rules are thought of as being points in a space with a dimension equal to the number of inputs (figure 4), a chromosome, in ordering these points, describes a path through this space. In simple terms, what we wish our ordering operator to do is to minimise the length of this path.

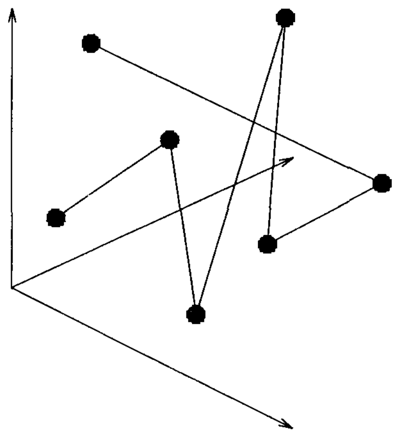

Fig. 4. Chromosomal path through input space

This is in fact the well known travelling salesman problem (TSP), a complex NP-complete problem that has no analytical solution for a large number of points. A successful applications of genetic algorithms has been in producing solutions to the TSP [3]. Using a CDC, however, it is very difficult to change the order of a chromosome without badly disrupting it, so explicit ordering is not feasible. Instead, it is possible to do it implicitly using the GA itself. The method is simple - an approximation of the degree of epistasis is made by taking the sum of the distances between the centres of antecedent sets that are adjacent on the chromosome, weighted by the radius of the sets. The reciprocal of this measure, called the *chromosomal fitness*, is included in the objective function, so that one of the goals of the GA is to minimise this measure.

If N set centre positions in input space are given by the vectors $S_1, S_2, \ldots, S_n$, their radii $r_1, r_2, \ldots, r_n$ then the chromosomal fitness is given by

$$\frac{1}{n}\sum_{i=1}^{n-1} u(0)(r_i + r_{i+1} - ||S_i - S_{i+1}||)$$

Where $u(0)$ is the unit step function, included so that two sets do not interact by a negative amount if they are a long way apart in input space.

More complex measures can be used, for example calculating some weighted sum of the distance between sets that are close on the chromosome but not adjacent, with the weight being inversely proportional to the distance between the sets on the chromosome, or by calculating chromosomal fitness to account for the volume of input space occupied by set overlaps instead of using the linear method shown above. It was found experimentally that for the applications discussed in this paper that more complex measures than the one shown did not provide a significant advantage.

It was found that the best way to combine chromosomal fitness and raw fitness in the objective function was to multiply them together following scaling to ensure that they had similar geometric ranges (i.e. the ratio initial value to final value was about the same). This implicitly rewards the algorithm for improving the attribute that is worst, as a unit increase in the smaller quantity produces the greatest increase in overall fitness. Although other methods of combining multiple objectives are often used, for example Pareto optimality [7] simple multiplication was found to be effective.

It also avoids the problem of the algorithm becoming totally concentrated on improving one attribute, which sometimes happens if they are summed. This is true of any case where a GA has to improve a number of attributes in parallel.

Using this technique avoids the need for time-consuming explicit reordering algorithms and enhances the GAs natural power of self optimisation.

The results of using a chromosomal fitness measure are shown in section 3.

Although the use of chromosomal fitness is a disadvantage initially as the GA will be biased towards some strings purely on the basis of their structure and not their fitness as regards the behaviour of the corresponding rule base, this is more than compensated for later by the increased efficiency of the algorithm due to the order of the strings.

3 Algorithm applied to Cart-pole balancing

In this section we present the results of applying the algorithm to the inverted pendulum stabilisation problem, often used as a benchmark for controllers. We present comparisons with standard GA methods used for generating fuzzy rule bases, and briefly with another rule based method [17].

The cart-pole problem (see figure 5) is interesting because two variables have to be controlled using a single control input. There are four system states, and the dynamics of the system are governed by the following equations [5]

$$\ddot{x} = \frac{F - \mu_c sign(\dot{x}) + \tilde{F}}{M + \tilde{m}}$$

$$\ddot{\theta} = -\frac{3}{4l}(\ddot{x}\cos\theta + g\sin\theta + \frac{\mu_p\dot{\theta}}{ml})$$

where

$$\tilde{F} = ml\dot{\theta}^2 \sin\theta + \frac{3}{4} m \cos\theta \left(\frac{\mu_p \dot{\theta}^2}{ml} + g \sin\theta\right)$$

$$\tilde{m} = m\left(1 - \frac{3}{4}\cos^2\theta\right)$$

where the cart mass $M = 1.0$ kg, pole mass $m = 0.1$kg, pole half-length $l = 0.5$m, friction of cart on track $\mu_c = 0.0005$N, friction at hinge between cart and pole $\mu_p = 0.000002$kg m, cart position is x and pole deviation from vertical is θ. For comparison, the parameter values are taken from [5], but the simulation used here differed in that it included noise and had a lower sampling rate (20 Hz as opposed to 100 Hz).

There are four control inputs - θ, $\dot{\theta}$, x and $\dot{x}$. The control output is the lateral force applied to the cart.

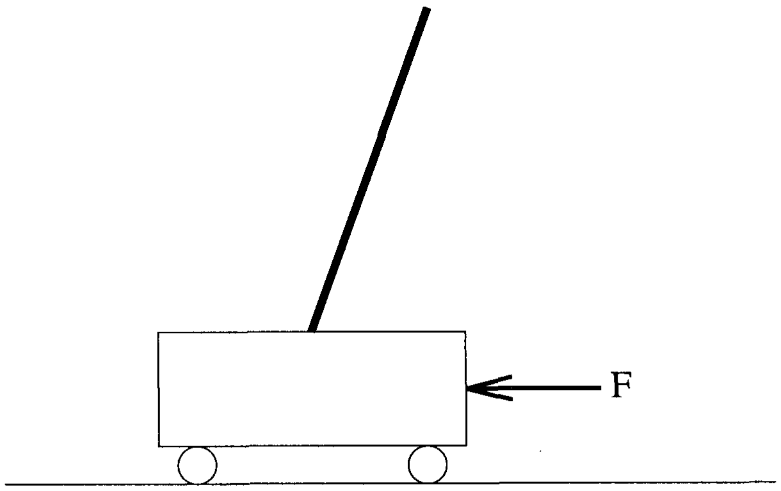

Fig. 5. Cart-pole system

The system was simulated on a computer, using an Euler scheme to calculate derivatives. Random noise was added to simulated sensor inputs to the controller, and to the actuator outputs. This was done in order to more closely model the real world, and also so that the controller was robust enough to cope with the inevitable differences that would result from transferring it from a simulated environment to a real one. The presence of noise actually enhanced learning as well as leading to more robust controllers [12].

The simulation had an update rate of 20 Hz, as did the controller. The scoring function used to assess the fitness of solutions over a trial was:

$$S = \sum_{test} (1 - x^2)$$

expressed as a percentage. If the controller failed (defined to be mod$x > 1.0$ or mod$\theta > 0.26$) the trial was stopped, so that a controller which failed after only 10 % of the maximum allowed time could only score a maximum of 10 %. Each controller was tested for up to 40 simulated seconds, from 12 different starting states. The best controller in each generation was also tested for 10 minutes from a random starting position, to test robustness and generalisation capability. A controller capable of stabilising the pendulum whilst keeping the base within the specified tolerance for the long test was usually evolved within the first 10 generations. The population was 30.

3.1 Chromosomal fitness

Figure 6 shows a comparison of GA convergence rates with (curve A) and without (B) a chromosomal fitness term included. The curves shown are of raw fitness (chromosomal fitness was only used to determine the probability of copying to the next generation), and are obtained by averaging the best score in each generation over six independent runs, as a single GA run is very noisy.

Three curves are shown; the two upper curves are of GAs using chromosomal fitness, and the lower curve is of a GA not using chromosomal fitness. Two versions of the GA using chromosomal fitness were included to illustrate that the difference between use and non-use of chromosomal fitness is significant and not random.

The population used was 30, the mutation rate was 0.005 (the probability that a single codon would be subject to mutation, not the probability of changing a codon) and the probability that any pair of chromosomes selected for copying were crossed over was 0.6.

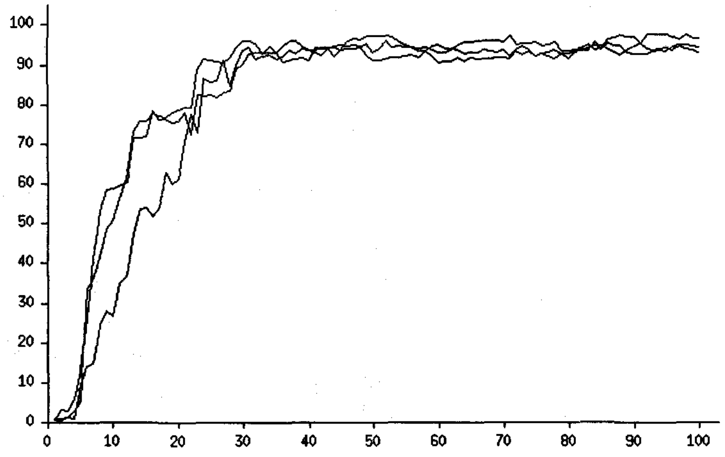

Fig. 6. Comparison of convergence rate with and without chromosomal fitness

3.2 GA parameter settings

The parameter setting of the GA were varied to examine the effect that this would have on convergence. Figure 7 shows the convergence curves obtained using mutation only, with the mutation rate set to 0.05, 0.005 and 0.0005. The best convergence is obtained with the rate set to 0.005, and the very noisy curve is with the highest mutation rate. It can be seen that mutation is actually a fairly efficient search operator.

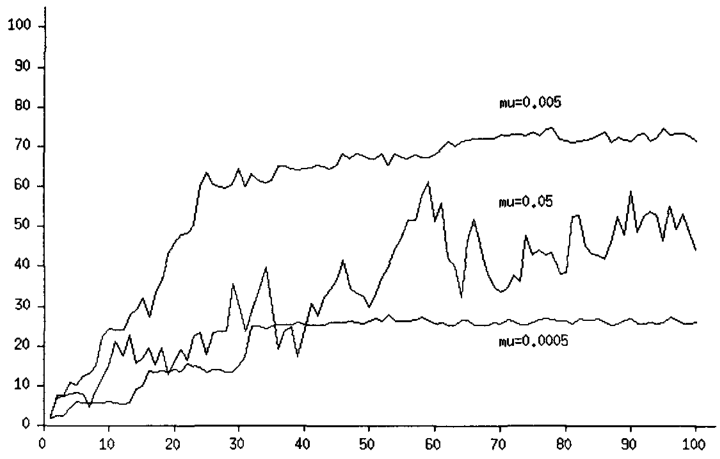

Fig. 7. Convergence of GA using various mutation rates

The effect of varying crossover is shown in figure 8. The two curves shown are produced by GAs with crossover rates of 1.0 (upper curve) and 0.2.

3.3 Comparison with standard coding

Figure 9 shows the convergence of standard GAs when applied to the same problem. The curve that quickly converges is a fixed length, real coded chromosome with the number of rules fixed at 20 (with CDC the number of rules was usually in the range 10 - 30), and the plot shown is the mean of six runs, again using the best score of each generation. The other curve is obtained using a similar representation, but with a variable length chromosome, and crossover sites constrained so that viable chromosomes are always produced.

It can be seen in figure 9 that although allowing the length of the chromosome to vary and picking independent crossover sites on each chromosome improves performance when compared to a standard GA, the GA using CDC significantly outperforms the variable length GA, indicating that some other mechanism than increased variability due to chromosomal mixing is involved in the improvement.

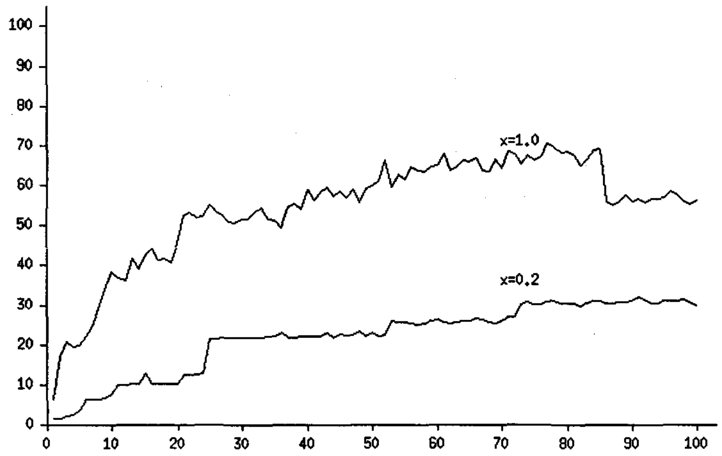

Fig. 8. Convergence of GA using various crossover rates

It is thought that this is due to second order effects that become possible in the presence of junk.

It is interesting to note that the convergence of the fixed length GA is initially faster than that of the variable length GA, although the latter eventually surpasses the former. It is believed that this is caused by the latter's far larger search space, and subsequently the initial period is spend building up small blocks of good rules which then start combining to produce the increase in convergence speed seen later on.

Cooper and Vidal describe a GA for evolving fuzzy inverted pendulum stabilisers in [5]. As a performance measure, they use the ability of the controller to stabilise the pendulum within certain limits for a long (hours of simulated time) trial. Their algorithm could achieve this in 10000 function evaluations, our algorithm frequently produces controllers capable of long term stabilisation within 1000 function evaluations. A controller is deemed to be capable of long term stabilisation if it can balance the pendulum (within the limits mentioned earlier) for a period of 24 hours of simulated time.

Varšek et. al. describe a rule based controller attributed to Makarovič [17], which in our trial had a fitness of 95% when the parameters were tuned by hand. Not only do many of the controllers produced using the algorithm described here outperform this, Makarovič's controller, and the controllers similar to it that use GAs for parameter determination in [17], imposed rigid structures on the input space, either using fixed partitioning, or decision trees to rank state variables. This type of structure would not be suitable for problems with a high dimensionality, or problems with a large degree of asymmetry, such as the mobile robot controllers described in the next section.

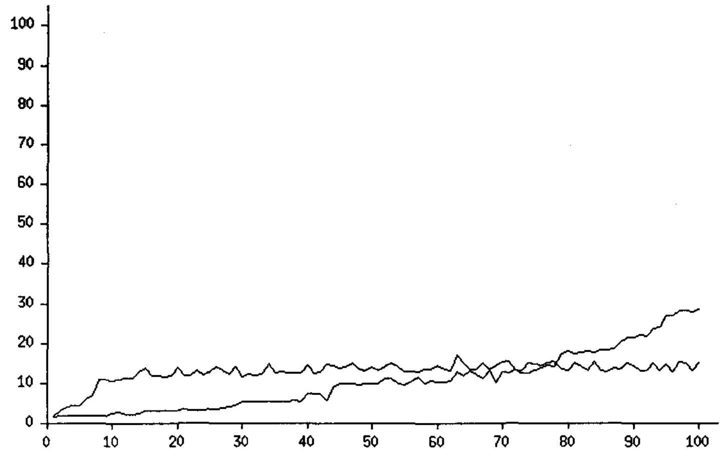

Fig. 9. Convergence rate of standard GAs

4 Problems in Robotics

Mobile robots provide a far greater challenge than simple 'toy' control problems. In the inverted pendulum, the system state can be measured directly and only a single control output is inferred. The task of balancing is also very easy to specify. A mobile robot has an array of sensors, which lead to a redundant, non-linear mapping from sensor inputs to states. There are normally at least two actuator outputs to be inferred from the input data, and the relationship between current state and next state is often highly configuration dependent and non-linear. Unpredictable environmental changes also affect the robot.

It is also far harder to classify behaviour quantitatively for a mobile robot which may need to execute complex manoeuvres than for a simple controller whose behaviour consists of tracking a set point.

The objective function used by the GA is the only method of judging the fitness of a behaviour. Good choice of objective function is therefore of paramount importance to the success of the algorithm. In this section we briefly analyse the application of a simple objective function, similar to the one used in the cart-pole problem, to robotics.

Let the state of a robot with respect to its environment at time t be $\mathbf{x}_t$. Define also the sensor reading vector to be S_t, E the state of the environment, which is entirely external to the robot, but with respect to which a task is defined, and the actuator output A_t.

These quantities are related through the equations:

$$S_t = f_1(\mathbf{x}_t, E)$$

$$A_t = f_2(S_t)$$

$$\mathbf{x}_{t+1} = f_3(A_t)$$

which are likely to be very non-linear and redundant. f_2 is the controller function; the only way of modifying fitness is by modifying this function. More concisely:

$$\mathbf{x}_{t+1} = f_1 f_2 f_3(\mathbf{x}_t, E)$$

this can be written as the timestep $\Delta t \rightarrow 0$ as a differential equation:

$$\frac{d\mathbf{x}}{dt} = g(\mathbf{x}, E)$$

Solving this differential equation for a given set of boundary conditions $\mathbf{x}_0$ (the initial state) gives us a trajectory through state space:

$$\int_{trial} \frac{d\mathbf{x}}{g(\mathbf{x}, E)} = \int dt$$

let:

$$\mathbf{x}_t = h(t, E)$$

This trajectory is of course affected by noise in sensor readings and actuations, and changes in the environment, but there is nothing that can be done about this other than ensuring that the controller is robust enough to deal with uncertainty.

The trajectory that results is also highly dependent on the boundary conditions of the equation. From this trajectory, the fitness of the controller must be calculated. For control surface approximation the ideal measure of fitness would be some integral of deviation from an ideal control surface. As we have no idea what the ideal control surface looks like, this is not possible and we have to infer error from indirect means, such as defining an ideal trajectory through state space and measuring the sum of the deviation from this - thus giving us a fitness measure for a single trial:

$$fitness = \int_{trial} |B(t) - h(t, E)| dt$$

Where $B(t)$ is the ideal trajectory through state space for given boundary conditions, or in simple cases a set point. This is obviously a very indirect method of measuring the fitness of a surface, and can lead to many complications. As noise and small changes in boundary conditions may lead to large divergences in state space locii, very different scores may be obtained for similar controllers which can make the problem GA deceptive. One way round this is to conduct a large number of short trials, avoiding divergence. As all controllers are run on a computer, they are in fact using discretised versions of the equation above. Using a large timestep allows large jumps to be made over the control surface, avoiding discontinuities which also cause divergence. The problem which presents the greatest difficulty, however, is the choice of $B(t)$, the desired behaviour.

4.1 Behaviour modification using GAs

Simply specifying $B(t)$ as a trajectory through state space is not good enough, as a perfectly good controller may well produce an identical trajectory at a different rate to the one specified. Some sort of dynamic programming algorithm may be able to match actual and desired behaviour independently of time, or a neural network could be trained to recognise 'good' behaviour, but that defeats the object of using a GA; that no training data need be supplied. Furthermore, the desired trajectory through state space is likely to be dependent on the initial conditions, and will need to adapt to account for changing environments. Apart from very simple cases, for example tracking, it is highly improbable that a designer could design an objective function that specified all aspects of a robot's behaviour under a range of conditions.

One possibility is to use proximity to final desired state as an objective function, but for most tasks this is far too simple as it will not reward partial solutions if they happen to fail in an area of state space distant from the intended final state.

The problem of objective function specification can be viewed as trying to specify a particular behaviour that is desirable, where a behaviour is simply a locus or set of locii in state space.

A fuzzy rule can be thought of as causing a very simple behaviour in an AGV, or defining a short locus through state space, usually only in a specific area of state space. The emergent behaviour caused by all the rules acting together is the locus that the robot follows.

The traditional approach to designing robot controllers to achieve a certain behaviour is to decompose the high level behaviour required in to lower level, simpler behaviours. It is hard to see how some equation defining a set of locii could be decomposed in to simpler equations, but behavioural decomposition can easily be done intuitively. For example, a manufacturing operation may be very hard to define using a single equation describing a locus or locii in the robot's state space, but can easily be described using terms such as "pick up" and "insert", which are lower level behaviours than "assemble object". In a sense low level behaviours are like sub goals that must be achieved in order to attain some global goal.

To achieve a high level behaviour by manipulating rule bases with a GA it is necessary first to develop low level competences which can then be further modified and enhanced to produce an emergent high level behaviour.

This suggests an evolved version of a Brooks type architecture [4] with a GA learning various behaviours at the lowest level by manipulating individual rules, and also at a higher level by manipulating combinations of lower-level controllers.

A GA could build a high-level behaviour on the basis of successfully evolved simpler controllers, as fuzzy rule bases can be combined in the same way as individual rules. A GA that can successfully manipulate single rules will be likely to be able to manipulate pre-evolved groups of rules which already implement simple behaviours. For this to happen, a two level evolution procedure is necessary.

Initially the GA learns easy to define low level behaviours, and once a reasonable degree of proficiency has been attained, the low level rule bases can be manipulated by the GA to form a controller capable of performing a higher level task. A simple objective function is sufficient as the controllers would already have a degree of competence at the behaviour desired.

This kind of hierarchical design assumes operator influence to choose intermediate level behaviours, but without this the GA would be unable to solve the complete problem, or to bridge the 'complexity gap' between the high-level behaviour desired and the low level behaviours it has available to manipulate. What is needed is some kind of operator supplied 'stepping stone' in the form of intermediate level behaviours. It is likely this is how complex behaviours arose in biology; humans and higher vertebrates didn't appear overnight, rather the complex behaviours exhibited by such creatures are based on simpler ones inherited from their ancestors, but subsequently combined and modified to produce an emergent, higher level behaviour.

One of the applications in the next section demonstrates how this multi level controller design may be achieved in practise.

5 Applications in robotics

In this section we present some results obtained using the algorithms and techniques discussed above. Two behaviours are learned using the genetic algorithm; a simple controller which moves along a corridor using simulated sonar readings, and a controller which executes a multi point turn. The robot moves in a simulated corridor environment, using eight sonar, two each on the front and rear and two on each side, to judge distance from the walls. The sonar are simulated bore sight, including noise and specularity. The robot has two independent driving wheels, and the control outputs of the system are signals controlling the angular position of the axles. This is only a kinematic simulation of a dynamic system. However, the dynamics were taken in to account by using a heuristic to calculate if the vehicle was likely to skid from the difference between two sets of sequential control signals, and reducing the fitness of any controller which caused excessive skidding.

Although simulations have to be used with caution when developing controllers as inevitably many real world effects are unmodelled, this particular simulation has been used in the past to develop controllers which have subsequently been successfully implemented on a real robot.

5.1 Corridor tracker

This is a very simple application of the algorithm, as only a single level of behaviour is learned, and it is easy to specify in a number of ways.

Following trials, the best measure of fitness was found to be distance moved parallel to the axis of the corridor before collision, rather than some summed measure of perpendicular distance from the centre of the corridor. The length of a trial is limited as controllers quickly arise which are capable of avoiding the walls indefinitely. This method of assessing fitness avoids the integral of section 3 by assessing behaviour according to final position in state space. A consequence of measuring fitness entirely by distance travelled is that the robot does not try to stay in the centre of the corridor, and an off-centre path scores as well as a centred one as long as this does not lead to excessive collisions, as can be seen in figure 10.

Paths produced by a typical controller are shown in figure 10. The controller came from the 40th generation of a GA run with 30 individuals, requiring about 10 minutes of CPU time to run on a sun4. Although the controller had no knowledge of previous inputs, large changes in control signals can be avoided by keeping the control surface fairly flat, and the algorithm produced controllers which did not skid at all in a few generations.

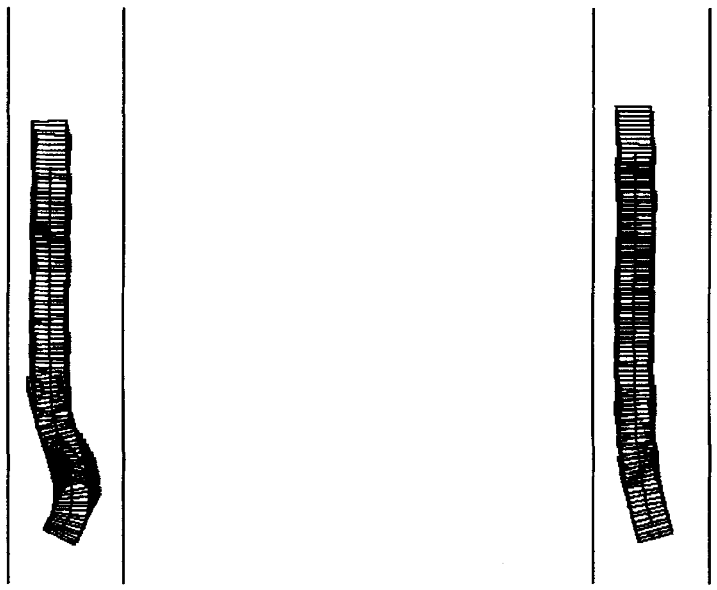

Fig. 10. Paths produced by evolved corridor tracker

5.2 Multi point turn

Performing a multi point turn is a complex manoeuvre, not least due to the fact that the control surface must have discontinuities. It was found that the GA was unable to learn a multi point turn starting from scratch using only total distance turned over a number of time steps as an objective function, so the behavioural approach to learning was used. The GA was run in the normal manner, with the objective function changing during the run to switch from learning simple local behaviours to learning the more complicated global behaviour. The initial phase of learning consisted of testing each controller from a number of starting points in state space, and allowing the controller to evaluate a single step (which could be quite a long move). The score of the controller was then based on how much the bearing of the vehicle had changed, minus a heavy penalty for colliding with a wall. The penalty for skidding was suspended, as the controller could be thought of as planning arcs rather than being a reactive motion controller. The controller effectively learned many simple behaviours in parallel, each of which would obtain the maximum change in bearing from a particular position. When combined these behaviours cause a five point turn to be performed.

A second phase of learning was then done, in which each controller was tested for a longer trial, and fitness was assessed purely on the basis of final state after a number of steps. The second phase of learning was started when no further progress was being made in the first stage, that is, the algorithm appeared to have converged.

When the second phase objective function was used from scratch, it was found that the controllers produced would learn how to perform a single leg of a turn with great precision, but never got any further than this. It is thought that this is due to the fact that the algorithm could initially get large gains in fitness by optimising single leg moves, but the controllers so produced were incapable of producing multi point turns; a good example of a GA deceptive function.

Following the initial phase of learning, the best controller in the final generation was typically capable of performing a five point turn 50 % of the time, the rest of the time becoming "stuck" half way round rather than colliding with a wall. After the second phase of learning, the success rate was over 85 %.

For this trial, the region of state space for which the controller learned to perform a turn was limited to the central regions of the corridor, with the vehicle pointing almost straight down the axis. For a controller to learn how to perform a turn from any starting position would require an extended learning period.

The initial phase of learning needed to produce the controllers used here took about 30 minutes on a sun4, with the second phase taking about the same.

Figure 11 shows some typical turns.

The convergence rate of the GA can be seen in figure 12, which is the mean logarithm of average fitness of four GA runs. After 100 generations, the objective function was changed to assess the performance of each controller as a multi point turner rather than to assess them as capable of performing simple low level behaviours. The curve shows the fitness as assessed, with a scaling factor used in the second phase of learning to maintain continuity of fitness (this is purely to improve the appearance of the graph). It can be seen that the learning rate increases when the objective function is changed, having converged in the first phase of learning.

The two stage learning process is based on the assumption that a GA, like any system that learns, can learn a complex process by breaking it down and learning the components in parallel. Once this has been achieved, improving the whole process can be undertaken.

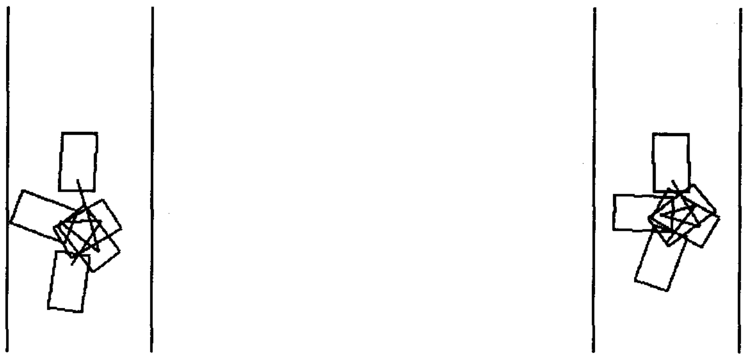

Fig. 11. Multi point turns performed by evolved controller

Obviously the learning process is not limited to two stages, there may be several levels of behavioural decomposition possible, or necessary. At present a human operator must decide on the way the problem is decomposed. However, with large class of problems this is not difficult. Broadly speaking, there are two ways of assessing behaviour; locally, or globally. Local behaviour assessment involves examining behaviour at a number of points in state space, and scoring controllers based solely on the behaviour at a number of points. Global behaviour assessment involves running long trials, and scoring controllers based on performance in an extended trial passing through many areas of state space.

Often, as is the case with performing a three point turn, it is beneficial to use a local method of behavioural assessment for the first phase of learning, and a global method for the final phase. It can be seen when learning to perform a three point turn that using a local method will only get so far, and a more global method of assessment needs to be used to "fine tune" the controllers. Using such a method from the start, however, will not work. To use a parallel from human learning, there are many stages in learning to be undertaken before a violinist can perform Brahms' concerto, or even attempt to perform it.

With more complex systems, similar reasoning applies. The role of controller designer becomes more one of tutor, as the right method of assessment has to be chosen. Using today's single stage learning algorithms this problem does not arise, but it is likely that it will be a focus for research in the future.

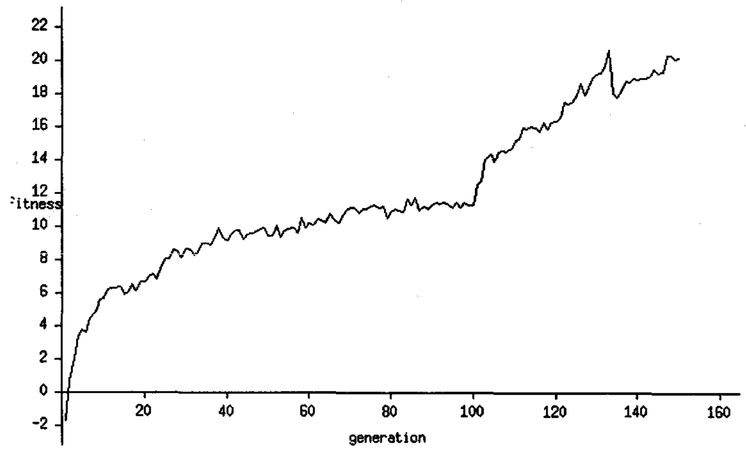

Fig. 12. Convergence rate for two stage learning

6 Conclusion

This paper has described a novel coding scheme for describing fuzzy rule bases for use in genetic algorithms, although the coding scheme could easily be extended to describe a great variety of systems. We have also described a method of minimising epistasis, and thus maximising the efficiency of the coding using a very simple method which utilises the power of the GA to improve its own performance.

The coding scheme leads to enhanced convergence when applied to simple control problems and compared to other similar schemes, although if a rule structure is assumed in advance the performance of the controllers produced can be beaten by conventional schemes. Such a structure, however, could not realistically be determined for more complex problems. We have examined the application of our algorithm to more demanding tasks, and shown how it will produce AGV controllers capable of simple tracking tasks. For more advanced behaviours, the simple approach described is insufficient, and we have described a method of avoiding the pitfalls associated with increasing complexity, and give a successful example of its implementation.

6.1 Further work

Although the coding scheme and algorithm have produced some interesting results, the problems solved have all been rather simple. We are at present working on applying the algorithm to more complicated behaviours, such as parallel parking in a space of varying size and position, without the use of external markers to indicate position.

The possibility of automatic generation of intermediate level behaviours is being considered, as although for tasks such as the multi point turn, behavioural decomposition is easy, for something more involved decomposition may not be so easy. The power of the GA is in producing emergent effects from a number of simple behaviours, and it is often very difficult even to predict these, far less design them, so it seems likely that an automatic process for generating and testing intermediate levels of behaviour will be necessary if the algorithm is to be applied further, particularly if it is to be used for designing dynamic systems that have internal states and consequently have a greatly enhanced level of behavioural complexity.

References

1. Altenberg, L.; *The Evolution of Evolvability* Ch. 3 in Ed. Kinnear, K. E.; *Advances in Genetic Programming*; MIT Press, Cambridge, MA, 1994.
2. Angeline, P. J.; *Genetic Programming and Emergent Intelligence* Ch. 4 in Ed. Kinnear, K. E.; *Advances in Genetic Programming*; MIT Press, Cambridge, MA, 1994.
3. Beasley, D., Bull, D. R. and Martin, R. R.; *An Overview of Genetic Algorithms, Part 1, Fundamentals*; University Computing, Vol. 15, No. 2, 1993.
4. Brooks, R.; *A Robust Layered Control System for a Mobile Robot*; IEEE Trans. Robotics and Automation, Vol. 2, No. 1, Mar. 1986.
5. Cooper, M. G. and Vidal, J. J.; *Genetic Design of Fuzzy Controllers*; Proc. 2nd Int. Conf. on Fuzzy Theory and Technology, Durham, NC, 1993.
6. Davidor, Y.; *Genetic Algorithms and Robotics*; World Scientific, Singapore, 1991.
7. Goldberg, D. E.; *Genetic Algorithms in Search, Optimisation and Machine Learning*;
   Addison-Wesley, 1989.
8. Hoffmann, F. and Pfister, G.; *Automatic Design of Hierarchical Fuzzy Controllers Using Genetic Algorithms*; Proc. 2nd European Congress on Intelligent Techniques and Soft Computing (EUFIT '94), Aachen, Germany, 1994.
9. Holland, J. H.; *Adaptation in Natural and Artificial Systems (2nd ed.)*; MIT Press, Cambridge, MA, 1992.
10. Karr, C. L.; *Design of a Cart-Pole Balancing Fuzzy Logic Controller using a Genetic Algorithm*; SPIE Conf. on Applications of Artificial Intelligence, Bellingham, WA, 1991.
11. Kosko, B.; *Neural Networks and Fuzzy Systems*; Prentice Hall, Englewood Cliffs, NJ, 1992.
12. Leitch, D. and Probert, P.; *Context Dependent Coding in Genetic Algorithms for the Design of Fuzzy Systems*; Proc. IEEE/Nagoya University WWW on Fuzzy Logic and Neural Nets / Genetic Algorithms, Nagoya, Japan, 1994.
13. Levenick, J. R.; *Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology*; Proc. 4th Int. Conf. on Genetic Algorithms, 1991.
14. Nomura, H., Hayashi, I. and Wakami, N.; *A Self-Tuning Method of Fuzzy Reasoning by Genetic Algorithm*; Proc. Int. Fuzzy Systems and Intelligent Control Conf., Louisville, KY, 1992.
15. Pedrycz, W.; *Fuzzy Sets and Systems*; Research Studies Press, 1989.

16. Takagi, H. and Lee, M.; *Neural Networks and Genetic Algorithm Approaches to Auto-Design of Fuzzy Systems*; Proc. 8th Austrian Artificial Intelligence Conference, FLAI '93, Springer-Verlag, Berlin, 1993.
17. Varšek, A., Urbančič, T. and Filipič, B.; *Genetic Algorithms in Controller Design and Tuning*; IEEE Trans. on Systems, Man and Cybernetics, Vol. 23, No. 5, 1993.
18. Wang, L. X. and Mendel, J. M.; *Generating Fuzzy Rules by Learning from Examples*; IEEE Trans. Systems, Man and Cybernetics, Vol. 22, No. 6, 1992.
19. Zadeh, L.; *Fuzzy Sets*; J Information and Control, Vol. 8, pp. 338 - 353, 1965.