CHAPTER 23

EVOLUTION OF FUZZY RULE BASED CONTROLLERS FOR DYNAMIC ENVIRONMENTS

Jeff Riley and Vic Ciesielski

School of Computer Science and Information Technology, RMIT University
Melbourne, Australia
E-mail:(jeff.riley@optushome.com.au,vc@cs.rmit.edu.au)

Fuzzy logic controllers have been applied to a wide range of control problems, but are very difficult to build for situations where the environment changes quickly and there is a lot of uncertainty. This work investigates a new method of creating fuzzy controllers, in the form of reactive agents, for such environments. The framework for this investigation is the RoboCup soccer simulation environment, where the agents are in the form of simulated soccer players evolved to exhibit competent dribble-and-score behaviours. The method proposed uses a messy genetic algorithm to evolve a set of behaviour producing fuzzy rules which define the agents. The results presented indicate that the messy genetic algorithm is well suited to this task, producing good performance by reducing complexity, and that the agents produced perform well in their environment. The best agent evolved is consistently and reliably able to locate the ball, dribble it to the goal and score.

1. Introduction

If an agent is able to learn behaviours it exhibits in response to stimuli, it may adapt to unpredictable, dynamic environments. Even though we may be able to describe the overall goal we expect an agent to achieve, it is not always possible to precisely describe the behaviours an agent should exhibit in achieving that goal. If we can describe a function by which we evaluate the results of the agent's behaviour against the desired

outcome, that can be used by some reinforcement learning algorithm to evolve the behaviours necessary to achieve the desired goal.

Fuzzy Sets[24] are powerful tools for the representation of uncertain and vague data. Fuzzy inference systems make use of this by applying approximate reasoning techniques to make decisions based on such uncertain, vague data. However, a fuzzy inference system on its own is not usually self-adaptive and not able to modify its underlying rulebase to adapt to changing circumstances.

Genetic algorithms[10] are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection. By combining the adaptive learning capabilities of the genetic algorithm with the approximate reasoning capabilities of the fuzzy inference system, we produce a hybrid system capable of learning the behaviour an agent needs to exhibit in order to achieve a defined goal.

There is a large body of work in the area of quasi-intelligent autonomous agents[16]. In recent times some researchers have moved away from modelling intelligent behaviour by designing and implementing complex agents. While the traditional single, complex agent approach has been shown to be successful in specialized domains such as game playing, reasoning, and path planning[17], other approaches need to be considered. One such approach is the simple agent approach in which a group of simple agents co-operate to achieve some goal. The simple agent approach forms the basis of *Artificial Life*[13]. Several variations of the multiple simple agent approach are being, or have been, investigated by different researchers: Wooldridge and Haddadi present a formal theory of *on-the-fly* co-operation amongst a group of agents[23], and Baray investigates the complexity that arises from the interaction between agents and their environment[2]. The simple agent approach would seem to be a reasonable one, and one for which the machine learning techniques described may work well. In the work presented in this chapter, the focus is on using those techniques to create simple reactive agents, rather than quasi-intelligent, complex ones.

The traditional decomposition for an intelligent control system or agent is to break processing into a chain of information processing modules proceeding from sensing to action (Figure 1).

The agent architecture implemented in the work presented in this chapter is similar to the *subsumption architecture* described by Brooks[3]. This architecture implements a layering process where simple task

achieving behaviours are added as required. Each layer is behaviour producing in its own right, although it may rely on the presence and operation of other layers. For example, in Figure 2 the *Movement* layer does not explicitly need to avoid obstacle: the *Avoid Objects* layer will take care of that.

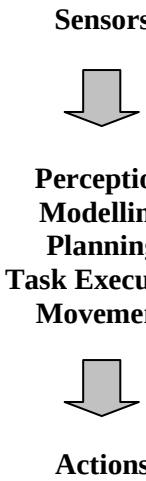


Figure 1 Traditional Agent Architecture

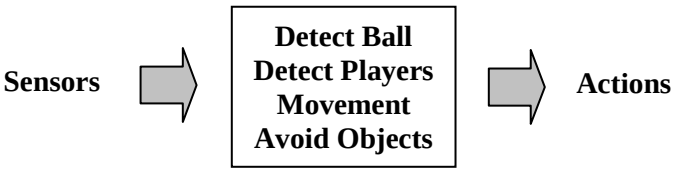


Figure 2 Brooks-style Layered Architecture for a Soccer Playing Agent

This approach creates agents with reactive architectures and with no central locus of control as described by Brooks[4]. For the work presented in this chapter the new behaviours, or behaviour producing rules, are evolved rather than designed.

This work investigates the use of an evolutionary technique in the form of a messy genetic algorithm to efficiently construct the rulebase for a fuzzy inference system to solve a particular optimisation problem. The flexibility provided by the messy genetic algorithm is exploited in the definition and format of the genes on the chromosome, thus reducing the complexity of the rule encoding from the traditional genetic algorithm. With this method the individual agent behaviours are defined by sets of fuzzy if-then rules evolved by a messy genetic algorithm. Learning is achieved through testing and evaluation of the fuzzy rulebase generated by the genetic algorithm. The fitness function used to determine the fitness of an individual rulebase takes into account the performance of the agent, based upon the number of goals scored, or attempts made to move toward goal scoring, during a game.

Previous work in the evolutionary optimisation of fuzzy system parameters can be divided into two main categories based upon the way in which the evolutionary algorithm is applied. These have become known as the *Pittsburgh* approach[20] and the *Michigan* approach[18].

The Pittsburgh approach considers each individual chromosome a complete set of rules, so the fuzzy inference system is represented by a single individual. With this approach reinforcement bandwidth is usually smaller and genetic crossover can be a cause of disruption.

The Michigan approach on the other hand considers each individual chromosome a single rule, so the fuzzy inference system is represented by the entire population. With this approach, because each individual in the population is competing with the others, care must be taken to balance cooperation and competition between individual rules.

A comparison of the Pittsburgh and Michigan approaches is presented by Pipe and Carse[19].

The genetic algorithm implemented in the work presented in this chapter is a messy genetic algorithm[8] which uses the Pittsburgh approach: each individual in the population is a complete ruleset.

There has been some work in the area of the application of evolutionary learning techniques to the challenges of RoboCup[1,7,15] but because the RoboCup environment is so large, complex and uncertain, attempts to learn the entire task have met with limited success. Andre[1], for example, did achieve some success in evolving some individual behaviours, while Luke[15] had some success evolving high-level behaviour using a pool of hand-coded low-level behaviours.

2. Goals

The primary goal of the work presented in this chapter is to investigate the potential of a fuzzy logic based controller in defining the behaviour of a reactive agent in a dynamic, uncertain environment, and the usefulness of using a messy genetic algorithm to evolve the rulebase for the controller. Furthermore, the work examines the hypothesis that the reduced complexity of the rule encoding also reduces the search space, allowing the algorithm to more quickly find reasonable solutions more quickly in a smaller, seemingly less diverse population. The framework for the investigation of this work is the RoboCup[12] soccer simulation environment, where the agents are in the form of simulated soccer players.

3. Method Description

3.1. *Overview*

Learning classifier systems[11] are an example of genetic algorithms incorporated into models of complex systems, where the classifier systems are used as models of behaviour ranging from simple stimulus-response to more complex cognitive behaviour. Classifier systems implement hierarchies of internal models that represent the environment, and the genetic algorithm uses intermittent feedback from the environment in order to discover the rules that represent those hierarchies.

This work implements a method involving the use of a messy genetic algorithm and a fuzzy inference system in which the messy genetic algorithm is used to determine, by simulated evolution, the fuzzy ruleset which defines the set of behaviours exhibited by reactive agents in response to stimuli.

An indicative example of previous work in which messy genetic algorithms are used to evolve fuzzy rules is given by Hoffmann and Pfister[9]. There a messy genetic algorithm was used to evolve a fuzzy controller for an autonomous vehicle capable of travelling to a destination and avoiding obstacles along the way. A significant difference between previous work and the work presented in this chapter

is that the agent or controller evolved here is able to cope with an uncertain, rapidly changing environment.

In addition to the primitives defined by the RoboCup system (*dash*, *kick*, *turn* etc.), the agent being evolved is endowed with a specific set of mid-level hand-coded soccer-playing skills. These are:

RunTowardBall: the agent *dashes* once in the direction of the ball, provided the direction to the ball is known.

RunTowardMyGoal: the agent *dashes* once in the direction of its own goal, provided the direction to the goal is known.

Dribble: the agent *kicks* the ball once in the direction it is facing, then *dashes* once in that direction.

DribbleTowardMyGoal: the agent *kicks* the ball once in the direction of its own goal, then *dashes* once in that direction, provided the direction to the goal is known.

KickTowardMyGoal: the agent *kicks* the ball once towards its own goal, provided the direction to the goal is known.

GoToBall: the agent *dashes* towards ball until it is within kicking distance of the ball, provided the direction to the ball is known.

DoNothing: the agent takes no action.

The agent will perform one of these actions in response to external stimuli; the specific response being determined by the fuzzy rulebase. If no action is indicated given the information known by the agent (that is, no rule fires), the agent will turn 90° in a randomly chosen direction in an effort to locate the ball or goal.

The external stimuli used as input to the fuzzy inference system is most of the visual information supplied by the soccer server: information regarding the location of opponents and team mates is not used at this stage, and only sufficient information to situate the agent and locate the ball is used.

3.2. *Genetic Algorithms*

The method investigated by this work results in a fuzzy rule base developed by the use of a *messy* genetic algorithm. In this method, fuzzy rulesets are encoded onto variable length chromosomes, and an initial

population of chromosomes is evolved to produce a fuzzy ruleset which defines the behaviours of the soccer playing agent.

3.2.1. Messy Genetic Algorithms

In classic genetic algorithms the chromosome is defined as a fixed length structure; commonly a fixed length bit string. With this definition each gene is guaranteed to occur only once, and its meaning is defined by its position in the structure. A messy genetic algorithm on the other hand, encodes a chromosome as a variable length structure comprised of tuples of values, with each tuple describing a gene. In this work, a gene is described by a triplet representing a fuzzy clause and connector, with the first element denoting the input variable, the second the fuzzy set membership (or fuzzy variable) of this input variable, and the third the clause connector. The rule consequent gene is specially coded to distinguish it from premise genes allowing multiple rules, or a ruleset, to be encoded onto a single chromosome. An example chromosome fragment is shown in Figure 3.

(Ball, Left, And)	(MyGoal, Far, Or)	(Dribble, Slow, *)

Figure 3 Messy Genetic Algorithm Example Chromosome Fragment

Some features of the chromosome in a messy genetic algorithm are:

- a gene is encoded as a tuple describing the gene's meaning, value and other relevant information.
- genes may occur multiple times.
- genes are not guaranteed to be present.
- genes may be permutated in any way.

For example, the chromosome fragments shown in Figure 4 are valid even though a gene is repeated. Furthermore, the chromosome fragments are equivalent even though the genes are ordered differently.

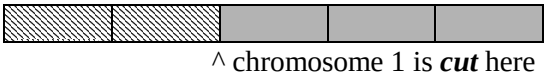
(Ball, Left, And)	(MyGoal, Far, Or)	(Ball, Left, And)

(Ball, Left, And)	(Ball, Left, And)	(MyGoal, Far, Or)

Figure 4 Valid and Equivalent Chromosome Fragments in a Messy Genetic Algorithm

For messy genetic algorithms, the selection and mutation operators are implemented in the same manner as for classic genetic algorithms. The crossover operator, however, is implemented as a combination of two new operators: *cut* and *splice*. The *cut* operator cuts each chromosome at a randomly chosen position, and since the chromosomes may be of different lengths, the resultant fragments may also be of different lengths. The *splice* operator concatenates the fragments produced by the *cut* operator, resulting in two new chromosomes of possibly different lengths from the original chromosomes. Figure 5 is an example of the *cut* and *splice* operations for a messy genetic algorithm.

It has been shown that messy genetic algorithms are useful tools for solving difficult optimisation problems. Recent work with messy genetic algorithms includes work on multiobjective optimisation[22] and the vehicle routing problem[21]. The work presented in this chapter uses the messy genetic algorithm to optimise the ruleset for the fuzzy inference system.



^ chromosome 1 is ***cut*** here

^ chromosome 2 is ***cut*** here

Figure 5a Messy Genetic Algorithm Cut Operation

second fragment of chromosome 2 is ***spliced*** to first fragment of chromosome 1

second fragment of chromosome 1 is ***spliced*** to first fragment of chromosome 2

Figure 5b Messy Genetic Algorithm Cut Operation

3.3. *Fuzzy Inference Systems*

A fuzzy inference system is a framework based on the concept of fuzzy set theory, fuzzy *if-then* rules and fuzzy reasoning. The fuzzy inference system is comprised of a number of fuzzy *if-then* rules, definitions of the

membership functions of the fuzzy sets operated on by those rules, and a reasoning mechanism to perform the inference procedure (Figure 6). The application of the fuzzy rule base by the inference procedure to external stimuli provided by the soccer server results in one or more fuzzy rules being executed and some action being taken by the client.

In this work the fuzzy rule base is developed by the use of a messy genetic algorithm. The messy genetic algorithm evolves the fuzzy rule base during a series of simulated training soccer games in which individuals are rewarded for goals scored. The membership functions of the input and output fuzzy sets are standard trapezoidal functions which are pre-defined and fixed, so not modified by the genetic algorithm.

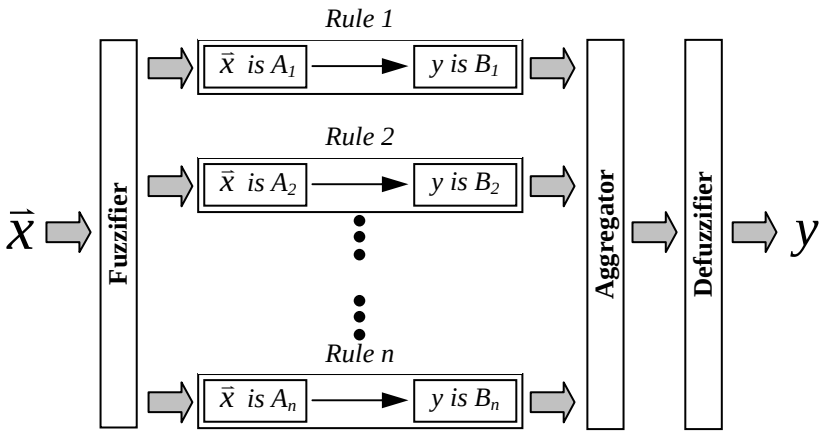


Figure 6 Fuzzy Inference System

The external stimuli given as input to the fuzzy inference system is fuzzified to represent the degree of membership of one of four fuzzy sets: direction, distance, speed and power. For example, the visual information supplied by the soccer server is interpreted as fuzzy relationships such as *Ball is Near*, *MyGoal is VeryFar*, *Ball is SlightlyLeft.*

To evolve a *dribble-and-score* behaviour, only that information required to locate the agent's goal, the ball, and to situate the agent is given as input to the agent.

The fuzzy rules developed by the genetic algorithm are of the form:

if *Ball* is *Near* and *MyGoal* is *Near* then *KickTowardMyGoal Soft*
if *Ball* is *Far* or *Ball* is *SlightlyLeft* then *RunTowardBall Fast*

The output of the fuzzy inference system is a number of *(action, value)* pairs, corresponding to the number of fuzzy rules with unique consequents. The *(action, value)* pairs define the action to be taken by the agent, and the degree to which the action is to be taken. For example:

(KickTowardMyGoal, power)
(RunTowardBall, speed)
(Turn, direction)

where *power*, *speed* and *direction* are crisp values representing the defuzzified fuzzy set membership of the action to be taken.

Only one action is performed by the agent in response to stimuli provided by the soccer server. Since several rules with different actions may fire, actions are assigned a priority and the highest priority action is performed.

3.4. *Detailed Method Description*

Input variables for the fuzzy rules developed by this method are fuzzy interpretations of the visual stimuli supplied to the agent by the soccer server. Output variables are the fuzzy actions to be taken by the agent. The universe of discourse of both input and output variables are covered by fuzzy sets, the parameters of which are predefined and fixed. Each input is fuzzified to have a degree of membership in the fuzzy sets appropriate to the input variable.

The encoding scheme implemented for this method exploits the capability of messy genetic algorithms to encode information of variable structure and length. The basic element of the coding of the fuzzy rules is a triplet representing a fuzzy clause and connector, with the first element denoting the input variable, the second the fuzzy set membership (or fuzzy variable) of this input variable, and the third the clause connector. The rule consequent gene is specially coded to distinguish it from premise genes allowing multiple rules, or a ruleset, to be encoded

onto a single chromosome. Chromosomes are not fixed length: the length of each chromosome in the population varies with the length of individual rules and the number of rules on the chromosome. The number of clauses in a rule and the number of rules in a ruleset is only limited by the maximum size of a chromosome. The minimum size of a rule is two clauses (one premise and one consequent), and the minimum number of rules in a ruleset is one.

The set of input variables for the premise clauses is:

(Ball, MyGoal)

and for the consequent clauses:

(Turn, Kick, KickTowardMyGoal, Dribble, DribbleTowardMyGoal, Run, RunTowardMyGoal, RunTowardBall, GoToBall, DoNothing)

The fuzzy variables for each of the fuzzy sets DISTANCE, POWER and DIRECTION which describe the input or action variables for both the premise and consequent clauses are:

DISTANCE:	(At, VeryNear, Near, SlightlyNear, MediumDistant, SlightlyFar, Far, VeryFar)
POWER:	(VeryLow, Low, SlightlyLow, MediumPower, SlightlyHigh, High, VeryHigh)
DIRECTION:	(Left180, VeryLeft, Left, SlightlyLeft, Straight, SlightlyRight, Right, VeryRight, Right180)

Premise clauses can be further modified by the use of a not operator. The set of possible clause connectors is:

(and, or, *),

where * indicates the connector is not used.

The DISTANCE, POWER and DIRECTION fuzzy sets are shown in Figure 7. The parameters for these fuzzy sets were not learned by the evolutionary process, but were fixed empirically. The initial values were set having regard to RoboCup parameters and variables, and fine-tuned after some experimentation.

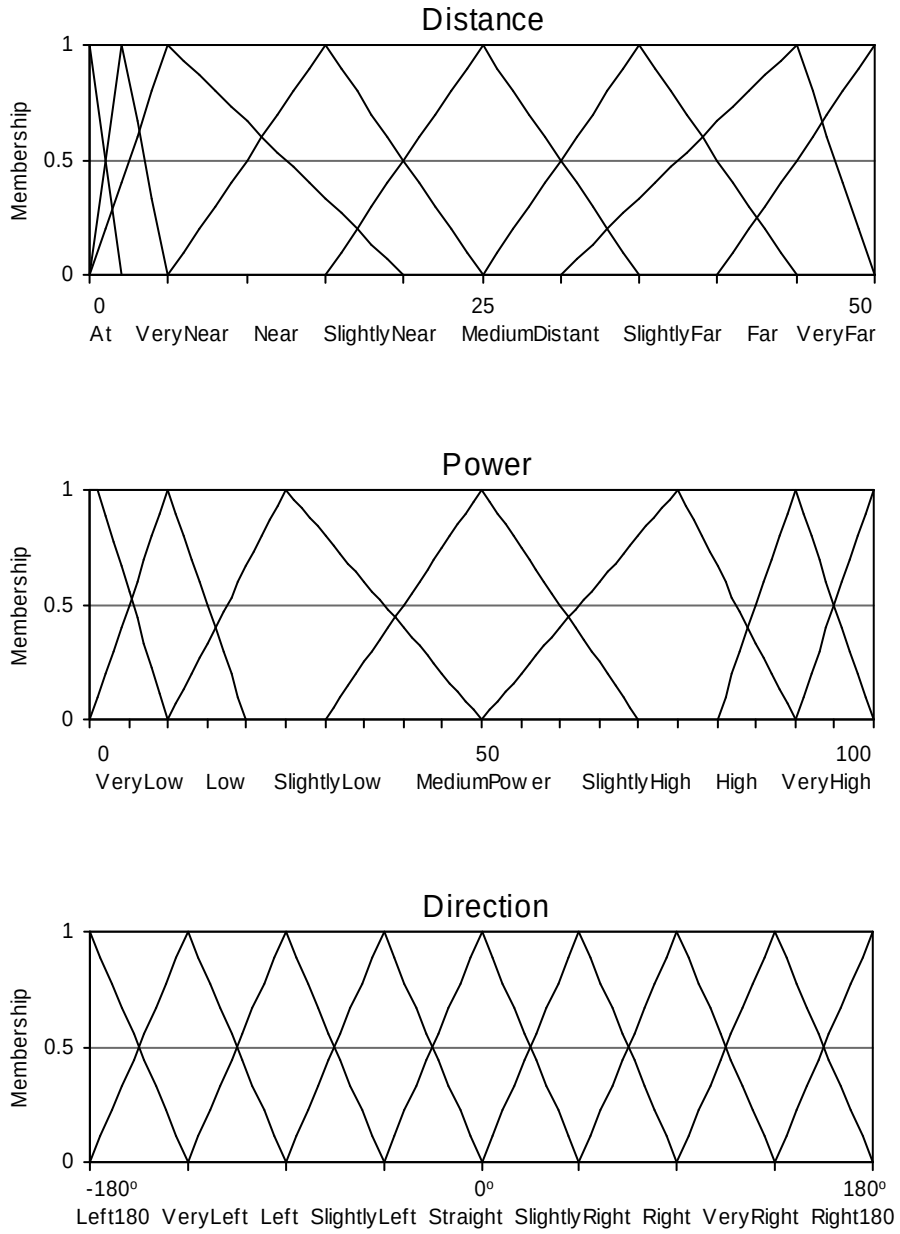
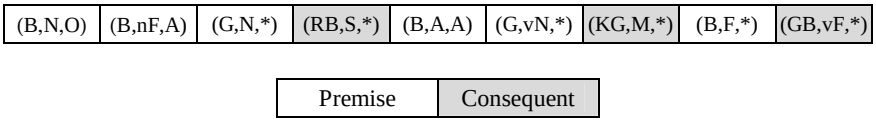


Figure 7 Distance, Power and Direction Fuzzy Sets

An example chromosome and corresponding rules are shown in Figure 8.



Rule 1: if *Ball* is *Near* or *Ball* is *not Far* and *MyGoal* is *Near* then *RunTowardBall Slow*
Rule 2: if *Ball* is *At* and *MyGoal* is *VeryNear* then *KickTowardMyGoal MediumPower*
Rule 3: if *Ball* is *Far* then *GoToBall VeryFast*

Figure 8 Chromosome and corresponding rules

The genetic operators implemented are cut, splice and mutation. As previously described, cut and splice are analogous to the crossover operation of classic genetic algorithms; the mutation operator is the same as that of the classic genetic algorithm. Since chromosomes are variable in length and can contain multiple rules, each chromosome represents a complete ruleset.

In contrast to classic genetic algorithms which use a fixed size chromosome and require *don't care* values in order to generalise, no explicit *don't care* values are implemented for any attributes in this method. Since messy genetic algorithms encode information of variable structure and length not all attributes, particularly premise variables, need be present in any rule, or indeed in the entire ruleset. In other words the format of the messy genetic algorithm implies *don't care* values for all attributes since any attribute (premise variable) may be omitted from any or all rules, so generalisation is an implicit feature of this method.

4. Results

In the trials for which the results are presented here:

- The *Roulette Wheel* method of selection for crossover was used, and the probability of crossover occurring after selection was 0.8.
- Each generation was mutated by selecting 10% of the population for possible mutation, then subjecting those selected individuals to a probability of mutation of 0.35. For each individual, a single gene was randomly selected for mutation: for a premise gene the

input variable, fuzzy variable or connector was mutated; and for a consequent gene the input variable or fuzzy variable was mutated. Mutation consisted of replacement by a randomly selected value.

Individuals were rewarded, in order of importance, for

- the number of goals scored in a game
- the number of times the ball was kicked during a game

A game was played with the only player on the field being the agent under evaluation. The agent was placed randomly on its half of the field and oriented so that it was facing the end of the field to which it was kicking, and the ball was placed at the centre of the field. A game was terminated when:

- the target fitness of 0.05 was reached
- the ball was kicked out of play
- 120 seconds expired
- 10 seconds of no player movement expired

The target fitness of 0.05 reflects a score of 10 goals in the playing time of 120 seconds. This figure was chosen to allow the player a realistic amount of time to develop useful strategies yet terminate the search upon finding a very good individual.

Two methods of terminating the evolutionary search were implemented. The first stops the search when a specified maximum number of generations have occurred; the second stops the search when the best fitness in the current population becomes less than a specified threshold. Both methods were active, with the first to be encountered terminating the search.

The results of several trials are presented below. Each trial consisted of a population of 200 randomly initialised chromosomes evolved over 25 generations.

Figure 9 shows the average fitness of the population after each generation for each of 10 trials, showing that the performance of the population does improve steadily and plateaus towards goal-scoring behaviour (i.e. a fitness of 0.5).

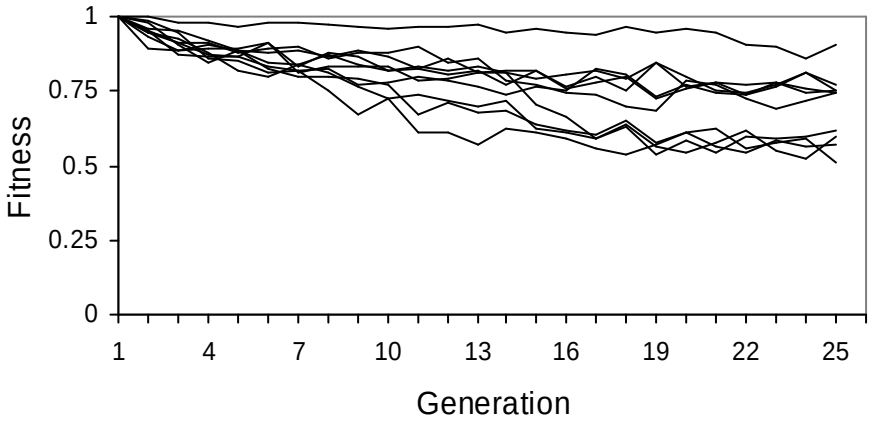


Figure 9 Average Fitness Curves for 10 Trials

Figure 10 shows the best individual fitness from the population after each generation for each of 10 trials, showing that good individuals are found after very few generations in contrast to the gradual improvement in average fitness (Figure 9).

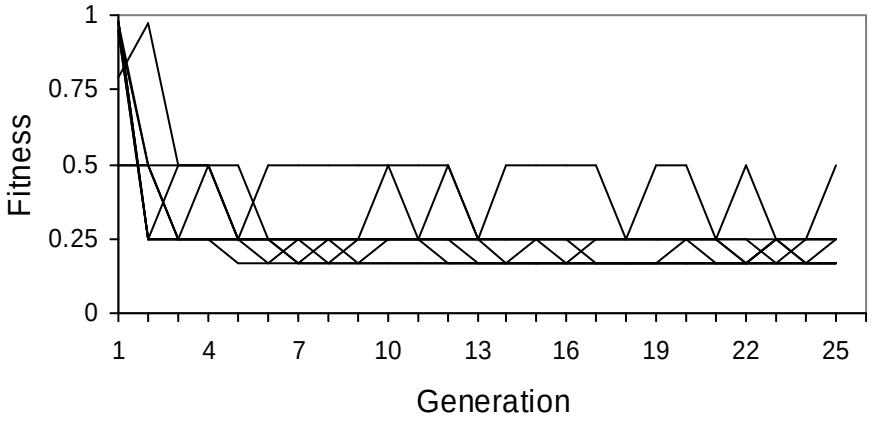


Figure 10 Best Fitness Curves for 10 Trials

Figure 11 is another visualisation of the progressive learning of the population from generation to generation, showing that not only do more players learn to kick goals over time, they learn to kick more goals more quickly. The histogram shows the average number of individuals, from a population of 200, which scored 0, 1, 2 or 3 goals from each generation of the 10 trials.

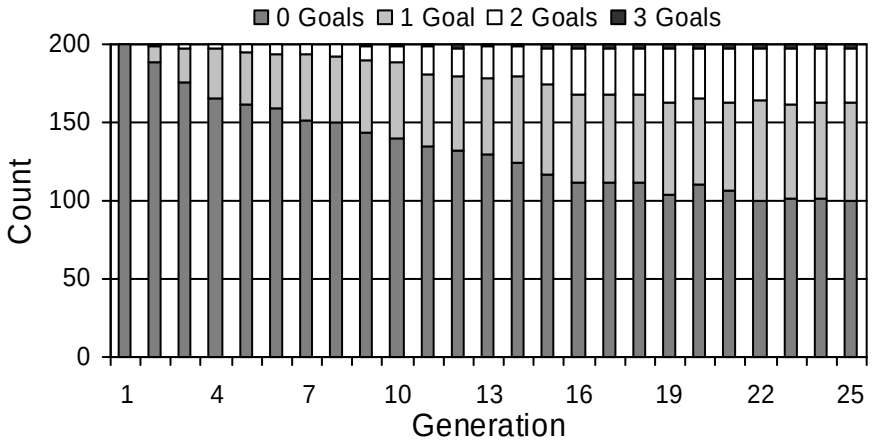


Figure 11 Goals Scored

The actual fitness function used was

$$f = \begin{cases} \begin{cases} 1.0 & , kicks = 0 \\ 0.5 & , kicks > 0 \end{cases} & , ticks = 0 \\ 1.0 - \dfrac{kicks}{2.0 \times ticks} & , ticks > 0 \end{cases} , goals = 0$$

$$\frac{1.0}{2.0 \times goals} \quad , goals > 0$$

where

goals = the number of goals scored by the agent
kicks = the number of times the agent kicked the ball
ticks = the number of soccer server time steps

The function chosen indicates a better fitness as a lower number so representing the optimisation of fitness as a minimisation problem. This function was chosen to reward agents for goals scored. Agents that do not score goals are rewarded for the number of times the ball is kicked on the assumption that an agent which actually kicks the ball is more likely to produce offspring capable of scoring goals. The expectation was that evolutionary pressures would cause the average fitness of the population to decrease, with individual fitness for some individuals decreasing more rapidly. The data presented indicates that this expectation was realised.

The rules evolved by the genetic algorithm for the best performing player from a typical evolutionary run (25 generations) were:

if *MyGoal* is *VeryNear* or *MyGoal* is *At* and *MyGoal* is *VeryRight* or *MyGoal* is *Right* and *Ball* is *SlightlyLeft* and *Ball* is *Near* and *Ball* is *VeryLeft* or *MyGoal* is *Right180* or *Ball* is *Far* or *MyGoal* is *Near*
then *Kick VerySoft*

if *MyGoal* is *VeryNear* or *Ball* is *VeryNear* and *Ball* is *SlightlyRight* and *Ball* is *Far*
then *GoToBall SlightlyHard*

if *MyGoal* is *not VeryRight* or *Ball* is *VeryFar*
then *DribbleTowardMyGoal Soft*

if *MyGoal* is *MediumDistant* or *MyGoal* is *not Left* and *Ball* is *not Right180* or *MyGoal* is *SlightlyRight* or *MyGoal* is *VeryLeft* or *MyGoal* is *VeryFar* and *MyGoal* is *not Left180* or *Ball* is *Left180* and *MyGoal* is *SlightlyNear*
then *DribbleTowardMyGoal MediumPower*

if *MyGoal* is *VeryNear* or *MyGoal* is *Left180* and *Ball* is *Near* and *MyGoal* is *Right*
then *Dribble Hard*

The player defined by this ruleset achieved a fitness value of 0.1667 by kicking 3 goals in the allotted time of 120 seconds. The best performing players from the trials were each tested in 100 trials of 120 seconds, with the player being placed in a different, randomly selected

starting position for each trial. The best performing players in these tests scored one or more goals in 60% of the trials. Typically players begin the game by hunting for the ball, then once the ball is located the players generally dribble the ball towards the goal in a reasonably direct route. Because the player developed is reactive and almost no state information is recorded (by the player), there are times when it loses sight of the ball or goal. These situations are characterised by the player momentarily hunting for the ball, or kicking the ball in the wrong direction.

The average time for each training run of 25 generations with a population size of 200 individuals was 100 hours. Several training runs were performed with a larger population of 1000 individuals with no significant increase in effectiveness, consistent with findings in other work in this area[14]. In all trials the average generational performance of the population tended to plateau after some time (see Figure 9), but a very good individual was found early in the search (see Figure 10).

5. Conclusions

This work investigates a method of creating reactive agents that uses a messy genetic algorithm to evolve fuzzy rules which define the agent's behaviour. The method consistently evolved players in very few generations that displayed very good goal scoring behaviour, thus demonstrating that this method can be used to successfully train a *dribble-and-score* behaviour in a reactive soccer playing agent. The RoboCup environment is a complex, dynamic and uncertain environment, and the results presented indicate that the method described can create controllers or agents for complex situations where the environment changes quickly and there is a lot of uncertainty. A useful next step is to use the method to evolve agents for the even more complex environment of a simulated game of soccer involving many players.

The good performance of the method with a small population and relatively few generations is likely to be due in part to the reduced complexity of the rule encoding afforded by the flexibility of the messy genetic algorithm. This would seem to reduce the search space, so allowing the algorithm to find reasonable solutions more quickly in a seemingly less diverse population.

The selection of mid-level, hand-coded composite skills rather than simple RoboCup primitives is likely to have had a positive effect on the performance since learning those skills is difficult and time consuming[5,6]. By pre-defining the composite skills the genetic algorithm was able to

search for the higher-level strategies necessary for a good *dribble-and-score* behaviour rather than the low-level skills.

Since this method produces human-readable rules which govern the behaviour of the agents, it is possible to gain some understanding of the (often novel) knowledge that the agent has learned through the evolutionary process. This is considered an advantage over many existing methods of automatically creating agents or controllers where often the learned behaviour is not apparent and not easily extracted. A useful avenue for further work, made possible by the human-readable form of the rules, is the post-processing and optimisation of the evolved rules.

References

1. Andre, D. and Teller, A. Evolving Team Darwin United. In Minoru Asada and Hioaki Kitano, editors, RoboCup-98: Robot Soccer World Cup II. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1999.

2. Baray, C. Evolution of Coordination in Reactive Multi-Agent Systems. PhD Thesis, Computer Science Department, Indiana University, Bloomington, Indiana, 1999.

3. Brooks, R. Robust Layered Control System for a Mobile Robot. A.I Memo 864, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1985.

4. Brooks, R. Intelligence without Representation. Artificial Intelligence, 47:139-159, 1991.

5. Ciesielski, V., Mawhinney, D. and Wilson, P. Genetic Programming for Robot Soccer. In Proceedings of the RoboCup 2001 International Symposium, Lecture Notes in Artificial Intelligence 2377, pp 319-324. Springer-Verlag, Berlin, 2002.

6. Ciesielski, V. and Lai, S. Y. Developing a Dribble-and-Score Behaviour for Robot Soccer Using Neuro Evolution. In Proceedings of the 5th Australia-Japan Joint Workshop on Intellignet and Evolutionary Systems, pp 70-78, Dunedin, New Zealand, 2001.

7. Ciesielski, V. and Wilson, P. Developing a Team of Soccer Playing Robots by Genetic Programming. In Proceedings of The Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems, pp101-108, Canberra, Australia, 1999.

8. Goldberg, D., Korb, B., and Deb, K. Messy Genetic Algorithms: Motivation, Analysis, and First Results. In Complex Systems, 3, 1989.

9. Hoffmann, F. and Pfister, G. Evolutionary Learning of a Fuzzy Control Rule Base for an Autonomous Vehicle. In Proceedings of the Fifth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, pp659-664. Granada, Spain, 1996.

10. Holland, J. Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press, 1975.

11. Holland, J., Holyoak, K., Nisbett, R. and Thagard, P. Induction: Processes of Inference, Learning, and Discovery. MIT Press, 1986.

12. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. RoboCup: The Robot World Cup Initiative. In Working Notes of the 1995 International Joint Conference on Artificial Intelligence Workshop on Entertainment and AI/Alife, pp19-24. Montreal, Canada, 1995.

13. Langton, C. (Ed.) Artificial Life. Addison-Wesley, 1989.

14. Luke, S. When Short Runs Beat Long Runs. In Proceedings of the 2001 Genetic and Evolutionary Computation Conference, pp74-80. San Francisco CA, USA, 2001.

15. Luke, S., Hohn, C., Farris, J., Jackson, G. and Hendler, J. Coevolving Soccer Softbot Team Coordination with Genetic Programming. In Hioaki Kitano, editor, RoboCup-97: Robot Soccer World Cup I. Lecture Notes in Artificial Intelligence No. 1395, pp398-411. Springer-Verlag, Berlin, 1999.

16. Maes, P. Designing Autonomous Agents. The MIT Press, 1990.

17. Nilsson, N. Artificial Intelligence: A New Synthesis. Morgan Kaufmann, San Francisco, CA, 1998.

18. Parodi, A. and Bonelli, P. A New Approach to Fuzzy Classifier Systems. In Proceedings of the Fifth International Conference on Genetic Algorithms, pp223-230. San Mateo CA, USA, 1993. Morgan Kaufman.

19. Pipe, A. and Carse, B. Autonomous Acquisition of Fuzzy Rules for Mobile Robot Control: First Results From Two Evolutionary Computation Approaches. In Proceedings of the 2000 Genetic and Evolutionary Computation Conference, pp849-856. Las Vegas NV, USA, 2000.

20. Smith, S. A Learning System Based on Genetic Adaptive Algorithms, Doctoral Thesis, Department of Computer Science, University of Pittsburgh, Pittsburgh. PA, USA, 1980.

21. Tan, K., Lee, T., Ou, K., and Lee, L. A Messy Genetic Algorithm for the Vehicle Routing Problem With Time Window Constraints. In Proceedings of the 2001 Congress on Evolutionary Computation, pp679-686. Seoul, Korea, 2001.

22. van Veldhuizen, D., and Lamont, G. Multiobjective Optimization with Messy Genetic Algorithms. In Proceedings of the Fifteenth ACM Symposium on Applied Computing (Evolutionary Computation and Optimization Track), pp470-476. Como, Italy, 2000.

23. Wooldridge, M. and Haddadi, A. Making It Up As They Go Along: A Theory of Reactive Cooperation. In W. Wobcke, M. Pagnucco, and C. Zhang, editors, Agents and Multi-Agent Systems -- Formalisms, Methodologies, and Applications (LNAI Volume 1441). Springer-Verlag, 1998.

24. Zadeh, L. Fuzzy Sets. Journal of Information and Control, Vol 8, 1965.