EVOLVING FUZZY RULES FOR REACTIVE AGENTS IN DYNAMIC ENVIRONMENTS

Jeff Riley
School of Computer Science
and Information Technology
RMIT University
Melbourne, AUSTRALIA
jeff_riley@hp.com
+61 3 9272 2967

Vic Ciesielski
School of Computer Science
and Information Technology
RMIT University
Melbourne, AUSTRALIA
vc@cs.rmit.edu.au
+61 3 9925 2926

ABSTRACT

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, enabling a performance improvement over traditional evolutionary methods 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 [20] 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 [11] 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.

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 [14], other approaches need to be considered. One such approach is the so-called dumb, or simple, agent approach in which a group of simple agents co-operate to achieve some goal. Several variations of this 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 in [19], and Baray investigates the complexity that arises from the interaction between agents and their environment in [1]. 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 paper, the focus is on using those techniques to create simple reactive agents, rather than 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 paper is similar to the *subsumption architecture* described by Brooks in [3]. This architecture implements a layering process where simple task achieving

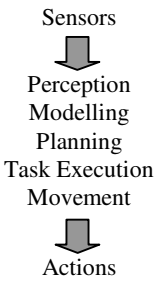

Figure 1: Traditional Agent Architecture.

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.

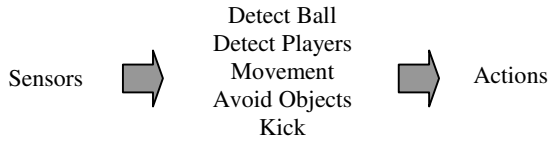

Figure 2: Brooks-style Layered Architecture for a Soccer Playing Agent.

This approach creates agents with reactive architectures and with no central locus of control as described by Brooks in [4] and [5]. For the work presented in this paper 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 [6], [12] and the *Michigan* approach [2], [15].

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 in [16]. A further approach, described in [7] and [9], uses an iterative approach to learning the fuzzy rules.

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

2. GOALS

The primary goal of this work 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 [13] soccer simulation environment, where the agents are in the form of simulated soccer players.

3. METHOD DESCRIPTION

3.1. Overview

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 in [10]. 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 paper is that the agent or controller evolved here is able to cope with an uncertain, rapidly changing environment.

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

TurnToBall: the agent *turns* to face the ball, provided the direction to the ball is known.

TurnToMyGoal: the agent *turns* to face its own goal, provided the direction to the goal is known.

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.

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 rule set 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 rule set, 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.

In 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.

(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.

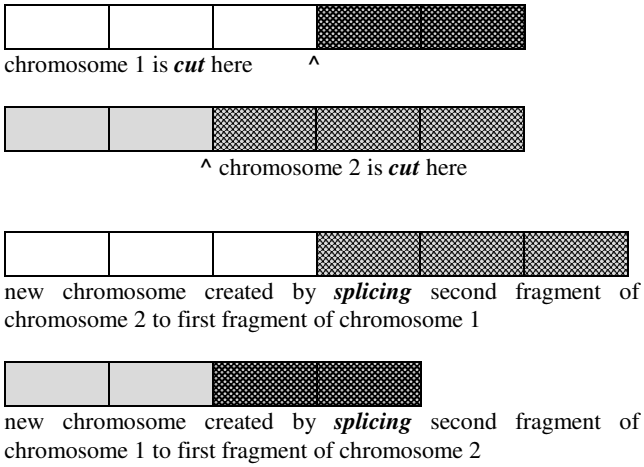

Figure 5: Cut and Splice Operations for a Messy Genetic Algorithm.

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 [18] and the vehicle routing problem [17]. The work presented in this paper uses the messy genetic algorithm to optimise the ruleset for the fuzzy inference system.

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.

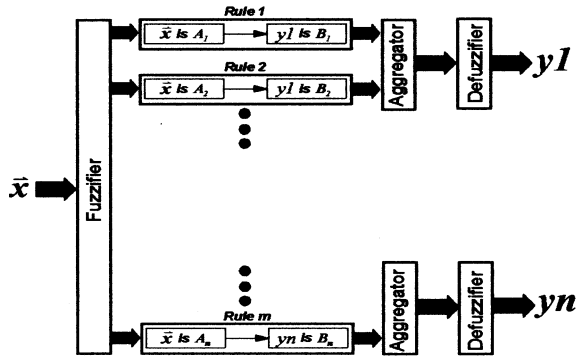

Figure 6: Fuzzy Inference System.

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:

The *Ball* is *Near*
MyGoal is *Very Far*
The *Ball* is a *Slightly Left*

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*
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 rule set, to be encoded onto a single chromosome. The number of rules in a rule set is only limited by the maximum size of a chromosome.

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, Very Near, Near, Slightly Near, Medium, Slightly Far, Far, Very Far)
POWER:	(Very Low, Low, Slightly Low, Medium, Slightly High, High, Very High)
DIRECTION:	(Left180, Very Left, Left, Slightly Left, Straight, Slightly Right, Right, Very Right, Right180)

Each of these 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. i.e. in the final premise and consequent clauses of a rule. An example chromosome and corresponding rules are shown in Figure 7.

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 rule base.

(B,N,O)	(B,nF,A)	(M,N,*)	(RB,S,*)	(B,A,A)	(MG,vN,*)	(KG,M,*)	(BA,F,*)	(GB,vF,*)

Premise	Consequent

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 *Very Near* then *KickTowardMyGoal Medium*
Rule 3: if *Ball* is *Far* then *GoToBall Very Fast*

Figure 7: Chromosome and Corresponding Rules.

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*. So a maximum of *3.5%* of the population was mutated. 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 is placed randomly on its half of the field and oriented so that it is facing the end of the field to which it is kicking. 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

Two methods of terminating the evolutionary search were implemented. The first stops the search when a specified maximum number of generations have evolved; 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.

Several trials were performed. Each trial consisted of a population of *200* randomly initialised chromosomes evolved over *25* generations. The results for the trials were remarkably similar, and the results for a typical trial are presented in Figure 8. Presented in Figure 8 is, for each generation of the trial, the average fitness of the individuals in the population, the standard deviation of the fitnesses of the individuals in the population, and the best individual fitness in the population.

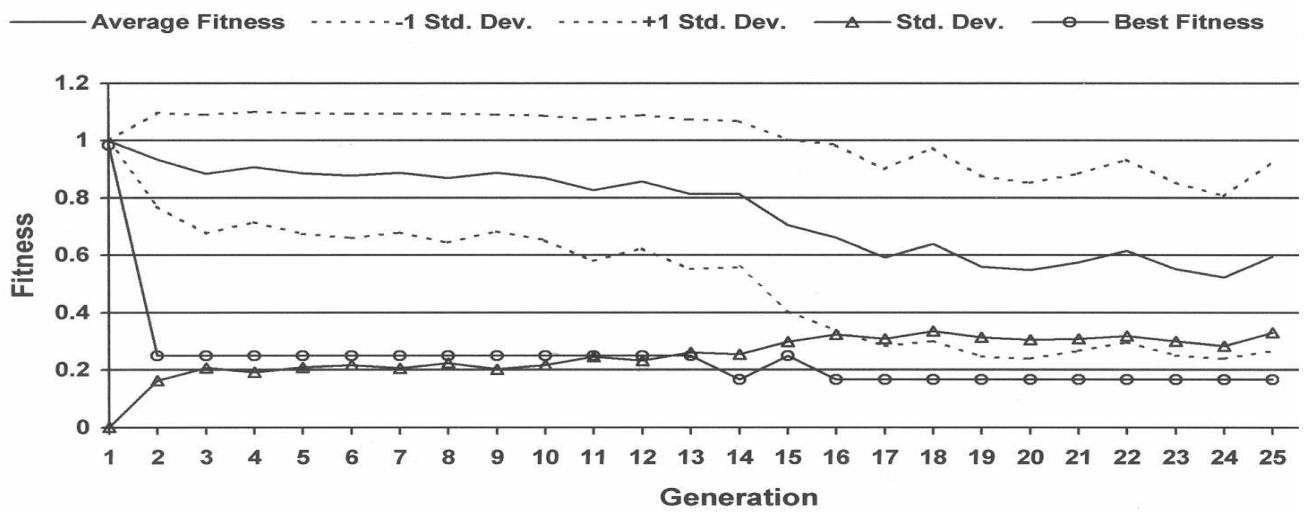

Figure 8: Trial Results.

The actual fitness function used was

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

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

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 individuals 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 *Very Near* or M*yGoal* is *At* and
MyGoal is *Very Right* or *MyGoal* is *Right* and
Ball is *a Little Left* and *Ball* is *Near* and
Ball is *Very Left* or
MyGoal is *Backward to the Right* and *Ball* is *Far* or
MyGoal is *Near*
then *Kick Very Soft*

if *MyGoal* is *Very Near* or *Ball* is *Very Near* and
Ball is *a Little Right* or *Ball* is *Far*
then *GoToBall Very Soft*

if *MyGoal* is *not Very Right* or *Ball* is *Very Far* and
Ball is *a Little Right* or *Ball* is *Far*
then *DribbleTowardMyGoal Soft*

if *MyGoal* is *Medium Distant* or M*yGoal* is *not Left* and
Ball is *not Backward to the Right* or
MyGoal is *a Little Right* or *MyGoal* is *Very Left* or
MyGoal is *Very Far* or
MyGoal is *not Backeward to the Left* and
Ball is *Backward to the Left* and
MyGoal is *Somewhat Near*
then *DribbleTowardMyGoal Very Soft*

if *MyGoal* is *Very Near* or
MyGoal is *Backward to the Left* 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. In subsequent tests, while not achieving a perfect record of kicking a goal in each test, the best performing player did succeed in kicking goals in more than *75%* of tests. Because the agent developed is reactive and almost no state information is recorded by the agent, there are times when it loses sight of the ball or goal. These situations are characterised by the agent momentarily hunting for the ball, or kicking the ball in the wrong direction.

The training and tests for this work were conducted on an HP715/100 Unix workstation with 64MB of memory running HP-UX 11.0. 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, both with a larger population of *1000* individuals, and with a reduced number of generations (*10*) with a population of *200*. In the case of the increased population, in several runs there was no reasonable player found, and when there was it took longer to occur. In the case of the reduced number of generations, though the population did not converge, a reasonable player was found within the *10* generations.

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 results obtained demonstrate that the method can be used to successfully train a *dribble-and-score* behaviour in a reactive soccer playing agent. This indicates that the method can create controllers or agents for uncertain complex environments, and that a useful next step is to use the method to evolve agents for the more complex environment of a simulated game of soccer involving many players.

The good performance of the method with a smaller population and fewer generations is likely to be due to the reduced complexity of the rule encoding afforded by the flexibility of 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.

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.

6. REFERENCES

[1] Baray, C. Evolution of Coordination in Reactive Multi-Agent Systems. *PhD thesis, Computer Science Department, Indiana University, Bloomington, Indiana, 1999.*

[2] Bonarini, A. An Introduction to Learning Fuzzy Classifier Systems. In *P.L. Lanzi, W. Stolzmann and S.W. Wilson (Eds.), Learning Classifier Systems - from Foundations to Applications, Lecture Notes in Artificial Intelligence, 83-104. Springer Verlag, Berlin Heidelberg, Germany.*

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

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

[5] Brooks, R. Intelligence without Reason. *A.I Memo 1293, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, April 1991.*

[6] Carse, B., Fogarty, T., and Munro A. Evolving Fuzzy Rule-based Controllers using Genetic Algorithms. *Fuzzy Sets and Systems 80(3):273-293, 1996.*

[7] Cordón, O., and Herrera, F. A Three-stage Evolutionary Process for Learning Descriptive and Approximate Fuzzy Logic Controller Knowledge Bases From Examples. *International Journal of Approximate Reasoning, 17(4):369-407, 1997.*

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

[9] González, A., and Pérez, R. SLAVE: A Genetic Learning System Based on an Iterative Approach. *IEEE Transactions on Fuzzy Systems, 7(2):176-191, 1999.*

[10] 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 (IPMU-96): 659-664. Granada, Spain, 1996.*

[11] Holland, J. Adaptation in Natural and Artificial Systems. *Ann Arbor: The University of Michigan Press, 1975.*

[12] Hwang, W., and Thompson W. Design of Fuzzy Logic Controllers Using Genetic Algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'94): 1383-1388. Piscataway NJ, USA, 1994. IEEE Computer Press.*

[13] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. RoboCup: The Robot World Cup Initiative. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence (IJCAI'95), Workshop on Entertainment and AI/Alife, Montreal, Canada, 1995.*

[14] Nilsson, N. Artificial Intelligence: A New Synthesis. *Morgan Kaufmann, San Francisco, CA, 1998.*

[15] Parodi, A., and Bonelli, P. A New Approach to Fuzzy Classifier Systems. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93): 223-230. San Mateo CA, USA, 1993. Morgan Kaufman.*

[16] 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 Genetic and Evolutionary Computation Conference (GECCO-2000):849-856. Las Vegas NV, USA, 2000.*

[17] 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 (CEC'2001): 679-686. Seoul, Korea, 2001.*

[18] van Veldhuizen, D., and Lamont, G. Multiobjective Optimization with Messy Genetic Algorithms. In *Proceedings of the Fifteenth ACM Symposium on Applied Computing (SAC'2000) (Evolutionary Computation and Optimization Track): 470-476. Como, Italy, 2000.*

[19] 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, June 1998.*

[20] Zadeh, L. Fuzzy Sets. *Journal of Information and Control, Vol 8, 1965.*