Evolving Fuzzy Rules for Goal-Scoring Behaviour in Robot Soccer

Jeff Riley
RMIT University
Australia

1. Introduction

If a soccer player is able to learn behaviours it should exhibit in response to stimuli, it may adapt to unpredictable, dynamic environments. Even though the overall objective a player is expected to achieve is able to be described, it is not always possible to precisely describe the behaviours a player should exhibit in achieving that objective. If a function can be described to evaluate the results of the player's behaviour against the desired outcome, that function can be used by some reinforcement learning algorithm to evolve the behaviours necessary to achieve the desired objective.

Fuzzy Sets (Zadeh 1965; Kandel 1986; Klir and Folger 1988; Kruse, Gebhardt et al. 1994) 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 (FIS) on its own is not usually self-adaptive and not able to modify its underlying rulebase to adapt to changing circumstances.

There has not been a great deal of success in the automatic generation of robot soccer players, and in fact hand-coded players, or players with hand-coded skills, generally outplay automatically generated players. Genetic algorithms (Holland 1975) are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection. By combining the adaptive learning capabilities of the genetic algorithm (GA) with the approximate reasoning capabilities of the fuzzy inference system, a hybrid system is produced, and the expectation is that this hybrid system will be capable of learning the behaviours a player needs to exhibit in order to achieve a defined objective – in this case developing goal-scoring behaviour. While the combination of genetic algorithms and fuzzy inference systems have been studied in other areas, they have not generally been studied in an environment as complex, uncertain and dynamic as the robot soccer environment.

2. Related Work

2.1 RoboCup

The Robot World Cup Initiative, RoboCup, has become an international research and education initiative, providing a standard platform and benchmark problem for research in

the fields of artificial intelligence and robotics. It provides a realistic research environment by using a soccer game as a platform for a wide range of research problems including autonomous agent design, multi-agent collaboration, real-time reasoning, reactive behaviour and intelligent robot control (Kitano, Asada et al. 1997a; Kitano, Asada et al. 1997b; Kitano, Tambe et al. 1997).
RoboCup currently consists of three major domains: RoboCupSoccer, RoboCupRescue and RoboCupJunior. The RoboCupSoccer domain includes a simulation league and is the environment used for the RoboCup part of this work.

2.1.1 The RoboCupSoccer Simulation League

The RoboCupSoccer Simulation League provides a simulated but realistic soccer environment which obviates the need for robot hardware and its associated difficulties, allowing researchers to focus on issues such as autonomous agent design, learning, planning, real-time multi-agent reasoning, teamwork and collaboration. The RoboCupSoccer simulator has been in continual development since its inception in 1995, and allows researchers to study many aspects of machine learning techniques and multi-agent systems in a complex, dynamic domain. The RoboCupSoccer environment is described in detail in (Noda 1995; Noda, Matsubara et al. 1998; Noda and Stone 2001).

2.2 The SimpleSoccer Environment

The RoboCupSoccer simulation league is an important and useful tool for multi-agent and machine learning research which provides a distributed, multi-agent environment in which agents have an incomplete and uncertain world view. The RoboCupSoccer state-space is extremely large, and the agent perception and action cycles in the RoboCupSoccer environment are asynchronous, sometimes resulting in long and unpredictable delays in the completion of actions in response to some stimuli. The large state-space, the inherent delays, and the uncertain and incomplete world view of the agents can increase the learning cycle of some machine learning techniques onerously.
There is a large body of work in the area of the application of machine learning techniques to the challenges of RoboCupSoccer (e.g. (Luke 1998a; Luke 1998b; Luke, Hohn et al. 1998; Ciesielski and Wilson 1999; Stone and Veloso 1999; Uchibe 1999; Ciesielski and Lai 2001; Riedmiller, Merke et al. 2001; Stone and Sutton 2001; Ciesielski, Mawhinney et al. 2002; Lima, Custódio et al. 2005; Riedmiller, Gabel et al. 2005)), but because the RoboCupSoccer environment is so large, complex and unpredictable, the extent to which such techniques can meet these challenges is not certain. The SimpleSoccer environment was designed and developed to address the problem of the complexity of the RoboCupSoccer environment inhibiting further research, and is described in detail in (Riley 2003) and (Riley 2007).

2.3 Machine Learning, Evolutionary Algorithms and Simulated Robot Soccer

Machine learning and evolutionary algorithms in various forms have been applied to the problem of robot soccer. Some representative examples, with emphasis on evolutionary algorithms, are described briefly in the following paragraphs.
Two early attempts to learn competent soccer playing skills from scratch via genetic programming are described in (Luke, Hohn et al. 1998) and (Andre and Teller 1999). Both set out to create complete, cooperative soccer playing teams, but neither achieved that

objective. The objective in (Luke, Hohn et al. 1998) was scaled back to attempt to evolve co-operative behaviour over hand-coded low-level behaviours. The players in (Andre and Teller 1999) developed some successful individual behaviours with the use of a sophisticated composite fitness function, but the objective of collaborative team behaviour was not realised.

In (Luke 1998b) a team of soccer players with a rich set of innate soccer-playing skills was developed, using genetic programming and co-evolution, that worked through sub-optimal behaviour described as "kiddie-soccer" (where all players chase the ball) to reasonable goal-scoring and defensive behaviour.

A layered learning technique was introduced in (Stone 1998) and (Stone and Veloso 2000), the essential principle of which is to provide the algorithm with a bottom-up hierarchical decomposition of a large task into smaller sub-tasks, or layers, and have the algorithm learn each sub-task separately and feed the output of one learned sub-task into the next layer.

The layered learning technique is based upon the following four principles (Stone and Veloso 2000):

- A mapping directly from inputs to outputs is not tractably learnable.
- A bottom-up, hierarchical task decomposition is given.
- Machine learning exploits data to train and/or adapt. Learning occurs separately at each level.
- The output of learning in one layer feeds into the next.

Stone and Veloso used the layered learning technique to produce good results when training robot soccer players for RoboCupSoccer (Stone and Veloso 2000).

Keepaway Soccer (Stone, Sutton et al. 2001) is a sub-domain of robot soccer in which the objective is not to score goals but to gain and maintain possession of the ball. There are two teams in keepaway soccer: the *keepers* and the *takers*. The task of the keepers is to maintain possession of the ball, while the objective of the takers is to take the ball away from the keepers. The keepaway soccer field is generally smaller than the RoboCupSoccer field, and no goal areas are required. The keepaway soccer teams are usually smaller than a full team in RoboCupSoccer, and are often numerically unbalanced (e.g. 3 vs 2 Keepaway Soccer (Kuhlmann and Stone 2004)).

Gustafson (Gustafson 2000) and Gustafson and Hsu (Gustafson and Hsu 2001; Hsu, Harmon et al. 2004) applied genetic programming and the layered learning technique of Stone and Veloso to keepaway soccer. For this method the problem was decomposed into smaller sub-problems, and genetic programming applied to the sub-problems sequentially - the population in the last generation of a sub-problem was used as the initial population of the next sub-problem. The results presented by Gustafson and Hsu indicate that for the problem studied layered learning in genetic programming outperformed the standard genetic programming method.

Asada et al. (Asada, Noda et al. 1996) describe the *Learning from Easy Missions (LEM)* method, in which a reinforcement learning technique (*Q-learning, (Watkins 1989)*) is used to teach a robot soccer player to kick a ball through a goal. The reinforcement learning technique implemented requires the robot soccer player to be capable of discriminating a finite set of distinct world states and also be capable of taking one of a finite set of actions. The robot's world is then modelled as a Markov process, making stochastic transitions based on the current state and action taken. A significant problem with this method is that for a real robot in the real world, or the simulation of a real robot in the real world, the state and action spaces are continuous spaces that are not adequately represented by finite sets. Asada et al. overcome this by constructing a set of sub-states into which the representation of the robot's world is divided, and similarly a set of sub-actions into which the robot's full range of actions is divided. This is roughly analogous to the fuzzy sets for input variables and actions implemented for this work.

The LEM method involves using human input to modify the starting state of the soccer player, beginning with easy states and progressing over time to more difficult states. In this way the robot soccer player learns easier sub-tasks allowing it to use those learned sub-tasks to develop more complex behaviour enabling it to score goals in more difficult situations. Asada et al. concede that the LEM method has limitations, particularly with respect to constructing the state space for the robot soccer player. Asada et al. also point out that the method suffers from a lack of historical information that would allow the soccer player to define context, particularly in the situation where the player is between the ball and the goal: with only current situation context the player does not know how to move to a position to shoot the ball into the goal (or even that it should). Some methods suggested by Asada et al. to overcome this problem are to use task decomposition (i.e. find ball, position ball between player and goal, move forward, etc.), or to place reference objects on the field (corner posts, field lines, etc.) to give the player some context. It is also interesting to note that after noticing that the player performed poorly whenever it lost sight of the ball, Asada et al. introduced several extra states to assist the player in that situation: the *ball-lost-into-right* and *ball-lost-into-left* states, and similarly for losing sight of the goal, *goal-lost-into right* and *goal-lost-into-left* states. These states, particularly the *ball-lost-into-right* and *ball-lost-into-left* states are analogous to the default hunt actions implemented as part of the work described in this chapter, and another indication of the need for human expertise to be injected to adequately solve the problem.

Di Pietro et al. (Di Pietro, While et al. 2002) reported some success using a genetic algorithm to train 3 keepers against 2 takers for keepaway soccer in the RoboCup soccer simulator. Players were endowed with a set of high-level skills, and the focus was on learning strategies for keepers in possession of the ball.

Three different approaches to create RoboCup players using genetic programming are described in (Ciesielski, Mawhinney et al. 2002) – the approaches differing in the level of innate skill the players have. In the initial experiment described, the players were given no innate skills beyond the actions provided by the RoboCupSoccer server. The third experiment was a variation of the first experiment. Ciesielski et al. reported that the players from the first and third experiments – players with no innate skills - performed poorly. In the second experiment described, players were given some innate higher-level hand-coded skills such as the ability to kick the ball toward the goal, or to pass to the closest teammate. The players from the second experiment – players with some innate hand-coded skills – performed a little more adequately than the other experiments described. Ciesielski et al. concluded that the robot soccer problem is a very difficult problem for evolutionary algorithms and that a significant amount of work is still needed for the development of higher-level functions and appropriate fitness measures.

Using keepaway soccer as a machine learning testbed, Whiteson and Stone (Whiteson and Stone 2003) used neuro-evolution to train keepers in the Teambots domain (Balch 2005). In that work the players were able to learn several conceptually different tasks from basic skills to higher-level reasoning using "concurrent layered learning" – a method in which predefined tasks are learned incrementally with the use of a composite fitness function. The player uses a hand-coded decision tree to make decisions, with the leaves of the tree being the learned skills.

Whiteson et al. (Whiteson, Kohl et al. 2003; Whiteson, Kohl et al. 2005) study three different methods for learning the sub-tasks of a decomposed task in order to examine the impact of injecting human expert knowledge into the algorithm with respect to the trade-off between:

- making an otherwise unlearnable task learnable
- the expert knowledge constraining the hypothesis space
- the effort required to inject the human knowledge.

Coevolution, layered learning, and concurrent layered learning are applied to two versions of keepaway soccer that differ in the difficulty of learning. Whiteson et al. conclude that given a suitable task decomposition an evolutionary-based algorithm (in this case neuroevolution) can master difficult tasks. They also conclude, somewhat unsurprisingly, that the appropriate level of human expert knowledge injected and therefore the level of constraint depends critically on the difficulty of the problem.

Castillo et al. (Castillo, Lurgi et al. 2003) modified an existing RoboCupSoccer team – the 11Monkeys team (Kinoshita and Yamamoto 2000) – replacing its offensive hand-coded, state dependent rules with an XCS genetic classifier system. Each rule was translated into a genetic classifier, and then each classifier evolved in real time. Castillo et al. reported that their XCS classifier system outperformed the original 11Monkeys team, though did not perform quite so well against other, more recently developed, teams.

In (Nakashima, Takatani et al. 2004) Nakashima et al. describe a method for learning certain strategies in the RoboCupSoccer environment, and report some limited success. The method uses an evolutionary algorithm similar to evolution strategies, and implements mutation as the only evolutionary operator. The player uses the learned strategies to decide which of several hand-coded actions will be taken. The strategies learned are applicable only when the player is in possession of the ball.

Bajurnow and Ciesielski used the SimpleSoccer environment to examine genetic programming and layered learning for the robot soccer problem (Bajurnow and Ciesielski 2004). Bajurnow and Ciesielski concluded that layered learning is able to evolve goal-scoring behaviour comparable to standard genetic programs more reliably and in a shorter time, but the quality of solutions found by layered learning did not exceed those found using standard genetic programming. Furthermore, Bajurnow and Ciesielski claim that layered learning in this fashion requires a *"large amount of domain specific knowledge and programmer effort to engineer an appropriate layer and the effort required is not justified for a problem of this scale."* (Bajurnow and Ciesielski 2004), p.7.

Other examples of research in this or related areas can be found in, for example, (Luke and Spector 1996) where breeding and co-ordination strategies were studied for evolving teams in a simple predator/prey environment; (Stone and Sutton 2001; Kuhlmann and Stone 2004; Stone, Sutton et al. 2005) where reinforcement learning was used to train players in the keepaway soccer environment; (Lazarus and Hu 2003) in which genetic programming was used in a specific training environment to evolve goal-keeping behaviour for RoboCupSoccer; (Aronsson 2003) where genetic programming was used to develop a team of players for RoboCupSoccer; (Hsu, Harmon et al. 2004) in which the incremental reuse of intermediate solutions for genetic programming in the keepaway soccer environment is studied.

3. The Player

3.1 Player Architecture

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

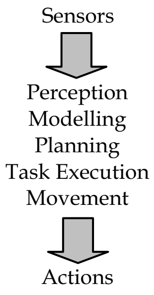

Fig. 1. Traditional Control Architecture

The control architecture implemented for this work is similar to the subsumption architecture described in (Brooks 1985). 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 Fig. 2 the *Movement* layer does not explicitly need to avoid obstacles: the *Avoid Objects* layer will take care of that. This approach creates players with reactive architectures and with no central locus of control (Brooks 1991).

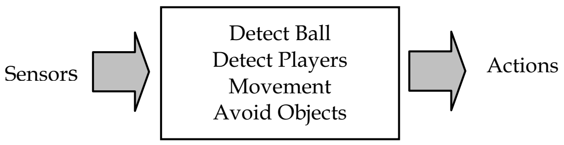

Fig. 2. Soccer Player Layered Architecture

For the work presented here, the behaviour producing layers are implemented as fuzzy if-then rules and governed by a fuzzy inference system comprised of the fuzzy rulebase, definitions of the membership functions of the fuzzy sets operated on by the rules in the rulebase, and a reasoning mechanism to perform the inference procedure. The fuzzy inference system is embedded in the player architecture, where it receives input from the soccer server and generates output necessary for the player to act Fig. **3**.

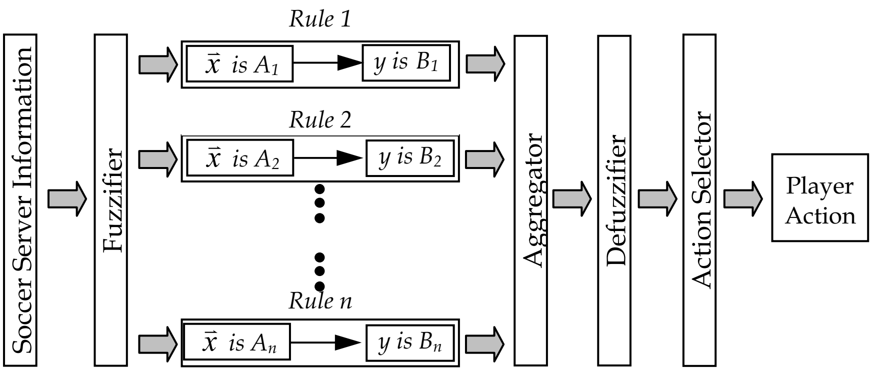

Fig. 3. Player Architecture Detail

3.1.1 Soccer Server Information

The application by the inferencing mechanism of the fuzzy rulebase to external stimuli provided by the soccer server results in one or more fuzzy rules being executed and some resultant action being taken by the client. The external stimuli used as input to the fuzzy inference system are a subset of the visual information supplied by the soccer server: only sufficient information to situate the player and locate the ball is used. The environments studied in this work differ slightly with regard to the information supplied to the player:

- In the RoboCupSoccer environment the soccer server delivers regular *sense*, *visual* and *aural* messages to the players. The player implemented in this work uses only the *object name*, *distance* and *direction* information from the visual messages in order to determine its own position on the field and that of the ball. The player ignores any aural messages, and uses the information in the sense messages only to synchronise communication with the RoboCupSoccer server. Since the information supplied by the RoboCupSoccer server is not guaranteed to be complete or certain, the player uses its relative distance and direction from all fixed objects in its field of vision to estimate its position on the field. The player is then able to use the estimate of its position to estimate the direction and distance to the known, fixed location of its goal. The player is only aware of the location of the ball if it is in its field of vision, and only to the extent that the RoboCupSoccer server reports the relative direction and distance to the ball.

- In the SimpleSoccer environment the soccer server delivers only regular visual messages to the players: there are no aural or sense equivalents. Information supplied by the SimpleSoccer server is complete, in so far as the objects actually with the player's field of vision are concerned, and certain. Players in the SimpleSoccer environment are aware at all times of their exact location on the field, but are only aware of the location of the ball and the goal if they are in the player's field of vision. The SimpleSoccer server provides the *object name*, *distance* and *direction* information for objects in a player's field of vision. The only state information kept by a player in the SimpleSoccer environment is the co-ordinates of its location and the direction in which it is facing.

3.1.2 Fuzzification

Input variables for the fuzzy rules are fuzzy interpretations of the visual stimuli supplied to the player by the soccer server: the information supplied by the soccer server is fuzzified to represent the degree of membership of one of three fuzzy sets: *direction*, *distance* and *power*; and then given as input to the fuzzy inference system. Output variables are the fuzzy actions to be taken by the player. The universe of discourse of both input and output variables are covered by fuzzy sets (direction, distance and power), 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.

Both the RoboCupSoccer and the SimpleSoccer servers provide crisp values for the information they deliver to the players. These crisp values must be transformed into linguistic terms in order to be used as input to the fuzzy inference system. This is the fuzzification step: the process of transforming crisp values into degrees of membership for linguistic terms of fuzzy sets. The membership functions shown in Fig. 4 on are used to associate crisp values with a degree of membership for linguistic terms. 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 RoboCupSoccer parameters and variables, and fine-tuned after minimal experimentation in the RoboCupSoccer environment.

3.1.3 Implication and Aggregation

The core section of the fuzzy inference system is the part which combines the facts obtained from the fuzzification with the rule base and conducts the fuzzy reasoning process: this is where the fuzzy inferencing is performed. The FIS model used in this work is a Mamdani FIS (Mamdani and Assilian 1975). The method implemented to apply the result of the antecedent evaluation to the membership function of the consequent is the correlation minimum, or *clipping* method, where the consequent membership function is truncated at the level of the antecedent truth. The aggregation method used is the min/max aggregation method as described in (Mamdani and Assilian 1975). These methods were chosen because they are computationally less complex than other methods and generate an aggregated output surface that is relatively easy to defuzzify.

3.1.4 Defuzzification

The defuzzification method used is the *mean of maximum* method, also employed by Mamdani's fuzzy logic controllers. This technique takes the output distribution and finds its mean of maxima in order to compute a single crisp number. This is calculated as follows:

$$z = \sum_{i=1}^{n} \frac{z_i}{n}$$

where z is the mean of maximum, z_i is the point at which the membership function is maximum, and n is the number of times the output distribution reaches the maximum level.

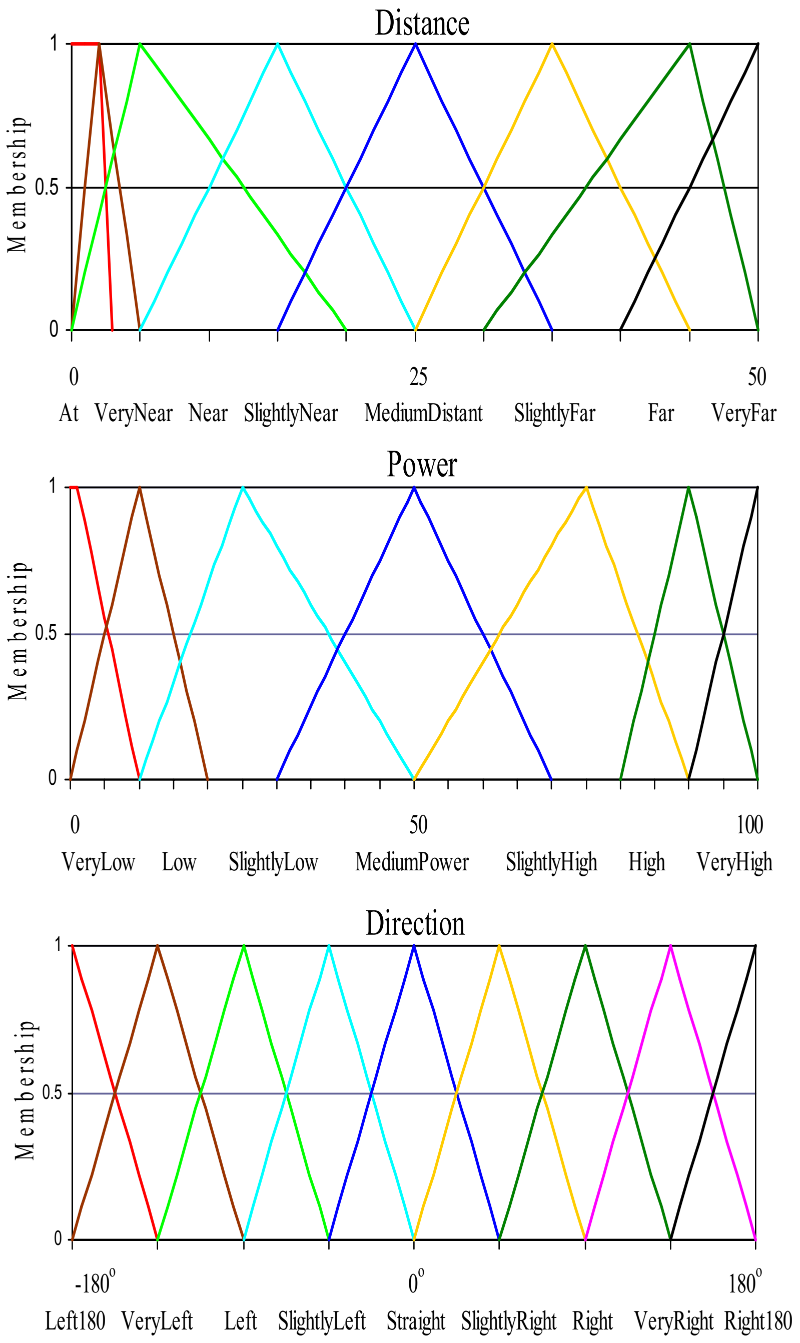

Fig. 4. Distance, Power and Direction Fuzzy Sets

An example outcome of this computation is shown in Fig. **5**. This method of defuzzification was chosen because it is computationally less complex than other methods yet produces satisfactory results.

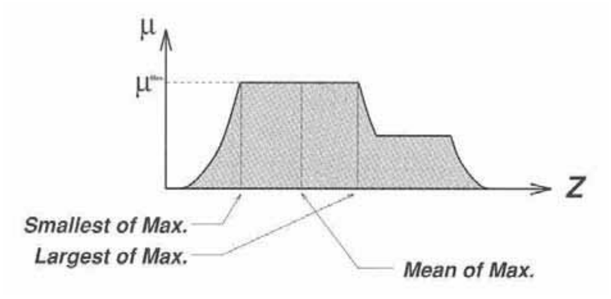

Fig. 5. Mean of Maximum defuzzification method
(Adapted from (Jang, Sun et al. 1997))

3.1.5 Player Actions

A player will perform an action based on its skillset and in response to external stimuli; the specific response being determined in part by the fuzzy inference system. The action commands provided to the players by the RoboCupSoccer and SimpleSoccer simulation environments are described in (Noda 1995) and (Riley 2007) respectively. For the experiments conducted for this chapter the SimpleSoccer simulator was, where appropriate, configured for RoboCupSoccer emulation mode.

3.1.6 Action Selection

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 player, and the degree to which the action is to be taken. For example:

(KickTowardGoal, power)

(RunTowardBall, power)

(Turn, direction)

where *power* and *direction* are crisp values representing the defuzzified fuzzy set membership of the action to be taken.
Only one action is performed by the player in response to stimuli provided by the soccer server. Since several rules with different actions may fire, the action with the greatest level of support, as indicated by the value for truth of the antecedent, is selected.

3.2 Player Learning

This work investigates the use of an evolutionary technique in the form of a messy-coded genetic algorithm to efficiently construct the rulebase for a fuzzy inference system to solve a particular optimisation problem: goal-scoring behaviour for a robot soccer player. The flexibility provided by the messy-coded 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 player behaviours are defined by sets of fuzzy if-then rules evolved by a messy-coded 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 player based upon the number of goals scored, or attempts made to move toward goal-scoring, during a game.
The genetic algorithm implemented in this work is a messy-coded genetic algorithm implemented using the Pittsburgh approach: each individual in the population is a complete ruleset.

4. Representation of the Chromosome

For these experiments, a chromosome is represented as a variable length vector of genes, and rule clauses are coded on the chromosome as genes. The encoding scheme implemented exploits the capability of messy-coded genetic algorithms to encode information of variable structure and length. It should be noted that while the encoding scheme implemented is a messy encoding, the algorithm implemented is the classic genetic algorithm: there are no primordial or juxtapositional phases implemented.
The basic element of the coding of the fuzzy rules is a tuple representing, in the case of a rule premise, a fuzzy clause and connector; and in the case of a rule consequent just the fuzzy consequent. 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.
For single-player trials, the only objects of interest to the player are the ball and the player's goal, and what is of interest is where those objects are in relation to the player. A premise is of the form:

(Object, Qualifier, {Distance | Direction}, Connector)

and is constructed from the following range of values:

Object:	{ BALL, GOAL }
Qualifier:	{ IS, IS NOT }
Distance:	{ AT, VERYNEAR, NEAR, SLIGHTLYNEAR, MEDIUMDISTANT, SLIGHTLYFAR, FAR, VERYFAR }
Direction:	{ LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }
Connector:	{ AND, OR }

Each rule consequent specifies and qualifies the action to be taken by the player as a consequent of that rule firing thus contributing to the set of *(action, value)* pairs output by the fuzzy inference system. A consequent is of the form:

(Action, {Direction | Null}, {Power | Null})

and is constructed from the following range of values (depending upon the skillset with which the player is endowed):

Action:	{ TURN, DASH, KICK, RUNTOWARDGOAL, RUNTOWARDBALL, GOTOBALL, KICKTOWARDGOAL, DRIBBLETOWARDGOAL, DRIBBLE, DONOTHING }
Direction:	{ LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }
Power:	{ VERYLOW, LOW, SLIGHTLYLOW, MEDIUMPOWER, SLIGHTLYHIGH, HIGH, VERYHIGH }

Fuzzy rules developed by the genetic algorithm are of the form:

if Ball is Near and Goal is Near then KickTowardGoal Low
if Ball is Far or Ball is SlightlyLeft then RunTowardBall High

In the example chromosome fragment shown in Fig. 6 the shaded clause has been specially coded to signify that it is a consequent gene, and the fragment decodes to the following rule:

if Ball is Left and Ball is At or Goal is not Far then Dribble Low

In this case the clause connector *OR* in the clause immediately prior to the consequent clause is not required, so ignored.

(Ball, is Left, And)	(Ball, is At, Or)	(Goal, is not Far, Or)	(Dribble, Null, Low)

Fig. 6. Messy-coded Genetic Algorithm Example Chromosome Fragment

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. Since the cut, splice and mutation operators implemented guarantee no out-of-bounds data in the resultant chromosomes, a rule is only considered invalid if it contains no premises. A complete ruleset is considered invalid only if it contains no valid rules. Some advantages of using a messy encoding in this case are:

- a ruleset is not limited to a fixed size
- a ruleset can be overspecified (i.e. clauses may be duplicated)
- a ruleset can be underspecified (i.e. not all genes are required to be represented)
- clauses may be arranged in any way

An example complete chromosome and corresponding rules are shown in Fig. 7 (with appropriate abbreviations).

(B,N,O)	(B,nF,A)	(G,N,A)	(RB,n,L)	(B,A,A)	(G,vN,O)	(KG,n,M)	(B,L,A)	(T,L,n)

Premise	Consequent

Rule 1: if Ball is Near or Ball is not Far and Goal is Near then RunTowardBall Low
Rule 2: if Ball is At and Goal is VeryNear then KickTowardGoal MediumPower
Rule 3: if Ball is Left then Turn Left

Fig. 7. Chromosome and corresponding rules

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, or need be, implemented for any attributes in this method. Since messy-coded 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. A feature of the messy-coded genetic algorithm is that the format implies *don't care* values for all attributes since any premise may be omitted from any or all rules, so generalisation is an implicit feature of this method.

For the messy-coded genetic algorithm implemented in this work the selection operator is implemented in the same manner as for classic genetic algorithms. Roulette wheel selection was used in the RoboCupSoccer trials and the initial SimpleSoccer trials. Tests were conducted to compare several selection methods, and elitist selection was used in the remainder of the SimpleSoccer trials. Crossover is implemented by the *cut* and *splice* operators, and mutation is implemented as a single-allele mutation scheme.

5. Experimental Evaluation

A series of experiments was performed in both the RoboCupSoccer and the SimpleSoccer simulation environments in order to test the viability of the fuzzy logic-based controller for the control of the player and the genetic algorithm to evolve the fuzzy ruleset. The following sections describe the trials performed, the parameter settings for each of the trials and other fundamental properties necessary for conducting the experiments.

An initial set of 20 trials was performed in the RoboCupSoccer environment in order to examine whether a genetic algorithm can be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player which produces consistent goal-scoring behaviour. This addresses part of the research question examined by this chapter.

Because the RoboCupSoccer environment is a very complex real-time simulation environment, it was found to be prohibitively expensive with regard to the time taken for the fitness evaluations for the evolutionary search. To overcome this problem the SimpleSoccer environment was developed so as to reduce the time taken for the trials. Following the RoboCupSoccer trials, a set of similar trials was performed in the SimpleSoccer environment to verify that the method performs similarly in the new environment.

Trials were conducted in the SimpleSoccer environment where the parameters controlling the operation of the genetic algorithm were varied in order to determine the parameters that should be used for the messy-coded genetic algorithm in order to produce acceptable results.

5.1 Trials

For the results reported, a single trial consisted of a simulated game of soccer played with the only player on the field being the player under evaluation. The player was placed at a randomly selected position on its half of the field and oriented so that it was facing the end of the field to which it was kicking. For the RoboCupSoccer trials the ball was placed at the centre of the field, and for the SimpleSoccer trials the ball was placed at a randomly selected position along the centre line of the field.

5.2 Fitness Evaluation

The objective of the fitness function for the genetic algorithm is to reward the fitter individuals with a higher probability of producing offspring, with the expectation that combining the fittest individuals of one generation will produce even fitter individuals in later generations. All fitness functions implemented in this work indicate better fitness as a lower number, so representing the optimisation of fitness as a minimisation problem.

5.2.1 RoboCupSoccer Fitness Function

Since the objective of this work was to produce goal-scoring behaviour, the first fitness function implemented rewarded individuals for goal-scoring behaviour only, and was implemented as:

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

where *goals* is the number of goals scored by the player during the trial.

Equation 1 RoboCupSoccer Simple Goals-only Fitness Function

In early trials in the RoboCupSoccer environment the initial population of randomly generated individuals demonstrated no goal-scoring behaviour, so the fitness of each individual was the same across the entire population. This lack of variation in the fitness of the population resulted in the selection of individuals for reproduction being reduced to random choice. To overcome this problem a composite fitness function was implemented which effectively decomposes the difficult problem of evolving goal-scoring behaviour essentially from scratch - actually from the base level of skill and knowledge implicit in the primitives supplied by the environment – into two less difficult problems:

- evolve ball-kicking behaviour, and
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

In the RoboCupSoccer trials, individuals were rewarded for, in order of importance:

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

This combination was chosen to reward players primarily for goals scored, while players that do not score goals are rewarded for the number of times the ball is kicked on the assumption that a player which actually kicks the ball is more likely to produce offspring capable of scoring goals. The actual fitness function implemented in the RoboCupSoccer trials was:

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

where

goals	=	the number of goals scored by the player during the trial
kicks	=	the number of times the player kicked the ball during the trial
ticks	=	the number of RoboCupSoccer server time steps of the trial

Equation 2 RoboCupSoccer Composite Fitness Function

5.2.2 SimpleSoccer Fitness Function

A similar composite fitness function was used in the trials in the SimpleSoccer environment, where individuals were rewarded for, in order of importance:

- the number of goals scored in a game
- minimising the distance of the ball from the goal

This combination was chosen to reward players primarily for goals scored, while players that do not score goals are rewarded on the basis of how close they are able to move the ball to the goal on the assumption that a player which kicks the ball close to the goal is more likely to produce offspring capable of scoring goals. This decomposes the original problem of evolving goal-scoring behaviour into the two less difficult problems:

- evolve ball-kicking behaviour that minimises the distance between the ball and goal
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

The actual fitness function implemented in the SimpleSoccer trials was:

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

where

goals	=	the number of goals scored by the player during the trial
kicks	=	the number of times the player kicked the ball during the trial
dist	=	the minimum distance of the ball to the goal during the trial
fieldLen	=	the length of the field

Equation 3 SimpleSoccer Composite Fitness Function

The difference between the composite fitness function implemented in the RoboCupSoccer environment and the composite fitness function implemented in the SimpleSoccer environment is just an evolution of thinking – rewarding a player for kicking the ball often when no goal is kicked could reward a player that kicks the ball very often in the wrong direction more than a player that kicks the ball fewer times but in the right direction. The SimpleSoccer implementation of the composite fitness function rewards players more for kicking the ball closer to the goal irrespective of the number of times the ball was kicked. This is considered a better approach to encourage behaviour that leads to scoring goals.

5.2.3 Fitness Values

To facilitate the interpretation of fitness graphs and fitness values presented throughout this chapter, following is an explanation of the fitness values generated by the fitness functions used in this work. All fitness functions implemented in this work generate a real number R, where $0.0 < R \leq 1.0$, $R = 1.0$ indicates no ball movement and $R \approx 0.0$ indicates very good performance – smaller fitness values indicate better performance.
For ball movement in the RoboCupSoccer environment where a composite fitness function is implemented, fitness values are calculated in the range $x \leq R \leq y$, where $x = 0.5$ and $y = 1.0$. For ball movement in the SimpleSoccer environment where a composite fitness function is implemented, fitness values are calculated in the range $x \leq R \leq y$, where $x \approx 0.5$ and $y \approx 0.77$. Where a simple goals-only fitness function is implemented, ball movement alone is not rewarded: if no goals are scored the fitness function assigns $R = 1.0$. In both environments all fitness functions assign discrete values for goal-scoring, depending upon the number of goals scored. Table 1 summarises the fitness values returned by the various fitness functions.

		Simple Goals-only Fitness Function	**RoboCupSoccer Composite Fitness Function**	**SimpleSoccer Composite Fitness Function**
No Goals Scored	**No Ball Movement**	1.0000	1.0000	1.0000
	Ball Movement	*n/a*	[0.5, 1.0]	[~0.5, ~0.77]
Goals Scored	**1**	0.5000	0.5000	0.5000
	2	0.2500	0.2500	0.2500
	3	0.1667	0.1667	0.1667
	4	0.1250	0.1250	0.1250
	5	0.1000	0.1000	0.1000
	6	0.0833	0.0833	0.0833
	7	0.0714	0.0714	0.0714
	8	0.0625	0.0625	0.0625
	9	0.0556	0.0556	0.0556
	10	0.0500	0.0500	0.0500
	...	…	…	…
	n	$0.5/n$	$0.5/n$	$0.5/n$

Table 1. Fitness Assignment Summary

Parameter	Value
Maximum Chromosome Length	64 genes
Population Size	200
Maximum Generations	25
Selection Method	Roulette Wheel
Crossover Method	Single Point
Crossover Probability	0.8
Mutation Rate	10%
Mutation Probability	0.35

Table 2. Genetic Algorithm Control Parameters

In initial trials in the RoboCup environment players were evaluated over five separate games and then assigned the average fitness value of those games. Since each game in the Robocup environment is played in real time, this was a very time consuming method. The results of experiments where the player's fitness was calculated as the average of five games were compared with results where the player's fitness was assigned after a single game and were found to be almost indistinguishable. Due to the considerable time savings gained by assigning fitness after a single game, this is the method used throughout this work. Since players evolved using the average fitness method are exposed to different starting conditions they may be more robust than those evolved using single-game fitness, but the effect is extremely small considering the number of different starting positions players could be evaluated against and the fact that the starting positions of the player and ball really only affect the first kick of the ball.

5.3 Control Parameters

The genetic algorithm parameters common to all 20 initial trials in both the RoboCupSoccer and SimpleSoccer environments are shown in Table 2.
A game was terminated when:

- the target fitness of 0.05 was reached
- the ball was kicked out of play (RoboCupSoccer only)
- the elapsed time expired:
  - 120 seconds real time for RoboCupSoccer
  - 1000 ticks of simulator time for SimpleSoccer
- A period of no player movement or action expired
  - 10 seconds real time for RoboCupSoccer
  - 100 ticks of simulator time for SimpleSoccer

The target fitness of 0.05 reflects a score of 10 goals in the allotted playing time. This figure was chosen to allow the player a realistic amount of time to develop useful strategies yet terminate the search upon finding an acceptably 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 the specified target fitness. Both methods were active, with the first to be encountered terminating the search. Early stopping did not occur in any of the experiments reported in this chapter.

6. Results

The following sections describe the results for the experiments performed for both the RoboCupSoccer and the SimpleSoccer environments. Discussion and analysis of the results is also presented.

6.1 RoboCupSoccer Initial Trial Results

Fig. **8** shows the average fitness of the population after each generation for each of the 20 trials for the RoboCupSoccer environment, showing that the performance of the population does improve steadily and, in some of the trials, plateaus towards a fitness of 0.5, or goal-scoring behaviour. Fig. **9** shows the best individual fitness from the population after each generation for each of 20 trials for the RoboCupSoccer environment, showing that good individuals are found after very few generations, in contrast to the gradual improvement in average fitness shown in Fig. **8**.

Fig. **10** 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, for the initial 20 RoboCupSoccer trials, the average percentage of the population which scored 0, 1, 2, 3, 4 or more than 4 goals for each generation. The maximum number of goals scored by any individual was 3.

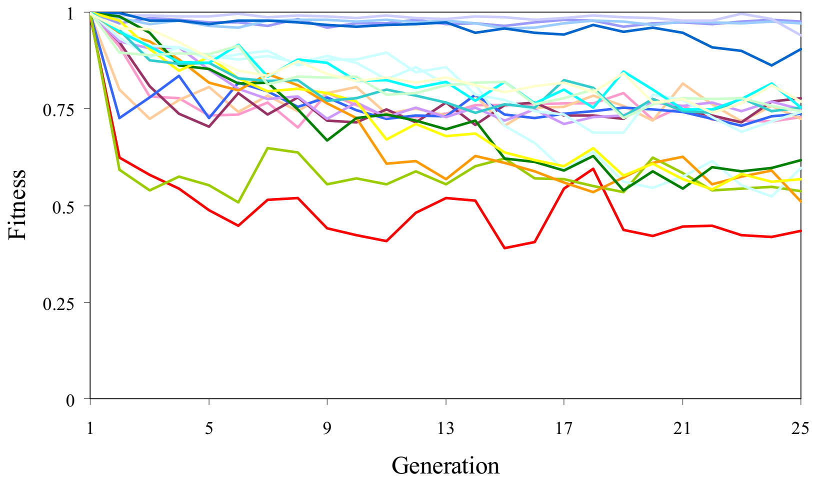

Fig. 8. RoboCupSoccer: Average Fitness - Initial 20 Trials

6.2 SimpleSoccer Initial Trial Results

Fig. **11** shows the average fitness of the population after each generation for each of the 20 trials for the SimpleSoccer environment, and as for the RoboCupSoccer trials, this graph shows that the performance of the population does improve steadily and plateaus, but unlike the RoboCupSoccer trials the average performance of the population does not approach a fitness of 0.5, or goal-scoring behaviour. Fig. **11** also shows that the average fitness curves for the SimpleSoccer trials are more tightly clustered than those of the

RoboCupSoccer trials, and plateau towards a fitness of around 0.75 which, in the SimpleSoccer environment indicates ball-kicking behaviour rather than goal-scoring behaviour.

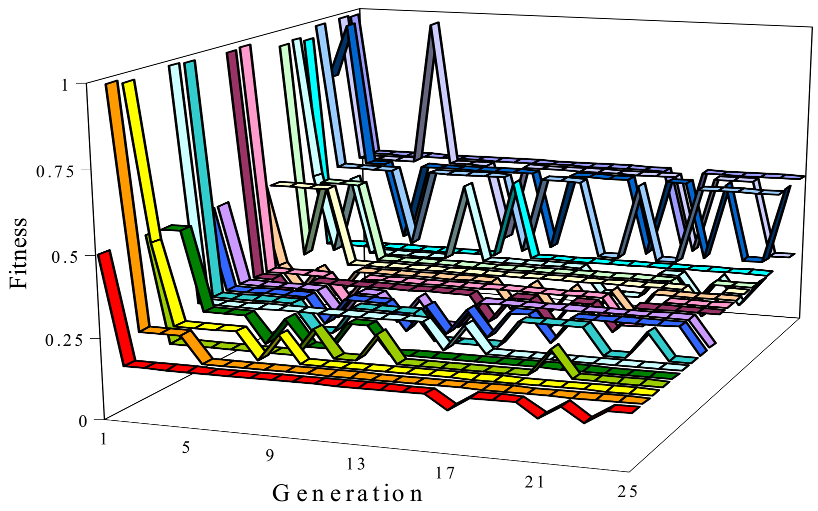

Fig. 9. RoboCupSoccer: Best Fitness - Initial 20 Trials

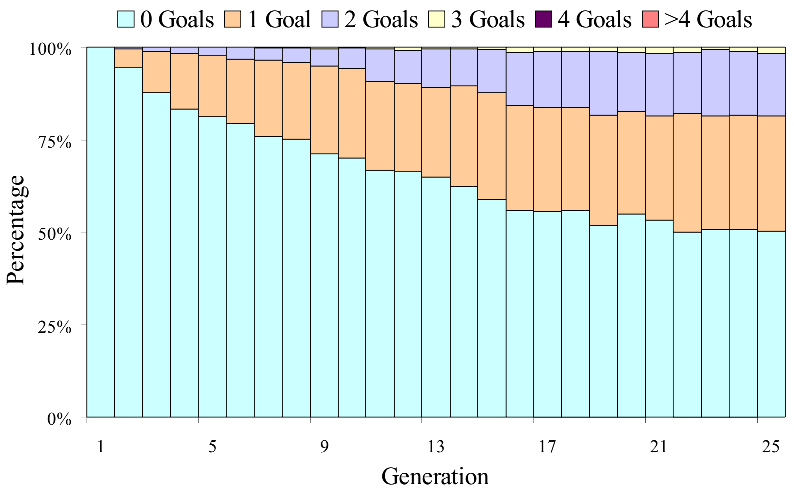

Fig. 10. RoboCupSoccer: Frequency of Individuals Scoring Goals

Fig. **12** shows the best individual fitness from the population after each generation for each of 20 trials for the SimpleSoccer environment and, as for the RoboCupSoccer trials, this graph shows that good individuals are found after very few generations. It is evident from a comparison of Fig. **9** and Fig. **12** that while good individuals are found quickly in both environments, the algorithm seems to be more stable in the RoboCupSoccer environment. The data shows that once a good individual is found in the RoboCupSoccer environment, good individuals are then more consistently found in future generations than in the SimpleSoccer environment.

Fig. **13** shows, for the initial 20 SimpleSoccer trials, the average percentage of the population which scored 0, 1, 2, 3, 4 or more than 4 goals for each generation. The maximum number of goals scored by an individual was 10. The contrast with the equivalent graph for the RoboCupSoccer environment (Fig. **10**) is striking since, although some individuals in the SimpleSoccer environment scored more goals than any individual in the RoboCupSoccer environment, the average goal-scoring behaviour of the population was less developed in the SimpleSoccer environment. This inconsistency is likely to be an indication that the combination of parameters used for the SimpleSoccer environment causes the genetic algorithm to converge more quickly than in the RoboCupSoccer environment, and a possible explanation for the lower average performance of the population when compared to that of the RoboCupSoccer environment as seen in Fig. **11**. Since these SimpleSoccer experiments were performed primarily as a comparison with the RoboCupSoccer experiments the genetic algorithm parameters were kept the same, but the soccer simulator implementations differ considerably and no tuning of simulator parameters to ensure similar performance was performed.

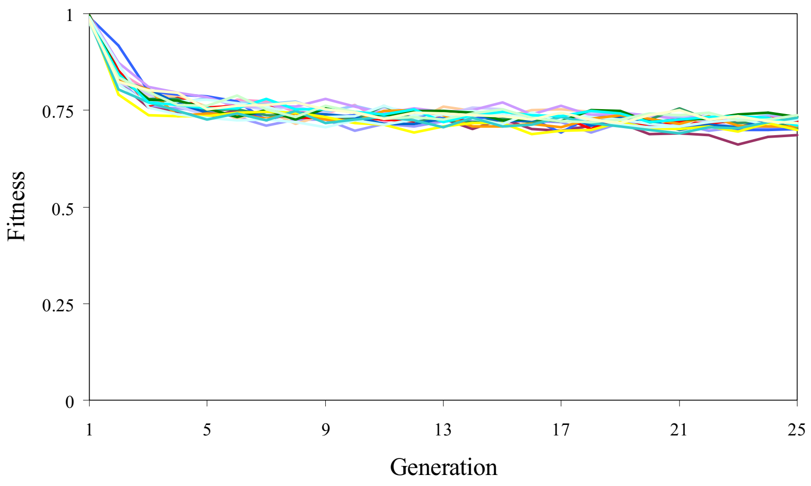

Fig. 11. SimpleSoccer: Average Fitness – Initial 20 Trials

6.3 SimpleSoccer as a Model for RoboCupSoccer

While the difference in the results of the experiments in the RoboCupSoccer and SimpleSoccer environments indicate that SimpleSoccer is not an exact model of RoboCupSoccer (as indeed it was not intended to be), there is a broad similarity in the results which is sufficient to indicate that the SimpleSoccer environment is a good simplified model of the RoboCupSoccer environment. Because SimpleSoccer is considered a reasonable model for RoboCupSoccer, and to take advantage of the significantly reduced training times provided by the SimpleSoccer environment when compared to RoboCupSoccer, all results reported in the remainder of this chapter are for experiments conducted exclusively in the SimpleSoccer environment.

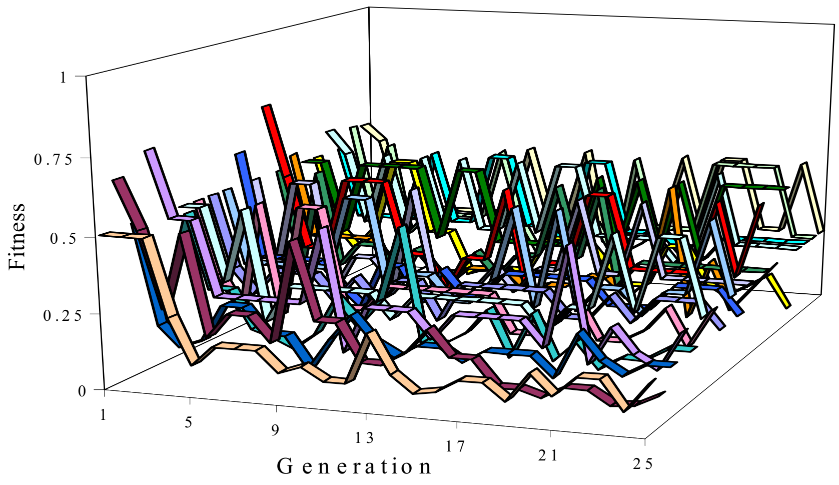

Fig. 12. SimpleSoccer: Best Fitness - Initial 20 Trials

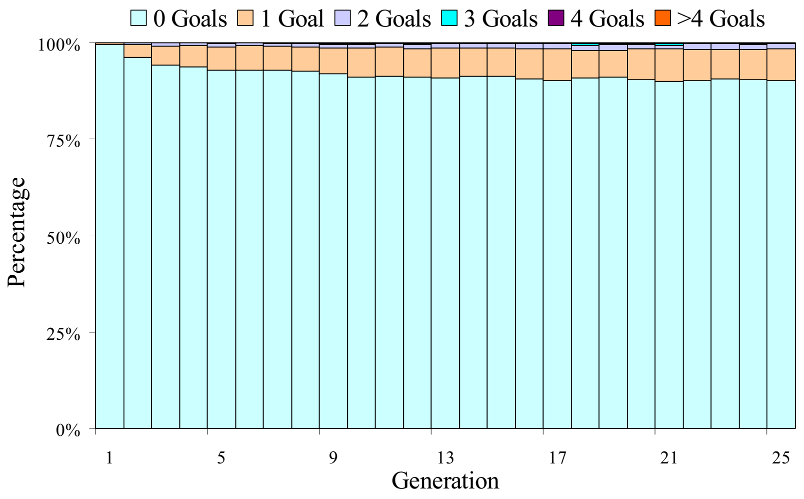

Fig. 13. SimpleSoccer: Frequency of Individuals Scoring Goals

6.4 GA Parameter Determination

Several experiments were conducted in order to determine a set of genetic algorithm parameters conducive to producing acceptable results. The following GA parameters were varied in these trials:

- Maximum Chromosome Length
- Population Size
- Selection Method
- Crossover Method
- Mutation Rate
- Maximum Generations

The values for the parameters shown in Table 2 on are used as control values, and in each of the trials presented in the following sections the value for a single GA parameter is varied. In the experiments conducted a series of 10 trials was performed for each different value of the parameter being varied, and in each case, with the exception of the experiment varying the maximum number of generations, the results presented are the averages of the 10 trials – each line on the graphs shown represents the average of the 10 trials. For the experiment varying the maximum number of generations, only 10 trials were conducted, and the results for each trial is reported individually – each line on the graph represents a single trial.

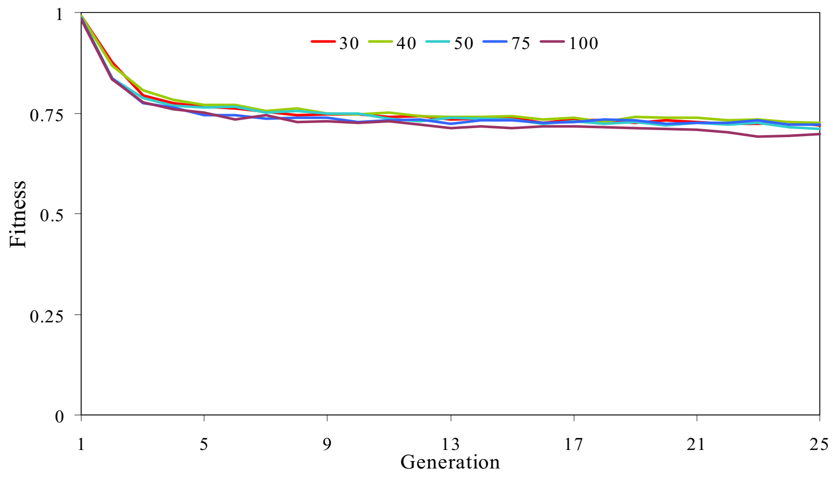

Fig. 14. Average Fitness: Maximum Chromosome Length Variation

6.4.1 Maximum Chromosome Length

While the actual length of the chromosome, measured as the number of genes on the chromosome, may vary depending upon the location of the cut point during the cut-and-splice operation of crossover, the maximum length of the chromosome is fixed throughout the evolutionary process. In order to determine if the maximum length of the chromosome is a significant factor in determining the quality of the evolutionary search, and if so what value is a good value, a series of trials was performed with different maximums for the chromosome length. Fig. **14** shows the average fitness of the population throughout the evolutionary process. It is evident from Fig. **14** that while a maximum chromosome length of 100 offers a very slight advantage, it is not significant. This is further substantiated by Fig. **15** which shows the best fitness in the population throughout the evolutionary process. The results shown indicate that while the method is not sensitive in any significant way to variations in the maximum chromosome length, a maximum chromosome length of somewhere between 50 and 100 genes, and most probably between 50 and 75 genes, produces less variation in the best fitness over the duration of the process.

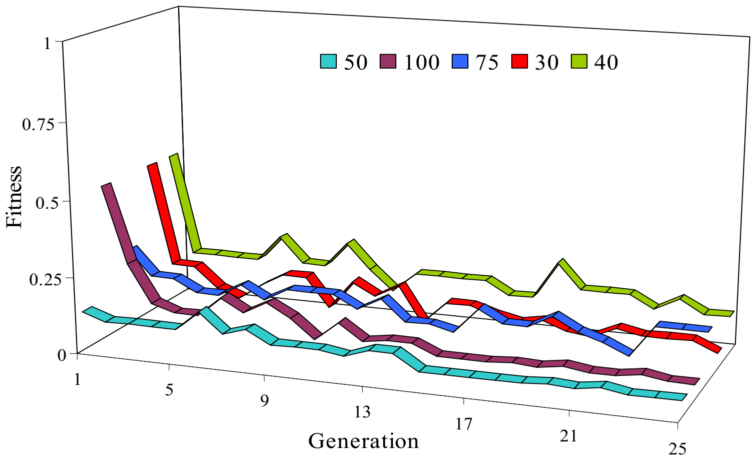

Fig. 15. Best Fitness: Maximum Chromosome Length Variation

Since the actual chromosome length may vary up to the maximum, the average chromosome length for each of the variations in maximum length was measured throughout the evolutionary process, as was the average number of valid rules per chromosome. These data are shown in Fig. **16** and Fig. **17** respectively. The chromosome lengths in the initial populations are initialised randomly between the minimum length of two genes (the smallest number of genes for a valid ruleset) and the maximum chromosome length, so the average chromosome lengths for the initial population shown in Fig. **16** are as expected. All trials show the average chromosome length rising in the initial few generations, then settling to around two-thirds of the maximum length. Given this, and since single-point crossover was used in these trials, with the cut point chosen randomly and chromosomes truncated at the maximum length after the cut-and-splice operation, the results indicate that chromosome length is unaffected by selection pressure. However, Fig. **17** shows the average number of rules per chromosome rising for all of the trials. This would indicate that there is some selection pressure for more rules per chromosome or shorter rules, but since the chromosome length is bounded, so is the number of rules per chromosome. Though outside the scope of this chapter, some further trials to investigate whether the pressure is for more rules or shorter rules, and the optimum number and/or length of rules per chromosome, would be useful work to undertake in the future.

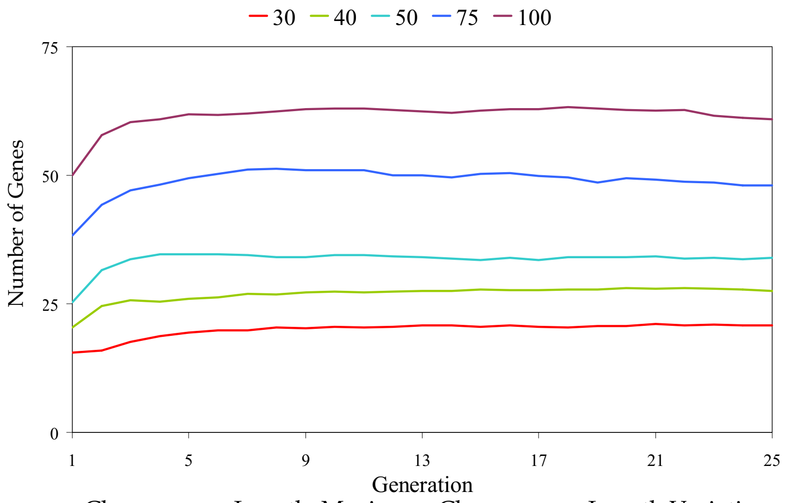

Fig. 16. Average Chromosome Length: Maximum Chromosome Length Variation

6.4.2 Population Size

Since the size of the population differs in this experiment, the number of generations was also varied according to the population size for each set of 10 trials to ensure the comparison between population sizes was for the same number of evaluations. Fig. **18** shows the average fitness of the population over 10,000 evaluations, and Fig. **19** shows the best fitness values for the same trials. It can be seen from the graphs that varying the population size has little effect on the population average fitness with only marginally better results for smaller population sizes, and a similarly small effect on individual best fitness, with larger populations producing slightly more stable results.

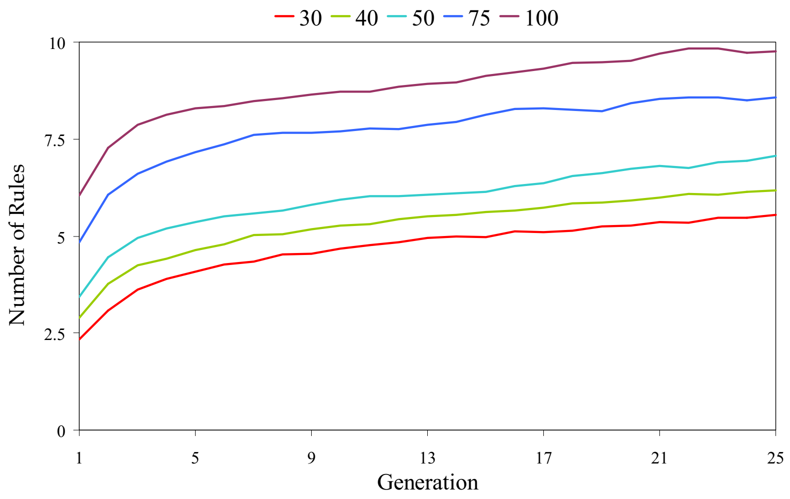

Fig. 17. Average Valid Rules per Chromosome: Maximum Chromosome Length Variation

Overall, the difference in performance between the population sizes tested is not significant, suggesting that it is the number of solutions evaluated, or the extent of the search, that is a significant factor affecting performance. This is consistent with the findings of other work in the area (Luke 2001).

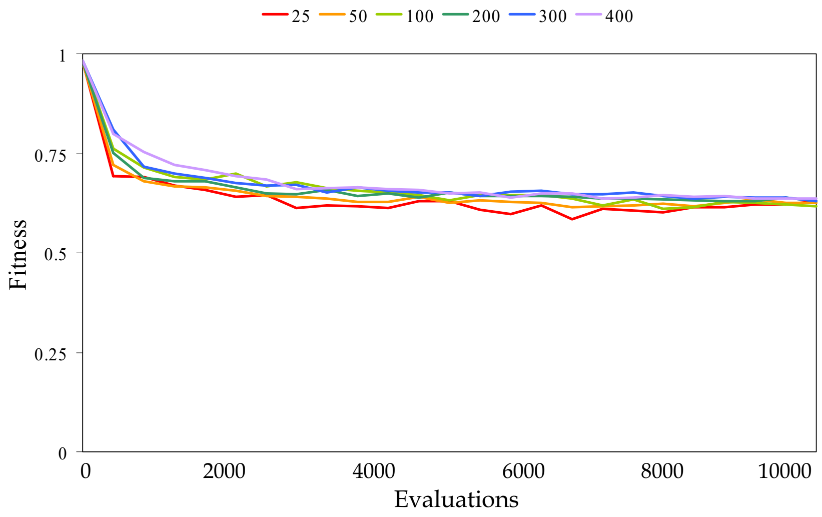

Fig. 18. Average Fitness: Population Size Variation

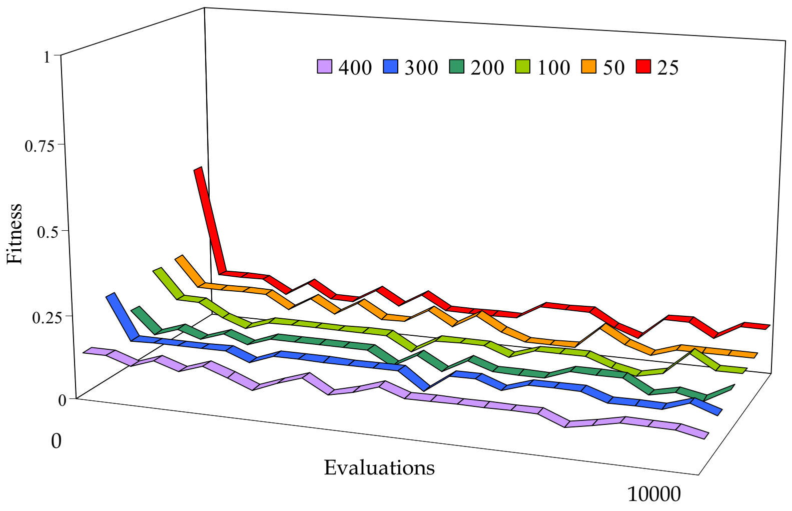

Fig. 19. Best Fitness: Population Size Variation

6.4.3 Selection Methods

Trials were conducted to compare the performance of three methods of selection: roulette wheel selection, tournament selection with tournament size = 2 and tournament selector = 0.75, and elitist selection with 5% retention. The population average fitness for the 10 trials conducted for each method is shown in Fig. **20**, and shows clearly that in terms of the population average fitness the method of selection is not a significant determinant. Fig**.** 21 shows the best fitness curves for these trials and shows that the elitist method produces similar results to the tournament method, with the roulette wheel method producing slightly less stable results. The good performance of the elitist method is probably due to the stochastic nature of the environment. Since the placement of the ball and the player is random, a player evaluated twice would likely be assigned different fitness values for each evaluation. The elitist method works well in this type of environment, allowing the better solutions to be evaluated several times thus allowing the reliability of the estimate of fitness to increase over time.

6.4.4 Crossover Methods

Since the chromosomes involved in crossover may be of different lengths, crossover methods that assume equal length chromosomes are not defined. The performance of two methods of crossover was compared: one-point and two-point. Fig. **22** shows the population average fitness over the duration of the evolutionary process, and Fig. **23** shows the best fitness values for the same period. It can be seen from this data that there is no meaningful difference in performance between the two methods, either with respect to the population average fitness or the best fitness achieved. While two-point crossover is more disruptive than one-point crossover, it is not clear from this data if a much more disruptive crossover method, such as uniform crossover, would significantly affect the performance of the method. It is likely that the messy-coding of the genetic algorithm and the rules-based nature of the representation causes the method to be somewhat less sensitive to disruptive crossover.

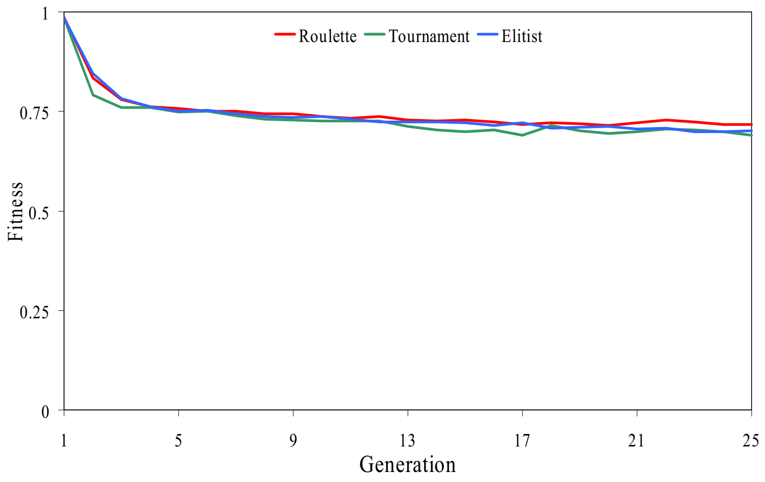

Fig. 20. Average Fitness: Selection Method Variation

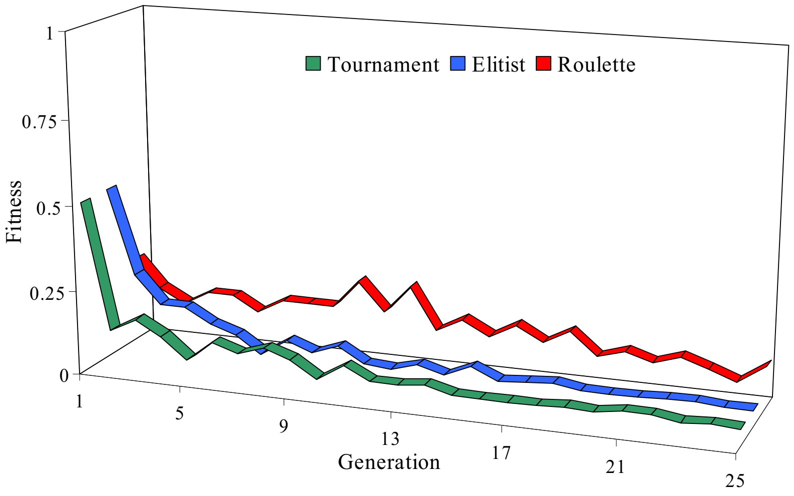

Fig. 21. Best Fitness: Selection Method Variation

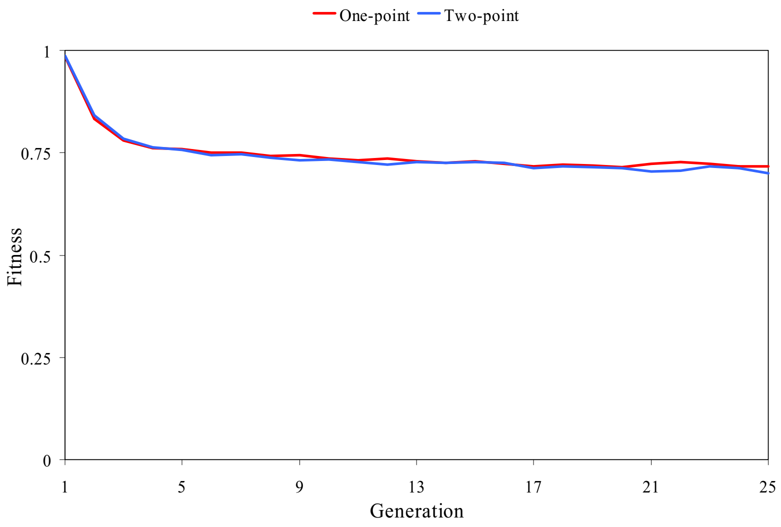

Fig. 22. Average Fitness: Crossover Method Variation

6.4.4 Mutation Rate

To determine the effect of the rate of mutation on the evolutionary process, 10 trials for each of several mutation rates were performed, and the averages of those trials presented. Fig. **24** shows the population average fitness for each mutation rate tested, and Fig. **25** the best fitness for those mutation rates throughout the evolutionary search. While varying the mutation rate has only a marginal effect on the population average fitness and, for the most part the individual best fitness, a mutation rate of 15% does seem to improve the population average fitness slightly, and the individual best fitness more markedly. This suggests that a mutation rate of 15% is the best balance between maintaining sufficient diversity in the population to help drive the evolutionary process while minimising the disruption to the good building blocks being created throughout the process.

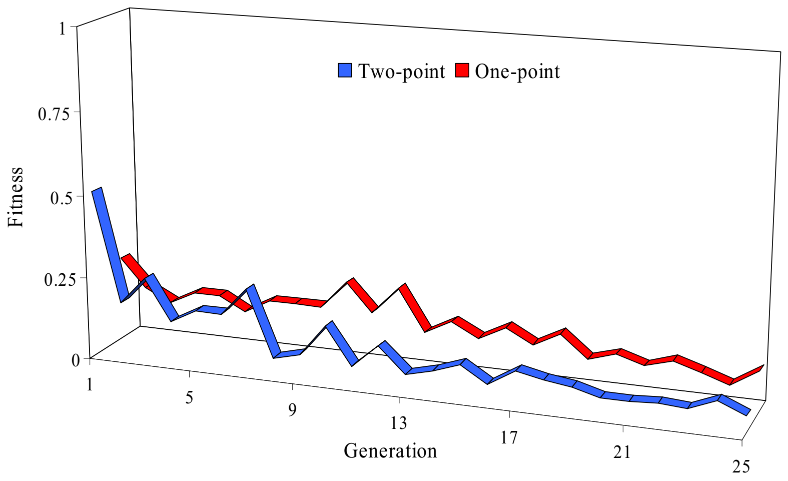

Fig. 23. Best Fitness: Crossover Method Variation

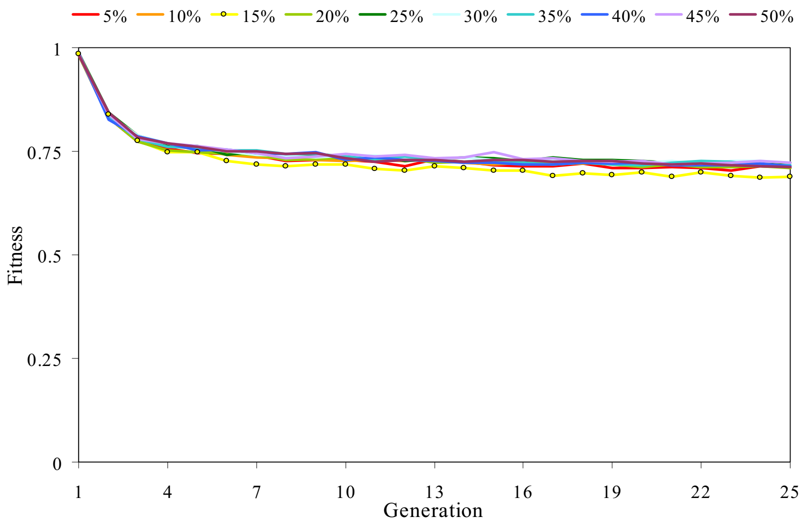

Fig. 24. Average Fitness: Mutation Rate Variation

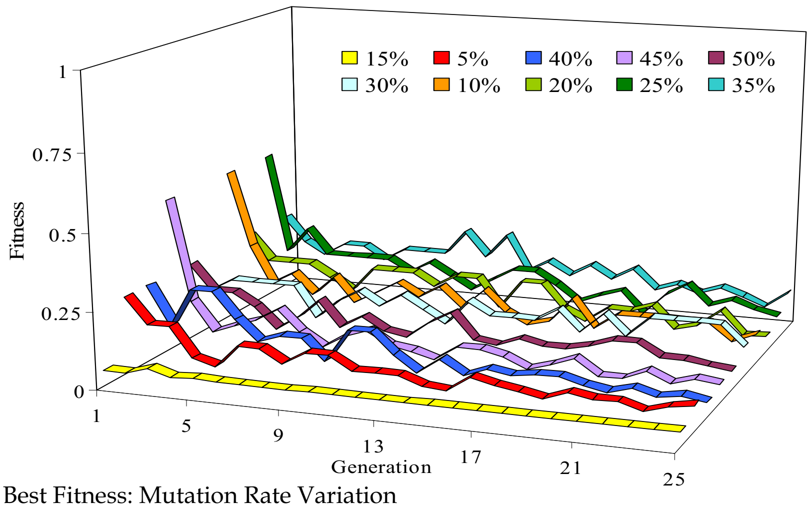

Fig. 25. Best Fitness: Mutation Rate Variation

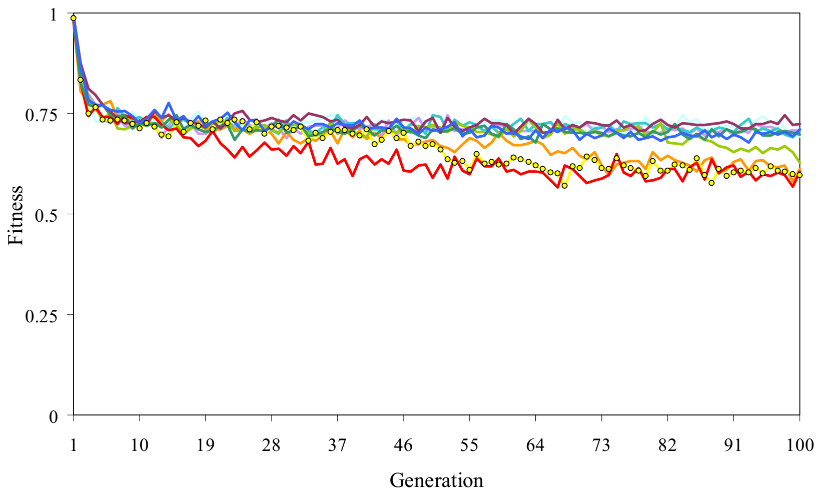

Fig. 26. Average Fitness: 100 Generations

6.4.4 Maximum Generations

In order to determine the effect of allowing the evolutionary process to continue for an extended time, a series of 10 trials was conducted with each trial continuing the evolutionary process for 100 generations. Two graphs of the results of these trials are presented. Fig. **26** shows the average fitness of the population for each of the 10 trials, and it can be seen that for more than half the trials the average fitness does not improve significantly after the tenth generation. Fig. **27** shows the best individual fitness from the population after each generation for each of the trials, and presents a similar scenario to that of the average fitness values: the best fitness does not improve significantly in most of the trials after the first few generations, though for a small proportion of the trials some significantly fitter individuals are evolved. These graphs suggest that although for some instances continuing to allow the population to evolve for an extended period can produce an improved population average, and that in those instances the best performing individuals from the population are consistently better, there is no real advantage in extending the evolutionary process. In almost every case an individual from the first 10 to 15 generations achieved the equal best fitness seen over the 100 generations, so given that the objective is to find good goal-scoring behaviour there would seem to be no real advantage in extended evolution of the population. This is a similar to the result reported in (Luke 2001), where Luke suggests that for some problems genetic programming encounters a critical point beyond which further evolution yields no improvement. Luke further suggests that performing many shorter experiments is likely to produce better results than a single very long experiment.

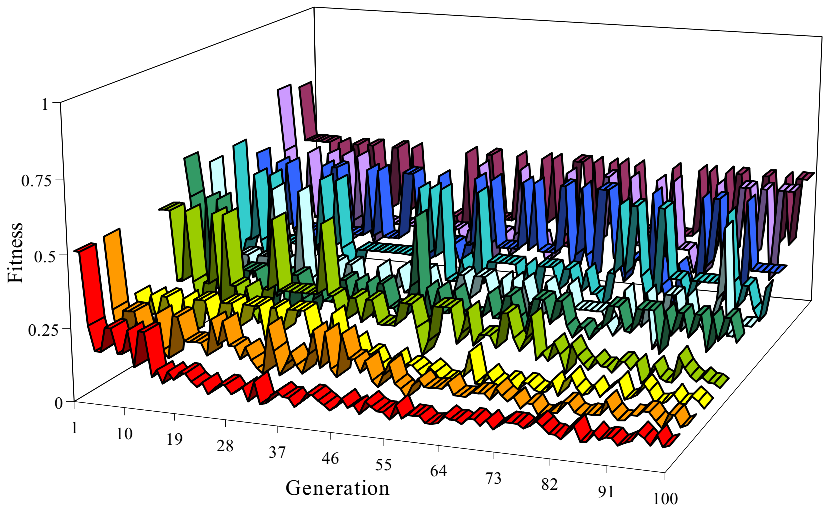

Fig. 27. Best Fitness: 100 Generations

6.4.5 Gene Pools

The trials reported in the previous section provide an opportunity to study, in broad terms, the genetic makeup of the population being evolved. For each of the 100 generations, Fig. **28** shows the average number of genes per chromosome, premises per chromosome, rules per chromosome, and valid rules per chromosome, with standard deviations for each. This graph shows that average chromosome length does not grow uncontrollably, and in fact plateaus at about 2/3 the maximum possible length. The average number of rules per chromosome, and hence the average number of consequents per chromosome, grows steadily throughout the evolutionary run. This agrees with the data presented earlier for the maximum chromosome length variation trials. It is interesting to note that the number of rules per chromosome is still increasing after 100 generations. Since the minimum number of genes per rule is 2 the number of rules per chromosome is bounded by half the chromosome length, and although the graph shows the number of rules approaching the upper bound, it has not reached that figure after 100 generations. Though outside the scope of this chapter, some more experimentation to observe the effect of reaching the upper bound would be useful work to undertake in the future.

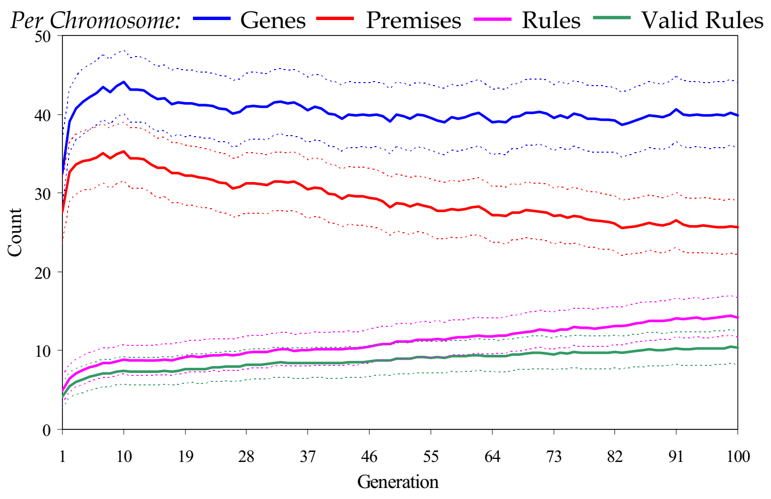

Fig. 28. Population Composition Mean and Standard Deviation: 100 Generations

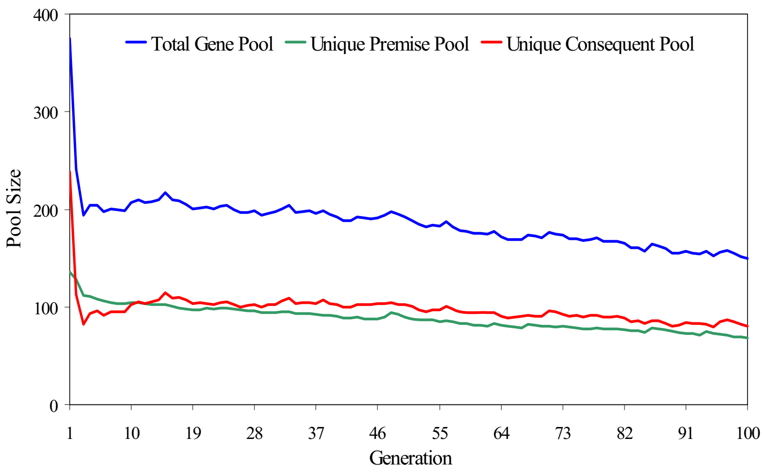

Fig. 29. Gene Pools: 100 Generations

The population gene pool sizes throughout the evolutionary process are show in Figure 29. The graph shows raw numbers of unique premise genes, unique consequent genes, and the total gene pool available to each generation of individuals. It is evident from the graph that the pool of unique premises falls slowly, but steadily, from the first generation, while the pool of unique consequent genes stays reasonably constant for close to 40 generations after an initial decrease. This is not unexpected, and is an indication that some selection pressure is evident, but that the number of rules remains fairly constant.

6.5 Performance Comparison – GA vs Random Search

In this section, in order to gauge the relative difficulty of the problem, the results obtained using the messy-coded GA search are compared to results obtained from random search. The messy-coded GA results shown are the average of the 10 trials conducted for the "maximum generations" experiments described earlier. The random search technique was simply the random generation and evaluation of a "population" of 500 individuals repeated 40 times – to equal the number of evaluations completed for the messy-coded GA trials. The "population" average fitness is shown in Fig. **30**, and the best individual fitness at intervals of 500 evaluations is shown in Fig. 31. The average fitness curves are included only to illustrate that the genetic algorithm is able to consistently improve the quality of the population for the duration of the evolutionary process: random search would not be expected to perform in the same manner. The best fitness curves (Fig. 31) show that although random search is able to find individuals that exhibit goal-scoring behaviour (i.e. fitness ≤ 0.5), evolutionary search finds better individuals and finds them more consistently, indicating that evolutionary search is not only a more successful search technique than random search, it is more robust.

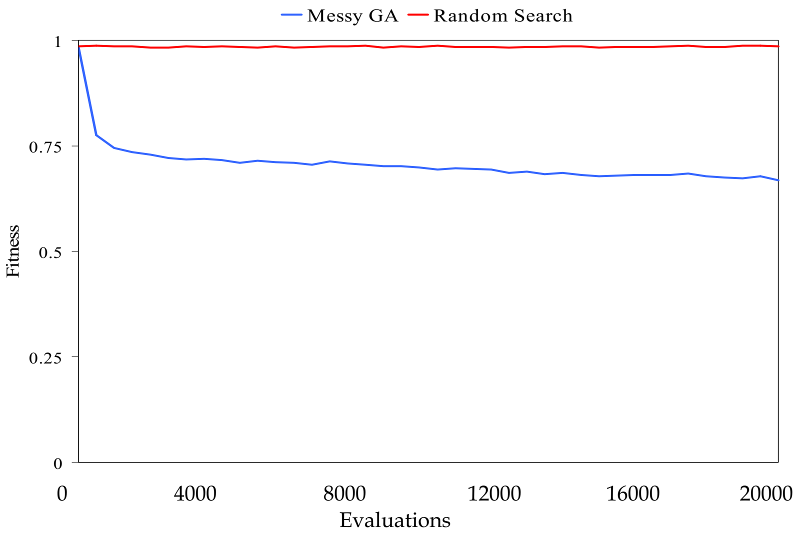

Fig. 30. Average Fitness: Random Search Comparison

As noted, random search does successfully find individuals with reasonably good goal-scoring skills. This result is a little surprising at first glance, but on closer inspection of the problem an explanation does present itself – the problem of finding goal-scoring behaviour *in the solution space defined*, whether by evolutionary search or random search, is not as difficult as it first seems, and this is because the solution space defined is not the one envisaged when first considering the problem. The solution space defined is one populated by players with mid-level, hand-coded skills available to them, as well as a "smart" default hunt action, which is a much richer solution space than the one usually envisaged when considering the problem of evolving goal-scoring behaviour "from scratch". As evidenced by the results of the random search shown here, the density of "reasonably good" solutions in the solution space is sufficiently high that random search will occasionally, and with some consistency, find one of those "reasonably good" solutions.

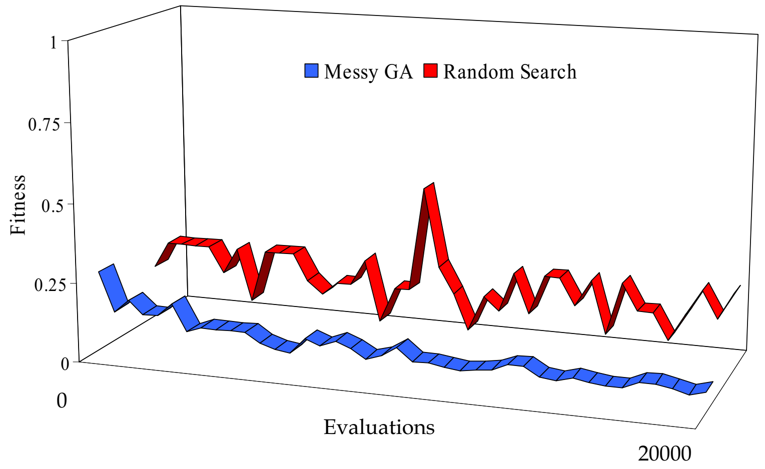

Fig. 31. Best Fitness: Random Search Comparison

7. Summary and Discussion

The work presented in this chapter has provided an implementation and empirical analysis of a fuzzy logic-based robot soccer player and the messy-coded genetic algorithm training algorithm. Several trials were performed to test the capacity of the method to produce goal-scoring behaviour. The results of the trials performed indicate that the player defined by the evolved fuzzy rules of the controller is capable of displaying consistent goal-scoring behaviour. This outcome indicates that for the problem of developing goal-scoring behaviour in a simulated robot soccer environment, when the initial population is endowed with a set of mid-level hand-coded skills, taking advantage of the flexible representation afforded by the messy-coded genetic algorithm and combining that with a fuzzy logic-based controller enables a fast and efficient search technique to be constructed.

Several experiments were performed to vary the genetic algorithm parameters being studied. The results of those tests indicate that within the range of the values tested, most parameters have little effect on the performance of the search. The *Maximum Chromosome Length* and *Selection Method* parameters had a marginal influence over the efficacy of the search, and although better performance was sometimes achieved after a long period of evolution, the *Maximum Generations* parameter is not considered to have a large effect on the performance of the algorithm after an upper bound of about 15 generations.

8. References

Andre, D. and A. Teller (1999). Evolving Team Darwin United. M. Asada and H. Kitano eds, RoboCup-98: Robot Soccer World Cup II. Lecture Notes in Computer Science, Springer-Verlag.

Aronsson, J. (2003). Genetic Programming of Multi-agent System in the RoboCup Domain. Masters Thesis, Department of Computer Science. Lund, Sweden, Lund Institute of Technology.

Asada, M., S. Noda, et al. (1996). "Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning." Machine Learning **23**(2-3): 279-203.

Bajurnow, A. and V. Ciesielski (2004). Layered Learning for Evolving Goal Scoring Behaviour in Soccer Players. G. Greenwood, ed., Proceedings of the 2004 Congress on Evolutionary Computation, Vol. 2, p. 1828-1835, IEEE.

Balch, T. (2005). Teambots Domain, http://www.teambots.org.

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

Brooks, R. (1991). "Intelligence Without Representation." Artificial Intelligence **47**: 139-159.

Castillo, C., M. Lurgi, et al. (2003). Chimps: An Evolutionary Reinforcement Learning Approach for Soccer Agents. Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics, Vol. 1, p. 60-65.

Ciesielski, V. and S. Y. Lai (2001). Developing a Dribble-and-Score Behaviour for Robot Soccer using Neuro Evolution. Proceedings of the Fifth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems, p. 70-78, Dunedin, New Zealand.

Ciesielski, V., D. Mawhinney, et al. (2002). Genetic Programming for Robot Soccer. Proceedings of the RoboCup 2001 Symposium. Lecture Notes in Artificial Intelligence, p. 319-324.

Ciesielski, V. and P. Wilson (1999). Developing a Team of Soccer Playing Robots by Genetic Programming. Proceedings of the Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems, p. 101-108, Canberra, Australia.

Di Pietro, A., L. While, et al. (2002). Learning in RohoCup Keepaway Using Evolutionary Algorithms. Langdon et al., eds, Proceedings of the Genetic and Evolutionary Computation Conference, p. 1065-1072, New York, NY, Morgan Kaufmann.

Gustafson, S. M. (2000). Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem. Masters Thesis, Department of Computing and Information Science, College of Engineering. Manhattan, KS, Kansas State University.

Gustafson, S. M. and W. H. Hsu (2001). Layered Learning in Genetic Programming for a Co-operative Robot Soccer Problem. Proceedings of the Fourth European Conference on Genetic Programming, Lake Como, Italy, Springer.

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

Hsu, W. H., S. J. Harmon, et al. (2004). Empirical Comparison of Incremental Reuse Strategies in Genetic Programming for Keep-Away Soccer. Late Breaking Papers of the 2004 Genetic and Evolutionary Computation Conference, Seattle WA.

Jang, J.-S., C.-T. Sun, et al. (1997). Neuro-Fuzzy and Soft Computing, Prentice Hall.

Kandel, A. (1986). Fuzzy Mathematical Techniques with Applications, Addison-Wesley, Reading MA.

Kinoshita, S. and Y. Yamamoto (2000). 11Monkeys Description. Veloso et al., eds, Proceedings of Robocup-99: Robot Soccer World Cup III. Lecture Notes In Computer Science, Vol. 1856, p. 550-553, Springer-Verlag, London.

Kitano, H., M. Asada, et al. (1997a). RoboCup: The Robot World Cup Initiative. Proceedings of the First International Conference on Autonomous Agents, p. 340-347, Marina Del Rey, CA.

Kitano, H., M. Asada, et al. (1997b). RoboCup: A Challenge Problem for AI. AI Magazine, 18(1): p.73-85. **18**.

Kitano, H., M. Tambe, et al. (1997). The RoboCup Synthetic Agent Challenge 97. Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, p. 24-29, Nagoya, Japan.

Klir, G. J. and T. A. Folger (1988). Fuzzy Sets, Uncertainty and Information, Prentice Hall.

Kruse, R., J. Gebhardt, et al. (1994). Foundations of Fuzzy Systems, Wiley.

Kuhlmann, G. and P. Stone (2004). Progress in Learning 3 vs. 2 Keepaway. D. Polani et al., eds, RoboCup-2003: Robot Soccer World Cup VII, Springer Verlag, Berlin.

Lazarus, C. and H. Hu (2003). Evolving Goalkeeper Behaviour for Simulated Soccer Competition. Proceedings of the Third IASTED International Conference on Artificial Intelligence and Applications, Benalmádena, Spain.

Lima, P., L. Custódio, et al. (2005). RoboCup 2004 Competitions and Symposium: A Small Kick for Robots, a Giant Score for Science. AI Magazine 6(2). **6**.

Luke, S. (1998a). Evolving SoccerBots: A Retrospective. Proceedings of the Twelfth Annual Conference of the Japanese Society for Artificial Intelligence, Tokyo, Japan.

Luke, S. (1998b). Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97. J. Koza et al., eds, Proceedings of the Third Annual Genetic Programming Conference, p. 204-222, Morgan Kaufmann, San Francisco.

Luke, S. (2001). When Short Runs Beat Long Runs. Proceedings of the 2001 Genetic and Evolutionary Computation Conference, p. 74-80, San Francisco CA.

Luke, S., C. Hohn, et al. (1998). Co-evolving Soccer Softbot Team Coordination with Genetic Programming. H. Kitano, ed., RoboCup-97: Robot Soccer World Cup I. Lecture Notes in Artificial Intelligence, p. 398-411, Springer-Verlag, Berlin.

Luke, S. and L. Spector (1996). Evolving Teamwork and Coordination with Genetic Programming. J.R. Koza et al., eds, Proceedings of the First Annual Conference on Genetic Programming, p. 150-156, Cambridge MA, The MIT Press.

Mamdani, E. and S. Assilian (1975). "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller." International Journal of Man-Machine Studies **7**(1): 1-13.

Nakashima, T., M. Takatani, et al. (2004). An Evolutionary Approach for Strategy Learning in RoboCup Soccer. Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, p. 2023-2028.

Noda, I. (1995). Soccer Server: A Simulator of Robocup. Proceedings of AI Symposium '95. Japanese Society for Artificial Intelligence, pp. 29-34.

Noda, I., H. Matsubara, et al. (1998). "Soccer Server: A Tool for Research on Multiagent Systems." Applied Artificial Intelligence **12**: 233-250.

Noda, I. and P. Stone (2001). The RoboCup Soccer Server and CMUnited: Implemented Infrastructure for MAS research. T. Wagner and O. Rana, eds, International Workshop on Infrastructure for Multi-Agent Systems (Agents 2000). Lecture Notes in Computer Science, p. 94-101, Barcelona, Spain.

Riedmiller, M., T. Gabel, et al. (2005). Brainstormers 2D - Team Description 2005. Team Description Papers, Proceedings of RoboCup 2005 (CD) (to appear).

Riedmiller, M., A. Merke, et al. (2001). Karlsruhe Brainstormers - a Reinforcement Learning Approach to Robotic Soccer. P. Stone, T. Balch and G. Kraetszchmar, eds, RoboCup-2000: Robot Soccer World Cup IV. Lecture Notes in Artificial Intelligence., Springer Verlag, Berlin.

Riley, J. (2003). The SimpleSoccer Machine Learning Environment. S.-B. Cho, H. X. Nguen and Y. Shan, eds, Proceedings of the First Asia-Pacific Workshop on Genetic Programming, p. 24-30, Canberra, Australia.

Riley, J. (2007). Learning to Play Soccer with the SimpleSoccer Robot Soccer Simulator. Robotic Soccer. P. Lima. Vienna, I-Tech Education and Publishing,**:** 281-306.

Stone, P. (1998). Layered Learning in Multiagent Systems. PhD Thesis, Computer Science Department, Technical Report CMU-CS98-187, Carnegie Mellon University.

Stone, P. and R. Sutton (2001). Scaling Reinforcement Learning Toward RoboCup Soccer. Proceedings of the Eighteenth International Conference on Machine Learning, Williamstown MA.

Stone, P., R. S. Sutton, et al. (2005). "Reinforcement Learning for RoboCup-Soccer Keepaway." Adaptive Behavior **13**(3): 165-188.

Stone, P., R. S. Sutton, et al. (2001). Reinforcement Learning for 3 vs. 2 Keepaway. Robocup 2000: Robot Soccer World Cup IV. P. Stone, T.R. Balch, and G.K. Kraetzschmar, eds. Lecture Notes In Computer Science, vol. 2019, p. 249-258, Springer-Verlag, London.

Stone, P. and M. Veloso (1999). Team-partitioned, Opaque-transition Reinforcement Learning. Proceedings of the Third International Conference on Autonomous Agents, Seattle WA.

Stone, P. and M. M. Veloso (2000). Layered Learning. Proceedings of the Eleventh European Conference on Machine Learning, p. 369-381, Springer, Berlin.

Uchibe, E. (1999). Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots. PhD Thesis, Osaka University.

Watkins, C. (1989). Learning from Delayed Rewards. PhD Thesis, King's College, University of Cambridge.

Whiteson, S., N. Kohl, et al. (2003). Evolving Keepaway Soccer Players through Task Decomposition. E. Cantu-Paz et al., eds, Genetic and Evolutionary Computation - GECCO-2003, volume 2723 of Lecture Notes in Computer Science, p. 356-368, Chicago IL, Spinger-Verlag.

Whiteson, S., N. Kohl, et al. (2005). "Evolving Keepaway Soccer Players through Task Decomposition." Machine Learning **59**: 5-30.

Whiteson, S. and P. Stone (2003). Concurrent Layered Learning. Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, p. 193-200.

Zadeh, L. (1965). "Fuzzy Sets." Journal of Information and Control **Vol. 8**.