Learning to Play Soccer with the SimpleSoccer Robot Soccer Simulator

Jeff Riley
RMIT University
Australia

1. Introduction

The RoboCup simulated soccer league (RoboCupSoccer) 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 (Kitano et al., 1995; Kitano et al., 1997). 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; Ciesielski & Wilson, 1999; Stone & Veloso, 1999; Uchibe, 1999; Ciesielski & Lai, 2001; Ciesielski et al., 2001; Riedmiller et al., 2001; Stone & Sutton, 2001; Bajurnow & Ciesielski, 2004; Riley & Ciesielski, 2004; Lima et al., 2005; Riedmiller et al., 2005; Riley, 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. More progress could be made more quickly if the complexity and uncertainty could be reduced: while tactics may differ due to uncertainty in the environment, high-level strategies learned in a less complex and more certain environment should transfer directly to a more complex and less certain environment.

SimpleSoccer[1] (Riley, 2003) was developed as an environment that reduces complexity and uncertainty sufficiently to increase the viability of machine learning techniques, yet retains sufficient complexity and dynamics to allow learning from SimpleSoccer to be directly transferrable to the RoboCupSoccer environment.

2. The SimpleSoccer Robot Soccer Simulator

The primary objective when creating the SimpleSoccer environment was to create an environment complex and dynamic enough that while low-level tactics may differ due to

[1] Full documentation and source code is located at http://www.rileys.id.au/SimpleSoccer.html

Source: Robotic Soccer, Book edited by: Pedro Lima, ISBN 978-3-902613-21-9, pp. 598, December 2007, Itech Education and Publishing, Vienna, Austria

the removal of systematic uncertainty, high-level strategies directly applicable to the RoboCupSoccer environment could be developed, thus providing a simple yet sufficiently accurate model of the RoboCupSoccer environment that allows rapid learning. The design objective was achieved by modelling only the attributes of the RoboCupSoccer environment necessary to allow ball and player interaction with the provision of basic player actions, and by not modelling the client-server environment and systematic uncertainty inherent in RoboCupSoccer. The SimpleSoccer environment is comprised of :

- the soccer field
- fixed landmarks - the goals
- the ball
- up to two teams with a maximum of eleven players each.

The SimpleSoccer environment was inspired in part by simplicity of the Ascii Soccer environment (Balch, 1995), but is a more complex environment which more closely models the RoboCupSoccer environment.

2.1 The Field

The soccer field in SimpleSoccer is represented by a two-dimensional grid with the goal markers being the only landmarks available to players (Fig. 1) The goal area for SimpleSoccer, in keeping with the RoboCupSoccer field and goals, is a defined area at each end of the field. The boundaries in SimpleSoccer, except for the goal areas, are hard barriers which impede movement of the ball and players: the ball does not rebound from the boundaries. Both the field size (length and width expressed as a number of cells) and goal size (in cells) are configurable.

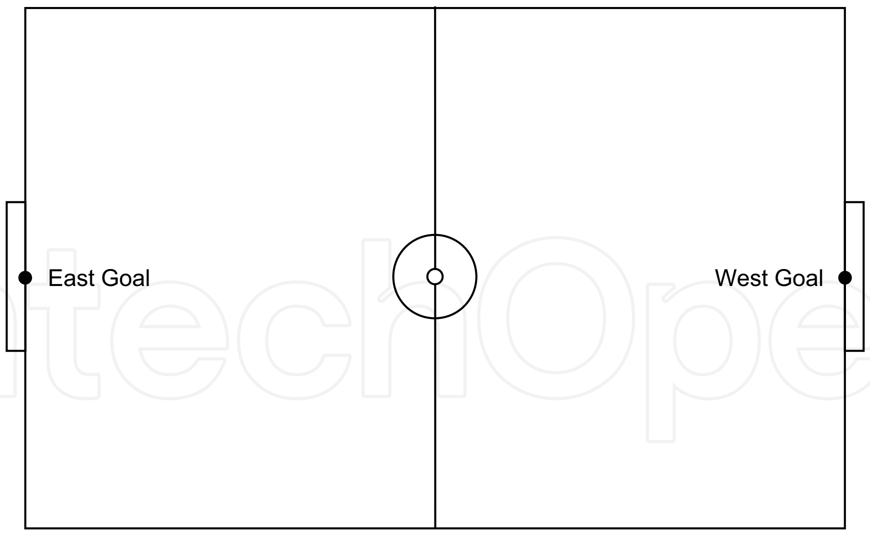


Fig. 1. Soccer field and landmarks in the SimpleSoccer environment (grid lines not shown)

2.2 The Players

Player movement and sensory capabilities in SimpleSoccer are similar to those of RoboCupSoccer. In the SimpleSoccer environment players may move in any direction, specified by a real-valued angle from 0.0 to 360.0 degrees relative to the player's current facing direction. Similarly, the ball can be kicked in any direction. Player and ball locations are specified by discrete grid co-ordinates, or cells: while movement and other actions can be in any direction, at the completion of an action, player and ball final locations are quantized to discrete cells. A player can only kick the ball if the ball is within a defined kickable distance (measured in cells) from the player.

Players in SimpleSoccer have a field of vision similar to that of RoboCupSoccer. Fig. 2 shows the range of a player's vision in the SimpleSoccer environment – players can see objects in a diamond-shaped area in the direction the player is facing. A player's viewing diamond is determined by the *view angle* and *view length*, and only objects of interest (ball, player or goal) within a player's viewing diamond can be seen by the player. The black circles shown in Fig. 2 represent objects on the field – only one is visible to the player in the diagram. At each time interval during a game all players are presented with the cell co-ordinates of, and direction (relative to the player's facing direction) and distance (number of cells) to any object of interest in the player's field of vision. Note that the information supplied to the player is limited to object location – no information regarding the movement of an object, either direction or speed, is supplied. The location, and hence direction to, an object is only known to a player if that object is visible to the player. Players may infer the location of objects based on previously known information, but this is likely to be less than reliable.

The detail of the visual feedback delivered to a player in the SimpleSoccer environment is the same irrespective of the player's vision parameters – only the size of the viewing diamond changes, the amount of detail does not. Unlike the RoboCupSoccer environment, players are not able to sense objects that are close but not visible to the player – the only sense available to players in the SimpleSoccer environment is visual.

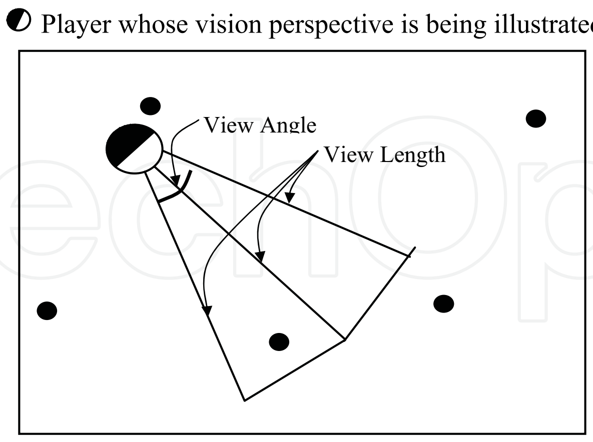


Fig. 2. The visible range of a player in the SimpleSoccer environment

2.2.1 Available Player Actions

The set of player actions provided by the SimpleSoccer simulator is a combination of some very basic, simple actions and some more complex hand-coded combinations of the basic actions. The complete set of actions available is listed in Table 1.

Action	Description
Turn	The player turns through the angle specified. *Argument:* direction.
Dash	The player dashes in the direction specified with the power specified. *Arguments:* direction, power, face.
Kick	If the ball is within a kickable distance from the player, the player kicks the ball in the direction specified with the power specified. *Arguments:* direction, power, face.
RunTowardGoal	If the direction to the player's goal is known, the player dashes once in that direction, otherwise no action is taken. *Argument:* power.
RunTowardBall	If the direction to the ball is known, the player dashes once in that direction, otherwise no action is taken. *Argument:* power.
GoToBall	If the direction to the ball is known, the player dashes towards the ball and continues to dash in that direction until the ball is within the kickable distance, otherwise no action is taken. *Argument:* power.
KickTowardGoal	If the direction to the player's goal is known, and the ball is within the kickable distance, the player kicks the ball once in the direction of its goal, otherwise no action is taken. *Argument:* power.
DribbleTowardGoal	If the direction to the player's goal is known, and the ball is within the kickable distance, the player kicks the ball once in the direction of its goal, then dashes once in the same direction. If the direction to the player's goal is not known, or the ball is not within the kickable distance, no action is taken. Argument: power.
Dribble	If the ball is within the kickable distance, the player kicks the ball once in the direction it is facing, then dashes once in the same direction. If the ball is not within the kickable distance, no action is taken. *Argument:* power.
DoNothing	The player takes no action.

Table 1. Available player actions

For each of the actions shown in Table 1:

- *direction* is specified in degrees in a clockwise direction relative to the direction the player is facing.
- *power* is specified as a percentage of maximum power and determines the number of cells the player or ball will travel as a result of the action.
- *face*, where specified, if true causes the player to turn to face in the direction specified after the completion of the action performed.

2.2.2 Player Default Action

If a player is unable to determine an action to be taken based on the information known, the player may, if so configured, perform a hand-coded default *hunt* action - on the basis that the most likely cause for a player not being able to determine an action is that the ball is not visible. The hand-coded hunt actions available as default actions are listed in Table 2.

Default Action	**Description**
Hunt Action 1	if the ball is not visible then
Goto Ball	dash in a randomly chosen direction
	else
	if ball is not in kickable distance then
	dash toward the ball
	else
	do nothing
Hunt Action 2	if the ball is not visible then
Locate Ball	dash in a randomly chosen direction
	else
	do nothing
Hunt Action 3	turn 90° in a randomly chosen direction
Random Turn	

Table 2. Player default actions.

2.3 The Game

A SimpleSoccer game is played between two teams, each with a minimum of zero and a maximum of eleven players. There must be at least one player present on the field, and the team sizes may be unequal, thus allowing for single player or single team training. The *East* team starts the game on the right-hand (or east) side of the playing field (as viewed by the observer) and kicks towards the *East Goal* (Fig. 1). Similarly, the *West* team starts the game on the left-hand (or west) side of the playing field and kicks towards the *West Goal*. At the start of play the ball is placed at the centre of the field, and only the team *kicking-off* may enter the centre circle until contact is made with the ball.

There is no referee in the SimpleSoccer environment, thus there are no free kicks for offside or other rule violations. The ball is never out of bounds; the boundaries (except for the goal areas) are hard barriers which impede movement of the ball and players. There is no concept of player momentum and stamina as implemented in the RoboCupSoccer environment.

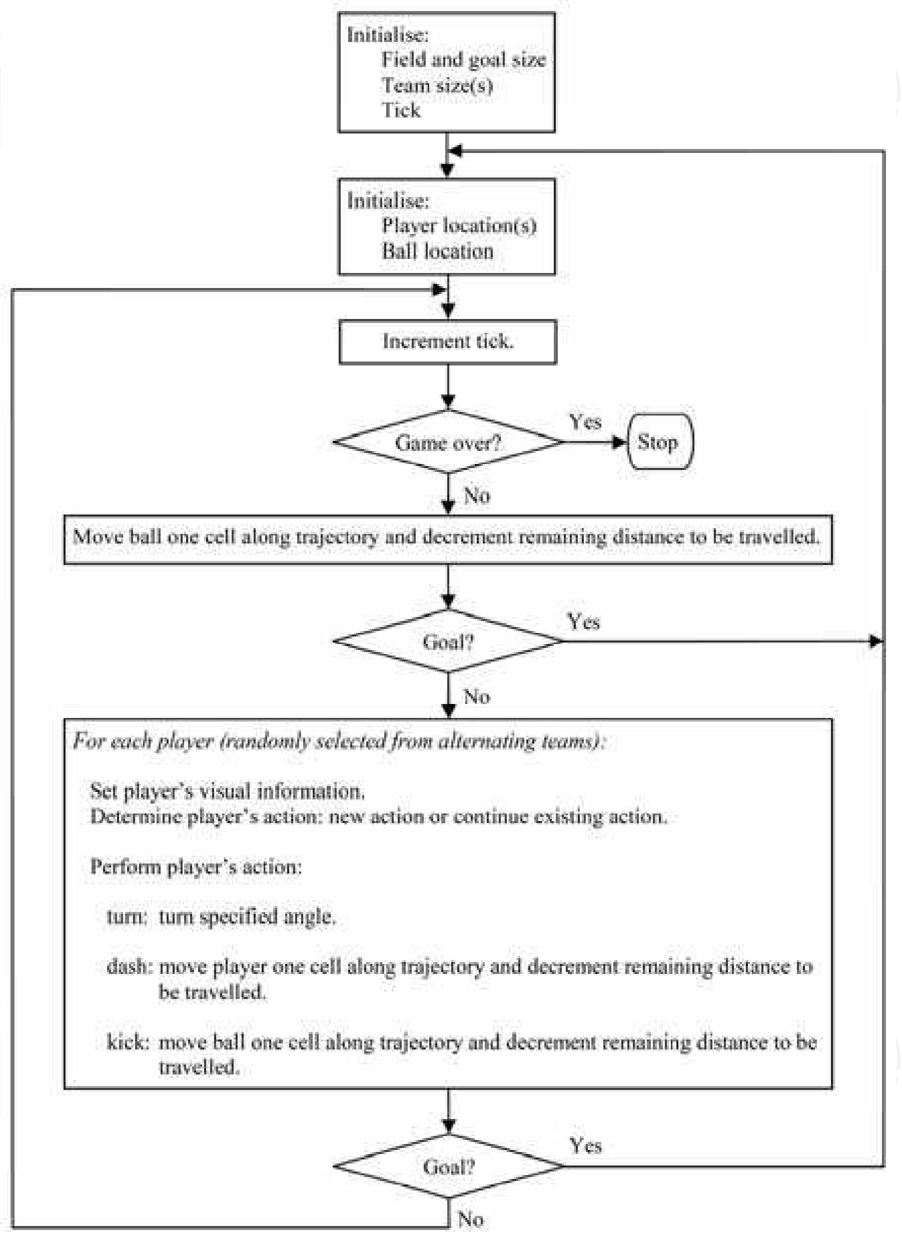

Initialise:
Field and goal size
Team size(s)
Tick
Initialise:
Player location(s)
Ball location
Increment tick.
Game over?
Yes
Stop
No
Move ball one cell along trajectory and decrement remaining distance to be travelled.
Goal?
Yes
No
For each player (randomly selected from alternating teams):
Set player's visual information.
Determine player's action: new action or continue existing action.
Perform player's action:
turn: turn specified angle.
dash: move player one cell along trajectory and decrement remaining distance to be travelled.
kick: move ball one cell along trajectory and decrement remaining distance to be travelled.
Goal?
Yes
No

Fig. 3. SimpleSoccer program flow

When the ball is kicked, the distance it will travel is calculated (as a number of cells) and the ball will travel that distance at a constant speed and direction unless it is kicked again, or it encounters a barrier (the field boundary or a player) or reaches a goal. Similarly, when a player dashes, the distance the player will travel is calculated (as a number of cells) and the player will travel that distance at a constant speed and direction unless it initiates a new action or encounters a barrier (the field boundary, a goal or a player).

The SimpleSoccer environment provides players with a single sensor that detects visual information about the field, such as the distance and direction to objects in the player's current field of view – no other information is provided to the player. There is no coach, and there is no communication of any kind between players. In contrast to the RoboCupSoccer environment, no random "noise" is introduced to the visual sensor information provided to the player – thus the information provided is complete and certain, and there is no loss of clarity of vision over distance.

A SimpleSoccer unit of time is a single tick corresponding to one iteration of the program's main loop (Fig. 3). At each tick the ball and players are moved, if necessary, a single cell (as a result of a previous action) and each player is presented with their new (visual) view of the state of the game, whereupon each player determines what action, if any, is to be taken and that action is begun (any previous action still in progress is superseded by the new action).

After each goal scored the ball is replaced at the centre of the field and the players replaced to their side of the field, and the game continues. The game is terminated when one of the following conditions is met:

- the maximum game time, measured in ticks, expires.
- the target number of goals is scored by any team.
- a period of no player action, measured in ticks, occurs.

3. Evolving Goal-Scoring Behaviour

The usefulness of the SimpleSoccer simulator as a simplified model for the robot soccer environment is demonstrated by using the environment to train a simulated robot soccer player to exhibit goal-scoring behaviour.

3.1 Overview

A messy-coded genetic algorithm (Holland, 1975; Goldberg et al., 1989) is used to evolve a population of simulated robot soccer players, with the SimpleSoccer simulator being used to evaluate the players' ability. The behaviour of the players is governed by a *fuzzy inferencing system* (Zadeh, 1965; Jang et al., 1997) with the ruleset for the fuzzy inferencing system being evolved by the genetic algorithm.

Players being evolved are endowed with a configurable subset of soccer-playing skills taken from the full set of skills shown in Table 1. In addition, if a player is unable to determine an action to be taken based on the information known to it, the player will perform one of the hand-coded default actions listed in Table 2.

Players perform one of the available actions, or the configured default action, in response to external stimulus; the specific response being determined by the fuzzy ruleset and the fuzzy inferencing system. The external stimulus used as input to the fuzzy inference system is the visual information supplied by the soccer simulator. The output of the fuzzy inference system is an (*action, value*) pair which defines 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. An example rule developed by the genetic algorithm is:

if Ball is Left and Goal is Left then Turn SlightlyLeft

The fuzzy inferencing system and messy-coded genetic algorithm are described briefly in the following sections, and in more detail in (Riley, 2005).

3.2 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. 4).

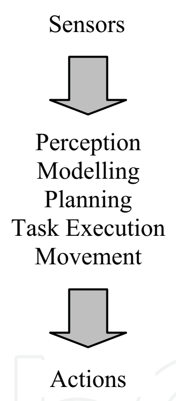


Fig. 4. Traditional control architecture

The control architecture implemented in this work is similar to Brooks' subsumption architecture (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. 5 the *Movement* layer does not explicitly need to avoid obstacles: the *Avoid Objects* layer, if present, will take care of that. This approach creates players with reactive architectures and with no central locus of control (Brooks, 1991).

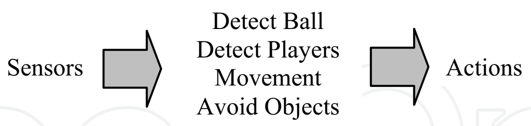


Fig. 5. Soccer player layered architecture

For the soccer player implemented for this work, 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.
- 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. 6).

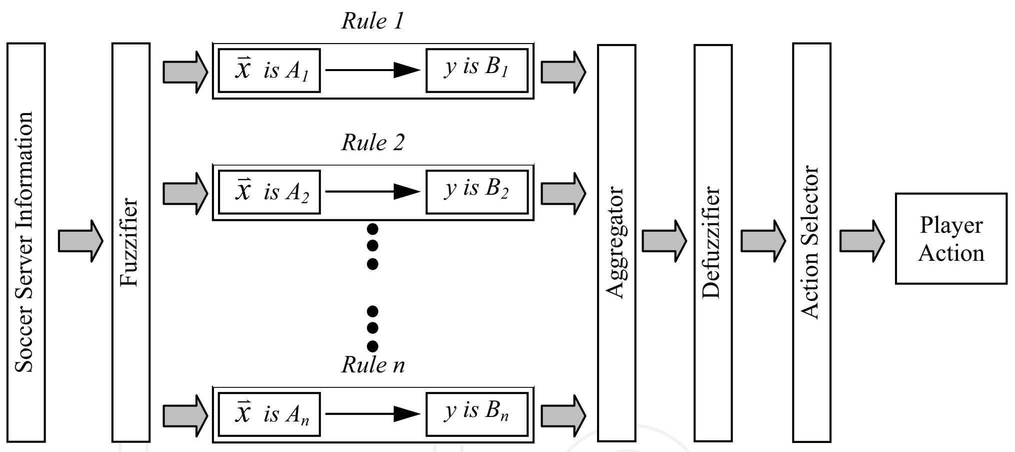


Fig. 6. Player architecture detail

3.2.1 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 player. 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 SimpleSoccer server delivers only regular visual messages to the players: there are no aural or sense equivalents of the aural and sense messages delivered by the RoboCupSoccer server in that environment. Information supplied by the SimpleSoccer server is complete, in so far as the objects actually in 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.

3.2.2 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.
The SimpleSoccer server provides crisp values for the information it delivers 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. An example of input variable fuzzification is shown in Fig. 7. In this example the crisp input variable x has a degree of membership (μ) of both fuzzy sets A_1 (0.6) and A_2 (0.1).

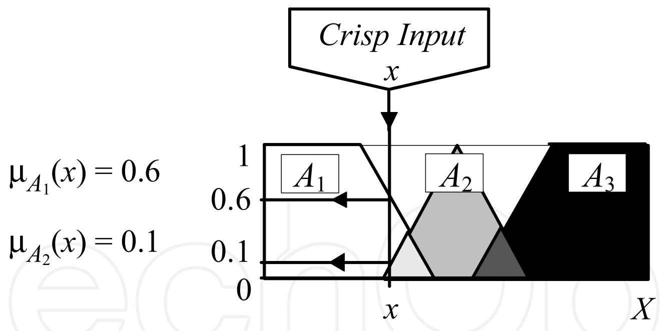


Fig. 7. Input variable fuzzification

The membership functions shown in Fig. 8 are used to associate crisp values with a degree of membership for the fuzzy sets *direction*, *distance* and *power*. The parameters for these fuzzy sets were not learned by the evolutionary process: they were fixed empirically. The initial values were set having regard to SimpleSoccer parameters and variables, and fine-tuned after minimal experimentation in the SimpleSoccer environment.

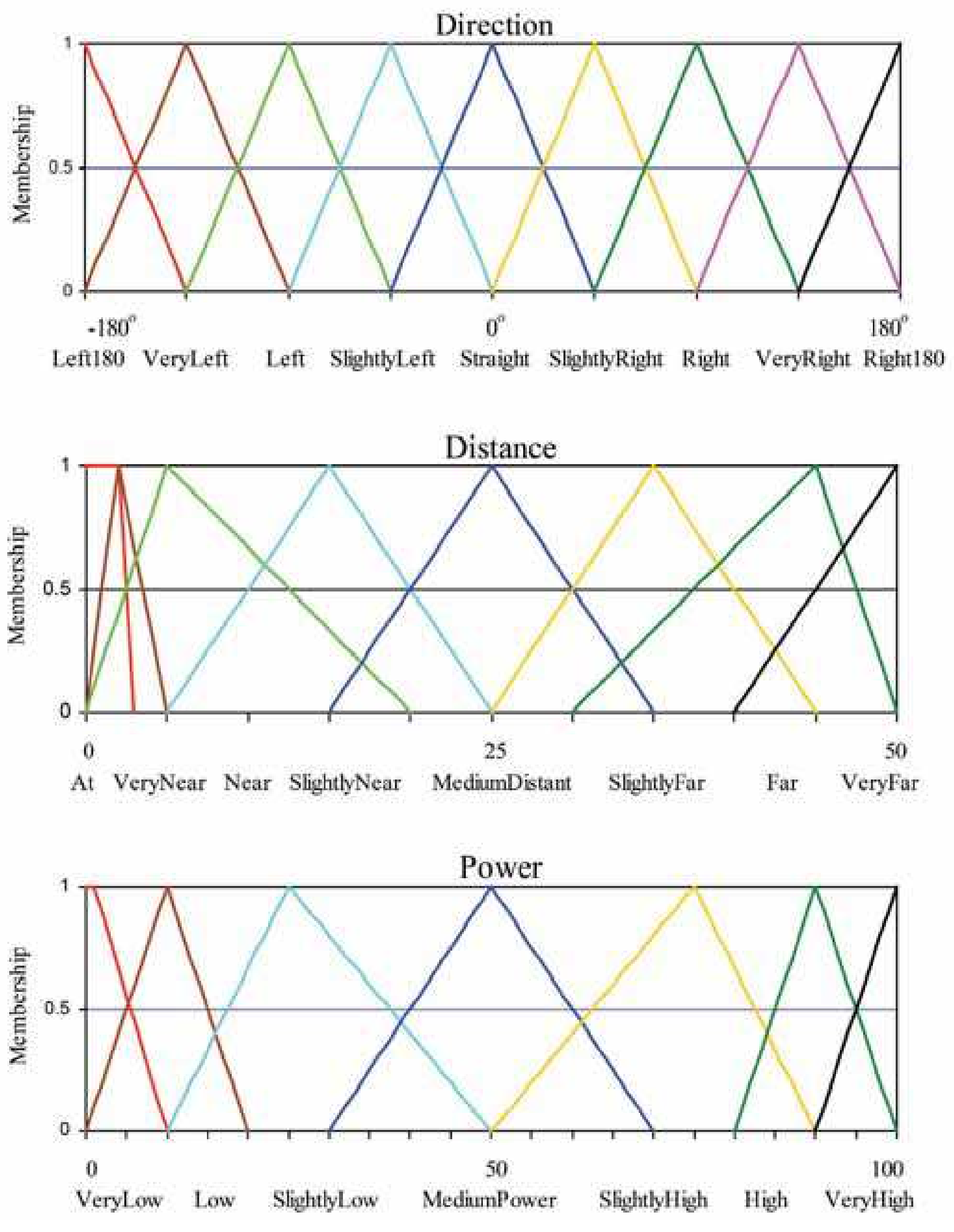


Fig. 8. Direction, distance and power fuzzy set membership

3.2.3 Implication and Aggregation

The core section of the fuzzy inference system is the part that 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.
After the input values are fuzzified they are applied to the antecedents of the fuzzy rules. For fuzzy rules with multiple antecedents, the fuzzy operators AND and OR are used as appropriate to obtain a single number that represents the result of the antecedent evaluation. This value is the degree to which the rule is true and is then applied to the consequent membership function. The evaluation of the antecedents is as follows:

- for the disjunction of rule antecedents, the fuzzy operator OR is defined by the fuzzy set operation union:

  $$\mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)]$$

- for the conjunction of rule antecedents, the fuzzy operator AND is defined by the fuzzy set operation intersection:

  $$\mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)]$$

The method implemented to correlate 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 (Fig. 9).

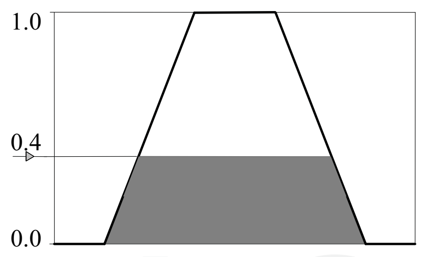


Fig. 9. Correlation minimum example

Aggregation is the process of combining the correlated fuzzy sets to produce a composite fuzzy region that represents the solution variable. The solution fuzzy region is then defuzzified if a crisp solution is required (as is the case in this work). The aggregation method used in this work is the *min/max* aggregation method. This method ORs the correlated consequent fuzzy set for each rule with the contents of the solution variable's output fuzzy region. This process takes the maximum of the consequent fuzzy set and the solution fuzzy set at each point along their mutual membership functions.
Fig. 10 is an illustration of a two-rule *Mamdani Fuzzy Inferencing System* (FIS) which implements the correlation minimum implication method and the min/max method of aggregation (Mamdami & Assilian, 1975).

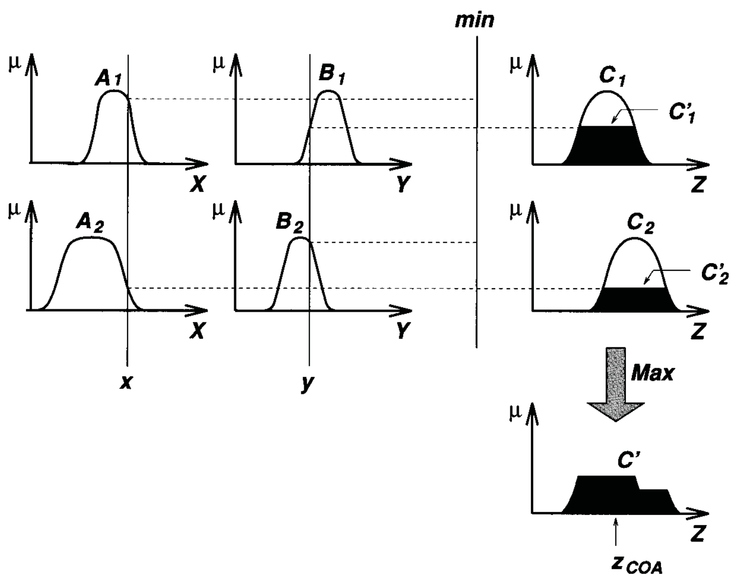


Fig. 10. 2-rule Mamdani FIS using Correlation Minimum implication and min/max aggregation. *Reproduced from (Jang et al., 1997)*

3.2.4 Defuzzification

The defuzzification method used is the *mean of maximum* method. 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. An example outcome of this computation is shown in Fig. 11.

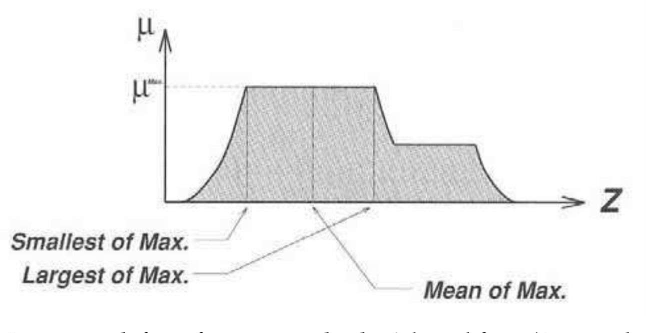


Fig. 11. Mean of Maximum defuzzification method. *Adapted from (Jang et al., 1997)*

3.2.5 Action Selection

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.

3.3 Player Learning

This work employs an evolutionary technique in the form of a messy-coded genetic algorithm to evolve the rulebase that defines the behaviour of a robot soccer player. A genetic algorithm (GA) is an adaptive search technique which maintains a population of potential solutions that evolves over time in accordance with the rules of the genetic operators implemented by the algorithm. Each member of the population has its fitness as a solution to the problem evaluated against some known criteria, and members of the population are then selected for reproduction based upon that fitness, with a new generation of potential solutions being generated from the offspring of (typically) the most fit individuals. The process of evaluation, selection, reproduction, recombination and mutation is iterated until an acceptable solution is shown (Fig. 12).

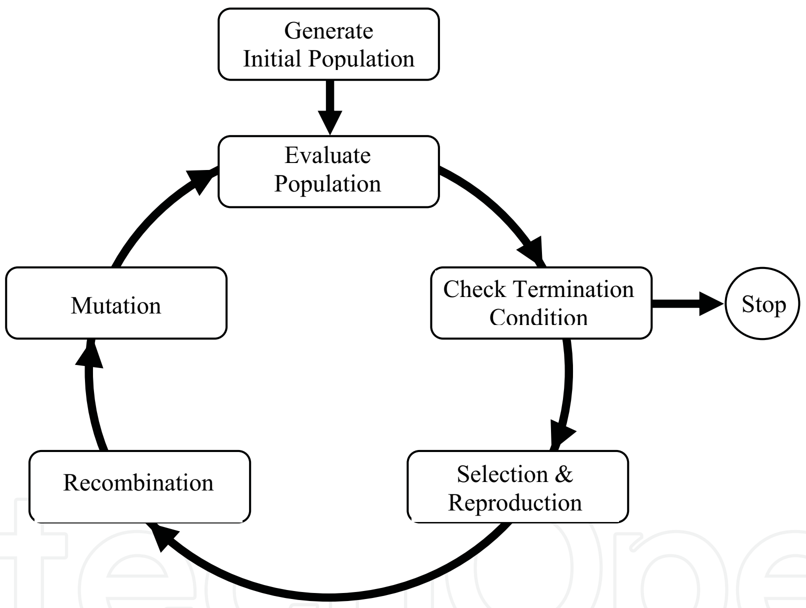


Fig. 12. The GA evolutionary cycle

The evaluation of the worth of an individual as a solution is achieved by the use of a fitness function. The objective of the fitness function is to numerically encode the performance of the individual with reference to the problem for which it is a potential solution. This is an extremely important part of the process, for without a fitness function which accurately evaluates the performance of potential solutions, the search will fail.

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. 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 implemented using the Pittsburgh approach, where each individual in the population is a complete ruleset (Smith, 1980).

3.3.1 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. The mutation operator is analogous to the mutation operator for classic genetic algorithms, whereas the classic crossover operation is replaced by a *cut-and-splice* operation (Goldberg et al., 1989). 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 flagged to distinguish it from premise genes thus 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, TOWARDBALL, TOWARDBOAL}
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 DribbleTowardGoal Low
if Ball is Far or Ball is SlightlyLeft then Run TowardBall High

In the example chromosome fragment shown in Fig. 13 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.

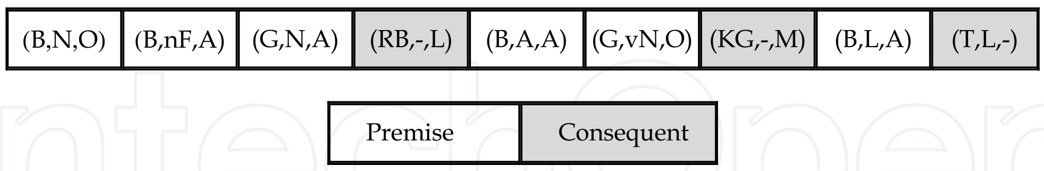
(Ball, is Left, And)	(Ball, is At, Or)	(Goal, is not Far, Or)	(Dribble, Null, Low)

Fig. 13. 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, which for this work was 64 genes. 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-and-splice and mutation operations implemented guarantee no out-of-bounds data in the resultant chromosomes, a rule is only considered invalid if it contains no premises. Any invalid rules are ignored when the ruleset is applied. A complete ruleset is considered invalid only if it contains no valid rules.

An example complete chromosome and corresponding rules are shown in Fig. 14 (with appropriate abbreviations). Some advantages of using a messy encoding in this case are:

- a ruleset is not limited to a fixed size
- a ruleset can be over specified (clauses may be duplicated)
- a ruleset can be under specified (not all genes are required to be represented)
- clauses may be arranged in any way



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. 14. Example 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.

3.3.2 Selection and Reproduction

Selection and reproduction are important processes for evolutionary algorithms. Individuals from the population are selected according to some criteria to be reproduced for the next generation. GA reproduction is essentially a cloning operation in which the individuals selected for reproduction are copied, and it is during the recombination process that the copies are mated to form new individuals. For genetic algorithms, selection and reproduction alone cannot introduce new individuals into the population: that is achieved throug the genetically-inspired *recombination* operators of crossover (*cut-and-splice* in the case of messy-coded GAs) and mutation. The purpose of selection and reproduction is to favour fitter individuals on the basis that the fitter an individual the more likely it will produce even more fit offspring.
A fitness-proportionate method of selection (Holland, 1975; Goldberg, 1989) known as "roulette wheel" selection was implemented for this work. With this method the number of times an individual is expected to be selected to reproduce is the ratio of the individual's fitness to the average fitness of the population. The implementation can be likened to a biased roulette wheel, where each individual in the current population has a slot on the roulette wheel proportional to that individual's fitness. The roulette wheel is spun once for each parent required, with the winning individuals being paired for reproduction.

3.3.3 Cut-and-Splice for Variable Length Chromosomes

Since the messy-coding implemented allows chromosomes of different lengths the crossover operation of the classic genetic algorithm needs to be modified. For messy-coded genetic algorithms the crossover operation is considered in its two distinct steps: the *cut* operation and the *splice* operation. 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. The cut-and-splice operation implemented in this work guarantees the operations will not result in out-of-bounds data in the resultant chromosomes. Fig. 15 is an example of the cut-and-splice operation for messy-coded chromosomes.

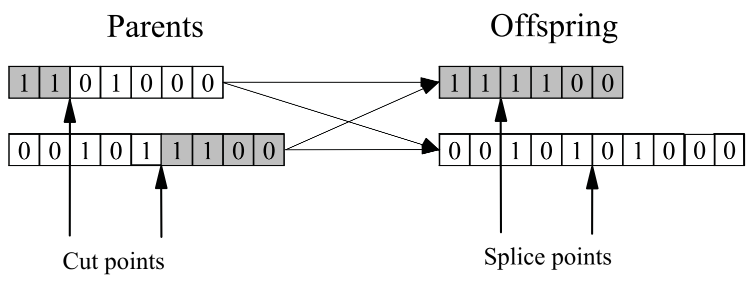


Fig. 15. Example cut-and-splice operation

3.3.4 Mutation

Mutation, which helps to maintain diversity in the population, is the arbitrary modification of individuals. The mutation scheme implemented in this work is a variation of random *single-bit* mutation, but in this case it is random *single-allele* mutation since the genes encoded in this work are integer values rather than single bits. This is a method in which a single allele is chosen randomly for modification to a random value. The mutation operator implemented guarantees mutations will not result in out-of-bounds data in the resultant chromosome.

3.4 Experimental Evaluation

A series of 20 trials was performed in order to test the viability of the fuzzy inferencing system 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 trials.

3.4.1 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. The fitness function used in these trials rewarded individuals 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 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, and
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

The actual fitness function implemented was:

$$f = \begin{cases} \begin{cases} 1.0 & ,kicks = 0 \\ 0.5 + \dfrac{dist}{2.0 \times fieldLen} & ,kicks > 0 \end{cases} & ,goals = 0 \\ \dfrac{1.0}{2.0 \times goals} & ,goals > 0 \end{cases} \quad (1)$$

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

Note that this fitness function indicates better fitness as a lower number, in effect representing the optimisation of fitness as a minimisation problem.

3.4.2 GA Control Parameters

The genetic algorithm parameters used in all 20 trials are shown in Table 3.

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

Table 3. Genetic algorithm control parameters

3.4.3 Simulator Control Parameters

The SimpleSoccer simulator parameters used in all 20 trials are shown in Table 4.

Parameter	**Value**
Field Length	61 cells
Field Width	31 cells
Goal Width	7 cells
Kickable Distance	1.0 cells
View Angle	90 degrees
View Length	5 cells
Maximum DASH distance	7.5 cells
Maximum KICK distance	15 cells
Player Skillset	All skills listed in Table 1
Default action	Hunt action 3: *Random turn*

Table 4. SimpleSoccer control parameters

3.4.3 Trial Results

For the results reported, each trial consisted of one complete execution of the genetic algorithm during which multiple simulated games of soccer were played, with the only player on the field being the player under evaluation.

For each game, 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, and the ball was placed at a randomly selected position along the centre line of the field. A game was terminated when one of the following conditions was met:

- the maximum game time of 1000 ticks expired.
- the target of 10 goals was scored, reflecting a fitness value of 0.05. 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.
- a 100 tick period of no player action occured.

A randomly generated population of players was generated and evolved over time by the genetic algorithm, with the evaluation of each member of the population being performed in the SimpleSoccer environment. The evolutionary search was stopped:

- after a specified maximum number of generations, or

- when the specified target fitness was reached by any player.

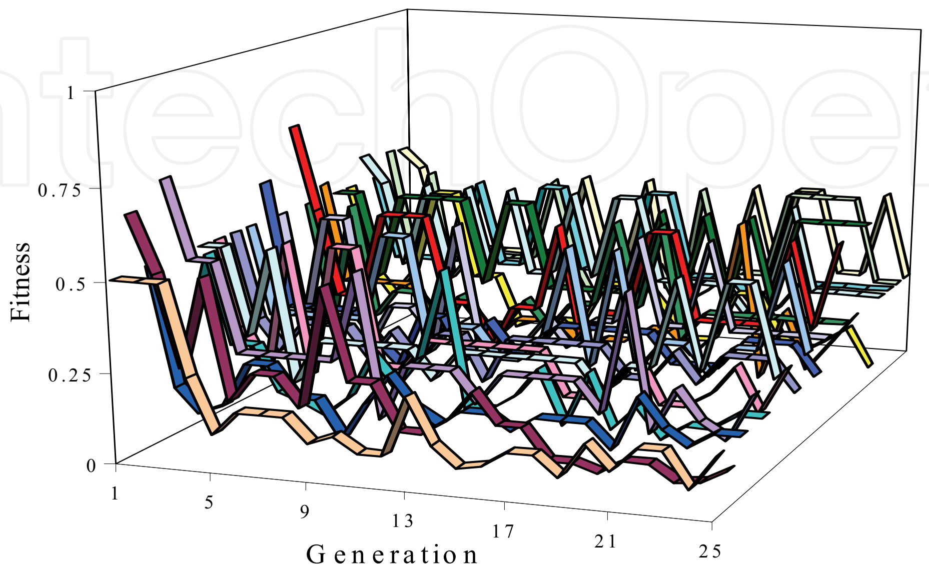


Fig. 16. SimpleSoccer: Best individual fitness

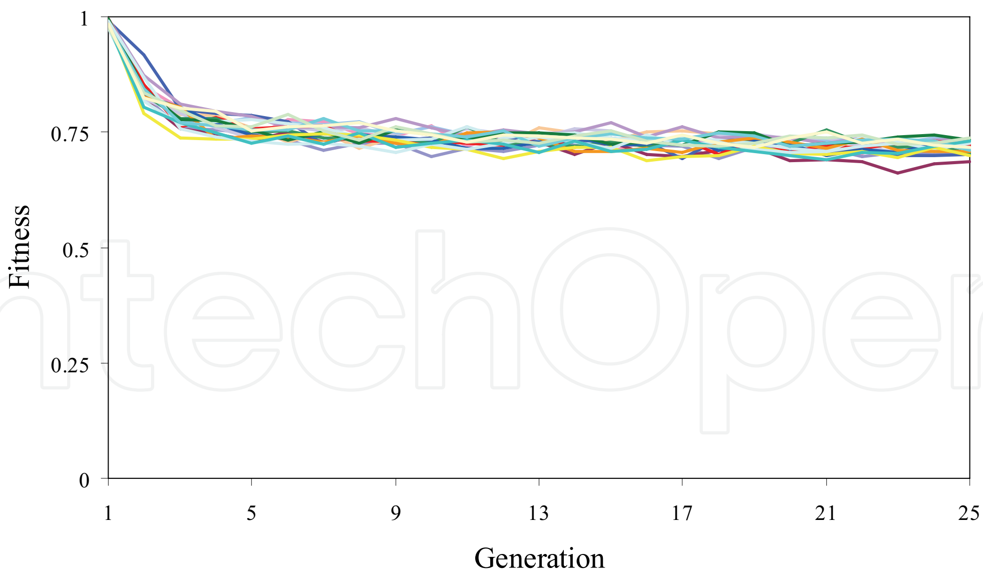


Fig. 17. SimpleSoccer: Population average fitness

Fig. 16 shows the best individual fitness from the population after each generation for each of the 20 trials. This graph shows that individuals able to score goals were found after very few generations, with some individuals being capable of scoring multiple goals in the allotted time.

Fig. 17 shows the average fitness of the population after each generation for each of the 20 trials. This graph shows that the average performance of the population improves steadily and plateaus, but while individual players do score goals, the population does not approach an average fitness of 0.5, or goal-scoring behaviour.

These results show that the method presented is capable of training a simulated robot soccer player to develop goal-scoring behaviour. The method uses a genetic algorithm to evolve the fuzzy rulesets that drive the soccer player's behaviour, with the evolutionary process being allowed to run for a maximum of only 25 generations which, while sufficient to demonstrate the effectiveness of the method, is probably not sufficient to evolve players with robust, consistent goal-scoring behaviour.

3.4.4 SimpleSoccer as a Model for RoboCupSoccer

To gauge the effectiveness of the SimpleSoccer environment as a model for RoboCupSoccer a further series of 20 trials was performed in the RoboCupSoccer environment. Similar simulator and GA control parameters were used. Game times for the RoboCupSoccer environment were limited to 120 seconds (real time) rather than a number of program ticks. The results of these trials are shown below.

Fig. 18 shows the best individual fitness from the population after each generation for each of the 20 trials. It is evident from a comparison of Fig. 16 and Fig. 18 that while good individuals are found quickly in both environments, the algorithm seems to produce more consistent behaviour in the RoboCupSoccer environment. These data show 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. 19 shows the average fitness of the population after each generation for each of the 20 trials. This graph shows 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. 19 also shows that the average fitness curves for the RoboCupSoccer trials are less tightly clustered than those of the SimpleSoccer trials (see Fig. 17), probably reflecting the more stochastic nature of the environment.

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 is not intended to be, there is sufficient similarity in the results to indicate that the SimpleSoccer environment is a good simplified model of the RoboCupSoccer environment.

Much of the motivation for creating the SimpleSoccer environment was the prohibitive time to train players in the real-time RoboCupSoccer environment and the need to reduce that training time to improve the efficiency and effectiveness of machine learning methods for training simulated robot soccer players. Table 5 shows the average number of seconds of real time for a single fitness evaluation in each of the environments used to evolve players for robot soccer, and from the data shown in Table 5 it is evident that the goal of creating a more efficient environment for machine learning techniques has been achieved.

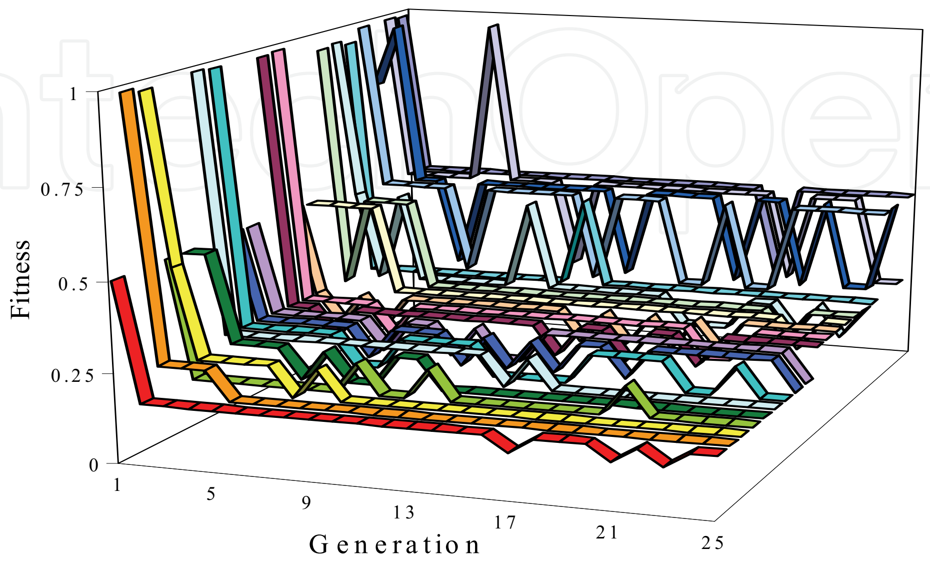


Fig. 18. RoboCuSoccer: Best individual fitness

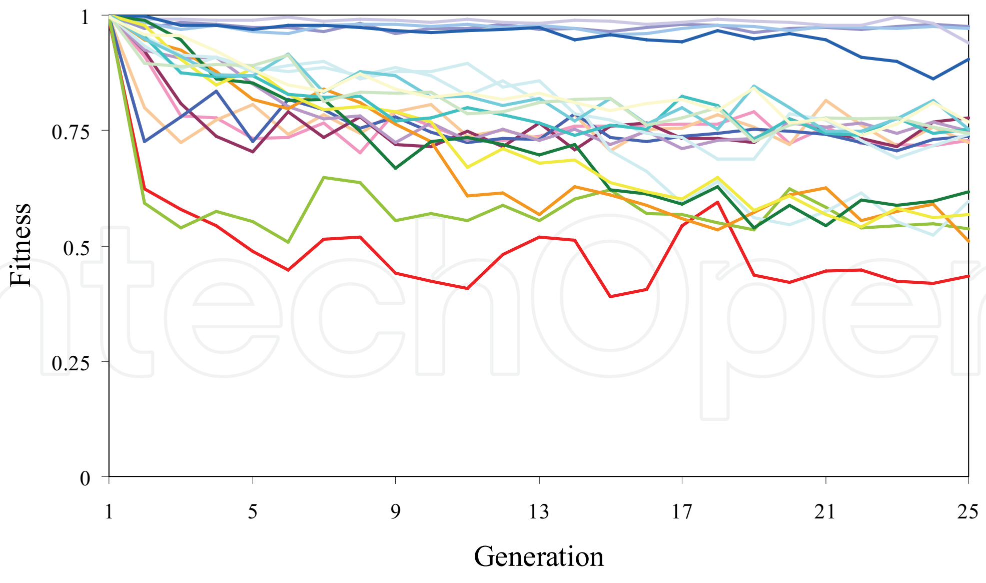


Fig. 19. RoboCupSoccer: Population average fitness

The RoboCupSoccer simulator used throughout this work was version 7.10, compiled and executed on an HP 9000/777 workstation running version 11.0 of the HP-UX operating system. The SimpleSoccer simulator was originally developed on an HP9000/777 workstation running HP-UX version 11.0, and was later ported to an Intel Pentium-based PC running Windows XP. Evaluation times are quoted for each of those systems. No trials using the RoboCupSoccer simulator were performed on the PC. Note that although the SimpleSoccer evaluation time is two orders of magnitude smaller on the PC, RoboCupSoccer evaluation times would not enjoy the same improvement if executed on the PC – the RoboCupSoccer evaluation times are constrained by the real-time nature of the simulator, and the training game times were 60 seconds. Any benefit from running the RoboCupSoccer simulations on faster hardware would be evident in the few seconds of overhead time only, and would not significantly reduce the evaluation time.

Simulator	Platform	Seconds/Evaluation
RoboCupSoccer	HP 9000/777 workstation. 120MHz PA-7200 CPU, 256MB RAM, HP-UX 11.0 Operating System	70.65
SimpleSoccer	HP 9000/777 workstation. 120MHz PA-7200 CPU, 256MB RAM, HP-UX 11.0 Operating System	10.20
SimpleSoccer	Compaq PC. 1.6GHz Pentium M CPU, 512MB RAM, Windows XP Operating System	0.112

Table 5. Evaluation times

4. Summary and Discussion

The goal of this work was to create an environment with similar complexity and dynamics to the RoboCupSoccer environment, but with reduced uncertainty, both in player perception and in the player's interaction with the environment. The motivation was to create an environment in which the training times of machine learning techniques would be reduced sufficiently so as to improve the viability of such techniques, and to show that players could be trained in this environment to display reasonable goal-scoring behaviour. The SimpleSoccer environment was developed for this purpose, and through some sample experiments it was shown that the SimpleSoccer environment does aid in the reduction of training times for some machine learning techniques.

The implementation of a messy-coded genetic algorithm which successfully evolves the ruleset for a fuzzy logic-based simulated robot soccer player was described. 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.

Furthermore, tests in which the initial population for RoboCupSoccer was seeded with players evolved in the SimpleSoccer environment suggest that there is significant benefit in using the SimpleSoccer environment as an heuristic to generate high quality initial solutions for the RoboCupSoccer environment (Riley, 2003; Riley, 2005). The evolution of players displaying reasonable goal-scoring behaviour is achievable in the SimpleSoccer environment in a fraction of the time it would take in the RoboCupSoccer environment, and only a few generations are required in RoboCupSoccer to refine the behaviours evolved in the SimpleSoccer environment. High-level strategies learned in the more certain SimpleSoccer environment are directly transferrable to the RoboCup environment, and when used as the starting point for further learning can help to reduce the training time in the RoboCup environment.

5. References

Bajurnow, A. & Ciesielski, V. (2004). Layered Learning for Evolving Goal Scoring Behaviour in Soccer Players, *Proceedings of the 2004 Congress on Evolutionary Computation,* Vol. 2, pp. 1828-1835, Portland OR, June 2004, G. Greenwood (Ed.), IEEE, Piscataway NJ.

Balch, T. (1995). The Ascii Robot Soccer Home Page. *http://www.cs.cmu.edu/~trb/soccer/*, 1995.

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

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

Ciesielski, V. & Lai, S.Y. (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*, pp. 70-78, Dunedin, New Zealand, November 2001, Wigham, P.; Richards, K.; McKay, B.; Gen, M.; Tujimura, Y. & Namatame, A. (Eds.).

Ciesielski, V., Mawhinney, D. & Wilson, P. (2001). Genetic Programming for Robot Soccer, *Proceedings of the RoboCup 2001 Symposium, Lecture Notes in Artificial Intelligence*, pp. 319-324, Seattle WA, August 2001, Birk, A.; Coradeschi, S. & Tadokoro, S. (Eds.), Springer NY.

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

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, ISBN 0-201-15767-5.

Goldberg, D., Korb, B. & Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems, vol. 3*, 1989, pp. 493-530.

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

Jang, J.-S.; Sun, C.-T. & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*, Prentice Hall, ISBN 0-13-261066-3, Upper Saddle River NJ.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa, E. (1995). RoboCup: The Robot World Cup Initiative, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Workshop on Entertainment and AI/ALife*, Montréal, Québec, Canada, August 1995, Morgan Kaufmann, San Francisco CA.

Kitano, H.; Tambe, M.; Stone, P.; Veloso, M.; Coradeschi, S.; Osawa, E.; Matsubara, H.; Noda, I. & Asada, M. (1997). The RoboCup Synthetic Agent Challenge 97, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 24-29, Nagoya, Japan, August 1997, Morgan Kaufmann, San Francisco CA.

Lima, P.; Custódio, L.; Akin, L.; Jacoff, A.; Kraezschmar, G.; Ng, B.K.; Obst, O.; Röfer, T.; Takahashi, Y. & Zhou, C. (2005). RoboCup 2004 Competitions and Symposium: A Small Kick for Robots, a Giant Score for Science. *AI Magazine* 26(2), 2005, pp. 36-61.

Luke, S. (1998a). Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Madison WI, July 1998, Koza, J.R.; Banzhaf, W.; Chellapilla, K.; Deb, K.; Dorigo, M.; Fogel, D.B.; Garzon, M.H.; Goldberg, D.E.; Iba, H. & Riolo, R.L. (Eds.), Morgan Kaufmann, San Francisco CA.

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

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

Riedmiller, M.; Merke, A.; Meier, D.; Hoffman, A.; Sinner, A.; Thate, O. & Ehrmann, R. (2001). Karlsruhe Brainstormers – a Reinforcement Learning Approach to Robotic Soccer. In: *RoboCup-2000: Robot Soccer World Cup IV*, Stone, P.; Balch, T. & Kraetszchmar, G. (Eds.), Springer Verlag, Berlin.

Riedmiller, M.; Gabel, T.; Knabe, J. & Strasdat, H. (2005). Brainstormers 2D - Team Description 2005. In: *RoboCup 2005: Robot Soccer World Cup IX.*, Bredenfeld, A.; Jacoff, A.; Noda, I. & Takahashi, Y. (Eds.), Springer Verlag, Berlin.

Riley, J. (2003). The SimpleSoccer Machine Learning Environment, *Proceedings of the First Asia-Pacific Workshop on Genetic Programming*, pp. 24-30, Canberra, Australia, December 2003, Cho, S-B.; Nguen, H.X. & Shan, Y. (Eds.).

Riley, J. & Ciesielski, V. (2004). Evolution of fuzzy rule based controllers for dynamic environments, In: *Recent Advances in Simulated Evolution and Learning, volume 2 of Advances in Natural Computation*, Tan, K.C.; Lim, M.H; Yao, X. & Wang, L. (Eds.), chapter 23, pp. 426-445. World Scientific, ISBN 981-238-952-0, Singapore.

Riley, J. (2005). Evolving Fuzzy Rules for Goal-Scoring Behaviour in a Robot Soccer Environment, *PhD Thesis*, RMIT University: Melbourne, Australia.

Smith, S. (1980). A Learning System Based on Genetic Adaptive Algorithms, *PhD Thesis,* Department of Computer Science, University of Pittsburgh: Pittsburgh PA.

Stone, P. & Sutton, R. (2001). Scaling Reinforcement Learning Toward RoboCup Soccer, *Proceedings of the Eighteenth International Conference on Machine Learning*, Williamstown MA, July 2001, Brodley, C.E. & Danyluk, A.P. (Eds.), Morgan Kaufmann, San Francisco CA.

Stone, P. & Veloso, M. (1999). Team-partitioned, Opaque-transition Reinforcement Learning, *Proceedings of the Third International Conference on Autonomous Agents*, pp. 206-212, ISBN 1-58113-066-X, Seattle WA, May 1999, Etzioni, O.; Müller, J. & Bradshaw, J. (Eds.), ACM Press, NY.

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

Zadeh, L. (1965). Fuzzy Sets. *Journal of Information and Control*, vol. 8, 1965, pp. 338-353.