

# Genetic Programming based Optimisation of Soccer Agents

Christopher Lazarus and Huosheng Hu

*Department of Computing and Electronic Systems  
University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK  
Email: [clazar@essex.ac.uk](mailto:clazar@essex.ac.uk), [hhu@essex.ac.uk](mailto:hhu@essex.ac.uk)*

---

## Abstract

This paper describes our approach on evolving a robot controller to generate a set of goalkeeper behaviours in the RoboCup domain based on the genetic programming (GP) technique. The goalkeeper agent is a software construct that operates in a simulated environment provided by the Soccer Server. Our research is to determine the conditions that need to be met for an evolved goalkeeper agent to perform adequately in a defensive situation. A framework is developed to test the agent's performance in the simulator. The impact of the sensors information that operates as the input to our GP implementation is addressed. The framework by incorporating it within two established soccer agents is evaluated, and then the offline performance of each agent is tested. The experimental results show that our approach is able to produce a robust robot controller and the framework is flexible enough to be readily incorporated into other existing soccer agent frameworks.

**Keywords:** Evolutionary Robotics, Genetic Programming, Simulated Soccer.

---

## 1. Introduction

RoboCup competition is an international robotics event, including several different leagues [1], such as simulation, small-sized, medium-sized, Sony AIBO, humanoid and RoboCup rescue leagues. This paper reports on work done for soccer agents in the simulation league.

Programming a team of robot to perform soccer competition tasks is nontrivial task. Our efforts focus on evolving behaviours for a goalkeeper agent by adopting the Genetic Programming (GP) technique and the focus of the research is on the design of a set of specifications for the desired behaviours that are able to emerge due to evolutionary pressure set out by the designer [2]. The work presented here is based on our previous work on evolving navigation and obstacle avoidance behaviours based on the optimisation of a single objective function [3].

However, our current work involves the optimisation of multiple objectives [4].

There have been many attempts by researchers to develop and improve ways on robot evolutions. Their solutions can be similar if not better than hand-coded solution, i.e. the traditional robot navigation solution. These works include evolution of both simulated and real robots. RoboCup is a rich domain for the exploration of these techniques, and gives us a platform where we can compare the performance of their techniques in same kind of environments. Given that more than one robot is involved in a soccer match, RoboCup also enables us to explore multi-robot cooperation algorithms. Several teams have used GP in evolving their soccer teams [2], and a comparison of these works can be seen in [5].

Competition between the GP evolved teams and hand-coded teams have been possible due to the open nature of the RoboCup soccer competition [1, 6].

Performances of GP evolved teams have usually been average. This paper shows that one of the reasons is not due to the GP implementation itself but due to other factors, such as architectural and other auxiliary functions. These functions could be the sensor interpretation routines, agent localisation and subsequently object detection routines.

The rest of this paper is organised as follows. Section 2 presents the methodology and a conceptual architecture developed. Section 3 describes the implementation of our agent architecture. The experimental results are presented and analysed in Section 4. Finally, a brief conclusion and further work are given in Section 5.

## 2. Methodology

The focus of our research is to find out the effects of using multiple objective fitness functions in our GP implementation. To accomplish this, we have devised a training ground situation to test the evolved behaviours of our goalkeeper. The architecture consists of a soccer server, a coach trainer agent, a hand-coded attacker agent and an evolving goalkeeper agent, as shown in Fig. 1.

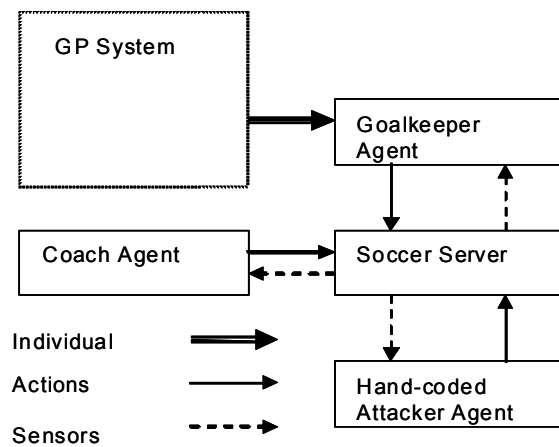


Fig. 1 System Configuration

The soccer server [6] accepts action commands every 100ms and sends back sensory information and runs the simulation. The coach agent performs the synchronisation among all the agents by relaying messages to let them know when a trial begins and ends. It also resets the positions of agents and ball. The hand-coded attacker agent is a simple agent that kicks the ball towards the centre of the goal. The goalkeeper agent is the one that has the capability to evolve based on the GP system that generates its control program.

Our control experiment uses a random goalkeeper agent that sends randomly generated commands to the soccer server instead of being generated by GP. Its fitness is evaluated during the experiment. Both random and evolving goalkeeper experiments are carried out to collect the data, and their performance is compared to measure the effectiveness of the GP implementation.

In designing our functions and terminals, we noted that if the performance of the GP algorithm is not quite what we expect then more complex functions and terminals may be added to the gene pool. One of the attractions of using GP is that detailed knowledge of the problem domain is not essential. The focus of the problem shifts from the careful design of the behaviours to the designing of specifications that encourages emergence of the desired behaviours [2]. In this case, specifications can be equated as the fitness criterion of each individual within the GP population.

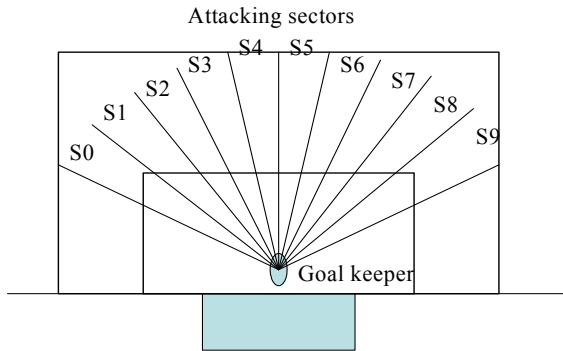
Multiple fitness functions are used in our current work. A goalkeeper has a single overriding objective, which is to stop any goal from being scored into its own net. To achieve this, there is a set of other objectives that need to be met. The first objective concerns navigation, another is localisation (position must be kept near the centre of the goal), another is blocking the ball trajectory, the ability to kick the ball out of the goal area and lastly to catch the ball when possible. Therefore the aim of our optimisation is to transform a vector of objectives to a scalar version and then proceed with the assignment of fitness values to each individual.

## 3. Implementation

Our GP system adopts lil-gp [7] and the client networking library libclient [8]. A coach agent is developed to provide synchronisation and monitoring during the experiment. The coach agent resets the experiment setting before the start of each trial case. This involves placing the goalkeeper, ball and attacker in their respective starting positions. Once in place, the ball is kicked by the attacker in a straight line directly towards the goal. There are 10 different ball placements for each evaluation (see Fig. 2). Each individual in the GP population representing the goalkeeper is tested on its performance in defending the goal from 10 different ball-starting positions.

The attacking area in front of the goal is divided equally into 10 sectors. At the start of each experiment, the 10 starting positions for the attacker are randomly generated within each of the 10

attacking sectors. Each attacker starting position is then tested sequentially in a clockwise order for each individual in the population. This variation in the attacker's starting position should enable a robust controller to be evolved. The evolved behaviours are not biased towards a particular attacking direction. We evaluated each individual by parsing the expression trees and sending a command every 10ms to the soccer server, instead of normal 100ms [6] to speed up the simulation, and only 1 command is



allowed per 10ms.

Fig. 2 A segment of the soccer field showing a goalkeeper at location (50, 0)

### 3.1 Evaluation time

The length of time steps allowed for an attacker to attack and score a goal has a direct impact on the length of time each evaluation takes. For the purpose of the experiments each evaluation takes 40 time steps. Each time step corresponds to a set period of 10ms. We allow the attacker 40 time steps per trial to give it enough time to try to score a goal.

### 3.2 Fitness measure

The fitness evaluation we adopted is called the Weighted Sum Approach

$$f = \sum_{i=1}^n w_i f_i \quad (0.0 \leq f \leq 1.0) \quad (1)$$

where  $f$  denotes the overall fitness of an individual,  $n$  denotes the number of fitness measures,  $w_i$  denotes the weights and  $f_i$  denotes the objectives to be optimised [4].

### 3.2 Fitness cases

There are 5 fitness cases that influence the evolution of desired behaviours: (i)  $f_1$  = Ball saving; (ii)  $f_2$  = Localisation; (iii)  $f_3$  = Ball locate; (iv)  $f_4$  =

Movements; (v)  $f_5$  = Positioning.

The rationales for the choice of the fitness measures in Table 1 are that we want the evolved goalkeeper to be actively moving towards the ball and intercepting it. The movements and positioning fitness cases reflect this desire. Fitness  $f_4$  and  $f_5$  might seem to overlap in functionality but without fitness  $f_4$ , the movement of the ball instead of the goalkeeper might be the cause for the proximity of the goalkeeper to the ball. Therefore, we encourage the selection of individuals that are active so that there are cases where the proximity of a goalkeeper to a ball is caused by the goalkeeper's own movements. We gave fitness  $f_4$  a higher weight because of similar reasons. Nevertheless, fitness  $f_4$  and fitness  $f_5$  are ineffective if they don't result in a ball saving event. So we give a higher weight for behaviours that resulted in a ball saving event.

Table 1 GP Parameter Settings

Max Generation	50
Population	
Type	Generational
Size	100
Initialisation	Half and Half
Selection Type	Tournament with size 7
Genetic Operators	
Crossover rate	0.9
Mutation Rate	0.1
Function Set	iflte, add, sub, mult, div, sin, cos, catch, dash, kick, turn, put, ADF0
Terminal Set	play_dir, play_pos, dist_to_ball, dir_to_ball, dir_to_ogool, dist_to_ogool, dist_to_opp, dist_to_opp, rand, get0, get1, ARG0, ARG1

Fitness case 1 is given by the equation

$$f_1 = 1 - \frac{g}{t_{\max}} \quad (2)$$

where  $g$  is the goal scored by an attacker and  $t_{\max}$  is the maximum trial case per evaluation

Fitness  $f_2$  is used to measure the goalkeeper's ability to estimate its own location within the playing field. If both initial and ending position of the goalkeeper within each trial is estimated then localisation is deemed as successful and fitness  $f_2$  is allocated.

Fitness  $f_3$  measures the goalkeeper's ability to locate the ball by checking whether the goalkeeper is able to estimate the distance between itself and the ball at the start and end of a trial. The goalkeeper is only able to do this if the ball is in its field of vision. Therefore this fitness can also be equated with the ability of the goalkeeper in turning towards the ball.

Fitness  $f_4$  is as follows

$$f_4 \quad \text{if } p_s \text{ is equal to } p_e \text{ then} \\ \quad \quad \quad \text{no movement is detected so } f_2 = 0 \\ \quad \quad \quad \text{else movement detected so } f_2 = 1$$

where  $p_s$  is the position of goalkeeper at the start of an evaluation, and  $p_e$  is the position of goalkeeper at the end of an evaluation

Fitness  $f_5$  uses a logarithmic scale function of the distance between the goalkeeper and the ball at the start of an evaluation and also at the end of an evaluation, ensuring that the relationship between the distance between the goalkeeper and the ball is logarithmic. This ensures that the fitness decreases following a logarithmic curve as the distance between the ball and the goalkeeper becomes larger.

$$f_5 \quad \text{if } \left(1 - \frac{\log(d_e)}{\log(d_s)}\right) \text{ is less than 0 then} \\ \quad \quad \quad \text{set } f_3 \text{ to 0} \\ \quad \quad \quad \text{else set } f_3 \text{ to } 1 - \left(\frac{\log(d_e)}{\log(d_s)}\right)$$

where  $d_s$  is the distance between goalkeeper and ball at the start of an evaluation greater than 1.  $d_e$  is the distance between goalkeeper and ball at the end of an evaluation greater than 1.

### 3.3 Termination-criteria

Termination of the GP run is set to the number of maximum generation predetermined at the beginning of the GP run.

## 4. Experimental Results and Analysis

Table 2 presents the experiments that have been carried out in this research and the extended experiment results are shown in Table 4. A summary of noise effects on 3 agents is summarised in Fig. 4.

The first two experiments, where the goalkeeper's behaviour was generated randomly, are

used as a benchmark for subsequent experiments. The starting position for the goalkeeper is set at location 50, 0 on the playing field. Three sets of weights were used during the experiments.

Table 2 Experiment runs

Experiments	Behaviour generated by	Memory	ADF	Weights
1	Random	N/A	N/A	Equal
2	Random	N/A	N/A	Lexicographic
3	GP	Yes	No	Equal
4	GP	Yes	No	Lexicographic
5	GP	No	Yes	Equal
6	GP	No	Yes	Lexicographic
7	GP	Yes	Yes	Equal
8	GP	Yes	Yes	Lexicographic
9	GP	Yes	Yes	SQRT (Lex.)

As shown in Table 3, one set of weights is designed to give equal importance to all of the 5 fitness measures. The second set of weights is designed to bias the importance of the fitness measures in a lexicographical manner in terms of fitness. The third set of weights is derived from normalised square roots of the lexicographic weights. The third set of weights has a smoother distribution compared to the second set. This is shown in Fig. 3. This allows for the fitness measures to be more uniformly distributed across the fitness range but would still be skewed according to the fitness measures ranking.

All of the experiments based on lexicographic weights have lower overall fitness when compared with runs that are based on equal weights. Lexicographic weights biased the fitness measurement too much towards fitness 1 which is ball saving. This results in GP not being able to capitalise on the contributions made by the other fitness measures. This also shows that even though ball saving is the main objective; it is not enough on its own to allow GP to explore potential solutions. By specifying that all of the fitness measures have equal weights, we see that the fitness is distinctively higher compared to using lexicographic weights. We can also see this assertion by comparing both of the random runs. The random run that uses equal weights has higher fitness compared to the random run that uses lexicographic weights.

Experiments 2, 3, 4 and 5 were run to observe whether the GP performance would be enhanced by the addition of either memory or ADF. The results

show that the GP with ADF has higher fitness than the GP with memory when equal weights are used. In contrast, the GP with memory has higher fitness than the GP with ADF when lexicographic weights are used. Looking at both the equal weights and the lexicographic weights groups, we can see that in both groups, the highest performing GP is the one that implements both memory and ADF in their architecture. The addition of either ADF or memory in isolation does improve the performance of the GP but its performance increases further if both of them are implemented together.

Table 3 Weights used during the experiments

Weights	Equal	Lexicographic	Square root (Lexicographic)
1	0.2	0.9	0.685942
2	0.2	0.09	0.216914
3	0.2	0.009	0.068594
4	0.2	0.0009	0.021691
5	0.2	0.00009	0.006859

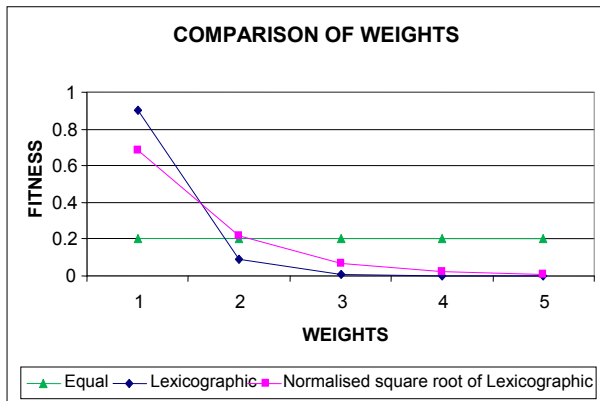


Fig. 3 Fitness distributions when lexicographic weights are normalized

When we compared performances of best of run individuals in contrast to populations, we start to see that the distinction caused by different weights set becomes less obvious. Experiments that use the lexicographic weights, has best of run individual nearing the performances from the experiments that uses equal weights. An additional experiment (experiment 9) that uses the normalised lexicographic weights was run to observe the effects of smoothing the distribution of the fitness measures. The population fitness for this experiment shows a marked improvement. However its population fitness is still below the population fitness for experiments that uses equal weights.

The next set of experiments was conducted to show whether our GP implementation has any effect when conducted with other agent implementations that has better sensor interpretation routines. For this work, we decided to test the GP framework using our own team, Essex Wizards [9] and a competitor, UvA Trilearn [10]. Our GP framework is incorporated into these teams' agent framework without much change due to the flexibility of the lil-gp [7] architecture.

In terms of mean population fitness, we see that the fittest population results from the agent using the UvA Trilearn framework. Again the lowest mean population fitness is from the libscient agent. This shows that other features of a soccer agent such as synchronisation, localisation and sensor interpretation algorithms has a direct impact on the fitness of the evolved behaviour tree.

Each of these evolved solutions is then tested using the same experiment scenarios. In these experiments the agent's behaviour trees are no longer being evolved. The best evolved behaviour tree at the end of the GP run for each agent is loaded at the start of the experiment. The agent then acts using this behaviour tree during the course of its run. An agent is assessed by performing 500 trials per run. Noise resulting from incorrectly kicked balls is discarded from evaluation. Fig. 4 shows a comparison of noise detected during the experiments. We see that better optimised agent architecture results in lower noises.

Our agent, who uses the libscient library, is not as optimised as the other two agents. We can see that the most noise is produced by the libscient agent with the randomly generated behaviour tree (see 2-1 in Fig. 4). This is reduced when the same agent uses a hand-coded behaviour tree (see 2-4 in Fig. 4). The least noise is observed in the evolved behaviour tree (see 2-7 in Fig. 4). GP has managed to optimise the behaviour tree by adapting to the level of noise in the simulation. This same task can be achieved by making sure that the synchronisation algorithm between the agent and the server are optimised.

An agent's performance is calculated based on the number of balls saved (caught and kicked by the goalkeeper agent). Table 4 and Fig. 4 show a comparison of performance between three different agent implementations. Each trial involves an attacker agent kicking a ball towards the goal. The scenario is similar to that shown in Fig. 2. We observe that the top two performances come from both the Essex Wizards and UvA Trilearn hand-coded agents. This result shows that their agents are highly optimised. Our GP evolved behaviour trees

produced disappointingly low performances when compared to the two hand-coded agents. This is to be expected since the two agents have performed exceptionally during past RoboCup competition.

Another reason is that the two hand-coded agents make use of high-level behaviours which are built on top of a set of lower level behaviours. Our GP agent is only given a set of low level commands (see Table 1). Our own hand-coded agent based on the libslcient library was not designed to be in the RoboCup competition hence it is not optimised. It is designed to be a control for comparison of low level behaviours. The hand-coded agent based on libslcient uses behaviours based only on low level commands to generate its behaviour. The libslcient evolved agent's performance is slightly better than the libslcient hand-coded agent's performance (see 2-4 vs. 2-7 in Table 4).

Table 4 Experiments (Performance of Agents) using best behaviour tree evolved during previous GP run

	Libslcient	Essex Wizards1999	UvATrilearn 2001
Random	(2-1) 7.72%	(2-2) 7.65%	(2-3) 4.61%
Hand-coded	(2-4) 5.14%	(2-5) 93.93%	(2-6) 79.4%
Evolved	(2-7) 5.8%	(2-8) 4.62	(2-9) 4.21

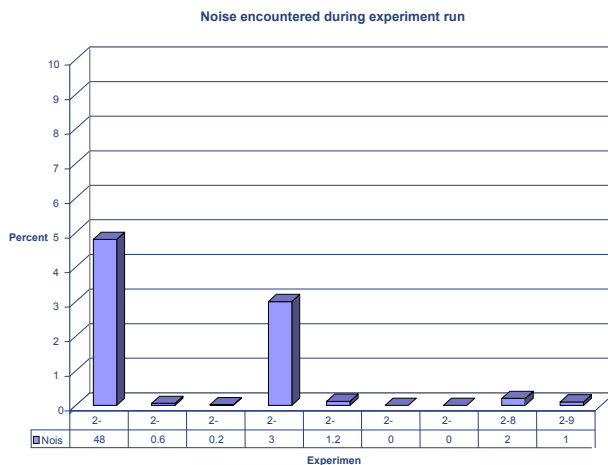


Fig. 4 Comparison of noise effects on the 3 agents

## 5. Conclusions and Future Work

This paper has shown that the selection of weights in the Weighted Sum Approach is important to the performance of GP implementation although most weight selection problems are solved heuristically. Considering the simulation results, each of the fitness

measure is equally important. Our attempts to bias the fitness measures by adjusting the weights shows lower overall population fitness. Nevertheless, the fittest individual within the population where the weights are not equally distributed still manages to be fitness-competitive to the fittest individual using equal weights. The addition of memory and ADF as part of an agent's memory and an individual's program tree respectively for the duration of the run increases the GP's performance. This corresponds with the increase in difficulty of the task that the goalkeeper agent is given.

The overall results show that although GP manages to evolve a good set of behaviours based on low level commands, which are still worse than behaviours optimised by human experience. Our future work is to investigate the way to improve the performance of the evolved goalkeeper further.

## References

- [1] H. Kitano, M. Tambe, P. Stone, M. Veloso, et al., The RoboCup Synthetic Agent Challenge, 97. Proc. of Int. Joint Conf. on Artificial Intelligence, 1997.
- [2] D. Andre and A. Teller, "Evolving Team Darwin United," in *RoboCup-98: Robot Soccer World Cup II*, M. Asada, Ed.: Springer-Verlag, 1998.
- [3] C. Lazarus and H. Hu, "Using Genetic Programming to Evolve Robot Behaviours," Proc. of the 3<sup>rd</sup> British workshop on Towards Intelligent Mobile Robots, Manchester, 2001.
- [4] C. M. Fonseca, P. J. Fleming, "Multi-objective Optimisation," in *Evolutionary Computations 2*, T. Bäck, D. Fogel, B, and Z. Michalewicz, Eds. UK: IOP Publishing Ltd., 2000.
- [5] J. Lundberg, "Survey over Genetic Programming Approaches to RoboCup", <http://www.dsv.su.se/~johank/courses/int4/robocup/papers/1999/int7/lundberg.pdf>, 2003.
- [6] I. Noda, "Soccer Server System", <http://sserver.sourceforge.net/NEW/>, 2003.
- [7] B. Punch and D. Zongker, "lil-gp Genetic Programming System", <http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html>, 1998.
- [8] I. Noda, "libslcient", <http://www.isi.edu/soar/galk/RoboCup/Libs/libslcient-4.01.tar.gz>, 1997.
- [9] H. Hu, K. Kostiadis, M. Hunter, M. Seabrook, "Essex Wizards '99 Team Description," *RoboCup-99: Robot Soccer World Cup*, 2000.
- [10] R. D. Boer, J. R. Kok, and F. C. A. Groen, "UvA Trilearn 2001 Team Description," *RoboCup 2001: Robot Soccer World Cup V*, 2001.