Res. Lett. Inf. Math. Sci., (2002) **3**, 15–23
Available online at http://www.massey.ac.nz/~wwiims/research/letters/

Evolution of a Robotic Soccer Player

Matthew Walker
I.I.M.S., Massey University Albany Campus, Auckland, New Zealand
m.g.walker@massey.ac.nz

Abstract
Robotic soccer is a complex domain where, rather than hand-coding computer programs to control the players, it is possible to create them through evolutionary methods. This has been successfully done before by using genetic programming with high-level genes. Such an approach is, however, limiting. This work attempts to reduce that limit by evolving control programs using genetic programming with low-level nodes.

1 Introduction

There's nothing like the thrill of watching expensive bits of machinery almost flying around a confined area, no longer under your direct control, but nonetheless your responsibility. Robot soccer provides such a research platform.

The robots move around the field with wheels rather than legs, and the field is not a standard grass paddock but rather a scaled-down version about the size of a ping pong table and with a wooden surface. Controlled by a master computer with a bird's eye view of the field, for the duration of the game, the robots may not be assisted by humans. Figure 1 shows a simulation of the game.

"Without human assistance" is what makes this task difficult, for programmers must develop algorithms to play every aspect of soccer. Programming soccer-playing is complex because there are so many levels. Firstly, the robots need to be told how to chase, kick, and stop the ball. They then need to be given jobs such as "be the goalie" or "kick for goal", and these jobs can change as the game progresses. Then there are individual strategies such as "take the ball up the field and kick for goal". And finally, the robots need to act as a team; "support him!" is not easy to code.

There have been many successful human-coded robotic soccer teams and each year there are a number of competitions where these robots demonstrate successful high-level and low-level control. That these teams exist shows that we can divide the problem into parts small enough to solve [3]. This does not say that the way we have chosen to divide the problem is the best way; indeed, it may not even be near optimal as there are many ways of coding a soccer team.

Rather than hand-coding the control program, it should be possible for the computer to learn how to play. How could the computer learn? One answer is to use genetic programming.

2 What is Genetic Programming?

Selective breeding of farm animals has occurred for thousands of years. Dairy cattle, pigs, beef cattle, sheep, poultry, goats, and horses have all been carefully bred by farmers to produce desirable traits. Modern day dairy cattle produce more milk, chickens grow faster and horses gallop quicker than any of their long-dead ancestors.

This principle, selective breeding, can also be harnessed to generate computer programs. If a program can somehow be bred and also classified as "better" than another, then a population of programs can be evolved. This is genetic programming.

Rather than the DNA found in biological life, programs are generally stored as trees. The tree structure of a program to calculate $3(x + 6)$ is shown in figure 2.

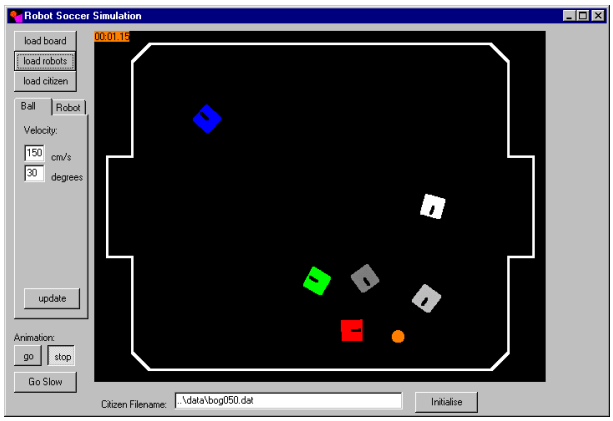

Figure 1: A simulation of a MiroSot three-a-side robotic soccer game.

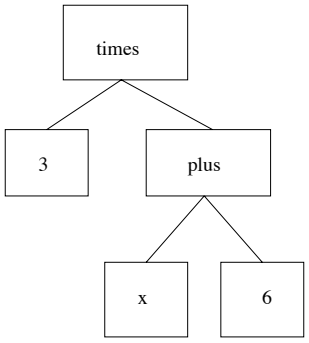

Figure 2: A typical, albeit simple, tree-structured program (individual) that returns $3(x + 6)$.

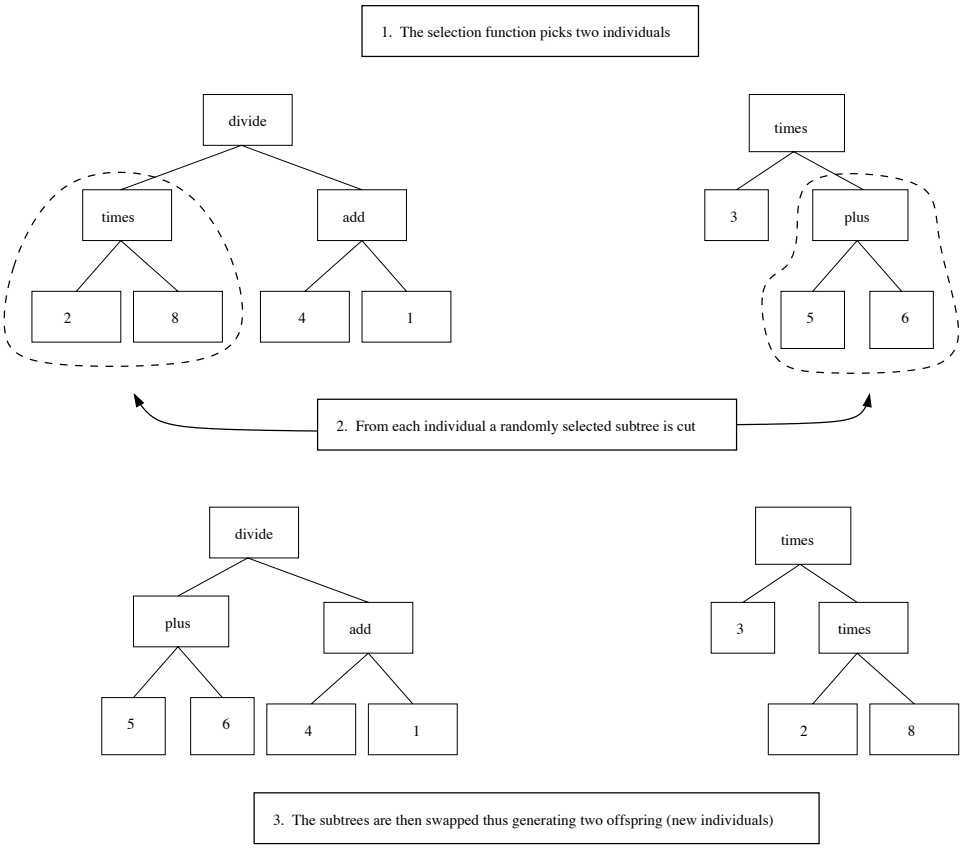

Figure 3: The crossover operation applied to two individuals.

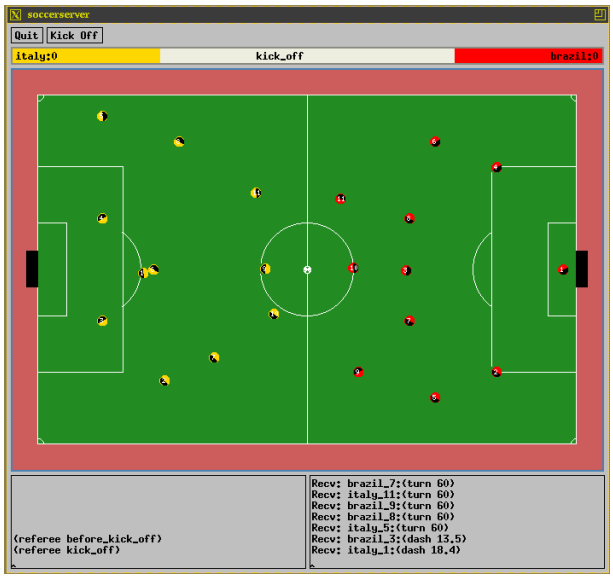

Figure 4: The RoboCup SoccerServer simulator.

Tree structures make it possible to breed programs. A number of genetic operators are available, the most common are *reproduction*, *mutation* and *crossover*. Reproduction makes an exact copy from one generation to the next. Mutation takes an individual's tree and makes an exact copy except that it alters one randomly selected node. Crossover takes two individuals and copies them to the new generation; it then cuts one subtree from each and swaps them around. Figure 3 is an example of crossover in action.

3 Other Work

Luke, et al. [6] worked on creating a program to control a team of robots for the RoboCup simulator league [2] (see figure 4). This league is effectively virtual soccer with eleven players on each side. Although it is never played with real hardware, it is a very challenging domain.

The nodes of their trees were primitive soccer-based notions such as: ball, a vector to the ball; opp-closer, a check on whether an opponent is closer to the ball than the robot and dribble, a human-coded function to dribble the ball. An example program tree, taken from their paper [6], can be seen in figure 5.

Luke, et al. attempted to evolve an entire team of virtual players but due to the massive computation requirements they were forced to make a number of simplifications. The most important simplification was to use the same program for each of the eleven robots. This way, only one program needed to be evolved, which would decrease the time required to find an acceptable solution, but specialisation of players, such as strikers and defenders, could not occur.

Their paper discuses the history of their evolution: from random movers, to kiddie-soccer players, to kiddie-soccer with defenders, and finally to an acceptable solution.

That they were able to evolve an acceptable team using genetic programming was the basis for this work.

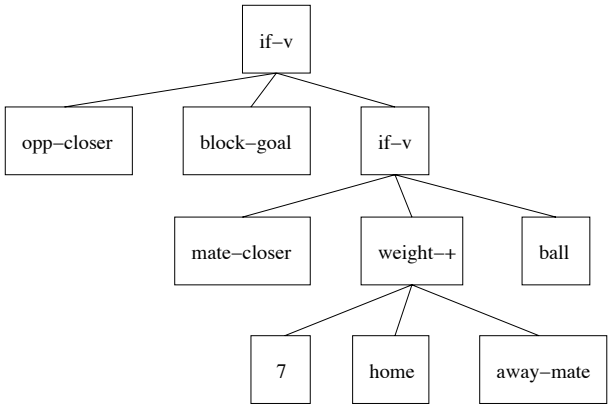

Figure 5: An example program tree from Luke, et al.

4 Objective

Unlike the virtual robots in the simulator league, this work aims to produce robots for real competitions. These real robots also need control strategies which, so far, have been hand-coded.

The problem with hand-coding is that it is time consuming and requires a programmer. The use of genetic programming allows this task to be given to the computer and it is possible that evolution may produce a solution that would not have been considered by humans. It is also possible that given sufficient time, the programs evolved may even be better than those hand coded. The objective of this work is to evolve such control and strategy algorithms.

The main problem with evolving control algorithms is that the robots must play literally thousands of soccer games. Given that their batteries last not more than a number of hours before needing a time-consuming recharge, and that after almost every game a robot will need to be repaired, and that during a game human interference is often required, thousands of games are not feasible in the time constraints of a few months.

Because the evolutionary process, genetic programming, *needs* all these thousands of games, a solution is required. The solution is to simulate the environment so that evolution can take place in a virtual world. This environment must accurately model the real world because, with the evolutionary process complete, the final result will be exported from this virtual world into the robots.

The robots have two wheels that are controlled by setting a value, from -127 to 127, for each wheel: 127 is full speed forward; -127 is full speed backward and zero is stopped. This structure dictates what the final result of the evolved program must be: two integer values to control the wheels.

It was decided not to copy the high-level nodes of Luke et al. (such as dribble and opp-closer) as this would limit the evolutionary process. Low level nodes allow the evolution of motion control as well as strategy algorithms. High level nodes, on the other hand, do not allow such optimisation of motion control.

To give more focus to general robot control it was decided to give only low-level abilities such as arithmetic (e.g. add and subtract), important game constants (e.g. the ball's position with ball_x and ball_y) and randomly selected constants. This means that as well as developing strategy, low-level knowledge, such as how to dribble the ball, will also be optimised by the genetic programming process.

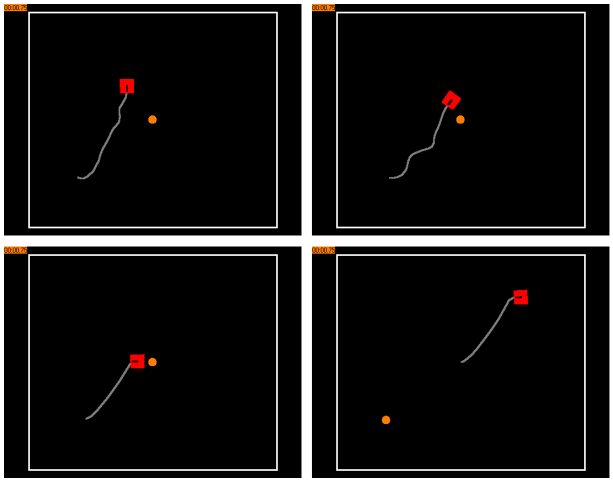

Figure 6: First attempt at ball following. Top left: best individual from generation zero (BOG 0). Top right: BOG 1. Bottom left: BOG 8; this movement pattern was used almost exclusively from BOG 8 to BOG 50. Bottom right: BOG 50; different initial positions but notice the same movement pattern as BOG 8.

5 Transferring to Real Systems

Moving the evolved code from a simulated environment to the real world has been successfully executed by Lee and Zhang [5] with Khepera robots.

Using a commonly available simulator which accurately simulates the Khepera robot light-finding problem domain, they were able to evolve a program to solve this problem. The genetic programming system produced a solution that, in the simulator, very quickly travelled towards the light source. They were able to take this program and use it to control a real Khepera robot which was equally successful.

Of major concern when moving programs from the simulator to the real world is that they will be optimised for a slightly different task. For example, the simulator for the Khepera problem assumes that there is no ambient lighting; in the real world however, removing ambient light is very impractical. To get around this problem Lee and Zhang used perceptrons to separate the sensors from the program. This way the perceptrons were trained to output expected values even in different environments.

Fortunately there are no sensors on the robots used in this study. Instead, all the game information comes from the camera looking down from above. The robots have access to variables whose values are derived from the camera's images (i.e. ball position, robot positions and robot orientations) and these variables can be accurately simulated. Thus, Lee and Zhang's perceptron interface technique is unnecessary in this domain.

6 Results

Before evolving a soccer team, a simple ball-following problem was attempted. It was expected that this would be a problem that was very quick to solve and that would test the simulator and genetic programming kernel. However, the problem proved so difficult that more advanced problems were not considered.

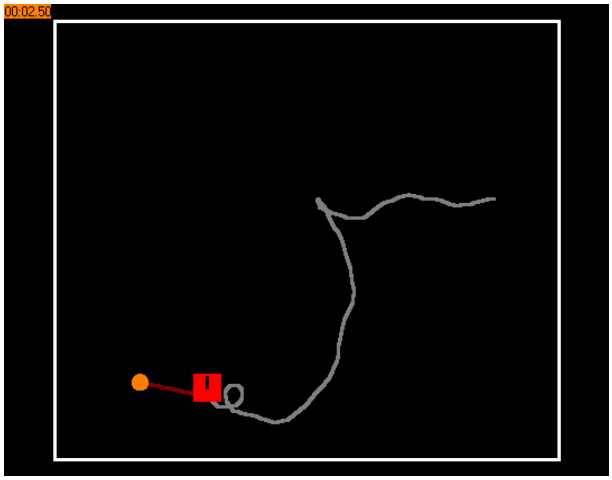

Figure 7: A further attempt at ball following: BOG 50 starting from an arbitrary position and running for 2.5 seconds.

The first scenario given to the kernel was that the robot's initial position was in the lower left and the ball was stationary in the centre. For every frame of the simulation (there were $\frac{1}{33}$ seconds per frame for 0.75 seconds) the distance between the robot and the ball was squared and this was added to the individual's raw fitness score. Thus, low raw fitness scores imply the robot moved quickly towards the ball. This problem was allowed 50 generations with a population of 500.

After only eight generations a very efficient solution was found (see figure 6). It travelled almost directly towards the ball at high speed. As the robot got closer it would slow down to avoid hitting the ball.

However when this program was applied to an alternative initial starting scenario, it was almost completely useless. Rather than follow the ball, as intended, the robot had learned to travel diagonally upwards from its initial position (see the bottom left image in figure 6). From this experience it is clear that it is important that the learning environment represents the actual problem.

A further attempt at learning to follow the ball was made. Rather than just one starting scenario, eight scenarios were used: the robot was initially placed in the centre with the ball in positions around the robot. The available terminals were also expanded. This was notably more successful for the general task of ball following, but it was also considerably more time consuming requiring more than 50 hours of computation. Figure 7 shows the final evolved program in action.

Although the program is not the most efficient of all ball-followers, it does display some interesting behaviour. It is important to realise that following the ball so closely as to actually hit it, makes the problem significantly more difficult. As soon as the robot contacts the ball, the ball moves off in a different direction: the robot now has to follow a moving ball. To counteract this problem, the evolved program executes very tight spins as it nears the ball.

A more detailed description of the simulator, GP kernel and the problems faced with this work, can be found in Walker [7].

7 Conclusions

From the experience of evolving a ball following program, problem specification is seen as very important. Certainly a solution for the first problem (with just one starting scenario) was evolved

quickly, but because it was not general, it was not useful. The later attempt, with eight starting scenarios, produced better results.

The later attempt was significantly assisted by the addition of extra terminals. Although these terminals provided no more functionality than what was initially available, their addition produced significantly better results. Were the GP kernel equipped with Koza's ADFs [4], Angeline's module acquisition [1] or some similar feature, the addition of extra terminals may not have been necessary.

The tight spinning behaviour observed in the program evolved in the later attempt is the type of behaviour that is both unexpected and useful: an element of strategy is already evolving. Within the larger scope of a full soccer game there is significantly more room to, and more advantage in, developing such unexpected strategies. It is for this reason that genetically evolved programs may become skilled adversaries: evolutionary search algorithms scan the nooks and crannies of the problem domain, finding solutions that humans may never consider.

The relative failure of this work compared to that produced by Luke, et al. was disappointing. Successful evolution of a soccer team was anticipated, yet poor quality ball-following was what eventuated.

The first issue is that of the available genes. Luke, et al. started with high-level genes (such as dribble and opp-closer) whereas I started with low-level genes (such as genes for arithmetic). The two approaches are significantly different. Luke, et al. do not allow the genetic process to alter fundamental abilities, such as how to dribble, instead the process focuses on soccer-playing strategies. With a low-level approach, fundamental abilities are just as likely to be optimised as general strategy. Thus the low-level approach gives more flexibility to the evolutionary process at the expense of computing resources.

The second issue is the available computing resources. Although 50 hours seemed a large amount of time for such a simple task as ball following, to produce their final team, Luke, et al. waited *several months'* time. Further to the amount of time, the super-computer they used outperformed the desktop I used by about a factor of 50. Effectively they used *several years* of my computer to produce their results. 50 hours pales into insignificance.

To slightly improve this issue, the environment could have been simplified. Rather than a ball that moved when the robot came too close, a static goal point might have been an easier initial problem.

A further improvement would have been to optimise the calculations performed by the simulator; they were very inefficient since the simulator was graphical.

References

[1] P. J. Angeline and J. B. Pollack. Evolutionary module acquisition. In D. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163, La Jolla, CA, USA, 25-26 1993.

[2] Mao Chen, Ehsan Foroughi, Fredrik Heintz, Zhan Xiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. Robocup soccer server: Users manual. This manual (for Soccer Server version 7.07) is available from the Web at sserver.sourceforge.net., June 2001.

[3] G. Sen Gupta, C. H. Messom, and H. L Sng. State transition based supervisory control for a robot soccer system. In M. Renovell, S. Kajihara, I. Al-Bahadly, and S. Demidenko, editors, *Proceedings of The First IEEE International Workshop on Electronic Design, Test and Applications 2002*, pages 338–342. IEEE, 2002.

[4] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[5] K. J. Lee and B. T. Zhang. Learning robot behaviors by evolving genetic programs. *Proceedings of the 26th International Conference on Industrial Electronics, Control and Instrumentation (IECON-2000)*, pages 2867–2872, 2000.

[6] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler. Co-evolving soccer softbot team coordination with genetic programming. In H. Kitano, editor, *RoboCup-97: Robot Soccer World Cup*, pages 398–411. Springer, 1997.

[7] Matthew Walker. Development of robot soccer strategies by genetic programming utilising a simulated environment. Massey Univeristy Honours project report. Available at www.massey.ac.nz/∼mgwalker/gp., 2000.