Evolutionary Solo Pong Players

Technical Report CSM-423 ISSN 1744-8050

Computer Science, University of Essex, UK

W. B. Langdon Riccardo Poli

Revision 1 July 2005 Correct crossover rate
First version 17 March 2005

Abstract

An Internet Java Applet `http://www.cs.essex.ac.uk/staff/poli/SoloPong/` allows users anywhere to play the Solo Pong game. We compare people's performance to a hand coded "Optimal" player and programs automatically produced by artificial intelligence. The AI techniques are: genetic programming, including a hybrid of GP and a human designed algorithm, and a particle swarm optimiser. The AI approaches are not fine tuned. GP and PSO find good players. Evolutionary computation (EC) is able to beat both human designed code and human players.

1 Introduction

Game playing is a long established human activity and has been widely used as a test bed for artificial intelligence (AI) experiments. Amongst these board games, like chess, go and draughts, have been successfully played by both rule based and evolutionary AI [Schaeffer *et al.*, 1993; Fogel, 2001], however AI can also be applied to other types of games. In particular continuous real-time games. Solo Pong is such a game (cf. Figure 1). Not only must the player decide where to place the paddle in order to keep the ball in play, including considering what will happen in the longer term but must plan to get it into position before the ball crosses the base line and so goes

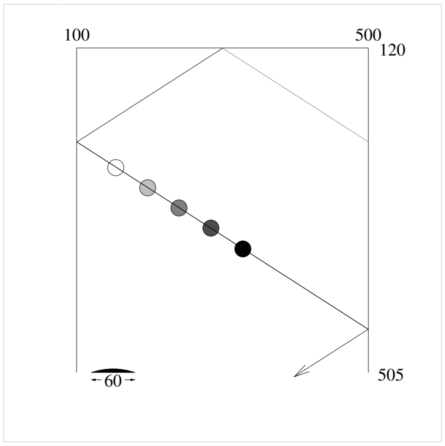

Figure 1: Pong game. The goal is to move the paddle (black arc) horizontally so ball that it bounces from it back into the top arena and so keep the ball in play for as long as possible. Five ball positions from the largest serve angle in the 21 training examples are shown.

out of play. To really master the game, the player must also consider what will happen after the next bounce.

Control applications tend not to use AI, so it is interesting to see two AI techniques being used for control. Not that this is the first time they have been used in control applications. For example, Koza has used genetic programming to evolve optimal control strategies [Koza, 1992; Koza *et al.*, 1997]. While PSO control applications tend to be in electrical power transmission, e.g. [Yoshida *et al.*, 2000].

The following section describes Solo Pong in more detail. Section 3 describes the Human and various computer algorithms which play Pong. The length of time they can keep each ball in play is given in Section 4, while their strategies are described in Section 5 before we conclude (6).

2 Solo Pong

In Solo Pong the user is presented with a rectangular two dimensional Squash Court enclosed on three sides (see Figure 1) containing an elastic ball. The bottom side is open but contains a banana shaped paddle, which the player must position so as to prevent the ball going out of the court. The ball speeds up every time it hits the paddle. The paddle has both inertia and is subject to friction. That is, the player cannot move the paddle instantaneously, instead the game requires both thought to predict where the paddle should be as well as dexterity with the mouse to control it. In human play, the frame rate was set at the best speed to allow sufficient thought as well as time to move the paddle. It is not a reaction only test. The following two sections describe in detail the motion of the ball and paddle.

2.1 Simulating Ball Position and Bounces

The ball moves with constant velocity until it strikes one of the walls or the paddle. Bounces from the walls are perfectly elastic. Should the ball pass the base line, it is considered lost and another ball is served from the centre of the top ball in a random direction (±1 radian). The paddle is not flat. This means, unless the ball strikes its centre, the ball's deflection angle will be different from the incidence angle. This effect increases linearly along the length of the paddle, so a ball just clipping it will have its horizontal velocity increased or decreased by 5 pixels/frame. Secondly the ball's vertical velocity is increases by 10% each time it strikes the paddle, so each ball gets progressively more difficult to play. Eventually the ball becomes so fast that integration instabilities in the simulator ensure none of the algorithms can keep it in play.

The simulation runs in discrete time, so changes in ball/paddle location and velocity are calculated at regular intervals (known as the frame rate, 10 s^{-1}). This effectively adds some noise to the system. In particular it makes it harder to predict the path of the ball after it bounces.

2.2 Simulating Paddle Location and Velocity

The paddle is moved by an external force which has to overcome both inertia and friction. Each frame its position and velocity are updated. It cannot pass the left and right walls but does not bounce off them as the ball does, nor do collisions with the walls drop its velocity instantaneously to zero. (So the simulator allows the paddle to appear to be at rest, since it cannot move through the walls, but still has inertia which must be overcome by the external force.) The external force cannot exceed 1 pixel frame^{-2}. The frictional force is proportional to the velocity of the paddle (both are unbounded). Friction $= -1.0 \times$ paddle velocity (expressed in units of pixels and frames).

3 Players – Human, Optimal, GP and PSO

3.1 Human

In order to assess the typical performance of human players, we asked eight subjects to play Solo Pong in the same experimental conditions. Each of the subjects was give a short introduction to the game and a few minutes to play the game and familiarise themselves with the paddle controls and game dynamics. All player were fit young or middle age adults with no visible disabilities. The simulation speed (10 frames per second) was chosen to be slow enough to achieve high scores by people but fast enough not to become boring.

When a human player controls the paddle, the mouse pointer horizontal position is constantly assessed and compared with the position of the paddle. A positive force is applied to the paddle when the paddle is to the left of the pointer, and vice versa. The force is proportional to the horizontal displacement between paddle and pointer, but it is clipped to the range $[-1, +1]$. This is the same clipping as is used for the automatic players described below.

3.2 Optimal Player

Since some automatic players are extremely good and can keep a ball in play for a long time their frame rate was increased ten fold to keep time for simulations reasonable. (In automatic play frame rates as high as 400 per second can be reached on an ordinary personal computer.)

All the automatic players are given the same information, their control output is fed into the paddle controller in the Pong simulator and their performance is calculated in the same way. The strategy of the optimal player, which we will call Optimus hereafter, is, given the current location and velocity of the ball, to predict where it will cross the base line. (Purely for illustrative purposes, the prediction is displayed as a small red dot on the screen.) Except when it comes into contact with the walls or the paddle, the ball moves in a straight line at constant speed. Thus when no bounces are involved, predicting the ball crossing point is straightforward, and it seems that even AI can learn to do this. However if the player waits until after the last bounce he will often not have sufficient time to move the paddle to intercept the ball. Apart from simulation noise, Optimus is coded to deal correctly with bounces from the walls. The AI approaches find this very hard.

Given the current location and velocity of the paddle, Optimus places the paddle at the predicted location as fast as possible leaving it at rest (effectively using a bang-bang type of control). Having the paddle at rest, means it can be driven to the left or the right with equal ease for the next bounce. Since predicting bounces from the banana shaped paddle is difficult, Optimus only tries to predict as far as the next base line crossing. Optimus knows the control laws used by the simulator for both ball and paddle, whereas the GP and PSO approaches have to learn these.

3.2.1 Optimal Paddle Control Law

Optimus drives the paddle with maximum force towards the predicted place where the ball will cross the base line until the paddle and prediction are sufficiently close that it must apply an opposing force to overcome the paddle's inertia and bring it to rest. Naturally this speed reduction is also done with maximum force.

Assuming continuous time, rather than the discrete time used by the simulator; let v represent the velocity of the paddle. Then $\dot{v} = F - fv$. Where F is the applied force and f is the frictional coefficient. The solutions to this differential equation are linear combinations of decaying exponentials. In particular when slowing, $v = -v_\infty + (v_0 + v_\infty)e^{-ft}$ (note $v_\infty = F/f$ is the paddle's terminal speed and v_0 is its current velocity). Integrating this gives the distance travelled as $x = (v_0 + v_\infty)(1 - e^{-ft})/f - v_\infty t$. Optimus wants to bring the paddle to rest ($v = 0$) so this will take $t = \frac{1}{f}\log(1 + v_0/v_\infty)$ during which time it will travel $x = (v_0 - v_\infty \log(1 + v_0/v_\infty))/f$.

That is, Optimus continues to push the paddle towards the target until it is within x of its prediction, after which it directs the force away in order to bring the paddle to rest as fast as possible. So all Optimus has to do, each frame, is to calculate $\frac{1}{f}(v - v_\infty \log(1 + v/v_\infty))$ and decide if this is bigger or smaller than the distance between the paddle's current position and where Optimus wants it to be. (Given the discrete nature of the simulation, in practice this control strategy gives rise to small oscillations about the target set point.)

Table 1: GP Solo Pong Parameters

Function set:	$+ - \times$ DIV[a]
Terminal set:	110 terminals, including: ball_x, ball_y, paddle_x in pixels, ball velocity (x, y) and paddle velocity (x) in pixels per frame (hybrid has a 7^{th} terminal, Optimus' prediction). The remaining terminals are constants uniformly randomly chosen in the range $-2 \ldots +2$
Fitness:	21 balls are served from the middle of the top line of the court (401×385) each at a different angle, $-1 \ldots +1$ every 0.1 radians. Initial vertical velocity is 10 pixels s^{-1}
Selection:	steady state binary tournaments for both parent selection and who to remove from the population
Initial pop:	Trees randomly grown with max depth of 6 (root=0)
Parameters:	Population 5000. 10% crossover rest mutation 2% chance of mutation per tree node.
Termination:	generation 20

[a] DIV is protected division I.e. if $|y| <= 0.001$ $\text{DIV}(x, y) = x$ else $\text{DIV}(x, y) = x/y$.

3.3 Genetic Programming

Genetic programming is a well know automatic program induction method [Koza, 1992; Banzhaf *et al.*, 1998; Langdon and Poli, 2002]. We used a Java version of tinyGP, extended to include double precision constants [Poli, 2004]. Details are summarised in Table 1. GP evolves a function. Each frame, the evolved function, as with the optimal player etc., is given the current state of the ball and paddle. It returns a number which is truncated to the range $-1 \ldots +1$ and this is used to force the motion of the paddle.

Each node within the 90% of offspring created by copying a single parent (rather than by crossover) is subjected to a 2% chance of random mutation. This counteracts bloat [Langdon *et al.*, 1999] since larger programs tend to suffer more mutations. The fitness function used to guide the evolution of the GP and the other two players (described in the following sections) is also given in Table 1.

3.4 Optimus & GP Hybrid

A hybrid of the Optimus player and genetic programming is evolved by first running the prediction part of the optimal algorithm. This prediction is passed to the GP as a seventh input (leaf) and GP is evolved as before.

3.5 Particle Swarm Optimisation

An initial study of the functions evolved by tinyGP suggested that many evolved trees were simple polynomials. This was used as inspiration for a fixed representation to be used by the particle swarm [Kennedy *et al.*, 2001; Crichton, 2002]. This allows all constant, linear and quadratic interactions between the same six inputs as used by the optimal and GP algorithms, making a total of $1+6+21=28$ terms. Each is scaled by a double precision parameter, initially chosen at random from the range $0\ldots+1$. For each parameter the PSO also maintains a velocity (initially zero) and remembers the parameter value associated with the best performance achieved by the particle. In preliminary experiments we tested various other alternative initialisation strategies, including more standard ones with symmetric ranges and non-zero initial velocities. However, these provided no noticeable performance improvement with respect to the simple one we finally adopted.

As with genetic programming (cf. Sections 3.3-3.4), we used a standard PSO without special mutation operators, restricted neighbourhoods or problem specific parameter tuning. However we have not tried a hybrid combination of PSO and Optimus.

The particle swarm consists of 20 such individuals. In addition to remembering the best set of parameters seen by each particle, separately the swarm maintains the best set seen overall. This becomes the final set used. At each iteration the fitness of the each individual is calculated (in the same way as for the GP) and the best statistics are updated. Then the velocities and positions of each particle are updated before running the next iteration. A total of 1000 iterations are used. At each iteration, the velocity in each of the 28 dimensions is changed by a random fraction of the distance between where the particle is and the best point it has seen, plus a random fraction of the distance between it and the best point seen by the whole swarm. Both random fraction are chosen uniformly from the range $0\ldots0.5$. As is usual in PSO, the parameters are clipped to lie in the expected range of sensible values. Guided by the results of GP runs, we chose $-5\ldots+5$.

4 Results

The players performance on the 21 fixed training cases and in general is summarised by Table 2. Even with the high variation between individual balls, it is clear that each type of player's performance is significantly different from the others.

Table 2: Pong performance. Average time ball is kept in play. All times normalised to 10 frames s^{-1}. The 4^{th} column refers to the number of times when the paddle did not return the ball even once. I.e. the game scored an *ace* against the player.

Player	Training	Balls	unplayed	Mean	(SD)
Human	-	10	5	16	(13)
	-	10	5	25	(24)
	-	15	-	26	(24)
	-	10	4	21	(19)
	-	10	6	15	(18)
	-	10	7	10	(12)
	-	10	4	17	(15)
		10	7	10	(10)
	-	10.6	5.7	18	(18)
PSO	40	200	79	32	(31)
Optimus	-	200	32	50	(25)
GP	59	200	141	126	(151)
GP Hybrid	91	130	44	564	(548)

There are two differences between training and testing: 1) in training balls are initially served at an angle chosen at random between $-1 \ldots +1$ rather than at 21 fixed angles. 2) the limit of 1000 frames is removed, the ball remains in play until it is lost. With all the players, the length of time they can keep a ball in play is very variable. It appears to depend mainly on whether the player can return the initial serve or not. Balls at a large angle strike the walls multiple times and are most difficult (see Figure 1).

5 Strategies

5.1 Human

Even over only ten balls each, there is clearly high variations between players. People would complain of having had several difficult serves to play. We view the immediate serving of the next ball immediately after the last going out of play as part of the game. The means the player has to concentrate through out. Loss of concentration can be immediately punished by failing to return a succession of balls. To prevent user fatigue the number of balls used to collect data was limited to ten (there was no limit on the number available for practise).

The best strategy appears to be to carefully predict where the ball will cross the base line and position the paddle there as fast as possible. The best human players were able to use the off-centre bounces from the paddle to straighten the ball to be nearly vertical, but most others could not do that.

5.2 Optimus

The human designed player is clearly very good. Its principle strength is predicting where the ball will cross the stop line. In the Java applet this is indicated by a red dot. As noted in Section 2.1 the simulator does not simulate Newtonian physics exactly and some noise is injected at each bounce. Optimus does not know about this and is incapable of learning it, thus its long range predictions are slightly inaccurate. Nevertheless they are good enough for most situations. The hand coded strategy does not try to play more than one return in advance.

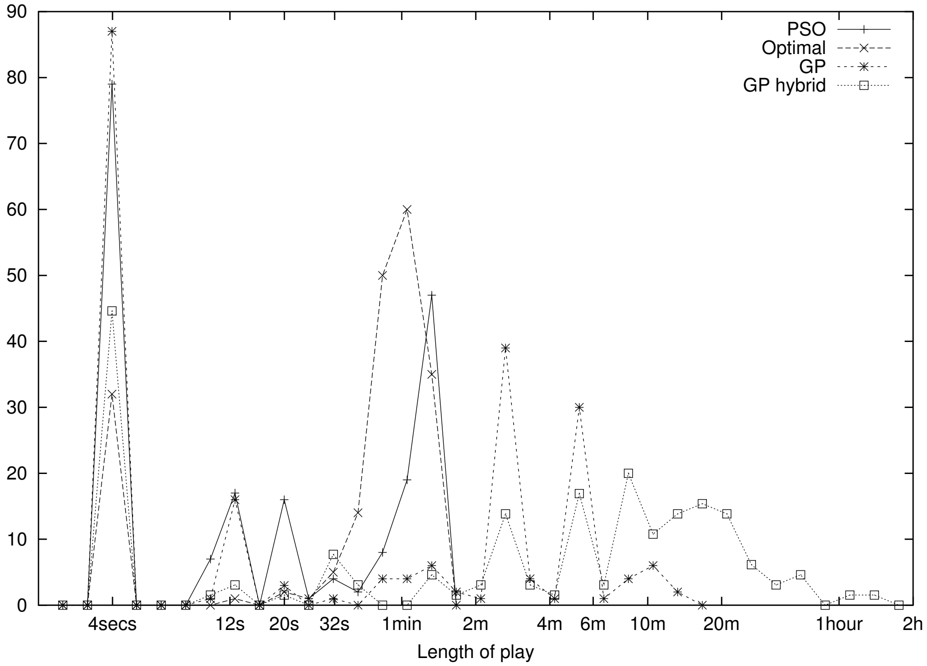

Figure 2: Histogram of lengths of time individual balls are in play. (Data for the GP Hybrid has been rescaled to be in the same proportions as the others.) The peak at 4 seconds corresponds to serves which were not returned. Peaks at 12 and 20 seconds can be explained as the player returning the serve once (12) or twice (20). While the third peak may be due to the time taken to transit the court seven times being somewhat more than 7×4 seconds and so players failing to return the ball a third time tends to fall into data collection bin 32 rather than 28. The other peaks remain unexplained. Nevertheless it is clear that the GP hybrid, GP and PSO keep the ball in play longer than the hand coded player and the human players. (Human data given in Table 2). See also Figure 6 for a discussion of the tail of the distribution.

5.3 Genetic Programming

All the evolved players learn to exploit the banana shape of the paddle so as to deflect the ball not at the same angle (which is what Optimus is effectively trying to do) but closer to the vertical. A vertical ball is both easier to predict and requires less movement of the paddle making the next bounce easier to deal with. Figure 3 describes the evolved program. Since the subtree under the protected division operation typically returns a small value, to a first approximation we can interpret the GP strategy as follows. In order to catch the ball the program constantly tries to reduce the difference both in x position and in speed between paddle and ball (see term multiplied by 0.62). If the ball does not move too fast and the trajectory is not too angled, this is sufficient to hit the ball with the middle of the paddle. When the paddle matches both the ball's position and speed, the term $\mathrm{ball}_{\mathrm{x-velocity}}$ becomes prominent. This modifies the basic strategy just described: unless the ball's trajectory is vertical, it will try to hit the ball with one of the sides of the paddle in such a way as to straighten the ball's trajectory. This is what provides evolved players an edge over Optimus.

5.4 GP Hybrid

The hybrid program (see Figure 4). is the clear winner. It appears that giving GP the same prediction as is calculated for Optimus enables GP to concentrate on the finer parts of controlling the ball with the paddle. However it is not presented with more information than before, it is given the same information but crucially it is presented in a different way. (Optimus' prediction, which is given to the hybrid GP, is also calculated from this information. So no new data is being given to the GP.) It is clear the new presentation makes evolving superb players possible.

While slightly simplistic, if we compare the hybrid strategy with previous GP one (Figure 3) the principle structural difference is to replace the ball's current position with the prediction of where it will be.

Figure 2 suggests the hybrid strategy has traded a small reduction in the number of initial serves it successfully returns for the ability to play the others for much longer.

5.5 Particle Swarm Optimisation

The swarm finds 28 non-zero coefficients to the quadratic polynomial used to drive the paddle. The polynomial is very fast to evaluate but difficult to interpret in terms of a game strategy. The fitness function does not seem to

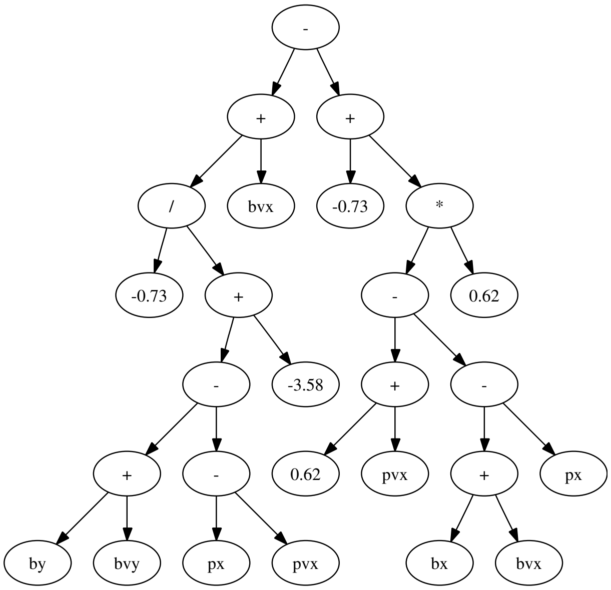

Figure 3: GP evolved strategy to play Pong (simplified by collapsing constant expressions). Given the magnitude of the various parameters This is approximately $\text{ball}_{\text{x}-\text{velocity}} + 0.62 \times \Big((\text{ball}_x - \text{paddle}_x) + (\text{ball}_{\text{x}-\text{velocity}} - \text{paddle}_{\text{velocity}})\Big)$

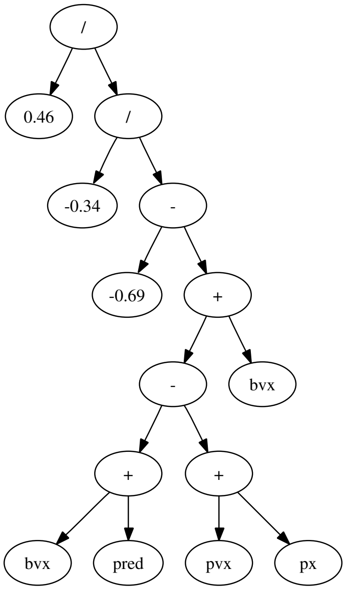

Figure 4: Hybrid evolved strategy to play Pong (after original 51 node tree simplified by collapsing constant expressions). Given the magnitude of the various parameters this is approximately $1.34((\text{prediction} - \text{ball}_x) + (\text{ball}_{\text{x−velocity}} - \text{paddle}_{\text{velocity}}) + \text{ball}_{\text{x−velocity}} + 0.69)$

be well suited to PSO, with large areas of the 28 dimensional parameter space having identical scores. Nevertheless the swarm produces players nearly as good as the hand-coded optimal strategy.

5.6 Discussion

The optimal player was coded knowing the Newtonian laws of motion for the whole system. Thus given the current state the entire future should be known. In practice (as in real life) the physics simulator introduces noise. It is not surprising the optimal player does so well, however given the size of the court, the ball speed etc. there are some serves, which even it cannot return. This explains the initial peak in length of time each ball is played, see Figure 2. Subsequent smaller peaks corresponding to balls being initially returned but lost the next time and even (for some players) after two successful returns.

For illustration purposes, Figure 5 shows five typical paths taken by balls across the court when the paddle is controlled by evolved players. Note how the player has learnt to straighten the path so when the ball next returns to the base line it has moved only a small amount. However eventually the exponential increase in ball speed defeats the player. The four other traces correspond to higher angle initial serves. These are harder to play (so the traces are shorter). When the angle is high the ball will bounce from the walls. Except Optimus, this defeats many players.

Figure 2 makes it clear for hard serves, the length of play distribution is dominated by discrete effects associated with how long it takes the ball to pass from one end of the court to the other. If failure to return the ball was an independent random event, i.e. the chance of returning each bounce is approximately constant, the length of time the ball was kept in play would be an exponential distribution. Of course, we cannot expect our players to match this simple model, since the speed of the ball increases each time it hits the paddle. However, Figure 6 hints that an exponential distribution is compatible with our observations, particularly for the hybrid player. The exponential distribution has a standard deviation equal to its mean. Looking back to Table 2 we see this is approximately true of every player. This indicates although the players ability to return each ball increases with time thanks to their trajectory-straightening strategy, this increase in performance is matched by a corresponding increase in difficulty due to the ball speeding up over time. So, the number of balls lost per unit of time is approximately constant, thereby leading to an approximately exponential distribution.

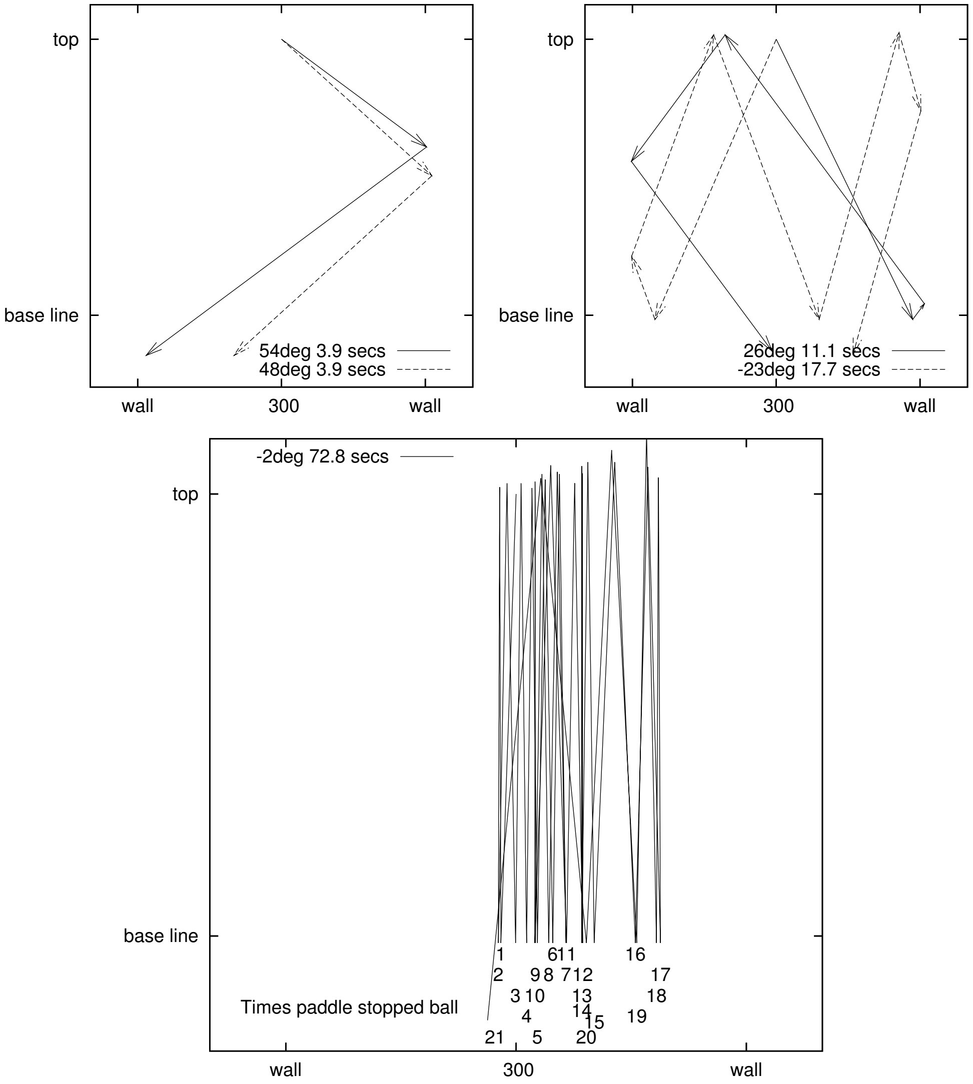

Figure 5: Examples ball play. Here we show a PSO player but GP is similar. The ball is initially served from the top in the centre (300). Larger initial server angles are more difficult to reach, leading to many misses and a high variance in performance. The lower box shows a long game (72.8 seconds) in which the PSO returns the ball 21 times. Good players learn to use the banana shape of the paddle to progressively reduce the angle, making the next return more vertical and so easier to play. Near the middle of the rally there are periods when the ball moves almost vertically (6-10, 11-15, 16-19). However the exponential increase in ball speed makes it harder to control. After return 19, the angle increases and the PSO player can only return it twice more (20, 21).

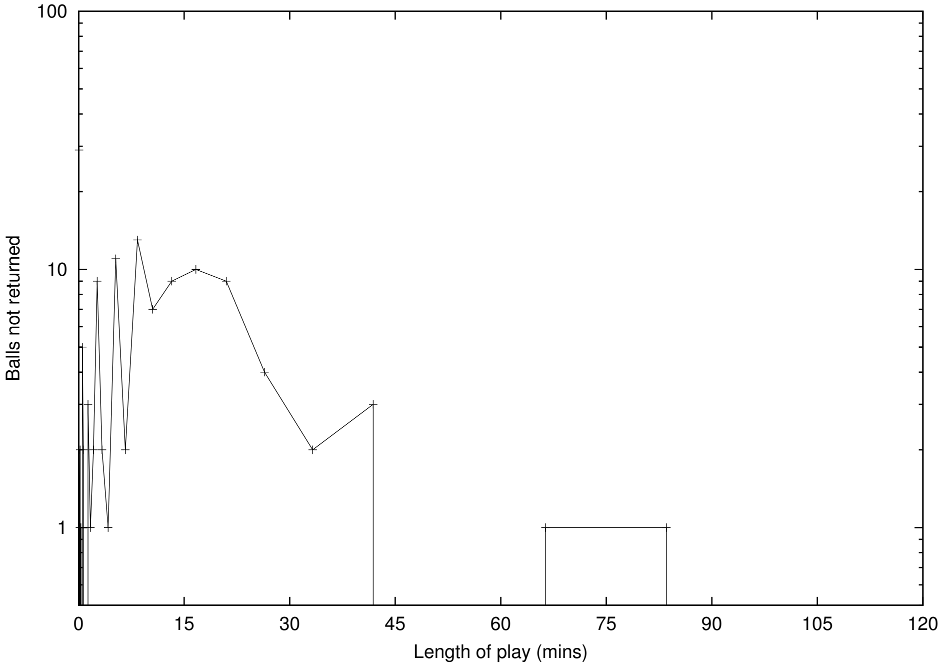

Figure 6: Histogram of lengths of time individual balls are in play for GP hybrid (same data as Figure 2) An exponential tail after 15 minutes of play is hinted at by linear (on this log scale) reduction in balls lost with time played. Note there are few data after 30 minutes, and so they are subject to large amounts of noise.

6 Conclusions

Despite tuning the speed of the game to suit Human players all of the computer algorithms were able to beat even the best human player.

The use of particle swarms for game playing is not well established, so we are pleased that not only did PSO beat all the Human players but also came near the Optimus player. It was also faster to train than either genetic programming approach.

Not surprisingly the optimal player is hard to beat. Nevertheless it is not omnipotent and looks ahead only as far as where the paddle must be placed next. The first genetic programming approach is given the same information but managed to evolve a program which exceeds it. The combination of genetic programming and the optimal player readily evolves programs which not only intercept the ball but control it leading to enormous scores.

Acknowledgements

We would like to thank Simon Lucas, Pierre Collet and anonymous referees for helpful suggestions, and the volunteers for their participation in this study. Also, we would like to acknowledge support from EPSRC grant GR/T11234/01.

References

[Banzhaf *et al.*, 1998] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann, 1998.

[Crichton, 2002] Michael Crichton. *Prey.* HarperCollins.

[Fogel, 2001] David B. Fogel. *Blondie24: Playing at the Edge of AI.* Morgan Kaufmann Publishers, 2001.

[Kennedy *et al.*, 2001] James Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm Intelligence.* Morgan Kaufmann, San Francisco, 2001.

[Koza *et al.*, 1997] John R. Koza, Forrest H Bennett III, Martin A. Keane, and David Andre. Evolution of a time-optimal fly-to controller circuit using genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 207–212, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Langdon and Poli, 2002] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[Langdon *et al.*, 1999] William B. Langdon, Terry Soule, Riccardo Poli, and James A. Foster. The evolution of size and shape. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, 1999.

[Poli, 2004] R. Poli. Tinygp. See TinyGP GECCO 2004 competition at `http://www.cs.essex.ac.uk/staff/sml/gecco/TinyGP.html`.

[Schaeffer *et al.*, 1993] Jonathan Schaeffer, Norman Treloar, Paul Lu, and Robert Lake. Man versus machine for the world checkers championship? *AI Magazine*, 14(2):28–35, 1993.

[Yoshida *et al.*, 2000] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.