Measuring Progress in Coevolutionary Competition

Pablo Funes and Jordan B. Pollack
Brandeis University Department of Computer Science
415 South St., Waltham MA 02454, USA.
pablo@cs.brandeis.edu

Abstract

Evolution, as trial-and-error based learning methods, usually relies on the repeatability of an experience: Different behavioral alternatives are tested and compared with each other. But agents acting on real environments may not be able to choose which experience to live. Instead, the environment provides varying initial conditions for each trial.

In competitive games for example, it is difficult to compare players with each other if they are not able to choose their opponents. Here we describe a statistics-based approach to solving this problem, developed in the context of the Tron system, a coevolutionary experiment that matches humans against agents on a simple video game. We are now able to show, among the results, that the complex interactions led the artificial agents to evolve towards higher proficiency, while at the same time, individual humans learned as they gained experience interacting with the system.

1. Introduction

In the last edition of SAB we presented the Tron system, (Funes et al., 1998) the first example of animal-animat coevolution, between an agent species and a living species. Part of the analysis of this experiment was, at the time, inconclusive: Was the agent species learning? We could tell that agents were winning more frequently, but this could have been due to other effects: their human opponents getting worse over time, for example. In a coevolutionary environment, the Red Queen effect (Cliff and Miller, 1995) makes it difficult to evaluate progress, since the parameter for evaluation of one species is the other, and vice versa. A higher number of wins does not necessarily imply better performance.

To analyze the performance of Tron agents evolving vs. human players we have now applied a statistical method that gives a mathematically sound evaluation of agent and human players alike, allowing us to compare all individual players with each other, even when it is possible that they have never played together.

1.1. Coevolution in Competition

The most basic way to assign fitness to players in a competitive/coevolutionary environment is to sum up all wins (Angeline and Pollack, 1993, Hillis, 1991, Axelrod, 1987). More advanced is the use of fitness sharing strategies (Beasley et al., 1993, Juillé and Pollack, 1996, Rosin, 1997). Different researchers have tried to reduce the number of games to be played in each generation: large savings can be obtained by matching players against a sample instead of the whole population — "finding opponents worth beating" (Sims, 1994, Rosin and Belew, 1995). The assumption, however, that one can choose the opponents, could not be upheld in our case, where human opponents come and go at their will, and an entirely different approach to scoring was needed.

The Tron experiment assayed a fitness sharing-inspired fitness function: for agent a the fitness is

$$F(a) = \sum_{\{h:p(h,a)>0\}} \left(\frac{s(h,a)}{p(h,a)} - \frac{s(h)}{p(h)} \right) \left(1 - e^{\frac{p(h)}{10}} \right) \quad (1)$$

(where s(h,a) is the number of games lost minus the number of games won (*score*) by a human opponent h against a; p(h,a) is the total number of games between the two; s(h) is the total score of h; and p(h) is the number of games that h has played.)

We knew that different agents would play against some of the same and some different humans, so simply summing up all wins would not suffice. Instead we compared winning ratios: according to eq. 1, agents get positive points when they do better than average against a human, and negative points for doing worse than average. The more experienced the human is, the more valuable those points are.

This function was relatively successful in finding good Tron agents, but had problems that we did not foresee. Over time, a strong group of agents formed that were reliably better than average, thus surviving for many generations. As these agents had seen hundreds of humans over their history, and were better than average, even though not necessarily the best, they had too many points to be challenged by newer ones.

Similar problems arise when one tries to compare the performances of past and present players. A well-known strategy for evaluating coevolutionary progress in presence of the Red Queen effect is to take a sample set, an advanced

From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior. Meyer, *et.al. (eds.)*. MIT Press, 2000. Pages 450-459.

generation for example, and use them to evaluate all players (Cliff and Miller, 1995, Pollack and Blair, 1998). This is impossible here: we cannot recreate the behavior of humans who played in the past. Some fixed agents could conceivably be kept in the population for evaluation purposes, but even if one or a few agents were to be present in all generations, most people would play against them only a few times, yielding a measure of low confidence. At the onset of the experiment, we were not willing to sacrifice performance, nor slow down the evolutionary pace by keeping fixed losers inside the population (if they were winners, they would not have to be kept alive artificially, but without an oracle we could not choose them in advance).

The need for a more accurate evaluation of performance in coevolution was thus twofold: not only we wished to study the evolution of the experiments, comparing today's and yesterday's humans and robots; we were also looking for a better measure to further evolve the artificial population in the future.

In what follows we succinctly describe the Tron system, then the statistical analysis tools, to go in more detail over the results obtained.

2. Tron

2.1. Internet Evolution

A machine that learns by playing games may acquire knowledge either from external expertise (playing with a human or human-programmed trainer), or by engaging in *self-play*. Tesauro (Tesauro, 1992) was able to obtain strong backgammon players, having one neural network play itself and adjusting the weights with a variant of Sutton's TD algorithm (Sutton, 1988). Although it worked for backgammon, self-play has failed on other domains. Our group obtained similar results to those of Tesauro's, using hill-climbing, a much simpler algorithm (Pollack and Blair, 1998). This demonstrates that elements unique to backgammon, more than the TD method, enable learning to succeed. Self-play remains an attractive idea because no external experience is required. In most cases, however, the learning agent explores a narrow portion of the problem domain and fails to generalize to the game as humans perceive it.

Attaining knowledge from human experience has proven to be difficult as well. Today's algorithms would require millions of games, hence rendering training against a live human impossible in practice. Programmed trainers have led (as in self-play above) to the exploration of an insufficient subset of the game space: Tesauro (Tesauro, 1990) tried to learn backgammon using human knowledge through a database of human expert examples, but self-play yielded better results. Angeline and Pollack (Angeline and Pollack, 1993) showed how a genetic program that learned to play tic-tac-toe against several fixed heuristic players was outperformed by the winner in a self-playing population.

Today's expert computer players are programmed by humans; some employ no learning at all (Newborn, 1996) and some use it during a final stage to fine-tune a few internal parameters (Baxter et al., 1998).

With the advent of the Internet, evolving against thousands of humans becomes possible. We conceived the idea of a species of software agents that evolve on the web, playing games with humans they encounter: only the better agents survive, so a niche on the Internet exerts the evolutionary pressure that drives the virtual species.

2.2. Tron Agents

An agent engaging in games on user's browser programs is constrained by the Java Virtual Machine of the browser, an environment very limited in speed and resources. Thus we used Tron, a game with minimalistic memory, CPU and graphics requirements.

Tron, (also known as "Light Cycles") got its name from a movie (Walt Disney Studios, 1982) and became popular during the 80's. It is a real-time video game that requires quick reactions and spacial-topological reasoning at the same time. In this game, players move at constant, identical speeds, erecting walls wherever they pass and turning only at right angles. As the game advances, the 2D game arena progressively fills with walls and eventually one opponent crashes, losing the game. In our version, the two players (one human, one agent) start in the middle region of the screen, moving in the same direction (fig. 1). The edges are not considered "walls"; players move past them and reappear on the opposite side, thus creating a toroidal game arena, 256×256 pixels in size.

Our Tron agents perceive the world through sensors that evaluate the distance in pixels from the current position to the nearest obstacle in eight relative directions: Front, Back, Left, Right, FrontLeft, FrontRight, BackLeft and BackRight. Every sensor returns a maximum value of 1 for an immediate obstacle, a lower number for an obstacle further away, and 0 when there are no walls in sight.

Each robot-agent is a small program, representing one Tron strategy, coded as a Genetic Programming (GP) s-expression (Koza, 1992), with terminals {A, B, …, H (the eight sensors) and $\Re$ (random constants between 0 and 1)}, functions {+, -, * (arithmetic operations),% (safe division), IFLTE (if-then-else), RIGHT (turn right) and LEFT (turn left)}, maximum depth of 7 and maximum size of 512 tokens. An agent reads its sensors and evaluates its s-expression every third time step: if a RIGHT or LEFT function is output, the agent makes the corresponding turn; otherwise, it will keep going straight.

When a visitor opens the Tron web page[1], her browser loads and starts a Java applet. The applet receives the GP

1. http://www.demo.cs.brandeis.edu/tron

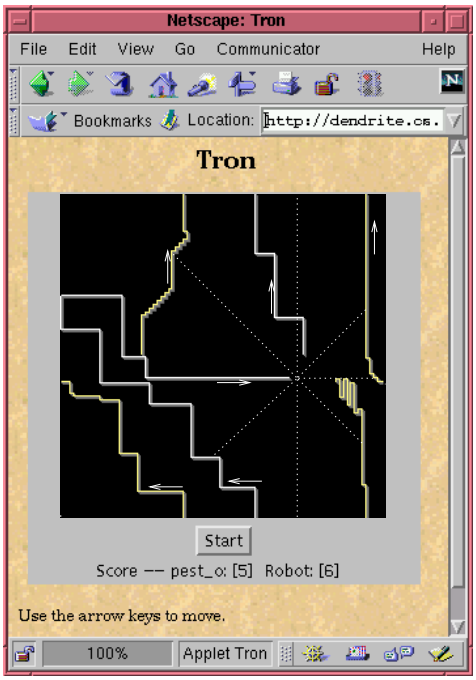

Figure 1: The Tron game. Tron runs as an applet inside an Internet browser. Arrows have been added to indicate direction of movement, and dotted lines to show the sensors of the artificial agent.

code for an agent from our web server and uses it to play one game with her. The human moves by pressing the arrow keys and the agent according to its s-expression. When the game ends, the applet reports the result (win or loss) to the server, and receives a new agent for the next game. This cycle continues until the human stops playing.

2.3. Evolving the Tron species

The system maintains a population of 100 agents. For each game, an agent is drawn at random from this population. Results are stored in a database. A generation lasts until all 100 agents have played a minimum number of games: new agents play at least 10 games, while veterans from previous generations play only 5 games (thus about 18% of games are played by rookies who have not seen humans before). With the current system reaching a high proficiency level, the fact that some novice strategies are always present is a benefit for beginner humans who play for the first time: there are always some games that the system plays more naively, allowing the humans to win occasionally instead of being frustrated by an overwhelming opponent.

When all agents have completed their minimum number of games, the current generation finishes: agents are sorted by fitness; the worst 10 are eliminated and replaced by 10 fresh ones, supplied by a separate *novelty engine*. A new generation begins (fig. 2).

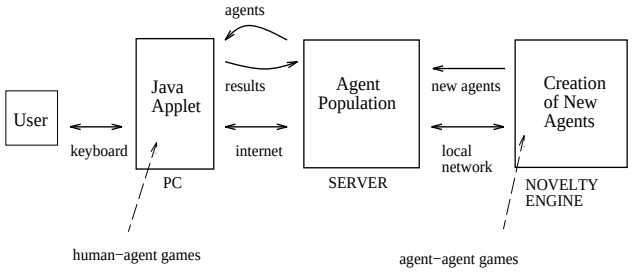

Figure 2: Scheme of information flow. Agents travel to users' computers to play games. Those with poorest performances are eliminated. A novelty engine creates new players. The better ones are added to the population, filling the empty slots.

2.4. Creating New Opponents by Coevolution

The Tron architecture uses a separate *novelty engine* — the "background" part of the system — as the source of new individuals. This module coevolves a population of 1000 agents by playing them against each other.

Even though self-play does not provide enough information to know which strategies will perform well against people, this method is much better than blind recombination for creating interesting new agents.

The novelty engine plays all the individuals in its population against a training set of 25 agents. Fitness is evaluated, and the bottom half of the population is replaced by random mating with crossover of the best half. Fitness sharing is used to promote diversity in the population. The training set consisted a fixed part, the top 15 players from the foreground population, and a coevolutionary part, 10 more agents replaced on each iteration — with a fitness sharing criterion of "finding opponents worth beating" (adapted from Rosin, 1997). Full details of this configuration are given on (Funes et al., 1998).

Later analysis suggested us that having 10 fixed players during the coevolutionary process — they only changed with the slowly changing Internet population of agents — was suboptimal, so we reduced the fixed set to just one player. Fitness from the foreground is fed back into the novelty engine now by reintroducing the best agents directly into the coevolving population, allowing them to evolve against their kin (see section 4.4). The novelty engine now runs continuous coevolution, each agent playing 25 games, one against the fixed 'champion against humanity', and 24 more against the representatives chosen from the previous iteration.

3. Paired Comparisons

Paired comparisons models are statistical methods that estimate the relative strengths or preferences of a group of participants. The "Elo ratings" for Chess (Elo, 1986) are one

example of such method. Chess poses some problems akin to ours, as one would like to ask, say, was Capablanca better than Fisher? Even if the two players did play each other, one might not have been at the peak of his abilities at the time. All the information from opponents they played in common, and how good they performed, should be put together. We have followed the maximum likelihood approach described by Joe (Joe, 1990), applied by the author to the Chess problem among others.

Elo's model — adopted today for many other games, including the so-called "game ladders" on the Internet — assigns a low ranking to a novice, who can slowly climb up as she wins games against other ranked players. Maximum likelihood statistics such as Joe's are better suited to our problem because they compute the most feasible ranking for all players, without presuming that young ones are bad.

The goal of paired comparison statistics is to deduce a ranking from an uneven matrix of observed results, from which the contestants can be sorted from best to worst. In the knowledge that crushing all the complexities of the situation into just one number is a huge simplification, one wishes to have the best one-dimensional explanation of the data.

Each game between two players (P_i, P_j) can be thought of as a random experiment where there is a probability p_{ij} that P_i will win. Games actually observed are thus instances of a binomial distribution experiment: Any sample of n games between P_i and P_j occurs with a probability of

$$\mathrm{P(sample)} = p_{ij}^{\,w_{ij}}(1-p_{ij})^{n-w_{ij}} \qquad (2)$$

where w_{ij} is the number of wins by player P_i.

We wish to assign a *relative strength* parameter (RS) λ_i to each of the players involved in a tournament, where $\lambda_i > \lambda_j$ implies that player P_i is better than player P_j.

A probability function F such that F(0)=0.5 is assumed arbitrarily; we use the logistic function

$$\mathrm{F}(x) = \frac{1}{1+e^{-x}} \qquad (3)$$

The model describes the probabilities p_{ij} as a function of the RS parameter λ_i for each player,

$$p_{ij} = F(\lambda_i - \lambda_j) \qquad (4)$$

so the outcome of a game is a probabilistic function of the difference between both opponent's strengths.

The observed data is a long sequence of games between opponent pairs, each one a either a win or a loss. According to eq. 4, the probability of that particular sequence occurring would have been

$$P = \prod_{i,j} F(\lambda_i - \lambda_j)^{w_{ij}}(1-F(\lambda_i - \lambda_j))^{n_{ij}-w_{ij}} \qquad (5)$$

for any choice of λ_i's.The set of λ_i's that best explains the observations is thus the one that maximizes this probability. The well known method of maximum likelihood can be applied to find the maximum for eq. 5, generating a large set of implicit simultaneous equations that are solved by the Newton-Raphson method.

An important consideration is, the λ_i's are not the true indeterminates, for the equations involve only paired differences, $\lambda_i - \lambda_j$. One point has to be chosen arbitrarily to be the zero of the RS scale.

A similar method permits assigning a rating to the performance of any smaller sample of observations (one player for example): fixing all the λ_i's on equation (5), except one, we obtain

$$\mathrm{wins} = \sum_i F(\lambda - \lambda_i) \qquad (6)$$

where λ is the only unknown — all the other values have already been calculated. The remaining indeterminate is easily solved with the same procedure.

If a given player's history, for example, is a vector $(w_1,...,w_N)$ of win/loss results, obtained against opponents with known RS's $\lambda_1,...,\lambda_N$, respectively, then eq. 6 can be solved iteratively, using a "sliding window" of size $n < N$, to obtaining strength estimates for $(w_1,...,w_n)$, then for $(w_2,...,w_{n+1})$, and so on. Each successive value of λ estimates the strength with respect to the games contained in the window only.

With this, we can do two important things: analyze the changing performance of a single player over time, and, putting the games of a group of players together into a single indeterminate, observe their combined ranking as it changes over time.

Altogether, the paired comparisons model yields:

- A performance scale that we have called *relative strength* (RS). The zero of the scale is set arbitrarily (to the one of a fixed sample player: agent 460003).
- An ordering of the entire set of players in terms of proficiency at the game, as given by the RS's.
- An estimation, for each possible game between two arbitrary players, of the win-lose probability (equation 4). With it, an estimation of exactly how much better or worse one is, as compared to the other.
- A way to measure performance of individuals or groups over time.
- A possible fitness measure: the better ranked players can be chosen to survive.

4. Performance of Human and Agent Players

Our server has been operational since September 1997; we have collected the results of all games between agents and humans; the system is still running. The results presented here are based on the first 525 days of data (204,093 games). A total 4037 human players and 3512 agent players have participated, each of them having faced just some of all potential opponents (fig. 3).

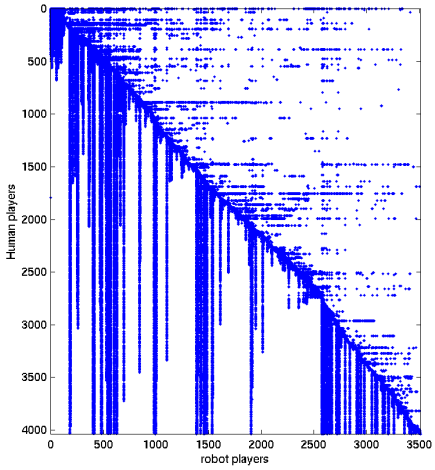

Figure 3: Who has played whom: A dot marks every human-robot pair who have played each other at least once. Both populations are sorted by the date of their first appearance. The long vertical lines correspond to robots that have been part of the population for a long time, and thus have played against most newcomers.

4.1. Win Rate

A basic performance measure is the *win rate* (WR),

$$win\ rate\ =\ \frac{games\ won}{games\ played} \tag{7}$$

which is the fraction of games that the artificial players win. The average win rate over the total number of games played is 0.55, meaning that 55% of all games completed have resulted in agent victories. The WR has been changing over time (fig. 4), in an oscillating fashion. This noisy behavior is a natural phenomenon in a coevolutionary environment, and occurs here more noticeably since one of the evolving populations consists of random human players. Each of the 4037 persons sampled here has a different level of expertise and has played a different number of games (another variable factor is the speed of the game on the user's machine, which may have a slower pace when the Java environment is too slow[2]). The increasing WR suggests, but not proves, that the robot population has been learning, getting better over time as a result of the selection process.

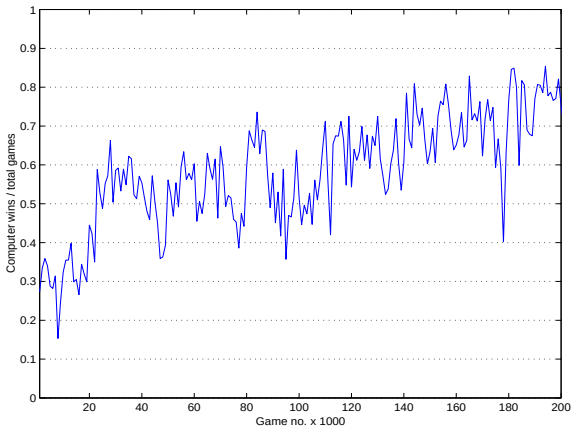

Figure 4: Computer win rate, sampled every 1000 games.

4.2. Statistical Analysis

An increasing WR is not a whole proof that our system was evolving towards better agents. It could be the case, for example, that humans became increasingly sloppy, losing more and more of their games while agents stayed more or less the same.

Applying the paired comparison model gave us more reliable information. We computed the RS for every computer and human player[3]. For the first time we were able to compare agents and humans with each other: tables (a) and (b) on fig. 5 list the 15 best and worst players, respectively. Each human and robot is labelled with a unique id. number: humans were numbered consecutively by their first appearance, and robots have id numbers all greater than 10,000 (the first 3 digits encode the generation number).

The top players table (fig. 5a) has 6 humans at the top, the best agent so far being seventh. The best player is a human, far better than all others: according to eq. 4, an estimated 87% chance of beating the second best!. This person

2. Our Java Tron uses a millisecond sleep instruction to pace the game, but different implementations of the Java Virtual Engine, on different browsers, seem to interpret it with dissimilar accuracies. The effect is more noticeable on machines with slow CPUs and early Java-enabled browsers.
3. Players who have either lost or won all their games cannot be rated, for they would have to be considered infinitely good or bad. Such players convey no information whatsoever to rank the others. Losing against a perfect player, for example, is trivial and has no information contents. Perfect winners/losers have occurred only on players with very little experience. There is one human (no. 228) who won all 37 games he/she played. Should we consider him/her the all-time champion? Perhaps. The present model does not comprehend the possibility of a perfect player. To eliminate noise, we only consider players with 100 games or more. All "unrated" players are far below this threshold.

must be a genius or, more likely, a user with a very old computer, running the applet way below its normal speed.

	Best Players (a)		Worst Players (b)	
	Strength	Player Id	Strength	Player Id
1.	3.55	887	-4.64	2407
2.	1.60	1964	-3.98	2068
3.	1.26	388	-3.95	3982
4.	1.14	155	-3.88	32
5.	1.07	1636	-3.75	1986
6.	1.05	2961	-3.73	33
7.	0.89	3010008	-3.69	3491
8.	0.89	3100001	-3.41	2146
9.	0.84	1754	-3.39	2711
10.	0.81	2770006	-3.36	3140
11.	0.81	3130004	-3.31	1702
12.	0.76	2980001	-3.31	1922
13.	0.70	1860	-3.30	2865
14.	0.66	2910002	-3.27	2697
15.	0.62	3130003	-3.22	2441

Figure 5: Best players (a) and worst players (b) tables. Only players with 100 games or more are been considered. Id. numbers greater than 10000 correspond to robot players.

The difference between the top group of human players (RS around 1.1) and the top agent players (RS's around 0.7) is about 60%. Seven out of the best 15 players are agents. We conclude that Tron is partially learnable by self-play, and that a few very good agent players have managed to survive.

The worst players table (fig. 5b) is composed of all humans. This does not indicate that all agents are good but rather, that most bad agents are eliminated before reaching 100 games.

4.3. Distribution of Players

The global comparative performance of all players is visualized on the distribution curves (fig. 6). Here we have plotted all rated players, including those with just a few games. The fact that agents and humans share similar average strengths indicates that the coevolutionary engine that produces new tron players, has managed to produce some good players. But at the same time, the wide spread of agent levels, from very bad to very good, shows us that there is a reality gap between playing against other robots and playing against humans: all agents that ever played against humans on the website were selected among the best from an agent-agent coevolutionary experiment that has been running for a large number of generations: our novelty engine. If being good against agents was to guarantee that one is also good against people, robots would not cover a wide range of capacities — they would all be nearly as good as possible, and so would fall within a narrow range of abilities.

4.4. Are New Generations Better?

It seems reasonable to expect that new humans joining the system should be no better, nor worse, on average, than those

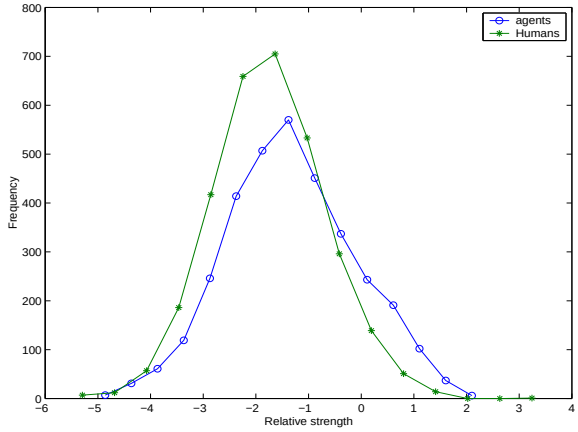

Figure 6: Strength distribution curves for agents and humans.

who came earlier. This is indeed the case, according to the data on fig. 7a: both good and not-so good people keep joining the system. Tron agents (fig. 7b) do show differences. It was our hope that feedback from the foreground population back to our background "novelty engine" could lead to the production of better agents.

Feedback from the foreground population into the background was introduced in two forms: a) From the onset of our experiment, the 15 best agents were used as part of the training in the novelty engine b) Around robot no. 2500 this strategy was changed: control experiments suggested that training against fixed control sets was suboptimal. From this point on, the fixed training set was reduced to just one agent. The main feedback used now consists in seeding the population with the 100 champions from the foreground, letting it evolve from there by pure coevolution. The improvement on the average quality of new Tron agents since no. 2500 is apparent in the graph (so is the bug that produced lousy agents for a few generations).

Our attempt for progressively increasing the quality of new agents produced by the novelty engine, by having them train against those best against humans, was partially successful: graph 7b shows a marginal improvement on the average strength of new players, 0-th to 2500-th. But noticeable better agents beginning at 2800 come to confirm the previous findings of other researchers (Angeline and Pollack, 1993, Tesauro, 1990) in the sense that the coevolving population used as fitness yields more robust results than playing against fixed trainers who can be fooled by tricks that have no general application.

5. Learning

We wish to study how the performance of the different players and species on this experiment has changed over time. Fig. 8 shows the sliding window method applied to one robot. It reveals how inexact — or "noisy" — the RS estimates are when too few games are put together. It is apparent

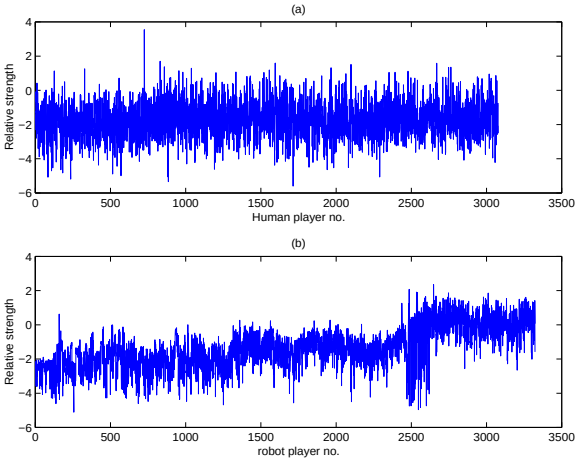

Figure 7: New humans (above) are about as good as earlier ones on average. New robots (below) may be born better, on average, as time passes, benefiting from feedback from agent-human games and improvements on the configuration of the novelty engine.

that 100 games or more are needed to obtain an accurate measure.

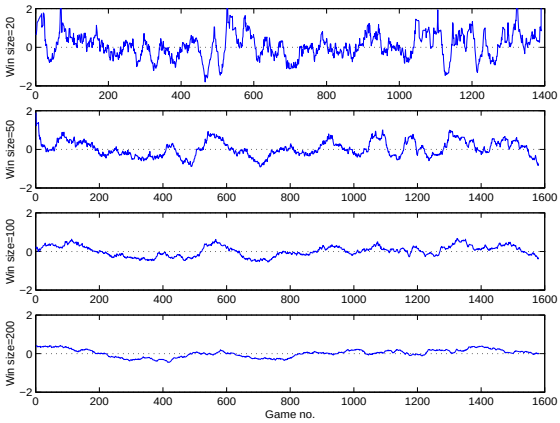

Figure 8: Performance of robot 460003 — which was arbitrarily chosen as the zero of the strength scale — observed along its nearly 1600 games, using increasingly bigger window sizes.

Since each individual agent embodies a single, unchanging strategy for the game of Tron, the model should estimate approximately the same strength value for the same agent at different points in history. This is indeed the case, as seen for example on figs. 8 (bottom graph) and 9a.The situation with humans is very different, as people change their game, improving in most cases (fig. 9b.)

5.1. Evolution as Learning

The Tron system was intended to function as one intelligent, learning opponent to challenge humanity. The strategy of

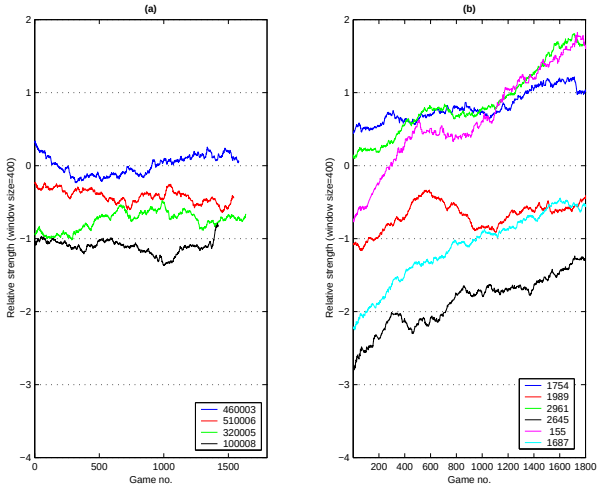

Figure 9: (a) Robot's strengths, (a) Robot's strengths as expected, don't change much over time. Humans, on the other hand, are variable (b).

this virtual agent is generated by the random mixture of Tron robots in the evolving population; 18% of the games being played by new, untested agents, exploring new strategy space. The remaining games are played by those agents considered best so far — survivors from previous generations, exploiting previous knowledge. In terms of traditional AI, the idea is to utilize the dynamics of evolution by selection of the fittest as a way to create a mixture of experts that create one increasingly robust Tron player.

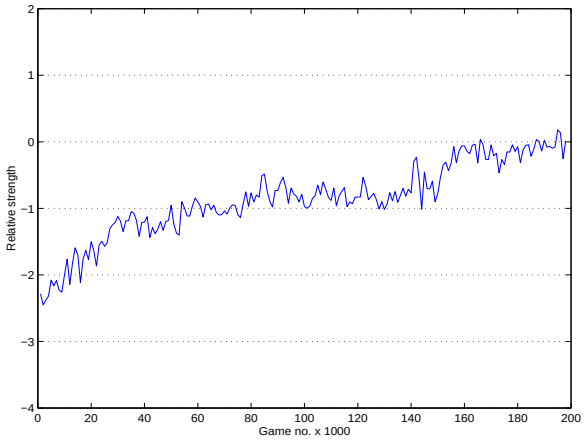

Figure 10: Relative strength of the Tron species increases over time, showing artificial learning.

We use the same formula that solves the ranking equation (6) for one player, but solving for all of the computer's games put together. The result is the performance history of the combined Tron agent. Fig. 10 shows that our system has been learning throughout the experiment, at the beginning performing at a RS rate below –2.0, and at the end around 0.

Now we can to go back to the human scale. The next graph, re-scales the RS values in terms of the percent of

humans below each value. *Beginning as a player in the lower 30 percent, as compared to humans, the Tron system has improved dramatically: by the end of the period it is a top 5% player* (fig. 11).

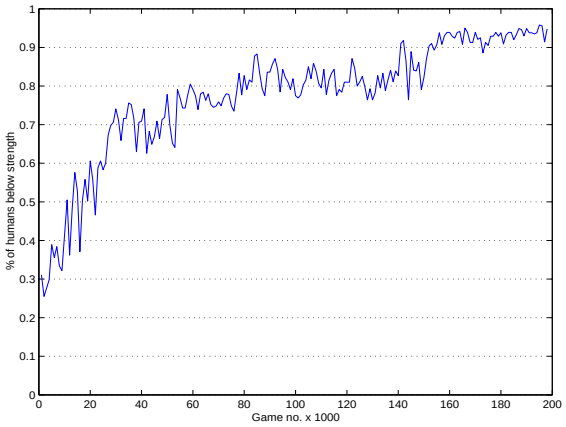

Figure 11: Strength values for the Tron system, plotted as percent of humans below. In the beginning our system performed worse than 70% of all human players. Now it is within the best 5%

5.2. Human Learning

Is the human species getting better as well? No. Redoing the same exercise of figure 10, but now tracing the strength level of all human players considered as one entity, we obtain a wavy line that does not seem to be going up nor down (fig. 12). This means that, although individual humans improve, new novices keep arising, and the overall performance of the species has not changed over the period that Tron has been on-line.

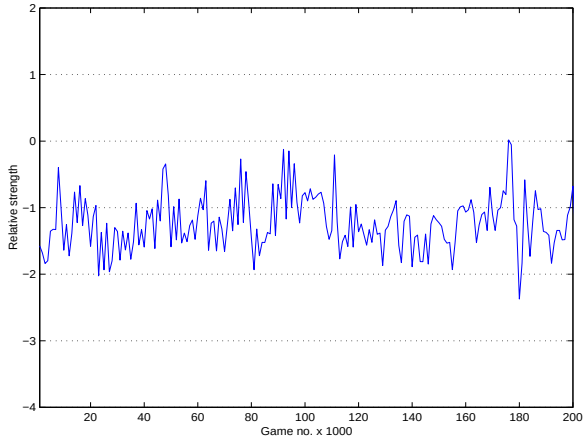

Figure 12: Performance of the human species, considered as one player, varies strongly, complicating things for a learning opponent, but does not present overall trends.

An altogether different image emerges when we consider humans on an individual basis. Although a large number of games are needed to observe significant learning, there is an important group of users who have played 400 games or more. On average, these humans raise from a performance of –2.4 on their first game, to –0.8 on their 400th game, improving approximately 1.5 points over 400 games (fig. 13). We must conclude that the learning rate is dramatically faster for humans, as compared to the approximately 100,000 games (against people) that our system needed to achieve the same feat (fig. 10).

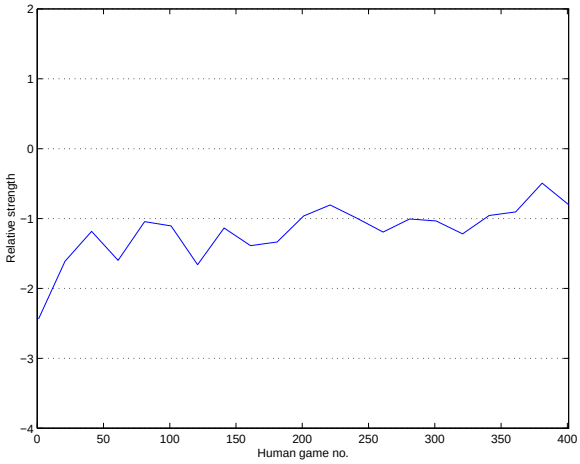

Figure 13: Average human learning: RS of players' *n*-th games up to 400. A first-timer has an estimated RS strength of -2.4; after a practice of 400 games she is expected to play at a -0.8 level. Only users with a history of 400 games or more were considered (*N*=78).

On fig. 14. we have plotted the learning curves of the 12 most frequent players. Many of them keep learning after 1000 games and more, but some plateau or become worse after some time.

6. Conclusions

In an effort to track the Red Queen, without having to play games outside those involved in the coevolutionary situation, we can think of each player as a relative reference. In Tron, each agent has a fixed strategy and thus constitutes a marker that gives a small amount of evaluation information. A single human, as defined by their login name and password, should also be relatively stable — in the short term at least. The paired comparisons model described here is a powerful tool that uses the information of all the interwoven relationships of a matrix of games (fig. 3) at the same time. Every player, with his/her/its wins and loses, contributes useful bits of information to evaluate all the rest.

There are degenerate situations where the present model would give no answer. If one has knowledge of games between players A and B for example, and also between C and D, but nor A nor B have ever played C or D, there is no

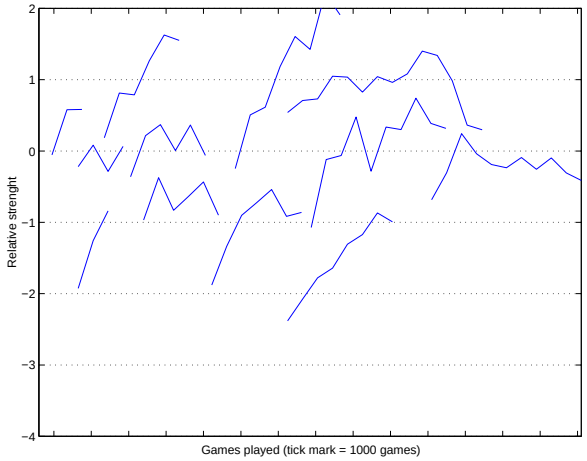

Figure 14: Individual learning: strength curves for the 12 most frequent players (curves start at different x values to avoid overlapping). All users change; nearly all improve in the beginning, but later some of them plateau or descend whereas others continue learning.

connectivity, and consequently no solution to equation (5). In the Tron case, connectivity is maintained throughout the experiment by the multitude of players who come and go, coexisting for a while with other players who are also staying for a limited time. The whole matrix is connected, and the global solution propagates those relative references throughout the data set.

With Tron we are proposing a new paradigm for evolutionary computation: creating niches where agents and humans interact, leading to the evolution of the agent species. There are two main difficulties introduced when one attempts this type of coevolution against real people:

- Interactions with humans are a sparse resource.
- Opponents are random and known tournament techniques for coevolution become unfeasible.

The first problem is common to all applications that wish to learn from a real, or even simulated, environment: interactions are slow and costly. We address this problem by nesting an extra loop of coevolution: while the system is waiting for human opponents, it runs more and more generations of agent-agent coevolution.

The second problem led us to develop a new evaluation strategy, based on the paired comparisons statistics. With it we have been able to prove that the system has indeed been learning through interaction with people, reaching the level of a top 5% player.

The paired comparisons model also gives us a candidate for a fitness function that could solve the problems of the first one. At the present moment, we have replaced our original formula (eq. 1) with the RS index, re-evaluated after each generation is run. The results will be presented in a forthcoming paper.

The widespread distribution of Tron agent capacities, from very good to very bad (fig. 5) indicates, on one hand, that evolving Tron agents by playing each other was not sufficient, as the top agents are usually not so special against people. But on the other, some of them are good, so expertise against other robots and expertise against people are not completely independent variables.

We think that this is the general case: evolutionary computation is useful in domains that are not entirely unlearnable; at the same time, there is no substitute for the real experience: simulation can never be perfect.

We have also been able to show here, how most humans — at least those who stay for a while — learn form their interaction with the system; some of them quite significantly. Even though the system was not designed as a training environment for people, but rather simply as an artificial opponent, the implications for human education are exciting: evolutionary techniques provide us with a tool for building adaptive environments, capable of challenging humans with increased efficiency due to the simultaneous interaction with a large group of people.

References

Angeline, P. J. and Pollack, J. B. (1993). Competitive environments evolve better solutions for complex tasks. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 264–270, University of Illinois at Urbana-Champaign. Morgan Kaufmann, San Mateo, Calif.

Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*. Pitman: London.

Baxter, J., Tridgell, A., and L.Weaver (1998). TDLeaf(lambda): Combining temporal difference learning with game-tree search. In *Proceedings of the Ninth Australian Conference on Neural Networks*, pages 168–172.

Beasley, D., Bull, D. R., and Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125.

Cliff, D. and Miller, G. (1995). Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Third European Conference on Artificial Life*, pages 200–218.

Elo, A. E. (1986). *The rating of chessplayers, past and present*. Arco Pub., New York, 2nd edition.

Funes, P., Sklar, E., Juillé, H., and Pollack, J. B. (1998). Animal-animat coevolution: Using the animal population as fitness function. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation*

of Adaptive Behavior, pages 525–533, University of Zurich. MIT Press, Cambridge, MA.

Hillis, D. (1991). Co-evolving parasites improves simulated evolution as an optimization procedure. In C. Langton, C. Taylor, J. F. and Rasmussen, S., editors, *Artificial Life II*. Addison-Wesley, Reading, MA.

Joe, H. (1990). Extended use of paired comparison models, with application to chess rankings. *Applied Statistics*, 39(1):85–93.

Juillé, H. and Pollack, J. B. (1996). Dynamics of co-evolutionary learning. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 526–534. MIT Press.

Koza, J. (1992). *Genetic Programming*. MIT Press, Cambridge.

Newborn, M. (1996). *Kasparov vs. Deep Blue: Computer Chess Comes of Age*. Springer Verlag, New York.

Pollack, J. and Blair, A. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240.

Rosin, C. D. (1997). *Coevolutionary Search Among Adversaries*. Ph.D. thesis, University of California, San Diego.

Rosin, C. D. and Belew, R. K. (1995). Methods for competitive co-evolution: finding opponents worth beating. In *Proceedings of the 6th international conference on Genetic Algorithms*, pages 373–380. Morgan Kaufman.

Sims, K. (1994). Evolving 3d morphology and behavior by competition. In Brooks, R. and Maes, P., editors, *Proceedings 4th Artificial Life Conference*. MIT Press.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Tesauro, G. (1990). Neurogammon wins computer olympiad. *Neural Computation*, 1:321–323.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277.

Walt Disney Studios (1982). Tron. movie.