The Internet as a Virtual Ecology: Coevolutionary Arms Races Between Human and Artificial Populations

COMPUTER SCIENCE TECHNICAL REPORT CS-97-197

Pablo Funes, Elizabeth Sklar, Hugues Juillé and Jordan Pollack

Volen Center for Complex Systems
Brandeis University
415 South St., Waltham MA 02254 USA
{pablo,sklar,hugues,pollack}@cs.brandeis.edu

Abstract

In this paper, we propose that learning complex behaviors can be achieved in a coevolutionary environment where one population consists of the human users of an interactive adaptive software tool and the "opposing" population is artificial, generated by a coevolutionary learning engine. We take advantage of the Internet, a connected community where people and software coexist. A new kind of adaptive agent can exploit its interactions with thousands of users—inside a virtual "niche"—to learn in a coevolutionary human-robot arms race. Our model is Tron, a simple dynamic game where introspective self-play quickly leads to collusive stagnation. We describe an application where thousands of small programs are sent to play with people through the Java interpreter running in their web browsers. The feedback provided by these agents is collected in our server and used to augment an ever improving fitness landscape for local robot-robot games. Speciation and fitness sharing provide diversity to challenge humans with a variety of different strategies. In this way, we obtain an evolving environment where human as well as artificial adaptation are simultaneously taking place.

1 Introduction

1.1 Evolutionary Learning

Evolutionary learning methods such as Genetic Algorithms (GA’s) provide general-purpose approaches to the problem of machine learning. With them we can build engines that create a succession of partial results whose success at dealing with a problem —hopefully —increases over time.

Evolutionary learning calls for three basic ingredients: (a) a representation that is capable of encoding each candidate solution, (b) mutation and crossover operators that can be applied to such a representation, and (c) a *fitness function* that is the standard measure against which to test each candidate solution during the iterative process [8, 7].

In natural as in artificial evolution, a population moves toward fitness optimality while maintaining variation over all the dimensions of the genetic space, including those dimensions that are not being selected. This iterated generation/selection process is regulated in genetic algorithms by the fitness function, which must be applied to each single individual as it is produced.

1.2 Too Many Fitness Evaluations

The need to evaluate the fitness of a large number of individuals is a critical factor that restricts the range of application of GA's. In many domains, a computer can do these evaluations very fast; but in others, the time spent by this process may render the GA solution impractical. Examples of the latter case include a computer playing a game with people and trying to learn from experience or a robot attempting to complete a task in the physical world.

Robots —provided they are reliable enough —can run repeated trials of the same experiment over a long time in order to learn using evolutionary computation techniques. Floreano and Mondada [4, 5] run their robots for several days in order to evolve controllers for basic tasks. Most evolutionary roboticists have preferred to rely on computer simulations to provide them with faster evaluations, but the crafting of appropriate simulations is also very difficult [12].

1.3 Using Humans for Fitness

Evolution of *interactive* (that which is used by humans) adaptive software faces similar difficulties. On the one hand, it is nearly impossible to design a fitness function whose virtual environment will prepare it to meet the enormous variation of human responses. On the other hand, if users themselves are asked to provide fitness evaluations, then hundreds of trials will be necessary and the process will take a long time. Humans—unlike robots—get tired of repetitive tasks. Humans act adaptively; they may react differently each time when faced with the same situation more than once. If users provide fitness evaluations, adaptive software would need to be able to filter out such sources of “noise” provided naturally by human users.

Our theory is that the Internet, with millions of human users, could be fertile ground for the evolution of interactive adaptive software. Instead of relying on a few selected testers, the whole community of users together constitute a viable gauge of fitness for an evolutionary algorithm that is searching to optimize its behavior.

1.4 The problem of generalization

Given that it is an arduous task to evaluate fitness for multitudes of individuals, is it possible to limit the search to just a few? The problem of lack of generalization or lack of transfer to a more general environment will defeat this alternative. The fact that an algorithm performs well in a certain group of test cases does not usually mean that it will generalize to a wider range of situations.

Neural networks are thought to have generalization capabilities [3 ch. 3, 23], successfully inducing, for example, a good backgammon player from a set of suggested moves [21]. Supporters of the Genetic Programing paradigm [11] will suggest that this may be the case for GP as well [17]. In [10], Juillé and Pollack argue that the dynamics of coevolutionary fitness help them get “perspicacious” solutions to a problem of recognizing 194 points arranged in a spiral pattern. The GP function they obtain indeed defines two roughly spiral surfaces that continue outside the boundary of the original test points.

1.5 Learning game playing

Game playing is one of the traditional domains of AI research. Ever since Samuel’s early experiments with checkers [19], we have hoped that the computer would be able to make good use of experience, improving its skills by learning from its mistakes and successes.

In his work with the game of backgammon, Tesauro began collecting samples from human games to provide a fitness measure for training neural networks [21]. Later, he abandoned this methodology and used introspective *self-play* [22]. Of the earlier approach, he argued that “building human expertise into an evaluation function [...] has been found to be an extraordinarily difficult undertaking” [22, p. 59].

Learning to play a game by self-play involves a problem of transfer as well. The fitness landscape (even in the coevolutionary case, where the “landscape” is redefined in every generation) might be an insufficient sample of the larger problem defined by the whole game and the way humans approach it. While learning backgammon [22, 15] is a success for coevolution, this same approach has failed in most other cases.

Real-time, interactive games (e.g. video games) have distinctive features that differentiate them from the better known board games. Koza [11 ch. 12] and others [17] evolved players for the game of Pacman. There has been important research in pursuer-evader games [16, 13] as well as contests in simulated physics environments [20]. But these games do not have human participants, as their environments are either provided by the game itself, or emerge from coevolutionary interactions inside a population of agents.

1.6 Coevolution of Interactive Adaptive Software

In this paper, we propose that learning complex behaviors can be achieved in a coevolutionary environment where one population consists of the human users of an interactive adaptive software tool and the “opposing” population is artificial, generated by a coevolutionary learning engine. An artificial “niche” must be created in order for the arms race phenomenon to take place, requiring that:

- A sufficiently large number of potential human users must exist.
- The artificial population must provide a useful environment for the human users, even when -in the early stages- many instances perform poorly.
- A (crude) estimation of the artificial population's performance must be extractable from its interaction(s) with the human users.

The collective environment of the Internet provides a virtual ecology where a kind of arms race between human and artificial populations may be possible.

As such, we have created an experimental Java-based learning environment on the Internet for the game called **Tron**, meeting the requirements mentioned above. First, we know that there is considerable interest in Java-based games in the Internet community. Second, our earlier experiments with Tron have shown us that, by self-play, we can produce players that are not entirely uninteresting when faced by humans. And third, each round of Tron results in a performance measure: a win, loss or tie. We started this experiment with the hope that the average over many games against a variety of human users would provide a useful fitness measure, even when a random population of game players is expected to have enormous variability.

2 System Description

2.1 Tron (Light Cycles)

Tron, a 1982 movie from Walt Disney Studios, showed a game in a virtual world where two futuristic motorcycles ran at constant speeds, making only right angle turns and leaving solid wall trails behind them. As the game advanced, the arena became filled with walls and eventually one opponent ran to its death by crashing into a wall. Also known as "Light Cycles", this popular game has been implemented on all kinds of computers with varying rules and configurations.

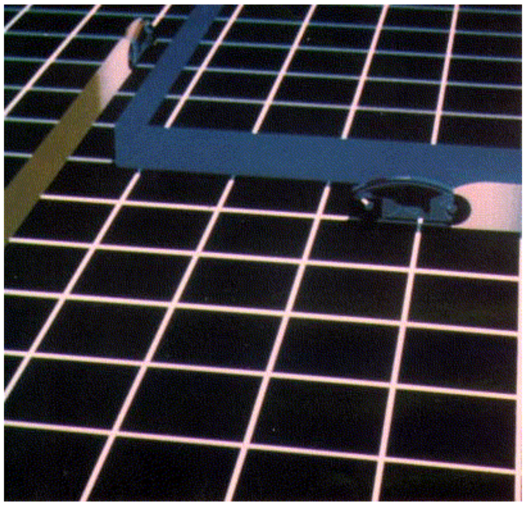

Figure 1. Still from the movie *Tron*

In our interpretation, the motorcycles are abstracted and represented only by their trails. Both players start in the middle region of the screen, moving in the same direction. The edges of the arena are not considered “walls”; players move past them and reappear on the opposite side, thus creating a “wraparound”, or toroidal, game arena. The size of our arena is 256×256 pixels.

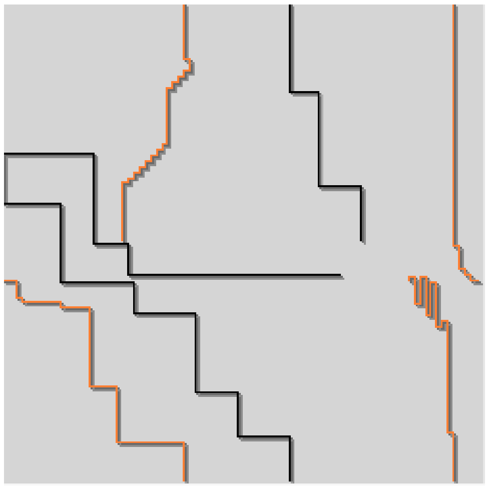

Figure 2. The Tron game arena. Both players need to avoid trails. The edges are connected.

2.2 Artificial (Robot) Players

We have crafted artificial or *robot* players with the capability to perceive the world in eight directions. Each robot is provided with eight simple “sensors”; each one evaluates the distance in pixels from the current position to the nearest obstacle in eight cardinal directions: Front, Back, Left, Right, FrontLeft, FrontRight, BackLeft and BackRight. Every sensor returns a maximum value of 1 for an immediate obstacle (i.e. a wall in an adjacent pixel), a lower number for an obstacle further away, and 0 when there are no walls in sight.

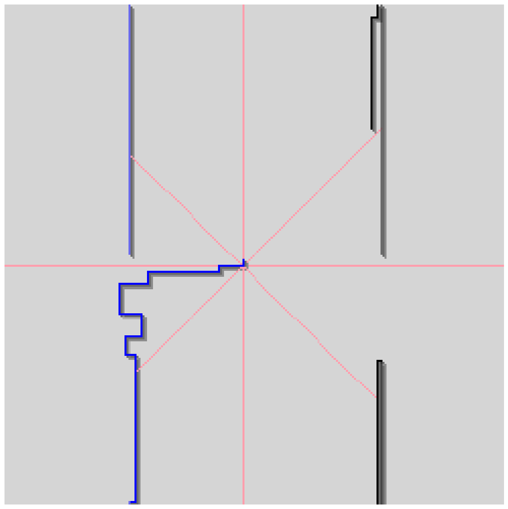

Figure 3. A Tron robot has eight sensory inputs.

2.3 Learning Tron by Self-Play

In earlier exploratory experiments [6], we used a genetic algorithm to learn the weights of a perceptron network to play tron. It became evident that while this simple architecture is capable of coding players that could perform interestingly when facing human opponents, such “good” weights were difficult to find in evolutionary or coevolutionary scenarios. Collusion [14] was likely to appear in most evolutionary runs in the form of “live and let live” strategies such as that shown in Figure 4.

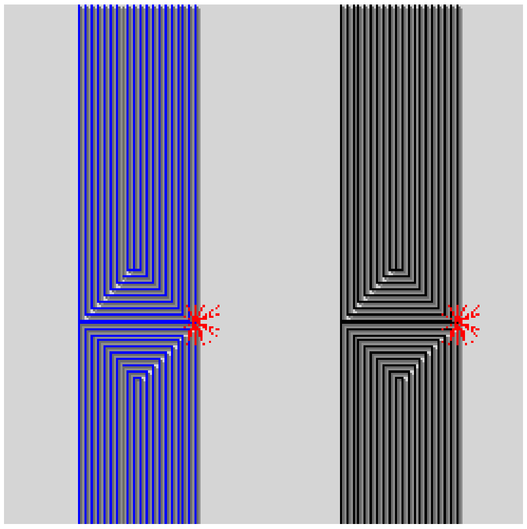

Figure 4. “Live and let live”: Two robot Tron players make tightest spirals in order to stay as far from the opponent as possible. This form of collusion is a frequent suboptimal equilibrium that prevents learning robot strategies by self-play in a coevolutionary arms race.

2.4 GP representation for Tron Robots

In the present study, we use Genetic Programming (GP) [11] as a means for coding artificial Tron players. The set of terminals is {_A,_B,..., _H (the eight sensors) and $\Re$ (random constants between 0 and 1)}. The functions are {+, -, * (arithmetic operations),% (safe division), IFLTE (if $a \leq b$ then-else), RIGHT (turn right) and LEFT (left turn)}. A maximum depth of 7 and a maximum length of 512 limit the valid s-expressions.

A robot player reads its sensors and evaluates its s-expression every three steps during a game. If a RIGHT or LEFT function is output, the robot makes the corresponding turn; otherwise, the robot will keep going straight.

2.5 The Tron Applet

We have written a Java applet and launched our game on the Internet. The architecture of the system takes advantage of the Java's ability to run a "client" on the user's local machine and a "server" on our host (Web server) machine. As shown in Figure 5, the Java Applet runs on the user's local machine; the Foreground and Background servers execute on our machine.

The Tron applet receives a GP s-expression (from our server), representing a Tron-playing strategy. The applet runs, playing one game with the human user, until a crash occurs. When the game ends, the applet opens a connection to our server, reports the results of the game and receives a new s-expression for the next game. This cycle continues until the human decides to quit playing.

We use a two-level server architecture to maintain two separate Tron-playing robot populations simultaneously, as illustrated in Figure 5. The Foreground Server plays games with humans, while the Background Server engages in self-play to filter brand new robot players that will be incorporated into the foreground population when the foreground process is ready for a new generation.

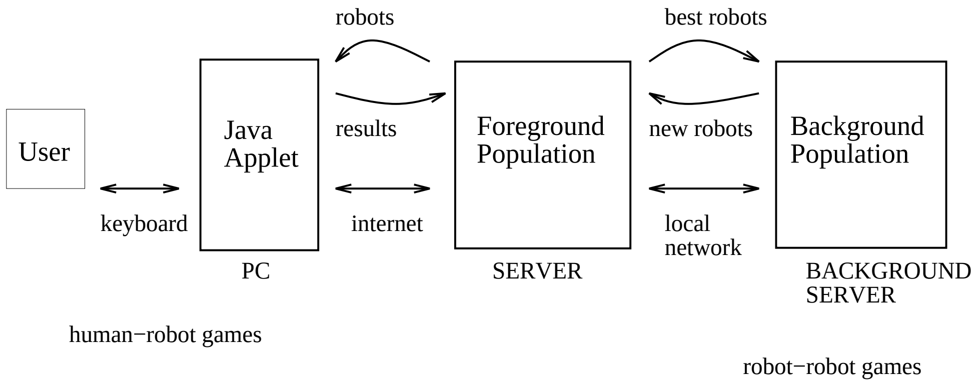

Figure 5. Scheme of information flow

The interaction between robots and humans occurs at a relatively slow pace: a few hundred games may happen every day. An evolving population idles, waiting while these games are accumulated to evaluate its individual agents. The role of the background process is to replace the usual reproduction stage, exploiting all this idle time to produce the best robot players that self-play can give us. Instead of raw crossover and mutations, the new individuals of the population will have been trained and filtered through self-play.

2.6 Foreground Server

The foreground portion of the Tron system controls the interplay between robots and humans. It maintains a population of 100 robots. Every time a player requests a new game, the server supplies one of these robots at random. When a user finishes a game, the foreground process saves the outcome in its database.

A generation in the foreground process lasts until all 100 robots have played a minimum number of games. The new robots that are playing for the first time in the current generation play a minimum of 10 games, while the "veterans" that have survived from previous generations play only 5 games. When all robots have completed their minimum number of games, the generation is finished and the next generation is started.

To start a new generation, the 100 current robots are sorted by fitness. The worst 10 are eliminated and replaced by 10 fresh robots, supplied by the background process. A new generation begins.

The fitness of robots is a shared fitness measure designed to promote speciation[2, 9] by giving points for doing better than average against a human player, and negative points for doing worse than average. For each robot *r*, the fitness is calculated as

$$fitness(r) = \sum_{\{h : \text{played}(h,r) > 0\}} \left(\frac{\text{lost}(h,r)}{\text{played}(h,r)} - \frac{\text{lost}(h)}{\text{played}(h)} \right) \left(1 - e^{-\frac{\text{played}(h)}{10}} \right) \quad (1)$$

where lost(*h*,*r*) is the number of games lost by each human opponent *h* against *r*, played(*h*,*r*) is the total number of games between the two, lost(*h*) is the total games lost by *h* and played(*h)* is the number of games that *h* has played. The measure is summed across all games played, not just those that belong to the current generation. The exponential factor on the right is a confidence measure that devalues the average scores of humans that have only played a few games.

An inherent exploitation/exploration bias occurs when we make 10 new robots play 10 times per generation and 90 veteran robots play 5 times each. This means that 20% of the games are played by the rookie robots, who have not been evaluated yet.

2.7 Background Server

The role of the background process is to supply the foreground process with good robot players for each new generation: the best that can be produced given the results that have been accumulated. We proceed as follows: every time the foreground population begins a new generation, the background receives the 15 best fit robots from the foreground process. This group of 15 robots comprises part of a training set against which a new population of 1000 random robots is generated and evolved. When the foreground process finishes one generation, it receives the 10 best robots that emerge from this procedure, adding them to its own population, and the cycle restarts (Figure 6).

The background process plays all the individuals in its population against the training set of 25 robots. Fitness is evaluated, and the bottom half of the population is replaced by random mating with crossover of the best half. The fitness function is defined as follows,

$$\mathrm{Fitness}_T(r) = \sum_{\{r' \in T:\ \mathrm{points}(r, r') > 0\}} \frac{\mathrm{points}(r, r')}{\mathrm{lost}(r')} \qquad (2)$$

where *T* is the training set, points(*r, r'*) = {0 if *r* loses against *r'*, 0.5 if they tie and 1 if *r* wins} and lost(*r'*) is the number of games lost by *r'*. Thus we give more points for defeating good players than bad players.

As indicated above, the training set consists of two parts. The first 15 members are fetched from the foreground process. The remaining 10 members of the training set are replaced each generation with a fitness sharing criteria. The new training set *T'* is initialized to the empty set and then new members are added one at a time, choosing the highest according to the following shared fitness function:

$$\mathrm{Fitness}_{T, T'}(r) = \sum_{r' \in T} \frac{\mathrm{points}(r, r')}{\left(1 + \sum \{\mathrm{points}(r'', r') : r'' \in T'\}\right)} \qquad (3)$$

This selection function is adapted from [18] and acts to decrease the relevance of a case that has already been “covered”, that is, when there is already a player in the training set that beats it.

When the best players from the foreground population re-enter the background as members of the training set, their genotype is isolated: they do not reproduce explicitly. They do so only implicitly as they disfavor players they can beat, and favor players that beat them. This is an arbitrary implementation decision whose effects we have not addressed yet (see section 5).

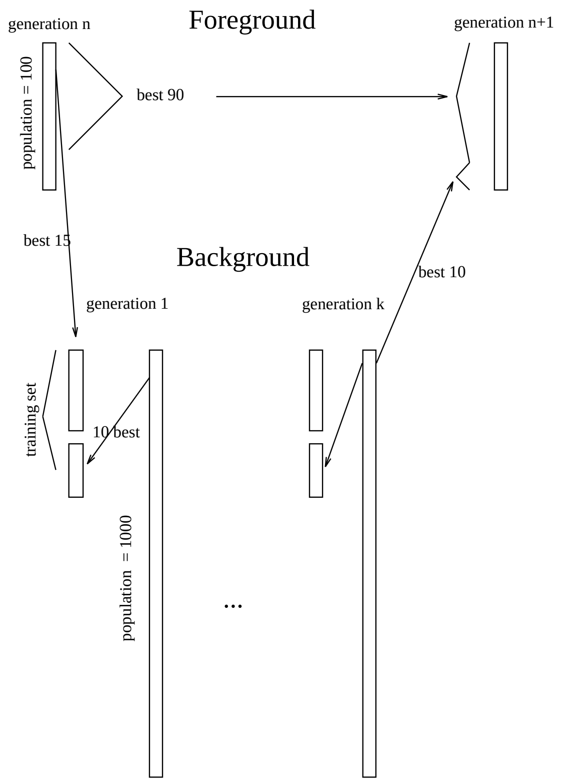

Figure 6. Scheme of foreground and background evolutionary populations.

3 Results

Our server has been operational for two months, and so far we have collected the results of 22,494 games. Twenty thousand games is a small number compared with the number of games played by the background process, which plays 25,000 games per generation, approximately one generation per hour. We are letting the system continue to run, but present here analysis of the data obtained thus far.

3.1 Robot Learning

Our basic performance measure is the *win rate*, that is, the fraction of games that the Tron-playing robots have won. The average win rate over the total number of games played is 0.33, meaning that 33% of all games completed have resulted in robot victories.

The following graph uses a sampling rate of 1000 to plot the evolution of the win rate over time.

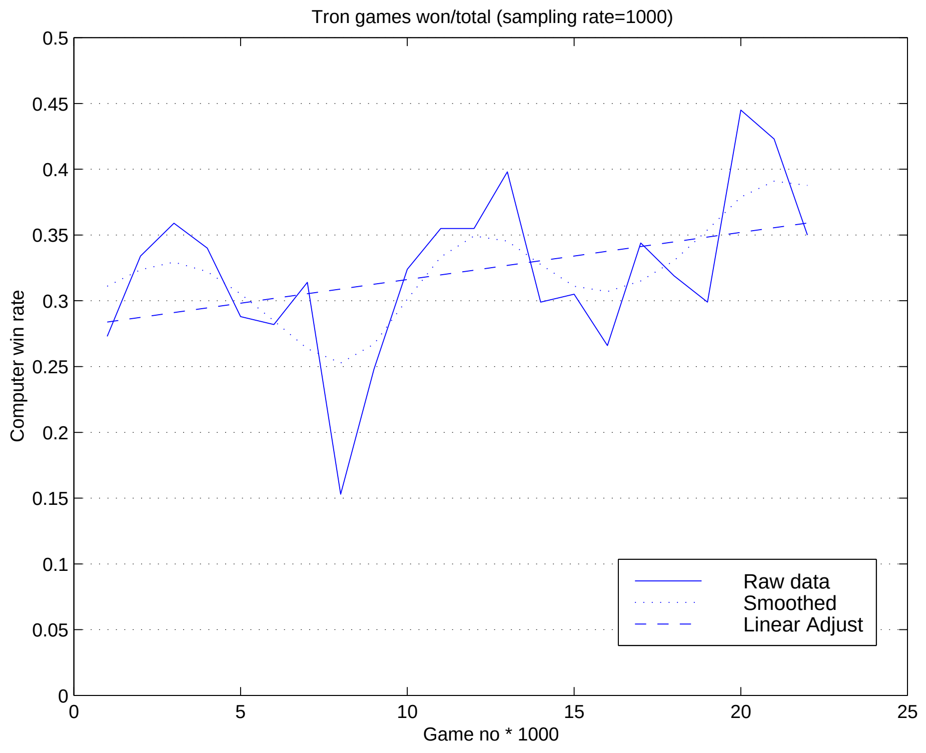

Figure 7. Evolution of the win rate[1].

The graph in figure 7 illustrates two important factors. First, there are oscillations. This is a natural phenomenon in a coevolutionary environment, and occurs here more noticeably since one of the evolving populations consists of randomly selected human players. Each of the 416 individuals sampled here has a different level of expertise[2], has played a different number of games, and so on[3].

The second important feature is that there is a visible trend toward improvement. The winning rate has gone up from roughly 28% to 35% over the time that the system has been operational.

3.2 Human Learning

Humans learn very fast, so the Tron system, like all interactive adaptive software, is chasing a moving target. The graph in Figure 8 illustrates the human learning rate. It shows the win rate of our system against all the first games of new players, then all their second games, and so on.

1. Smoothing obtained by convolution with $e^{-\left(\frac{x}{\alpha}\right)^2}$, normalized. $\alpha = 2000$.
2. Another variable factor is the speed of the game in the client machine, which goes down due to the low speed of Java interpreters inside Internet browsers.
3. We have called the low peak at 8000 games the “khith anomaly” because it consists mostly of games by the same one person, with login name “khith”.

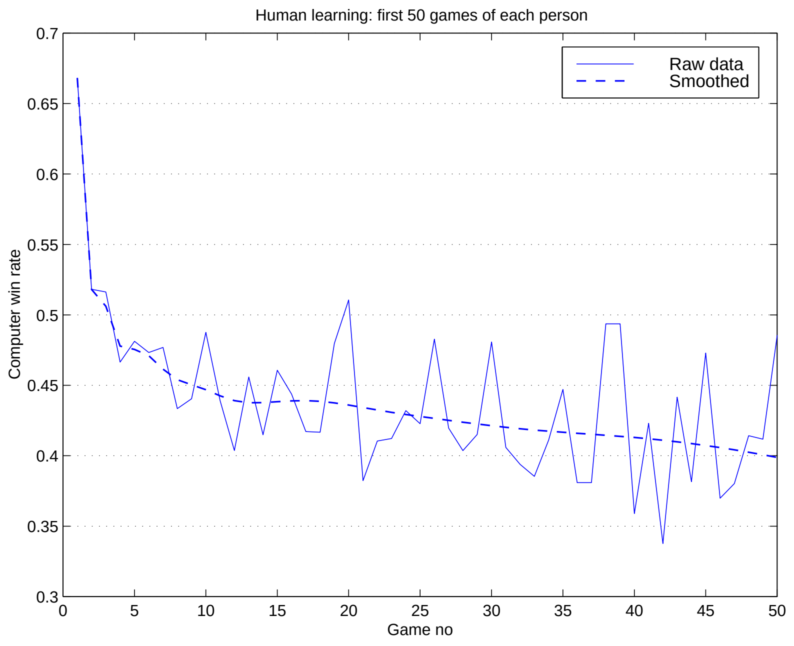

Figure 8. Averaged learning of human players[1]

The graph shows how quickly learning occurs in humans. The robots win 67% of the first games, but then only 52% of the second games, etc. By the 10th game, the rate descends to 44%. (We have used smoothing to average over the noisy data toward the right side of the graph.)

3.3 Overall Improvement

It would be desirable to factor out the influence of human learning from Figure 7 in order to visualize the learning rate of the Tron-playing robots. To do this, we examine only the first 10 games of every person that has played 10 or more games with our system, as shown in Figure 9.

1. Smoothing obtained by convolution with $e^{-\left(\frac{\ln x}{\alpha}\right)^2}$, normalized. $\alpha = 0.25$.

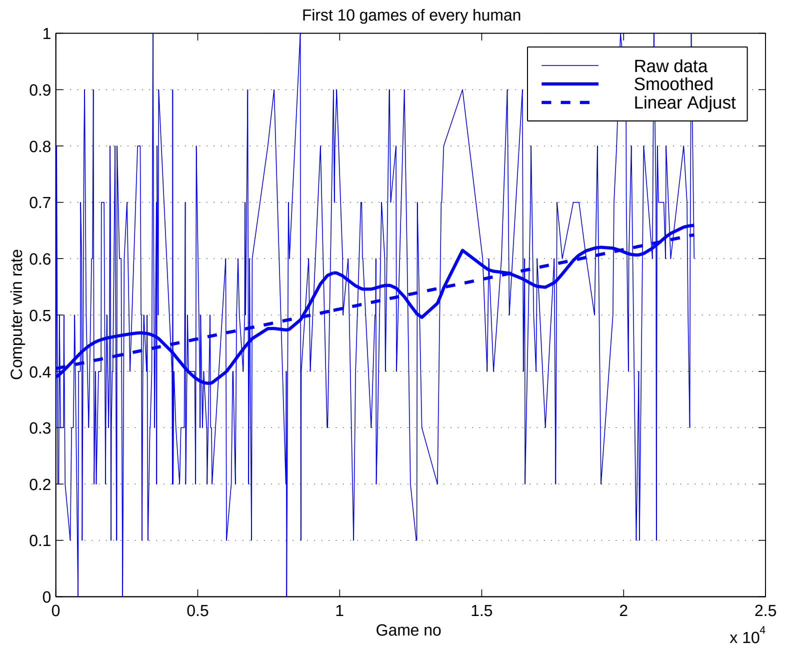

Figure 9. Performance of Tron during first 10 games with each opponent[1]

The raw data is very noisy since each point is the score of a single human. Some players log into the Tron system for the first time and win their first 10 consecutive games, showing a computer winning ratio of 0. Others lose their first 10 games and showing a computer winning ratio of 1. Most players perform somewhere in between. But both smoothing and a linear adjustment show that there is an overall trend toward robot improvement. Starting at around 40%, the computer winning ratio has increased to nearly 64%.

3.4 The Best Robots

Using our raw approximate measure (the win ratio), we obtain a score for each one of our robot players. But since luck determines the opponents of each robot, this ratio will only be an estimation that approaches a true value as the number of games played increases. In table 1, we have listed the best robot players, out of those who have played at least 50 games.

1. Smoothing obtained by convolution with $e^{-\left(\frac{x}{\alpha}\right)^2}$, normalized. $\alpha = 1024$.

Ranking	Robot Id No.	Wins/ Games	Games Played
1	330008	0.74	54
2	330001	0.74	54
3	330003	0.71	53
4	330004	0.70	56
5	280004	0.67	100
6	280006	0.62	94
7	330006	0.61	53
8	320002	0.61	58
9	170002	0.59	194
10	170008	0.58	185
...			
16	100009	0.56	243
...			
61	10053	0.35	288

Table 1: Best robots

At the moment, this table is dominated by strategies born in generation 33 (the current generation is 37). There are also two robots born in generation 28 and two more from generation 17. The rate for the four 33rd generation champions will probably decrease as the population of human players adapts to them. The most experienced robots above a 50% success rate are 5 individuals born in generation 10, who have won more than half of their games. In contrast, the best robot from the original population (generation 1) has played 288 games with a success rate of just 35%.

4 Strategy Analysis

4.1 The current champion

Following is the GP s-expression for the current champion, R.330008:

```
(+ (IFLTE _C _A (IFLTE (% 0.44444 (IFLTE _F _A (% (IFLTE
(RIGHT_TURN) 0.66667 _C (* 0.41270 0.84127)) (LEFT_TURN))
0.46032)) _F (% (IFLTE (RIGHT_TURN) 0.66667 (RIGHT_TURN) (*
0.41270 0.84127))(LEFT_TURN)) (LEFT_TURN)) _A) _E)
```

This can be roughly reduced to pseudocode as:

```
if FRONT >= LEFT then go straight
else
    if FRONT >= REAR_RIGHT then turn right
    else if 0.97 <= REAR_RIGHT then turn right
    else turn left
```

The code still leaves us wondering what the strategy is. Why should an obstacle closer in the front than to the left be an indication to go straight, when common sense dictates that it should be the other way around? The complex behavior of this player arises from its interactions with its changing environment.

When inserted in a Tron arena, this code begins with a defensive strategy: it does tight turns to keep itself as far away from its opponent as possible. It will keep doing these closed turns as long as its opponent does the same. It will deviate from this conservative behavior, if its opponent deviates, and switch to a more aggressive strategy. This technique protects the species (it is not surprising that 5 out of the 10 best players from its generation behave similarly) while trying to inflict damage on other strategies.

The Tit-for-Tat strategy described by Axelrod [1] for the Iterated Prisoner's Dilemma (IPD) game uses a similar approach. It maintains cooperation for as long as the other player does. But it retaliates after a defection, defecting as well. IPD induced this behavior by assigning more total payoff to two players cooperating than to one defection plus one win. In our case, the outcome of cooperation is a tie, which splits one point between the two players: the result is a zero-sum payoff matrix. Given this, one should not be surprised to obtain, from fitness sharing, both cooperating and non-cooperating species.

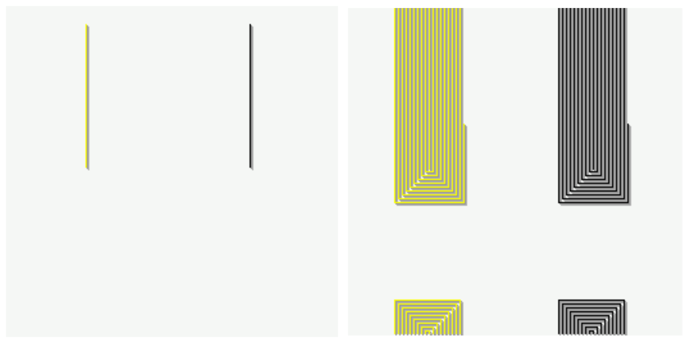
Figure 10. Next of kin: R. 330007 (gray) vs. R. 330008 (black) begin going straight (left), then make tight turns to stay as far away from each other as possible (right). This game ends eventually in a draw after the game arena is filled entirely.

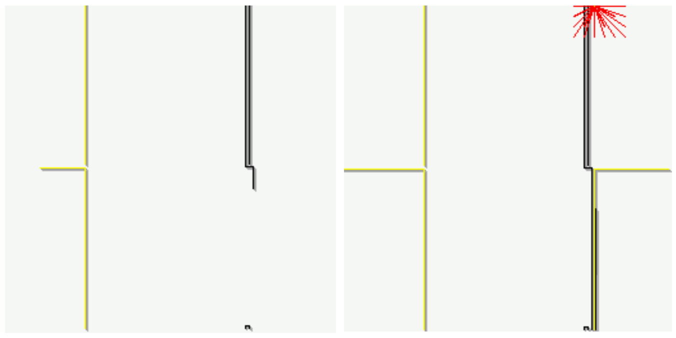

Figure 11. Tit-for-Tat: R. 330003 (gray) vs. R. 330008 (black). R. 330008 begins cooperating as in Figure 10, but as soon as 330003 deviates, it will change too, caging its opponent in a dead end to win the game.

Playing a few games manually against this robot shows that it performs some clever maneuvers, but also makes mistakes. Our conclusion here is that the Tron-playing robots have not yet evolved to play as well as they could against humans.

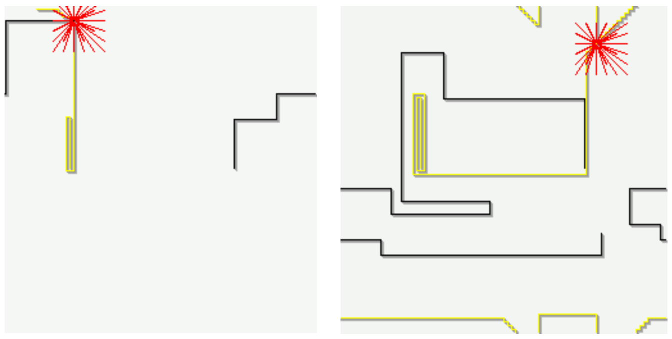

Figure 12. Humans are difficult: R. 330008 (gray) vs. one of us (black). On the left, the robot seems to be making harmless loops. When the human approaches, however, it cuts its path and makes him lose. On the second game we see R. 330008 escape a trap but later on loss unnecessarily.

4.2 Cooperation and Speciation

The following table summarizes the balance of robot forces at the end of generation 33:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	t	t	3	4	5	6	7	8	9	10
2		t	3	4	5	6	7	8	9	10
3			C	C	5	6	7	8	9	10
4				C	5	6	7	8	9	10
5					C’	C’	C’	C’	C’	C’
6						C’	C’	C’	C’	C’
7							C’	C’	C’	C’
8								C’	C’	C’
9									C’	C’
10										C’

Table 2: Games between the top 10 players of generation 33.

Table 2 indicates the outcomes of games between the 10 players of the background process that were selected to be incorporated as rookies in generation 33. The robot number indicates which player won. A “t” indicates a tie where no cooperation was observed; C and C’ are cooperative ties that fill the entire arena.

We observe that three species survived to be represented in this sample. Players 1 and 2 are non-cooperative; they exhibit aggressive behavior even playing with themselves. They are beaten by players 3-10. Players 3 and 4 cooperate with each other, but not with anyone else. Players 5-10 are cooperative only between themselves. In contrast, generation 1 contains only one species among its top 10 players. We have not made a thorough analysis of cooperative and speciation phenomena, but a quick glance through a few generations shows us that cooperative and non-cooperative species are present in most of them.

5 Conclusions

We have shown that evolutionary techniques can be successfully applied in the context of an environment of connected human users. The variable input of random players with diverse expertise and interests exhibits exploitable average trends. The trend toward collusion through cooperation in coevolutionary agents can be broken in adaptive software environments by incorporating the raw fitness evaluation emerging from its end-users.

Once an ecological niche is created, and by this we mean that a piece of adaptive software like our Tron game must attract a collectivity of users, a coevolutionary arms race will occur where human and artificial learning chase each other. We can observe human learning rates by averaging all performances of players at same levels of experience, and we can observe agent learning rates by looking at their performances against new incoming opponents along time.

The exploration vs. exploitation trade-off is linked to the problem of creating a niche. On one side we want to offer users our best level of performance, but on the other we need to take risks with un-tested versions. We address this problem in two ways. First, we have set our parameters so that only 20% of the games will be played by rookies. Second, we exploit the slow pace of human interactions, evolving new robots in the background, so that our novice robots will be as good as possible.

Regarding the algorithmic setup, several details were chosen arbitrarily. The way the background population is reset every time the front end completes a generation is questionable, and other approaches may be better, such as not resetting it at all, or doing it when the dynamics of its own evolutionary process indicate so. Genetic isolation between front and back end populations is an arguable factor, since it prevents new players from capitalizing on older genotypes.

The issue of diversity is a central point throughout our coevolutionary setup that needs to be emphasized. Shared fitness functions stimulate diversity by promoting original candidates in every new generation. Resetting the background population and genetic isolation are also promoting diversity, creating new solutions from scratch in each cycle.

Diversity and speciation are key in addressing the problem of transfer, or lack of generalization, by widening the spectrum of solutions produced by an introspective learner that needs strategies to cope with a wider environment than the one defined by a limited fitness function that describes only a few points of a larger problem.

The shared fitness function of robots vs. humans (eq. 1) addresses the problem of diversity of levels of expertise among the user population, but also attempts to prevent cooperative phenomena at this level, as robots are given points not for winning or losing, only for doing better or worse than the average of the other robots.

As perceived by each individual user, our virtual Tron is playing differently every time. The switching of random Tron programs for each individual game is responsible for this effect. At the same time, the overall level of play is going up over the days. This heterogenous behavior is part of the success; when we say that a certain robot is winning 74% of its games, this is a valid indicator of its level of play *given the collective of all other robots*. If we were to send the same identical robot over and over, humans would quickly learn strategies against it, and the system as a whole would be boring.

Finally, there are severe representational limitations to our Tron agents coming from the little perception they have. Whereas humans observe the entire state of the game in every screenshot, robots only “see” the nearest object in eight fixed directions. They have no perception of the position and heading of their opponents, and are incapable of analyzing the game board as a whole in order to make their decisions. Still, this work shows that computer learning can be successfully applied to the domain of complex dynamic games where human beings are the current experts.

6 References

[1] Axelrod, R. M. (1984) *The Evolution of Cooperation*. New York, Basic Books.

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

[3] Darwen, P. J. (1996) *Co-evolutionary Learning by Automatic Modularisation with Speciation*. University of New South Wales, 1996.

[4] Floreano, D. and Mondada, F. (1994). Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural Network Driven Robot. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson (Eds.), *From Animals to Animats III*, Cambridge, MA. MIT Press.

[5] Floreano, D. and Mondada, F. Evolution of Homing Navigation in a Real Mobile Robot. *IEEE Transactions on Systems, Man, and Cybernetics--Part B: Cybernetics*, 26(3), 396-407, 1996.

[6] Funes, Pablo (1996). The Tron Game: An experiment in Artificial Life and Evolutionary Techniques. Unpublished.

[7] Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

[8] Holland, John H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.

[9] Juillé, H. and Pollack, J. (1996) Dynamics of Co-evolutionary Learning. *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior.* MIT Press.

[10] Juillé, H. and Pollack, J. (1996). Co-evolving Intertwined Spirals. in *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, MIT Press.

[11] Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

[12] Mataric, M and Cliff, D. (1996). Challenges In Evolving Controllers for Physical Robots. In *Evolutionary Robotics,* special issue of *Robotics and Autonomous Systems,* Vol. 19, No. 1. 67-83.

[13] Miller, G. F. and Cliff, D. (1994) Protean Behavior in Dynamic Games: Arguments for the Co-Evolution of Pursuit-Evasion Tactics. *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94)*. D Cliff, P. Husbands, J.-A Meyer and S W Wilson, eds. MIT Press Bradford Books, pp.411--420.

[14] Pollack, J. B., and Blair, A.D. (1997) Why did TD-Gammon work? *Advances in Neural*

Information Processing Systems 9. 10-16.

[15] Pollack, J. B., Blair, A. and Land, M.(1996). Coevolution of A Backgammon Player. *Proceedings Artificial Life V,* C. Langton, (Ed), MIT Press.

[16] Reynolds, C.W. (1994). Competition, Coevolution and the Game of Tag", *Proceedings of Artificial Life IV.* R. Brooks and P. Maes, eds. MIT Press.

[17] Rosca, J. P. (1996). Generality versus Size in Genetic Programming. *Proceedings of the Genetic Programming 1996 Conference (GP-96).* The MIT Press.

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

[19] Samuel, A. L. (1959) Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development,* vol. 3, 210-229.

[20] Sims, K. (1994) Evolving 3D Morphology and Behavior by Competition. *Artificial Life IV Proceedings*, MIT Press.

[21] Tesauro, G. (1989) Neurogammon Wins Computer Olympiad. *Neural Computation I,* 321-323.

[22] Tesauro, G. (1995) Temporal difference learning and TD-Gammon. *Communications of the ACM,* 38(3): 58-68.

[23] Wolpert, D. H. (1990). A Mathematical Theory of Generalization. *Complex Systems 4*: 151-249.