On Evolving Robust Strategies for Iterated Prisoner's Dilemma

P. J. Darwen and X. Yao

Department of Computer Science
University College, University of New South Wales
Australian Defence Force Academy
Canberra ACT 2600 AUSTRALIA
Email: `darwen@canth.cs.adfa.oz.au`

16 November 1993

Abstract

Evolution is a fundamental form of adaptation in a dynamic and complex environment. Genetic algorithms are an effective tool in the empirical study of evolution. This paper follows Axelrod's work [2] in using the genetic algorithm to evolve strategies for playing the game of Iterated Prisoner's Dilemma, using co-evolution, where each member of the population (each strategy) is evaluated by how it performs against the other members of the current population. This creates a dynamic environment in which the algorithm is optimising to a moving target instead of the usual evaluation against some fixed set of strategies. The hope is that this will stimulate an "arms race" of innovation [3].

We conduct two sets of experiments. The first set investigates what conditions evolve the best strategies. The second set studies the robustness of the strategies thus evolved, that is, are the strategies useful only in the round robin of its population or are they effective against a wide variety of opponents?

Our results indicate that the population has nearly always converged by about 250 generations, by which time the bias in the population has almost always stabilised at 85%. Our results confirm that cooperation almost always becomes the dominant strategy [1, 2]. We can also confirm that seeding the population with expert strategies is best done in small amounts so as to leave the initial population with plenty of genetic diversity [7].

The lack of robustness in strategies produced in the round robin evaluation is demonstrated by some examples of a population of naïve cooperators being exploited by a defect-first strategy. This causes a sudden but ephemeral decline in the population's average score, but it recovers when less naïve cooperators emerge and do well against the exploiting strategies. This example of runaway evolution is brought back to reality by a suitable mutation, reminiscent of punctuated equilibria [12]. We find that a way to reduce such naïvity is to make the GA population play against an extra, static, high-quality strategy (not part of the GA population), as well as all the rest of the population. The strategies thus produced perform better against opponents that were included in the round robin (as expected) and, more significantly, better against opponents that were *not* included. That is, robustness is improved.

1 Introduction

Evolution works. In nature, evolution produces increasingly fit organisms in uncertain environments. Evolution is a proven form of adaptation to deal with a dynamic and complex environment. Emulating evolution shows great promise for machine learning.

The genetic algorithm (GA) [10] is a search algorithm that emulates some basic features of natural evolution. The GA works well in searching spaces with the same difficulties faced by machine learning when searching the space of possible strategies in games of conflict [4].

We use the GA to learn to play the iterated prisoner's dilemma (IPD) [1], a simple two-player non-zero-sum game which is widely studied in such diverse fields as machine learning, economics, political science, and mathematical game theory.

1.1 Background

Axelrod [1] investigated evolution and the IPD, and simulated a population (of strategies) in which each strategy plays IPD with every other individual [2, p 38]. In such a system, the environment (made up by the members of the evolving population) is constantly changing. He found that this dynamic environment produced strategies that performed very well against their population. Fogel [5] studied an almost identical system.

This begs a question: can a strategy evolved from competing against the its own population succeed against strategies not in its own population? Drawing an analogy from biology, can a successful predator on one continent be effective against the successful predators of some other, isolated continent? In machine learning, we want to produce strategies that are effective against all possible opponents, not just those in the local population. This desirable quality is "robustness".

Lindgren provided a radical insight [12]. He simulated a system similar to those of Axelrod [2] and Fogel [5]. He showed that (under certain conditions) when a population (of strategies) plays IPD against its own members, the high-performing strategies who dominate the population for long periods of time, are suddenly wiped out and replaced. The results of Lindgren's simulation bear an uncanny resemblance to the "punctuated equilibria" of natural evolution [3]. "In particular, the large extinctions that appear in these simulations should be studied in more detail, since these collapses are triggered by the dynamical system itself and do not need external catastrophes for their explanation." [12, p 310].

From the point of view of machine learning, Lindgren [12] demonstrated that co-evolution produced strategies that were not robust, i.e., the strategies did well against the local population, but when something new and innovative appeared (created by the the evolutionary process) they failed dismally. Fogel [5] also noticed that co-evolved strategies could do well against each other, but still have big flaws that could be exploited by the right opponent.

Why did they seemingly-invincible strategies fail? How can we prevent this disaster happening to strategies produced by evolutionary machine learning? This paper answers these questions.

1.2 The Answers

We study a system similar Axelrod's [2], and find a reason for Lindgren's catastrophic collapses [12].

It turns out that when a near-expert strategy dominates a population, the homogeneity in their behaviour (in the case of IPD, they all cooperate) means certain features atrophy. In the case of IPD, tit-for-tat decays; in a cooperative environment, it can pay to *not* retaliate, and this atrophy of a once-useful reaction opens the way for exploitation.

In a fanciful analogy, imagine a species of big-horned buffalo that must compete with other species of buffalo. Our species' huge horns make other lesser-horned species extinct. The now-dominant big-horned species then finds that their big horns are a liability (extra weight, etc.) and evolution selects for smaller horns. The now small-horned species dominates until a random mutation (or migration from elsewhere) allows a big-horned sub-species to exterminate their now-small-horned opponents.

In our paper, this drama occurs between the players of iterated prisoner's dilemma. This sheds a light on the robustness problem in strategies produced by co-evolution, and on evolution in nature.

2 Genetic algorithms

2.1 Introduction

Machine learning often involves searching a space of legal alternatives for near-optimal solutions [4, page 614]. The genetic algorithm (GA) is a powerful technique for searching spaces that suffer the following difficulties:-

1. The search space is combinatorially large;
2. Little is known *a priori* about the search space;
3. The search space contains fine-grained discontinuities, local optima, and other irregularities.

Games whose strategies involve identifying and exploiting an opponent are search spaces of this character [4]. Hence, the GA seems a suitable machine learning technique to learn to play iterated prisoner's dilemma (IPD).

2.2 Overview of the Genetic Algorithm

A genetic algorithm (GA) [10] maintains a population of sample points from the search space. It represents a point by a string of (usually binary) characters, known as a *genotype*.

For example, a common way to represent points over a continuous parameter $x : 0 \leq x < 1$ is to discretize the space into (say) 1024 points, and represent a point by a binary string of 10 bits (where $2^{10} = 1024$). Point 776 of the 1024 possible points would be represented by the binary string 0110000101 .

A GA begins with a population of these binary strings, usually initialised with random contents, and at each generation, the following happens:-

Evaluation: Each member of the population is evaluated according to the function to be optimised.

Selection: The better performers of the population are selected for reproduction, i.e., the worse performers are erased.

Reproduction: The genetic material of the better performers is made into a new population using various genetic operators, including crossover (emulating sexual reproduction), mutation, and numerous variations [14].

Mutation is changing a character in a genotype (from 0 to 1 or vica versa) in a new individual. The mutation rate is usually low; 1 mutation per 1000 bits copied is normal [6, p 14].

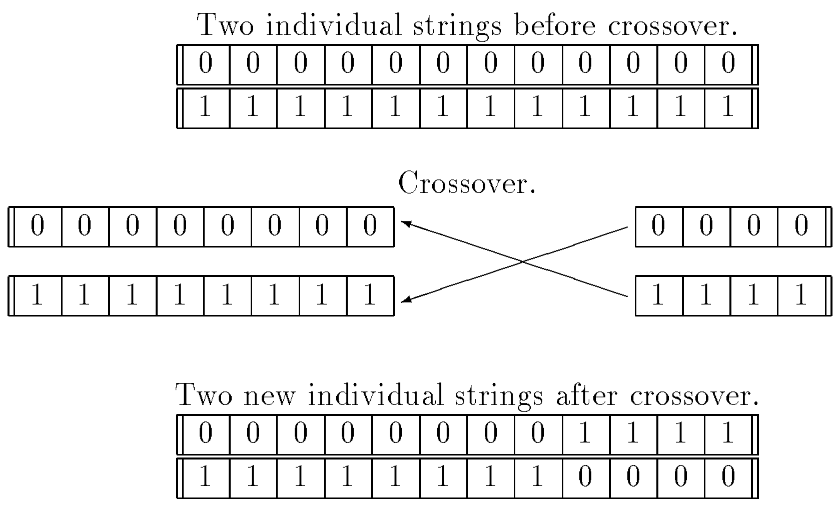

Figure 1: Crossover is when genetic material is swapped and recombined, creating two different individuals.

Crossover is the dominant mechanism of genetic rearrangement for both real organisms and genetic algorithms [11]. The simplest implementation of crossover is to pick two genotypes (usually strings of characters), randomly choose a common crossover point, and cut and paste to create two new genotypes, as in figure 1.

2.3 Why it works

The genetic algorithm quickly locates high-payoff "target" regions of the search space because each single string samples many regions. For example (where "*" indicates that a bit's value is unspecified), this string:-

1	0	0	1	1	0	1	1	0	0	0	0

is a member of this region (one quarter of the search space):-

1	0	*	*	*	*	*	*	*	*	*	*

as well as this region ($\frac{1}{32}$ of the search space):-

*	0	0	*	*	0	*	*	0	0	0	0

and so on. In general, if the binary strings are of length L, there are

$$\binom{L}{N}$$

distinct N-th order partitions of the the space (where an n^{th}-order partition divides the space into 2^n parts) [4]. This implicit parallelism allows a genetic algorithm's population to sample many more regions than its size.

A disadvantage: while a GA can rapidly find near-optimal solutions, other methods are better at going from near-optimal to optimal [11, 7]. An example from natural evolution is the giraffe: the giraffe's larynx (or voice box) is near its brain, and an optimal giraffe would have a short brain-to-larynx nerve. In reality, this nerve runs from the brain all the way down the giraffe's neck, loops around a major blood vessel, then runs back up the neck to the larynx. Like natural evolution, genetic algorithms usually give a near-optimum, but not always the optimum.

3 Strategy Generation in IPD

We use the GA to generate expert strategies to play the simple game of iterated prisoner's dilemma (IPD).

3.1 The Game

In Prisoner's Dilemma, each player can either cooperate with the other player, or defect. In iterated prisoner's dilemma (IPD), this step is repeated many times, and each player can remember previous steps. We used Axelrod's variant of prisoner's dilemma [2], is shown in figure 2.

	Cooperate	Defect
Cooperate	3 3	5 0
Defect	0 5	1 1

Figure 2: Axelrod's variant [2] of Prisoner's Dilemma as used in the runs described below.

The number of rounds per game is fixed at 50 rounds; that is, each pair of opponents play the game in figure 2 for 50 rounds, remembering the previous outcomes.

3.2 Software Implementation

Our implementation follows Axelrod [2]. Lindgren [12] used a variation of Axelrod's approach, and Fogel [5] used evolutionary programming which is strictly speaking not a GA. However, the three approaches are broadly similar, and our results are relevant to all of evolutionary machine learning.

Several excellent public-domain genetic algorithm packages are available. We used Grefenstette's GENESIS 5.0, and modified the evaluation procedure to evaluate individuals from how they score against other members of the population.

3.2.1 Genotype

We used the same genotype as Axelrod [2] Only the three most recent steps are remembered. Since each step has 4 possible outcomes (see figure 2), that means that there are $4 \times 4 \times 4 = 64$ possible histories of 3 steps. The genotype lists an action (cooperate or defect) to take for each of the possible 64 histories. These 64 actions are the parameters that the GA varies.

For example, if my previous 3 actions were cooperate-defect-cooperate (010), and my opponent's last 3 actions were defect-cooperate-cooperate (100), then we juxtapose these (opponent's history first) to get $100010 = 34$, and take the thirty-fourth action in my genotype. My opponent juxtaposes these to get $010100 = 20$, and does the action in the twentieth position of his genotype.

For the first 3 steps of a game of IPD, there are less than 3 previous steps to look back to, so the actions to take for these first 3 steps must also be specified in the genotype. The initial conditions are specified by an additional 6 bits, which makes 70 bits in the genotype. These extra 6 bits carry the assumed (pre-game) 3 steps of both players, so that the extra 6 bits alone indicate which of the (original) 64 actions to take on the first round. For rounds 2 and 3, the actual results of the game are updated to the history, so the history (as seen by an individual) contains both the pre-game "history" (carried by the genotype's extra 6 bits) as well as the actual events. In step 4 and afterwards, only the actual results are used.

3.2.2 The Round Robin

An individual strategy is evaluated by playing it against all the other members of the population. For a population of n individuals, the number of games to be played is $\sum_{i=1}^{n} i = (n-1)\frac{n}{2}$ if an individual plays against itself, and one less if it does not. The sum of the scores that an individual achieves in all these games is its fitness.

4 Results on Co-evolution

4.1 The Evolution of Cooperation

An interesting feature of evolution and iterated prisoner's dilemma is the evolution of cooperation. Even though the we simulate the most mercenary, dog-eat-dog form of evolution, cooperative strategies eventually proliferate and drive their non-cooperative competitors to extinction. Axelrod [1] describes this phenomenon in great detail.

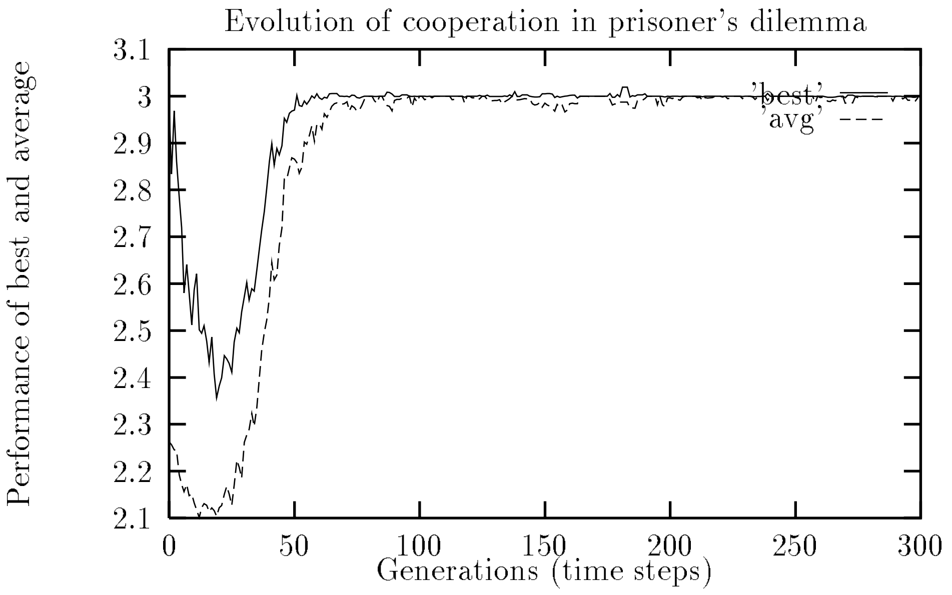

Figure 3: The evolution of cooperation in iterated prisoner's dilemma, starting from a random population. At first, non-cooperative strategies do well and performance plummets as non-cooperative strategies proliferate and they run out of victims. Eventually, cooperative strategies dominate and the performance rises to the level caused by mutual cooperation.

Figure 3 demonstrates the evolution of cooperation. From an initial population of random strategies, the population's average payoff plummets as non-cooperative strategies exploit more naïve genotypes. After a few dozen generations, gullible victims become extinct, and the population fills with strategies that cooperate when they can and retaliate against non-cooperative strategies, who are driven to extinction. In the end, only cooperative strategies survive and average per-step payoff indicates mutual cooperation.

4.2 Experiment on length of run

How long (if ever) does it take for the genetic algorithm to converge? That is, we are expecting evolution to usually produce a population of high-performance strategies that, once achieved, will change very little. Figure 3 indicates cooperation dominates after about 50 generations. Does further convergence take place after this? This experiment is simply to find how long the genetic algorithm usually runs until it stops evolving, starting from a random initial population.

We did thirty runs, each starting from a (different) random initial population of 100 individuals. The round robin was the same as Axelrod's [2]; all members of the population compete against every member of the population, *including* itself.

Figure 4 shows the average score of the GA population. Figure 5 shows the mean and standard deviation of the bias[1] of the population. The 30 runs indicate that 250 generations

[1]The *bias* of the population of a genetic algorithm is the average percentage of the most prominent value in each bit position. That is, a bias of 0.75 means that, on average, each position has converged to 75% zeros or 75% ones. The minimum bias is 0.5, the maximum is 1.

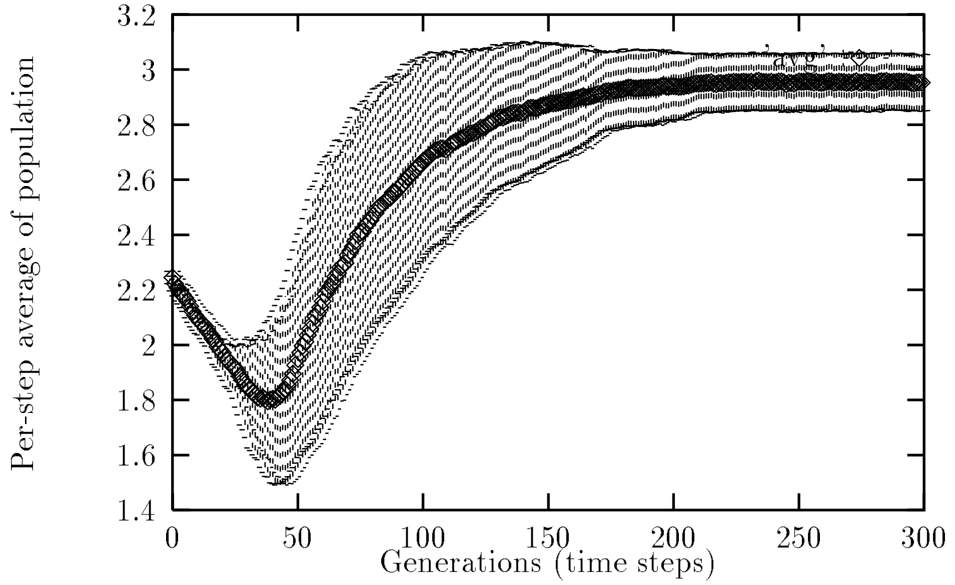

Figure 4: This shows the population's average per-step performance, and standard deviation, taken over 30 different runs. This indicates that, on average, the system has converged after about 250 generations.

is a satisfactory run length.

By 250 generations, the bias in the population has stabilised (figure 5) at 85%. Why not 100%? If there is still diversity in the population, why don't the better performers multiply and the worse performers die out?

Recall that the genotype lists actions to take under all contingencies. Diversity in the population means that individuals would choose different actions for the same contingency. We may infer that the variation in the population means this difference in opinion carries no benefit or penalty, and that some contingencies (represented by those diverse opinions) rarely arise in a cooperating population.

An analogy: imagine a species which lives where there are no flowering plants, and whose genes (by accident) make some individuals allergic to flowering plants. Since there are no flowering plants, this diversity carries no penalty or reward, and this diversity can continue despite the evolutionary pressure to succeed. Should flowering plants appear, though, selection will operate and quickly select for the better performers.

In our simulation, if no "flowering plants" appear then this diversity can continue to exist.

4.3 Experiment on seeding the initial population

One improvement to a round-robin genetic algorithm (i.e., a GA where the fitness of an individual is found from how it competes with the other individuals of the evolving population) is being investigated by Grefenstette [13]: seed the initial population with strategies known to be good.

We will find the best initial population that consists of a mix of a random initial popu-

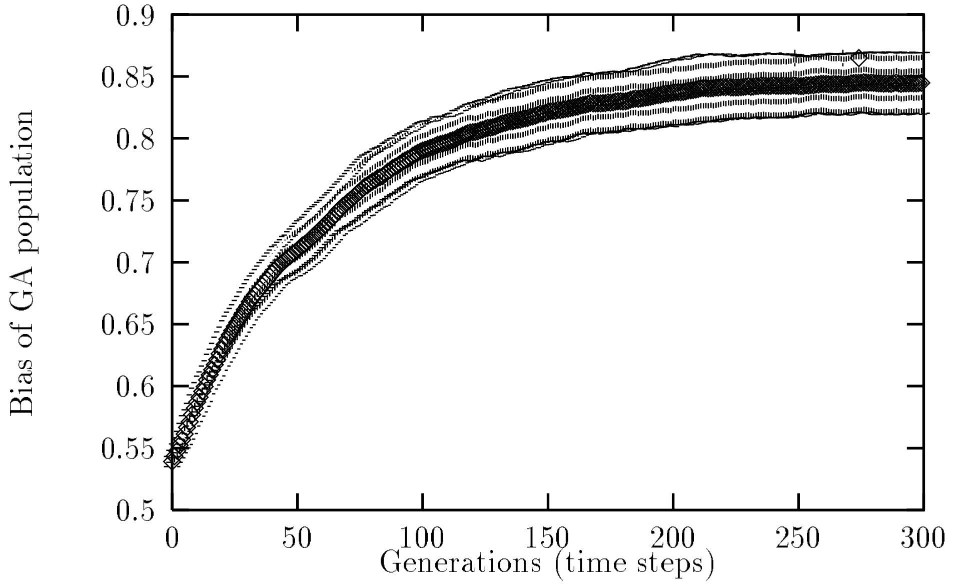

Figure 5: The bias of a genetic algorithm's population is the average per-bit level of convergence in the population; for example, a bias of 0.75 means that, on average, each position has converged to 75% zeros or 75% ones. This graph shows the mean and standard deviation of the bias for 30 runs of a genetic algorithm population playing repeated prisoner's dilemma in a round robin. The system has converged by 250 generations.

lation and a known effective strategy, tit-for-tat.

Each run used a GA population of 100 individuals. The proportion of the initial population that was tit-for-tat was varied from 0% to 25% in steps of 1%, and then from 30% to 70% in steps of 10%. For each mix of tit-for-tat and random strategies in the initial population, 30 runs were done (each of 250 generations), each using a different random component of the initial population.

Figure 6 shows how the average strategy of the final population scored against both tit-for-tat (where it nearly always scored 3, indicating mutual cooperation) and against a large number (> 1000)of random genotypes. For each run, the average genotype of the final population was played against both tit-for-tat and a large number of random genotypes, giving the mean and standard deviation for the 30 for each mix of the initial population. This is shown in figure 6.

As we expect from the evolution of cooperation in a co-evolving population [1], almost all final strategies do equally well against tit-for-tat.

Surprisingly, the best results against random strategies are *not* achieved when the initial population is all random, as one might expect. The best strategies are produced when about 10% of the initial population is tit-for-tat, and the rest random, as shown in figure 6. This indicates that genetic diversity produces better strategies, outweighing the effect of the environment matching the task. That genetic diversity aids performance agrees with Grefenstette [7].

This indicates that seeding an initial population is a worthwhile idea, but genetic diversity in a GA is still more important.

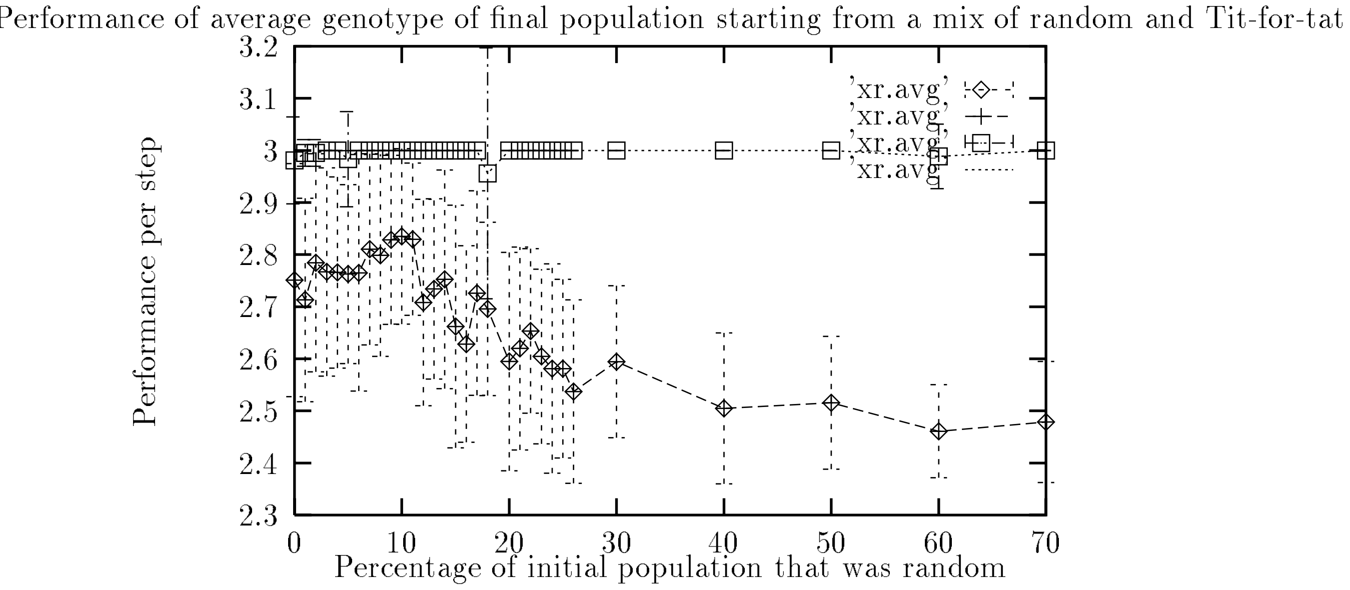

Figure 6: This shows the results of the final population against both tit-for-tat and a large number of random rules, when the initial population contains tit-for-tat. The best results are obtained when around 10% of the initial population is tit-for-tat.

5 Results on Robustness

5.1 Glitches and the robustness problem

There is a problem with strategies produced in a closed population of competing strategies: are the produced strategies good against all possible opponents, or are they useful only against the local population?

In figure 7, we see that after settling down to cooperation (similar to figure 3), in generation 222 a new mutation arises which successfully exploits its cooperative neighbours. This successful uncooperative strategy proliferates until there are so many uncooperative members of the population (and thus so few cooperative victims) that it becomes an unprofitable strategy, and cooperation returns.

5.1.1 The Cause of the Glitch

What causes the "glitch" in figure 7? Recall that the genotype of an individual is simply a look-up table that lists 64 actions (either "cooperate" or "defect"), corresponding to the 64 possible histories of the most recent 3 steps.

Consider this case: following a time of mutual cooperation, your opponent defected on you. Your opponent's last three actions are now defect-cooperate-cooperate, and your last three actions are cooperate-cooperate-cooperate. In the genotype's scheme (where "0" represents cooperation and "1" represents defection) this particular history of the last 3 steps (100-000, binary for 32) tells us to do the action listed in position 32 of our genotype.

Thus, this out-of-the-blue defection would cause the strategy to take the action (cooperate or defect) listed in bit 32 in the look-up table that forms the genotype. A strategy of tit-

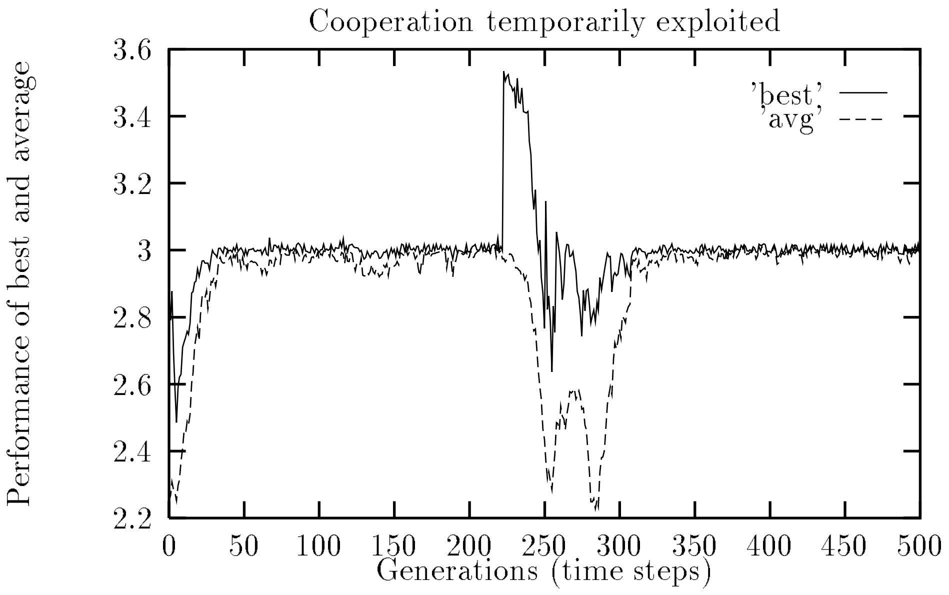

Figure 7: This shows that a new and innovative mutation can exploit a population that has cooperated for so long that they can be suckered.

for-tat would have this action listed as a defection (1), to retaliate. But what happens in a closed population of cooperators?

In a population of cooperators, unilateral defection rarely happens, and one of two things may happen to the part of the genotype (bit 32) that deals with this contingency:-

Genetic drift: There may be genetic drift: that is, since there is no selection pressure one way or the other on this part of the genotype, random mutations will fill this part of the genotype with garbage, since its contents will make no difference to fitness. Selection for retaliation or cooperation after a unilateral defection will not occur if you never have a unilateral defection.

Selection: The method that previously worked is selected against. In our example (figure 8), first there is selective pressure on bit 32 (the action to take after a unilateral defection) to retaliate; but after generation 130, there is selective pressure to *cooperate* after a unilateral defection. This is because in this particular local population dominated by cooperation, it pays to do this.

In figure 7, most individuals had a 0 (cooperate) in bit 32 by generation 200, which was an advantage in this particular cooperative population. Random mutation hit upon a strategy that could exploit this particular flaw, which exploited this over-cooperative flaw. This is shown by figure 8: before the successful exploitative mutation at generation 222, most of the population contained a 0 in bit 32, and would cooperate after suffering a unilateral defection. This was the flaw exploited by the new mutation.

A fanciful analogy may help explain this: imagine a species of big-horned buffalo competing with other species of buffalo. Our species' huge horns make lesser-horned species extinct. The now-dominant big-horned species finds their horns are a liability (extra weight to carry, etc.) and evolution selects for smaller horns. The now small-horned species dominates until

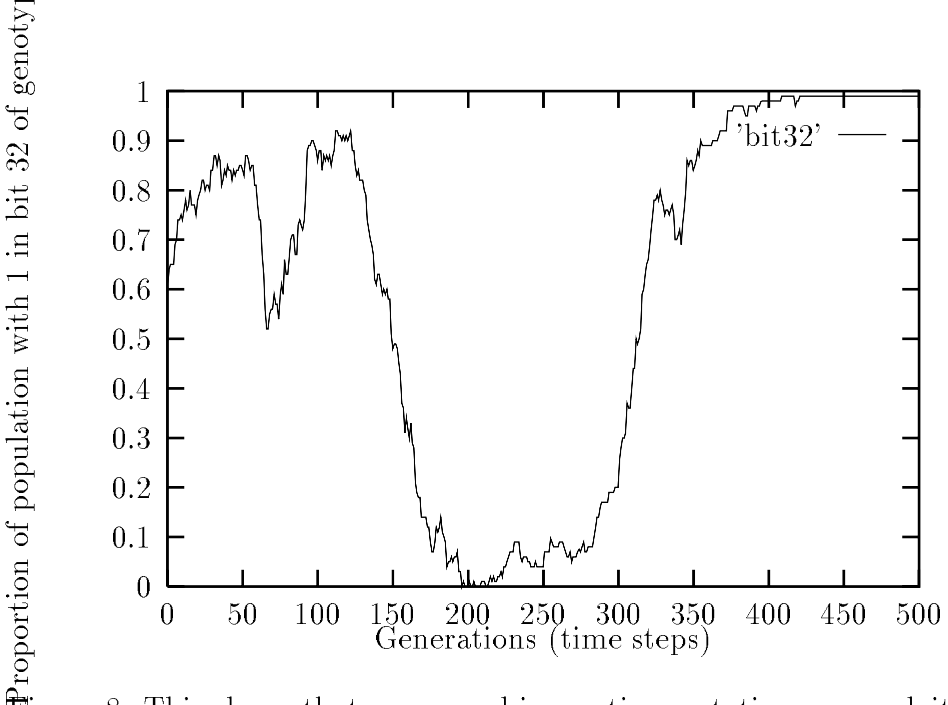

Figure 8: This shows that a new and innovative mutation can exploit a population that has cooperated for so long that bit 32 (the one that contains the action to take after a unilateral defection) is 0 (cooperate) for most of the population.

a random mutation (or migration from elsewhere) allows a big-horned sub-species to exterminate their small-horned opponents. In figure 8, put "horn size" on the y-axis and a similar drama occurs.

Another "glitch" is shown in figure 9. Lindgren [12], in a similar simulation, demonstrated that these events can happen repeatedly if the simulation runs for long enough.

We want to avoid disastrously naïve situations like this. We investigate two possible ways to avoid this kind of glitch:-

- Seed the initial population with some known expert rules, instead of a completely random initial population.

- Include a few extra, unchanging genotypes in the round robin, so that each individual in the population has to play every other member, plus the extra players. In the analogy of the old Western town, this is like having a resident bad man, so that people will know what to do in that contingency.

5.2 Experiment on static rules in round robin

We have seen how a strategy that evolved in a co-evolving population runs the risk of failing badly against new opponents. That is, there is a risk of blithely generating naïve strategies that are good against similar strategies but vulnerable to radically different ones. This situation is described in the previous section, is shown in figure 7, and also occurred in Fogel's similar approach evolutionary learning to play iterated prisoner's dilemma [5].

One solution is to expand the round robin in the genetic algorithm; as well as the other members of the evolving population, each individual could also play against extra, static

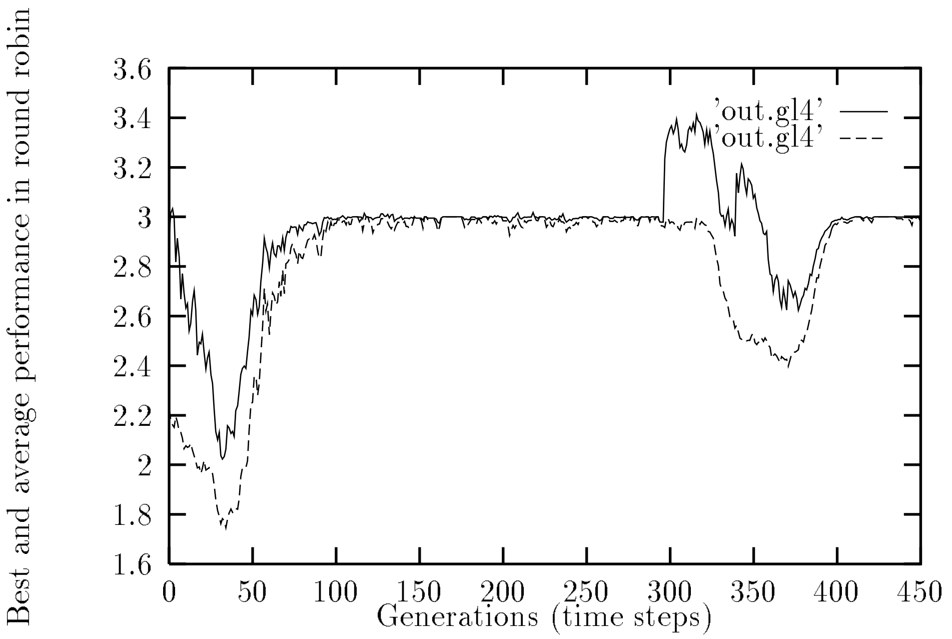

Figure 9: Another example of an innovative mutation that exploits a population which has cooperated for so long that they can be suckered.

strategies. The aim is to prevent over-specialisation to the local population, and produce versatile strategies that work against all opponents.

In this experiment, we add to the round robin. We do 4 sets of 100 runs (each of 250 generations; 100000 generations altogether) where the round robin contains:-

1. The usual round robin, with only the evolving population.

2. As for 1, but including a random genotype. This is a genotype of 70 bits just like the other members of the population, but its contents are random, and stay the same for a generation, before being replaced by a different random genotype in the next generation.

3. As for 1, but including the expert strategy tit-for-tat [1].

4. As for 1, but including both a (different) random strategy each generation, and the expert strategy tit-for-tat [1].

How can we evaluate the contents of a final population? That is, how do we tell if one round robin learned to play IPD better than another?

For each of the 400 runs, we followed the following procedure:-

1. Run for 250 generations.

2. From the final population (of 100 individuals), find the *average* strategy. That is, if a particular bit in the genotype is "0" for 50% or more of the population, it is 0 in the average genotype.

3. We evaluate this average strategy by running it against two strategies:-

In round robin	Against randoms	Against tit-for-tat
Population	2.54 ± 0.27	2.72 ± 0.47
Plus random	2.78 ± 0.23	2.66 ± 0.51
Plus tit-for-tat	2.55 ± 0.25	2.81 ± 0.40
Plus both	2.74 ± 0.23	2.72 ± 0.50

Table 1: The round robin of a genetic algorithm has extra rules added. For each round robin, 100 runs were done starting from a different random initial population each time. The average genotype of the final population scores best against an opponent that was added to the round robin.

Tit-for-tat This strategy is to verify that the strategy does cooperate in repeated prisoner's dilemma against an expert opponent.

Many random strategies We also play a rule set against a large number (> 1000) of random genotypes; that is, look-up tables filled with random contents. This is to verify that the strategy can exploit non-expert opponents.

Table 1 shows the average and standard deviation of the scores against tit-for-tat and the random genotypes. The evolution of cooperation causes the population to cooperate, so the average genotype usually scores 3 against tit-for-tat, and a few spectacular failures brought the average down in table 1.

Against the exploitable random genotypes, the populations which faced a random opponent in the round robin did better; as expected, the closer the learning situation is to the real thing, the better the results.

A genetic algorithm will specialise to fit the problem presented [8]. If we include (say) tit-for-tat in the round robin, we expect the strategy produced to do better against tit-for-tat; table 1 agrees. This result also agrees with Grefenstette [9], where training under variable conditions created rule sets that performed well under variable conditions.

As a method of overcoming the robustness problem in co-evolutionary machine learning, adding static strategies seems to work. Table 1 shows that adding a fixed strategy to the round robin causes the population to succeed against *those* opponents.

Usually we don't have the luxury of knowing the superior opponents *a priori*. The open question is still: how can co-evolution produce strategies that are robust performers against all possible opponents? That is, can a predator that evolved on one continent ever be sure of succeeding against a predator from another continent?

6 Discussion and Conclusion

Lindgren [12] and Fogel [5] noticed that strategies produced by the co-evolution of closed populations may do well in that closed population, but can fail against outsiders. we have demonstrated a possible cause of this lack of robustness: the homogeneity of some high-performing populations leads to the atrophy of techniques (such as retaliation against unilateral defection), opening the way for their exploitation. This is shown in "glitches" (see figure 7, where a homogeneous GA population that was doing well in its own predictable population was exterminated by a new, innovative mutation, which exploited the lack of retaliation against unilateral defection, as shown in figure 8.

This naïve vulnerability to innovation is the robustness problem in co-evolving genetic algorithms.

Adding static opponents to the round robin improves the results of the final population, as shown in table 1. As expected, the rules produced by a population do better against an opponent that was added to the round robin.

An open question is: can extra, fixed opponents produce strategies that are also good against opponents never seen before? The results show that this method has great promise for automatic strategy generation.

References

[1] Robert M. Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[2] Robert M. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 3, pages 32–41. Morgan Kaufmann, 1987.

[3] Richard Dawkins. *The Blind Watchmaker.* Longman, first edition, 1986.

[4] Kenneth A. Dejong. Genetic-algorithm-based learning. In Y Kodratoff and R Michalski, editors, *Machine Learning*, chapter 21, pages 611–638. Morgan Kaufmann, 1990.

[5] David B. Fogel. Evolving behaviours in the iterated prisoner's dilemma. *Evolutionary Computation*, 1(1):77–97, 1993.

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

[7] John J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 4, pages 42–60. Morgan Kaufmann, 1987.

[8] John J. Grefenstette. Strategy acquisition with genetic algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 14, pages 186–201. Von Nostrand Reinhold, 115 Fifth Avenue, New York NY 10003, 1991.

[9] John J. Grefenstette, Connie L. Ramsey, and Alan C. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381, October 1990.

[10] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, second edition, 1992.

[11] John H. Holland. Genetic algorithms. *Scientific American*, 267(4):44–50, July 1992.

[12] Kristian Lindgren. *Evolutionary Phenomena in Simple Dynamics*, volume 10 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 295–312. Addison-Wesley, 1991.

[13] Alan C. Schultz and John J. Grefenstette. Improving tactical plans with genetic algorithms. In *Proceedings of IEEE Conferenece on Tools for Artificial Intelligence*, pages 328–334. IEEE Society Press, November 1990.

[14] Xin Yao. An empirical study of genetic operators in genetic algorithms. *Microprocessing and Microprogramming*, 38:707–714, 1993.