Methods for Competitive Co-evolution: Finding Opponents Worth Beating (Submitted to ICGA 95)

Christopher D. Rosin Richard K. Belew

{crosin,rik}@cs.ucsd.edu
Cognitive Computer Science Research Group
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Abstract

Co-evolution refers to the simultaneous evolution of two or more genetically distinct populations with coupled fitness landscapes. In this paper we consider "competitive co-evolution," in which the fitness of an individual in a "host" population is based on direct competition with individual(s) from a "parasite" population. Competitive co-evolution is applied to three game-learning problems: Tic-Tac-Toe (TTT), Nim and a small version of Go. Two new techniques in competitive co-evolution are explored. "Competitive fitness sharing" changes the way fitness is measured, and "shared sampling" alters the way parasites are chosen for testing hosts. Experiments using TTT and Nim show a substantial improvement in performance when these methods are used. Preliminary results using co-evolution for the discovery of cellular automata rules for playing Go are presented.

1 Introduction

Co-evolution refers to the simultaneous evolution of two or more genetically distinct populations with coupled fitness landscapes. In this paper we consider "competitive co-evolution," in which the fitness of an individual in one population is based on direct competition with some individual(s) from another population. Such interactions have been hypothesized to occur in nature and modelled by game-theoretic constructions such as Maynard Smith's "evolutionary stable strategies," [10] and the Prisoners' Dilemma [2]. It also arises in the evolution of AI game-playing strategies where the range of potential strategies makes it difficult to establish the sort of fixed, "exogenous" fitness function typically used in genetic algorithms (GAs) [1]. More generally, a number of situations arise in complex software engineering applications where the construction of appropriate test suites to demonstrate the reliability of software is considered almost as important as the development of the software itself. By viewing the software solutions as one population and the test suites as another, competitive co-evolution allows the simultaneous search for both [8].

Note that there is a distinction between domains where the competition is symmetric (games such as Prisoners' Dilemma, for example [2]), and domains where it is asymmetric (for example, with a solution / test case distinction). Games (such as the ones used in this paper) may also be asymmetric; this can be due, for example, to the advantage enjoyed by the first player to move. In both the symmetric and asymmetric cases, individuals from each of the co-evolving populations must take turns testing and being tested. In this paper we use the term *host* to refer to the individual whose fitness is under consideration, and *parasites* to refer to the individuals that are testing the host.[1]

Since the parasites are also evolving with a fitness based on a competition's outcome, the success of a host implies failure for its parasites. Repeated failure by either host or parasite populations represents an evolutionary challenge that can lead to an evolutionary "arms race" [5]. New genotypes arise to defeat old ones. From the perspective of GA engineers, we desire that this process go on indefinitely. Each new parasite type should serve as a drive towards further innovation, creating ever-greater levels of complexity and performance.

Our goal for an implementation of competitive co-evolution is therefore the discovery of a diverse but consistently strong set of novel parasites, worth the computational effort of beating. Use of weak parasites as determinants of host fitness will only give rise to mediocre solutions; there is no drive to do better. Testing against a small set of parasites allows the evolution of hosts that may not prove robust in the face of broader experience. Co-evolution offers a potential solution to this problem. Each new, improved host type creates a new niche for those parasites that can defeat it. This can give rise to a wide diversity of niches, containing a wide variety of competitors. New types that evolve to compete well against such parasites are likely to be robust, and will offer new challenges in the competition.

As is common in software engineering, the problem of testing candidate solutions can be considered separately from that of developing good solutions. A perfect solution to the testing problem is a minimal and complete set of extremely difficult test cases, and is typically very difficult to identify for realistic problems. Note, however, that such a test set may well make a poor parasitic population, even if we knew what it was. Hosts competing against such rigorous parasites may all fare poorly. Evolution requires information from fitness *variation* to guide search. Co-evolution may provide this, too. In any generation, the parasites used will have varying degrees of evolutionary success, but none are likely to have evolved far beyond the difficulty required to defeat the current hosts. As the generations progress, we can expect a drive for more and more difficult test cases to arise. The result should be a pedagogical series of parasites that gradually increase in difficulty as host solutions gradually increase in competence.

In this paper competitive co-evolution is applied to three game-learning problems: Tic-Tac-Toe (TTT), Nim, and a small version (7x7) of Go. Two new techniques in competitive co-evolution are explored. "Competitive fitness sharing" changes the way fitness is measured, and "shared sampling" alters the way parasites are chosen for testing hosts. Both of these methods are intended to increase the useful diversity in co-evolving populations and support evolutionary innovation. Experiments using simpler versions of competitive co-evolution on Nim were unsuccessful, but this game can be solved using the two new techniques; similar improvement due to these

[1] Note that this terminology differs somewhat from Hillis'.

methods has been observed in Go.

Competitive fitness sharing and shared sampling are defined in detail, and the effect they have on evolution is considered. Most experiments are done using the more tractable TTT and Nim problems, since representations can be kept simple and the results comprehensible. Typical runs are shown for both problems, as well as replicated experiments that demonstrate the performance of the variations introduced. Preliminary results using co-evolution for the discovery of cellular automata rules for playing Go are presented.

2 Methods

In all experiments done here, two evolutionarily distinct populations are used, with competitions taking place between populations. Each population gets its turn as both hosts and parasites, when its fitness is being measured and when it is being used to measure fitness, respectively.

2.1 Competitive Fitness Sharing

The usual way to assign fitness based on success in a set of competitions is to add up the score in all competitions (where score is often a binary value indicating success/failure) [1, 8, 2]. This will be called "simple fitness" here. A different method, similar to fitness sharing, is also tested here, and improves performance on the test problems. In fitness sharing [7] a sharing function is defined, to measure similarity between individuals. An individual's fitness is then divided by the sum of its similarities with each other individual in the population. This rewards unusual individuals.

In the method used here, each parasite is treated as an independent resource to be shared by those hosts in the population that can defeat it. This is like having a separate sharing function for each parasite, and will be called "competitive fitness sharing", or just "sharing" for brevity in this paper. The fitness assigned to a host defeating parasites with the set of indices X is $\sum_{j \in X} \frac{1}{N_j}$, where N_j is the total number of hosts in the population defeating parasite j. Fitnesses are most comparable when all hosts compete against the same set of parasites, and this is the case for all experiments here.

The effect of this method is to reward hosts that defeat parasites few others can, even though the rewarded host might not defeat *as many* parasites as others can. This is most relevant when there do not yet exist hosts that can defeat all of the parasites. In such a case, any host that defeats parasites that others cannot may well contain important genetic material.

As an example of the advantage that sharing may provide, consider a situation in which a host type G_1 defeats I parasites which no other host type can defeat. Assume that types $G_2, G_3, \ldots, G_n$ defeat J parasites on average, with $J > I$. Thus, G_1 is important in that it contains information about defeating parasites that no others can, but it defeats a below-average number of parasites overall. Assume G_1 only had a small number of representatives in the population. This might be because it is a new innovation, or an old niche that recently became important. Or, it may simply be that random fluctuations carried its population to low numbers. Under simple fitness, G_1 would have below-average fitness, and would stand a good chance of becoming extinct this generation (with exact probability depending on the type of selection). Under competitive fitness sharing, though, individuals of type G_1 will receive substantially greater fitness, because they are rare: I fitness will be divided among a small number individuals. Other, more prevalent types will have lower per-individual fitness, because their fitness is being divided among a larger number of host individuals. Therefore, the small number of individuals of type G_1 stand a good chance of being selected, preventing the immediate extinction of G_1.

Because of this sort of situation, competitive fitness sharing is more able to allow the survival of important but poorly represented types. So, a greater number of diverse niches should be supportable under competitive fitness sharing. When many such niches exist they are necessarily small and subject to extremely low numbers due to sampling errors. Under simple fitness, there is a greater danger of extinction under these conditions than with competitive fitness sharing.

2.2 Opponent Sampling

It is desirable to reduce the computational effort expended in competition by testing each individual in the host population against only a limited sample of parasites from the other population. One simple way to collect a representative sample of parasites is to pick a random subset of them. This is one technique that is explored below.

However, information is available about individuals from the previous generation; this information can be used to try to select a more challenging sample of parasites (though parasites must now be selected from the previous generation's population). For example, parasites with the best fitness in the last generation might be preferentially selected for play in this generation. If there are several niches in the parasite population, however, this would tend to pick similar individuals from the niche that happened to have highest fitness, rather than selecting a diverse set of parasites from all niches.

An alternative technique is to select a subset of parasites that challenges all segments of the host population. A natural way to do this is to choose new sample members that have maximal competitive shared fitness within the current sample. In this way, each successive member of the sample is chosen to be one that competes well against hosts (from the previous generation) that the other members of the current sample do not compete well against. This technique of sampling the parasite population will be called "shared sampling", and is specified in detail by the algorithm in Figure 1. Note that the "opponents" here are the individuals that were used in the previous generation to rate the fitness of the parasite population being sampled.

```
Initialize current sample to be the empty set
For each opponent i from previous generation
    beat[i]=0 (# in current sample beating i)
While the current sample is not yet full
    For each parasite j not yet in sample
        samp_fit[j]=0 (fitness within sample)
        For each opponent k beat by j last gen.
            samp_fit[j] += 1/(1+beat[k])
    Let j be such that samp_fit[j] is maximal
    Add individual j to current sample
    For each opponent i from previous generation
        If j beat opponent i last gen.
            Increment beat[i]
```

Figure 1: Shared Sampling

It is interesting to note that shared sampling can also provide some of the effects of competitive fitness sharing described above, even when competitive fitness sharing is not being used. In addition to choosing parasites that challenge all hosts, shared sampling will tend to choose more representatives of parasite types that beat a large number of hosts. So, parasite types that are defeated by a large segment of the host population will generally be represented less in the sample than those defeated by few members of the host population. Therefore, even under simple fitness, hosts receive greater fitness for defeating parasites that few other hosts can defeat. This effect - that simple fitness with shared sampling can provide performance comparable to that of competitive fitness sharing - will be seen in some of the results below.

3 Structure of Experiments

3.1 Test problems and Representations

The test problems used for most experiments were the simple games, TTT and Nim. To test whether these methods can make progress on a much more difficult problem, strategies for the game of Go were co-evolved; this will be discussed in Section 5.

In TTT, players alternate placing their markers on a 3x3 grid. The first player to obtain 3 in a row horizontally, vertically, or diagonally wins. Games frequently end in a draw (which is the result of optimal play on both sides). In Nim, a position in the game consists of several piles of stones. Players alternate removing an arbitrary number of stones from a single pile. The player to take the last stone wins. The configuration used here starts with 4 piles, containing 3, 4, 5, and 4 stones. This configuration allows the first player to force a win with optimal play [4].

The representation used for both games has each gene corresponding to a single position in the game. In TTT, symmetric positions are removed, and the genes are ordered on the genome in the following, rather arbitrary way. A board configuration may be represented as a 9 digit base-3 number, with each digit being 0 to indicate that the square is empty, 1 to indicate that the opponent holds the square, and 2 to indicate that the moving player holds the square. All positions are generated in numerical order starting from 0 (all squares empty). Each position is assigned to a gene if: it is reachable in actual play, does not already contain a win, has at least two squares open (so that a choice needs to be made), and does not have a symmetric form (reflected and/or rotated) already present in a previous gene. This yields a genome with 593 board positions.

For Nim, the positions are placed on the genome in lexicographic order: (0,0,0,0), (0,0,0,1), ..., (3,4,5,4), where each quadruple gives the number of stones in each of the four piles. Actually, (3,4,5,4) is not needed in this representation, so a total of 599 genes are used.

In TTT, a control function is learned. Each gene can take on a number of allele values equal to the number of open squares in the corresponding position. This uniquely specifies a legal move from the position. For contrast, in Nim a binary evaluation function is used: each gene can take allele values 0 and 1. To make a move, legal moves are tested using the evaluation function, and the first position evaluated as 1 is selected as the move. Positions are tested in the following order. Piles are selected in order from first through fourth. For each pile containing $k > 0$ stones, moves leaving the pile with 0 up to $k-1$ stones are tested in order. If none if these positions is evaluated as 1, the first move in this ordering (not generally a good move) is selected by default.

These seem like simple problems. Certainly, perfect play is easily achieved for either of these games by game tree search. Note, however, that the task facing the GA is more difficult in two respects. First, our genomic representation assumes no knowledge about or searching of the games' underlying tree structure. Furthermore, the GA must also solve the problem of finding good parasites to test candidate strategies in order to solve the games. In [1], evolving TTT strategies in a competitive environment was found to be a difficult problem (under a different representation, and without symmetries eliminated, though).

3.2 Genetic Algorithm

In all TTT and Nim experiments presented here, two populations compete. The *first population* individuals are those searching for a perfect strategy, and the *second population* opposes them. In Nim, the first population seeks a perfect first player strategy, and therefore only plays first. In TTT, the first population seeks a strategy that never loses, going first and second.

In each generation, fitness is evaluated for each population in the following manner. A sample of the opponent population is selected as parasites to test the current host population. This sample might contain all opponents, or it might contain opponents from the previous generation when shared sampling is used. All host individuals compete against each parasite in this sample, with fitness assigned by simple fitness or by competitive fitness sharing. For TTT, all host/parasite competitions involve two games, with each player getting a chance to go first and second; under competitive fitness sharing, a parasite is treated as two independent shared resources, when it goes first and when it goes second.

The runs are terminated when a perfect strategy (meeting the criteria above) is found and is stable in the first population. Simple (i.e., single individual) elitism is always used, with the elite individual being displaced in the next generation only if a new individual with strictly greater fitness exists. This guarantees a perfect strategy to be stably held in the population once it makes it into the elite.

GA parameters were chosen through a modest amount of exploration of the parameter space. A few types of tournament and roulette selection were tried, with tournament selection found to be superior for these problems. Mutation rates were chosen on the basis of a few experiments that showed fairly high mutation rates to work well. Population size was chosen to be fairly large so that success was achieved consistently. Simple elitism was found to work well; large elite sets were somewhat detrimental to performance, and perfect strategies were repeatedly lost with no elitism. In general, parameters were chosen conservatively to give consistent results, rather than attempting to achieve optimal performance in every run. As long as a reasonably large sample of parasites is selected each generation, and reasonably large population sizes are used, eventual success of these methods does not seem to be critically dependent on other parameter value choices.

The GA operates on each population independently. Each child is created from the recombination of two parents. Each parent is selected in an independent 2-tournament. Crossover points are selected with probability 0.004 per gene. Mutations are chosen with probability 0.0066 per gene for TTT, and 0.0022 per gene for Nim. Each mutation selects a random legal allele value for the gene in TTT. Mutations in Nim flip the binary allele value. For TTT, each population has 400 individuals (for a total of 800), and for Nim, each has 500 individuals.

4 Results

4.1 Types of Data Obtained

Typical runs for Nim and TTT will be presented, followed by a comparison of the various techniques on these test problems. In analyzing results, performance is measured by the number of generations required to solve the game. Sharing and shared sampling provide a substantial performance improvement, and use of these techniques is required to solve Nim at all. To help understand these results, two additional aspects of experiments are presented.

The first of these is usage of positions in the game. Each time a move is made from a position, during any game in a single generation, a counter for that position is incremented. A "significant usage" threshold is set, and the number of positions with usage above this threshold is measured each generation, for both populations. This threshold is set at a level well above 0, with the goal being that significantly used positions are those whose usage has been strongly selected for by the

GA, rather than positions that simply happen to be used a few times due to a chance mutation. So, a larger number of significantly used positions should indicate a greater diversity of games, between individuals that are being selected for. This sort of diversity is important to test hosts more fully each generation and provide them with richer fitness information. For purposes of comparison, significant usage is averaged over all generations in a run, and the average of this quantity across multiple runs is presented. For TTT, only those positions with corresponding genes are counted.[2] Significant usage is substantially higher when sharing and shared sampling are used.

The second aspect analyzed is the loss of potentially important genetic material. As described above, sharing and shared sampling should reduce the extinction of host types that defeat parasites others cannot. Here, each generation, for each population, a count is made of the number of sampled parasites meeting two conditions. First, there existed hosts this generation which defeated the parasite. Second, no such host was selected to contribute genetic material to the next generation. This describes the sort of extinction considered above. Note that the loss of a single important host may be counted as several extinction events (one for each parasite which it beat, but which no other host could beat). For purposes of comparison, the average number of extinction events per generation is calculated, and the average of this quantity across multiple runs is presented. Extinction events are typically fairly rare, but there are fewer extinction events when sharing and shared sampling are used.

4.2 Typical Runs

Figure 2 shows the average number of wins for both populations during one run, in which sharing and shared sampling of 10% of the parasite population were used. TTT is the easier of the two problems: with parameters tuned well, perfect solutions could be found after about 1.5 million game-pairs (3 million games). In the early part of the run, many flaws in the first population are discovered by the second population and weeded out. In some runs (such as the one displayed), this initial stage is sufficient to find a perfect strategy. In other runs, after this initial stage, the best first population individuals typically defeat all opponents[3] in most generations. The second population occasionally discovers flaws in the first, which are then quickly corrected by the first population, until a perfect strategy is found.

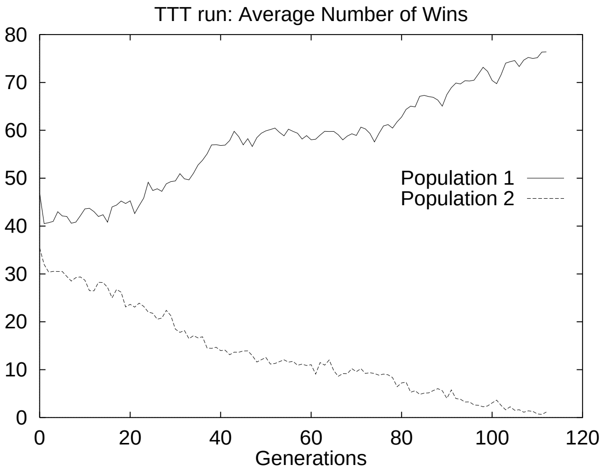

Figure 2: TTT average number of wins

Significant usage typically shows a quick initial rise, to a level that the particular configuration of the GA can support, followed by some fluctuations. These features are similar to those seen in significant usage for Nim (see below).

Nim is more difficult. This is probably due to several factors. First, there is greater constraint on the moves a perfect player can make. Second, the game tree is somewhat deeper and has substantially more distinct paths through it. Finally,

[2] This brings the measure in closer correspondence with gene usage [3].

[3] "defeat" here means "do not lose when going either first or second"

the representation is more epistatic, since unlike TTT, several genes enter into each move decision.

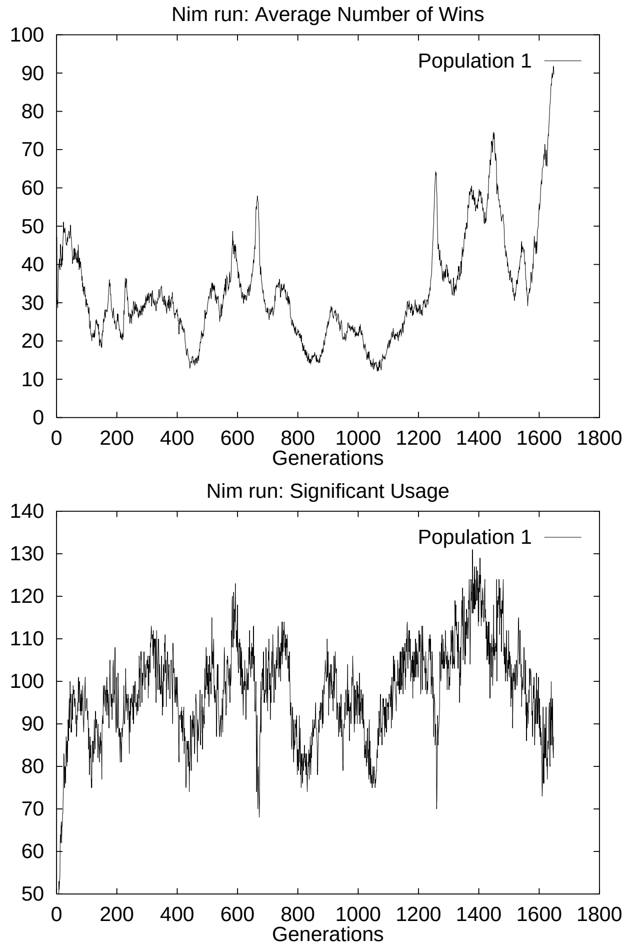

Figure 3: Nim average number of wins and significant usage

Figure 3 displays average wins and significant usage for the first population, during a run in which sharing and 20% shared sampling were used. The second population's average wins is a near-mirror image (though not quite, due to sampling). For Nim, a perfect solution can be found after about 150 million games with parameters tuned well. Typically, success in the competition for Nim is much more balanced between the two populations during most of the run. The first population spends much time trying strategies which employ an incorrect first move. Such strategies can never be perfect, but can be difficult to beat if most other moves had been optimized. Eventually, a strategy employing a correct first move is optimized to the point of perfection.

The significant usage threshold was set at 400 uses. Significant usage displays an initial rise, followed by fluctuations which are due in part to temporary convergence and failure in the population. When the population begins to converge to a temporarily successful type (fitness spikes at 670 and 1260 generations), diversity loss can be seen in the drop in significant usage. During periods of relative failure (for example at 450 generations), few types are strongly selected for, and significant usage also drops.

4.3 Comparison Results

4.3.1 TTT

Figure 4 shows the average performance over 10 runs of each of four variants, on TTT. The variants are the four possible combinations of random sampling / shared sampling and simple fitness / competitive shared fitness. Four levels of sampling were used: 1%, 4%, 10%, and 25%. Sharing and shared sampling clearly improve performance. The performance difference between random sampling with simple fitness, and shared sampling with shared fitness, is significant: $p \leq 0.002$ at all levels of sampling.

It is interesting that runs at the 1% sampling level were all successful, since this corresponds to the use of only 4 parasites to test a host population. This could not be pushed much lower, though: runs using only one parasite to test each host population were not successful.

Table 1 shows average number of significantly used positions per generation in TTT, with averages taken over 5 runs. The significant usage threshold was set at 320 uses. This corresponds to the number of games that 4 hosts play against all parasites. The qualitative results were fairly

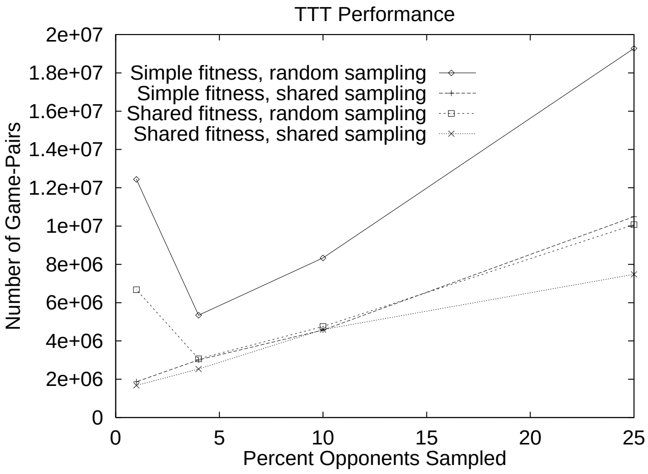

Figure 4: TTT performance comparison

robust to the choice of threshold. For these runs, 10% opponent sampling was used. P1 here refers to the first population serving as host to a sample of parasites from the second population, and P2 to the second population against a sample of the first population. The difference in usage between simple fitness with random sampling, and shared fitness with shared sampling, was found to be significant ($p < 0.001$ for both populations, treating the average for each run as an independent sample).

Fitness	**Sampling**	**P1**	**P2**
Simple	Random	47.57	36.65
Simple	Shared	50.50	40.43
Shared	Random	58.53	46.76
Shared	Shared	56.54	47.03

Table 1: TTT Significant Usage

Table 2 shows the average extinction events per generation for 5 runs of each of the four variants, on TTT. For this game, each parasite can yield two separate extinction events: one when it goes first and one when it goes second. For these runs, 10% opponent sampling was used. These extinction events are all for second population hosts playing against first population parasites. There were no events for first population hosts playing against second population parasites, probably because their evolutionary task was fairly easy. The difference in extinction events between simple fitness with random sampling, and shared fitness with shared sampling, is significant ($p < 0.001$).

Fitness	**Sampling**	**Extinctions**
Simple	Random	0.32
Simple	Shared	0.25
Shared	Random	0.0089
Shared	Shared	0.012

Table 2: TTT Average Extinctions per Generation

4.3.2 Nim

For Nim, simple fitness and shared fitness were tested without sampling (use entire parasite population), with random sampling, and with shared sampling. All Nim results presented are averaged over 5 runs, with 20% sampling of opposition when sampling is used. Lower levels of sampling also worked, but results were not as robust to changes in other parameters.

Table 3 shows performance results for the variants studied. When sampling is not used, each game is used twice: once with the population 1 individual playing host and once with the population 2 individual playing host; the results given in this case reflect the fact that each such game is only played and counted once, with the outcome used for both purposes. Shared fitness with shared sampling did the best. Note that runs using simple fitness, without sampling and with random sampling, never completed. Variations of these were run for up to 20000 generations, and though there was initial progress, these runs showed no hope of eventual success.

Table 4 shows significant usage for the six variants, on Nim. The significant usage threshold was

Fitness	Sampling	Avg	Std Dev
Simple	None	-	-
Simple	Random	-	-
Simple	Shared	193	65.4
Shared	None	885	425
Shared	Random	363	140
Shared	Shared	157	18.4

Table 3: Nim: Number of games to solve (in millions)

set at 400 for versions with sampling, and 2000 without. Both of these correspond to the number of games that 4 hosts play against all their parasites. For variants which could not find perfect strategies, the first 2500 generations of data were used. The difference between shared fitness with shared sampling, and simple fitness with random sampling, is significant. ($p < 0.02$ for both populations).

Fitness	Sampling	P1	P2
Simple	None	50.79	54.25
Simple	Random	55.31	61.26
Simple	Shared	92.74	90.63
Shared	None	91.39	95.93
Shared	Random	92.51	97.94
Shared	Shared	98.38	90.92

Table 4: Nim Significant Usage

Table 5 shows average extinction events per generation for the six variants, on Nim. For variants which could not find perfect strategies, the first 2500 generations of data were used. Note that when sampling is not used, there are potentially far more extinction events since there are more parasites. For the second population, the difference in extinction events between using simple fitness with random sampling, and shared fitness with shared sampling, is significant ($p < 0.01$).

Fitness	Sampling	P1	P2
Simple	None	2.73	0.212
Simple	Random	0.243	0.0369
Simple	Shared	0.0655	0.0101
Shared	None	0.00218	0.0
Shared	Random	0.00028	0.0
Shared	Shared	0.01558	0.0

Table 5: Nim Average Extinctions per Generation

5 Co-evolving Cellular Automata Rules for Go

The game of Go has proved difficult for computers. A description of the game, and some of the early work in getting computers to play it, can be found in [9]. The game involves placing stones at intersections on a grid. Go is fairly visual, and playing seems to involve much intuitive pattern matching. In addition, some properties of the game are nearly invariant of position on the grid. For these reasons, cellular automata (CA) seem to provide a natural representation of strategies for the game. Hand-designed cellular automata rules have been used in a program written by Allan Scarff, apparently with some success [6].

Here, each intersection on the Go board corresponds to one cell in the CA. Each cell contains 8 bits of state, all of which may be modified by the action of evolved rules. The first two bits show the current placement of a black or white stone, respectively, in the initial configuration, and all other bits are cleared initially. The third bit is used to signal where the next move is to be made. A move is made after the first CA timestep in which the CA selects a legal move.

CA rules are represented as a list of condition-action pairs, in the style of classifier system rules. Conditions consist of 0, 1, or "don't care" for all bits in the cells of a 3x3 neighborhood, and actions consist of 0, 1, or "don't modify" for each bit

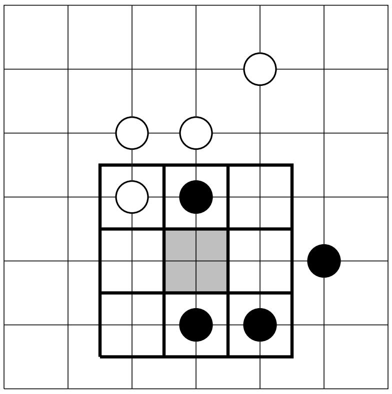

Figure 5: Go board with CA neighborhood

in the central cell (see Figure 5). Each timestep, for each cell, the action corresponding to the first matching condition in the list (if any) is taken. Genetic operators can mutate one of the condition or action bits in a rule, add a new random rule at the end of the list, swap two rules, or delete a randomly chosen rule. Crossover points are always chosen to be between rules in the list.

In an initial experiment, a 7 by 7 board was used. A population size of 600 was used, with the 300 individuals in the first population playing first, and the second 300 playing second. Fitness sharing, and a shared sample of 15 opponents, were used. To evaluate the success of evolved strategies, they were tested against Wally, a weak public-domain Go-playing program that has been used by other authors to test learned Go strategies [11].

After about 250 generations, some of the best strategies were able to consistently defeat Wally. These strategies were still not very good, though. In particular, they displayed a lack of ability to look at the board globally, sometimes making a move which is locally relevant, but incorrect globally because of some more pressing distant play. Interestingly, human beginners (and Wally) also tend to display this weakness. The CA rules did show an encouraging degree of competence, given that a fairly complex representation was used, with no information about playing well built in.

6 Conclusion

Competitive co-evolution was quite successful for the simple games used here. The fact that it solved these games indicates that it was able to solve the problem of finding parasites to test candidate strategies.

Two new techniques, competitive fitness sharing and shared sampling, were found to substantially improve performance on the test problems, and were required to completely solve Nim. Consideration of extinction events and significant usage suggests that these techniques protect important innovations and small niches, allowing a greater diversity of parasites to drive co-evolution at any point in time.

Finally, preliminary results on a more difficult game are promising. In the game of Go, coevolution is able to make progress on a difficult problem, using a complex representation with no knowledge about the task built in.

References

[1] Angeline, P.J., and J.B. Pollack. Competitive Environments Evolve Better Solutions for Complex Tasks. In *Proceedings of the Fifth International Conference on Genetic Algorithms* Morgan Kaufmann, 1993.

[2] Axelrod, R. Evolution of Strategies in the Iterated Prisoner's Dilemma. In *Genetic Algorithms and Simulated Annealing* Morgan Kaufmann, 1989.

[3] Bedau, M.A., and N.H. Packard. Measurement of Evolutionary Activity, Teleology,

and Life. In *Artificial Life II* Addison-Wesley, 1991.

[4] Berlekamp, E.R., J.H. Conway and R.K. Guy. *Winning Ways for Your Mathematical Plays* Academic Press, 1982.

[5] Dawkins, R. *The Blind Watchmaker* Norton, 1986.

[6] Durham, T. "A program with a touch of Zen". In *Computing Horizons* Addison-Wesley, 1985.

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

[8] Hillis, W.D. Co-evolving parasites improve simulated evolution as an optimization procedure. In *Artificial Life II* Addison-Wesley, 1991.

[9] Levy, D. (editor). *Computer Games* Springer-Verlag, 1988.

[10] Maynard Smith, J. *Evolution and the Theory of Games* Cambridge University Press, 1992.

[11] Schraudolph, N.N., P. Dayan, and T.J. Sejnowski. Temporal Difference Learning of Position Evaluation in the Game of Go. In *Neural Information Processing Systems 6* Morgan Kaufmann, 1994.