Methods for Competitive Co-evolution: Finding Opponents Worth Beating

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

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," (Maynard Smith 1982) and the Prisoners' Dilemma (Axelrod 1989). It also arises in the evolution of AI game-playing strategies where the range of potential opponents makes it difficult to establish a single, fixed, "exogenous" fitness function as is typically used in genetic algorithms (GAs) (Angeline 1993). 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 (Hillis 1991).

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 (this terminology differs somewhat from Hillis'). 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 (when its fitness is being measured) and parasites (when it is being used to measure fitness).

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. When the parasites evolve to overcome this failure, they create new challenges for the hosts; the continuation of this may lead to an evolutionary "arms race" (Dawkins 1986). New genotypes arise to defeat old ones. New parasite types should serve as a drive towards further innovation, creating ever-greater levels of complexity and performance by forcing hosts to respond to a wider range of more challenging parasite test cases.

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.

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 in an initial population forced to compete against such rigorous testing are all likely to 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 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 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

2.1 Competitive Fitness Sharing

The usual way to assign fitness based on success in a set of competitions is to sum the score (often a binary value indicating success/failure) across all competitions (Angeline 1993, Hillis 1991, Axelrod 1989). This measure will be called "simple fitness" here. In fitness sharing a sharing function is defined to take into account similarity among individuals (Goldberg 1987). An individual's fitness is divided by the sum of its similarities with each other individual in the population, rewarding unusual individuals.

Our method is similar to fitness sharing and also the "emergent fitness sharing" techniques found to promote generalization in an immune system model (Forrest 1993). In the "competitive fitness sharing" (and henceforth just "sharing" for brevity in this paper) method we propose 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. 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 defeats I parasites which no other host type can defeat. Assume that other host types defeat a greater number of parasites, on average. Thus, G 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 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 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 will receive substantially greater fitness, because they are rare: I fitness will be divided among a small number of 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 stand a good chance of being selected, preventing the immediate extinction of G.

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 way to get a representative sample of parasites is to pick a random subset of them. This is one technique explored below.

However, information is available about individuals from the previous generation; this can be used 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.

It would be preferable to choose parasites from the previous generation that challenge 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 competed well against individuals that the other members of the current sample did not compete well against. This technique will be called "shared sampling", and is specified in detail by the algorithm in Figure 1. Note that the "opponents" there are the individuals that were sampled in the previous generation to rate the fitness of the parasite population currently 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 (Berlekamp 1982).

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 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 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; since this is the initial position, it never needs to be evaluated (see below). This leaves a total of 599 positions on the genome.

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. By way of contrast, a binary *evaluation function* is used is used in Nim: 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 of 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 a complete set of tests for evaluating candidate strategies in order to solve the games. In fact, these games make good test problems for competitive co-evolution partly because it is difficult to find good test cases via simple methods like random sampling. Previous work has shown that evolving TTT strategies in a competitive environment is a difficult problem for the GA, and also that the use of co-evolved strategies to measure fitness improves results over the use of random or fixed strategies (Angeline 1993).

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, 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 the 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. The number of positions with usage above a "significant usage" threshold is measured each generation, for both populations. This threshold is set at a level well above 0, with the goal being to identify positions whose usage has been strongly selected for by the GA, rather than those that happen to be used a few times due simply to a chance mutation. A larger number of significantly used positions should therefore indicate a greater diversity of games played by selected individuals. 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 (this brings the measure in closer correspondence with the measure of gene usage in (Bedau and Packard 1991)). Significant usage is substantially higher with sharing and shared sampling.

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

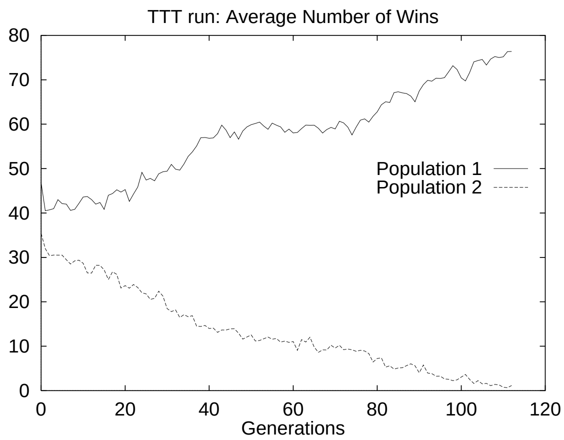

Figure 2: TTT Average Number of Wins

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 ("defeat" here means "do not lose when going either first or second") 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. Significant usage statistics for TTT are characterized by a quick initial rise, followed by some minor fluctuations.

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

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 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

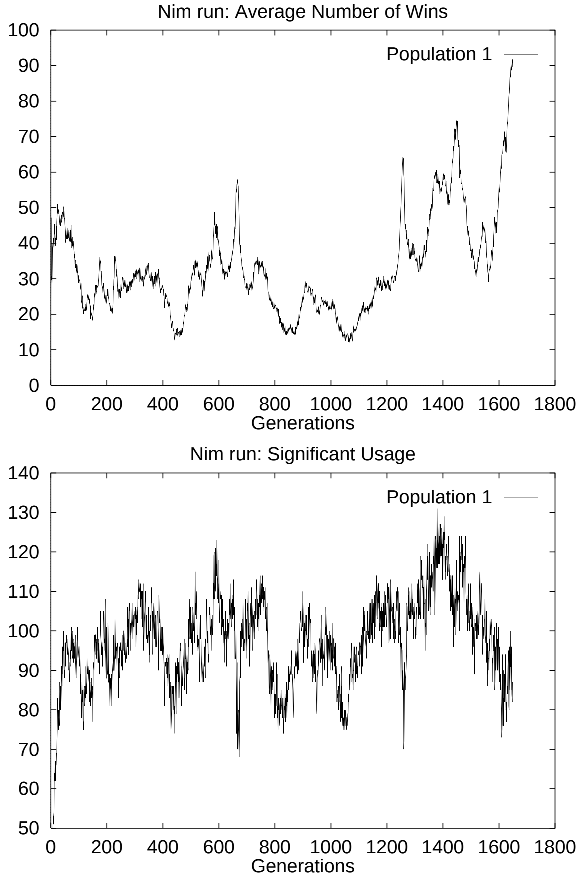

Figure 3: Nim Average Number of Wins and Significant Usage

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 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,

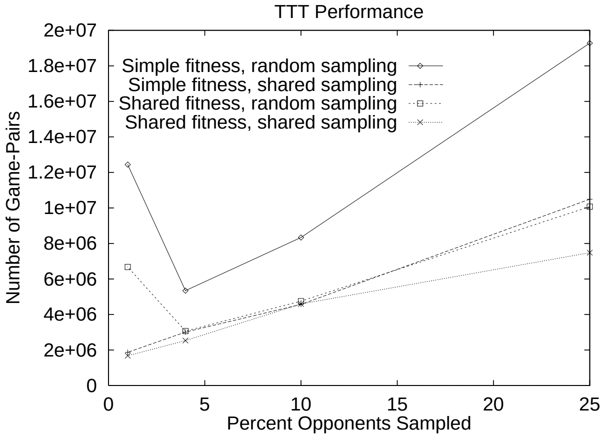

Figure 4: TTT Performance Comparison

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).

Table 1: TTT Significant Usage

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 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 ($p < 0.001$).

Table 2: TTT Average Extinctions Per Generation

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

4.3.2 Nim

For Nim, simple fitness and shared fitness were tested without sampling (using entire parasite population), with random sampling, and with shared sampling. All Nim results presented are averaged over 10 runs, with 20% sampling of parasites 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. Runs using simple fitness, without sampling and with random sampling, never completed. Variations of these were run for up to 20000 generations (5 billion and 2 billion games, respectively), and though there was initial progress, these runs showed no hope of eventual success.

Table 3: Nim: Number of Games to Solve (in Millions)

Fitness	Sampling	Avg	Std Dev
Simple	None	-	-
Simple	Random	-	-
Simple	Shared	201	60.6
Shared	None	806	330
Shared	Random	353	116
Shared	Shared	145	27.5

Table 4 shows significant usage for the six variants, on Nim. The significant usage threshold was set at 400 for versions with sampling, and 2000 without. These both 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 were used. The difference between shared fitness with shared sampling, and simple fitness with random sampling, is significant ($p < 0.001$ for both populations).

Table 4: Nim Significant Usage

Fitness	Sampling	P1	P2
Simple	None	44.29	46.92
Simple	Random	58.33	64.43
Simple	Shared	91.75	85.86
Shared	None	90.42	94.45
Shared	Random	91.86	97.18
Shared	Shared	97.70	97.15

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 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 ($p < 0.001$).

Table 5: Nim Average Extinctions Per Generation

Fitness	Sampling	P1	P2
Simple	None	2.79	1.92
Simple	Random	0.148	0.0334
Simple	Shared	0.0476	0.00781
Shared	None	0.00289	0.00014
Shared	Random	0.00025	0.00015
Shared	Shared	0.00796	0.0

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 (Levy 1988). 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 Go. Hand-designed cellular automata rules have been used in a program written by Allan Scarff, apparently with some success (Durham 1985).

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. Actions consist of 0, 1, or "don't modify" for each bit 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 7x7 board was used. A population size of 600 was used, with the 300 individuals in

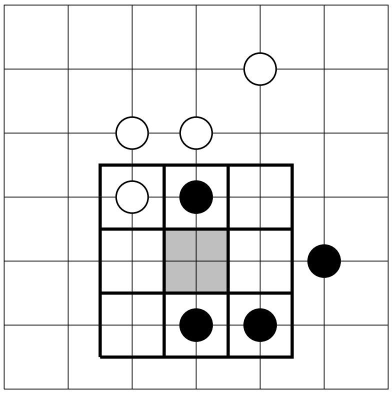

Figure 5: Go Board with CA Neighborhood

the first population playing first, and the second 300 playing second. Fitness sharing, and a shared sample of 15 opponents, were used. To evaluate evolved strategies, they were tested against Wally, a weak public-domain Go-playing program that has been used by others to test learned Go strategies (Schraudolph 1994). Wally played first during these tests.

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. This is natural, since a single CA rule can directly encode only a play conditioned solely on local information. However, the GA was able to get small sets of two or three rules to work together to produce some "nonlocal plays". For example, it is good for the first player to start the game with a stone near the center of the board. Initially, there is no information local to the center to distinguish it; only the edges may be so distinguished. An initial move near the center requires rules to mark location information into cells closer to the center, and another rule to use this information and make the move. The evolution of CA rules that can interact in this way is encouraging, as is the level of performance that has been achieved with no built-in knowledge about playing the game well. A similar experiment on a 9x9 board has yielded CA rules that can defeat Wally much of the time.

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, co-evolution produces interacting cellular automata rules that are able to defeat a weak computer opponent.

Acknowledgements

C.R. is an NSF Graduate Fellow. The authors gratefully acknowledge the support of Apple Computer.

References

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

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

Bedau, M.A., and N.H. Packard (1991). Measurement of Evolutionary Activity, Teleology, and Life. In *Artificial Life II*, Addison-Wesley.

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

Dawkins, R. (1986). *The Blind Watchmaker*, Norton.

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

Forrest, S., B. Javornik, R. Smith, and A.S. Perelson (1993). Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation* 1:3.

Goldberg, D.E., and J. Richardson (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, L. Erlbaum Assoc.

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

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

Maynard Smith, J. (1982). *Evolution and the Theory of Games*, Cambridge University Press.

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