Neuro-Evolution in Multi-Player Pente

Jacob Schrum (`schrum2@cs.utexas.edu`)
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA

Abstract

Pente is a derivative of the Japanese game Go-moku, both of which are normally played with only two players. We extend the game of Pente to three players and study the ability of neuro-evolution via the Enforced Sub-Populations (ESP) algorithm to evolve Pente players for 7 by 7 boards capable of beating pairs of opponents taken from a set of five simple hand-coded opponents. We also compare the performance of feed forward networks to that of simple recurrent networks and simple recurrent networks that *pay attention* to the board by reading inputs from it on every player's turn, not just their own. Evolving networks that beat all pairs of opponents in three-player games proves difficult, and we also find that against the given opponents feed forward networks are superior to either type of simple recurrent network.

1 Introduction

AI has been applied to abstract strategy games such as Checkers and Chess since its beginnings, but comparatively little attention has been given to multi-player abstract strategy games. Multi-player game search can be done using the max^n algorithm (Luckhardt and Irani 1986), which is an extension of the popular minimax algorithm for two-player games. Minimax assumes an optimal opponent, which is adequate in most two-player games, but in multi-player games the concept of optimal opponents is more difficult to define. This becomes an important issue when the search tree needs to be pruned, which is necessary to play any interesting games in reasonable time. Two-player search trees can be easily pruned with alpha-beta pruning, but multi-player pruning strategies are particularly sensitive to the ordering of nodes in the search tree and the tie-breaking mechanism used when searching (Sturtevant 2004).

This sensitivity makes learning based approaches to game AI more appealing. In particular, neuro-evolution has proven to be a successful technique for creating agents to play two-player abstract strategy games such as Chess (Fogel et al. 2004), Checkers (Fogel and Chellapilla 2002), Othello (Moriarty and Miikkulainen 1995) and Go (Lubberts and Miikkulainen 2001; Richards et al. 1998; Stanley and Miikkulainen 2004). The next step is to extend neuro-evolution to games supporting three or more players.

In games with more than two players it is possible and even likely that players may collude at times. We would like to evolve players that sense this collusion and respond appropriately in order to win the game. This requires that the networks have some sort of internal representation of how their opponents are playing. A recurrent network architecture provides such a representation. The recurrent links provide memory for the networks, allowing them to possibly sense and respond to collusion, especially if the networks monitor the board not only on their own turns, but on the turns of other players. By *paying attention* in this manner, the networks more closely approximate a human player that follows the action in a game while it is being played. For the most part, human players do not completely ignore the game between turns.

The following paper is a study of these issues as applied to the game of Pente, which is described below.

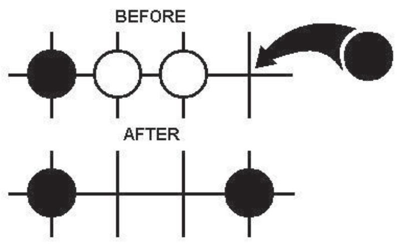

Figure 1: **Example Capture.** Black has captured two of White's stones with a single move. Five such captures result in a win for Black. (Adapted from Gabrel and Braunlich 2004)

2 The Game of Pente

Pente is a derivative of the Japanese game Go-moku. Both games are played on a 19 by 19 Go board on which players take turns placing colored stones until one wins by getting five stones in a row horizontally, vertically or diagonally. Pente adds to Go-moku the ability to capture pairs of adjacent opponent stones by bracketing them in on both ends (Figure 1). Captures only occur when the placement of a player's stone causes two of the opponent's stones to be surrounded, and not when the opponent plays a second stone between the player's stones: a player cannot make a move that results in capture of her own pieces (Figure 2). Multiple captures in a single move are also possible. In Pente, captures are not merely a way to remove enemy pieces from the board, but also a means to win: capturing five pairs of enemy stones results in victory for the player.

The game is biased in favor of the first player, whose first stone must be placed on the center point of the board. It has been claimed that the first player is guaranteed to win given

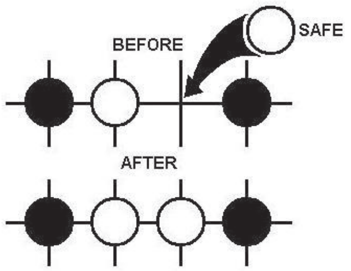

Figure 2: **Playing Between Pieces.** It is safe for White to play between Black's stones, because a capture can only result from Black bracketing White's pieces. (Adapted from Gabrel and Braunlich 2004)

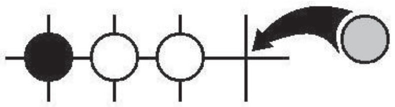

Figure 3: **Intervention to Stop Capture.** The Grey player makes a move that prevents White from being captured by Black, because Black already has four captures and would win otherwise.

perfect play, and this has actually been proven in the case of Go-moku (Allis et al. 1993) and Renju (another Go-moku derivative; Wàgner and Viràg 2001), though not yet in Pente, which has received considerably less formal study. However, general k-in-a-row games have been studied by Wu and Huang (2005).

Another unexplored aspect of the game is the ability to extend it to multiple players. In a multi-player game of Pente the rules are the same, except that play cycles amongst three or more players. Captures are still possible, but are only valid if stones of the same color are captured: it is not possible to capture pairs of differently colored stones. In games of three or more players an interesting dynamic emerges in which players will temporarily cooperate in order to prevent another player from winning. For example, Grey might make a move that prevents White from being captured by Black in order to prevent Black from getting five captures and winning the game (Figure 3). Players can also choose to ignore certain threats if they are confident that these threats will be noticed and dealt with by the other players. For example, suppose that Grey will win on her next move by getting five in a row if she is not blocked. Both White and Black get to move before Grey's next move, so either of them could block Grey, but Black decides to ignore Grey's threat and strengthen her own position, thus forcing White to block Grey.

It should be noted that no claims have been made as to the ability of the first player to win in the multi-player variation of the game, but it is reasonable to assume that the first player has an advantage over the second player and an even greater advantage over the third player. However, given a scenario in which both the second and third players always collude against the first player, it may be possible that a win for the first player is impossible. Whether or not this is the case is a question open to further study.

3 Approach

Of programs that play Go-moku, the most notable is the Victoria program (Allis et al. 1993) that used threat-space search and proof-number search to prove that Go-moku is a won game for the first player. Applications of soft-computing to the game of Go-moku include a Genetic Algorithm approach by Tang et al. (1999), a reinforcement learning approach by Freisleben (1995), and an approach using neural networks trained with backpropagation (Katz and Pham 1991).

This author knows of no prominent programs that play Pente, but there are several programs available on the internet. These programs all seem to make use of more traditional AI techniques such as minimax with alpha-beta pruning.

The above mentioned programs play the two-player variation of the game only. As far as this author knows, there are no multi-player programs that play Go-moku or Pente. This paper confines itself to the study of games with two and three players, but these games could easily be extended to four or more players. This study also restricts itself to 7 by 7 boards instead of the usual 19 by 19 boards because evolving agents to play on larger boards is considerably more difficult. We prefer to first evolve neural networks to play a simplified (smaller) version of the game to see if favorable results are obtained, and then scale these results up later if possible.

Before going into the details of how neural networks were evolved to play Pente, the next section first defines the network architecture used to play the game.

3.1 Network Architecture

The neural network players in this study all use the same architecture, which is capable of playing both two and three player games. The architecture is similar to the architecture used by Lubberts and Miikkulainen (2001) and Richards et al. (1998) to play Go and Moriarty and Miikkulainen (1995) to play Othello.

For each space on the 7 by 7 board there are three inputs: one for each of three players in a three-player game. This is a total of 147 inputs. For each space, the first input reports the presence of one of the network's stones. The next input reports the presence of a stone belonging to the player immediately following the network, and the third input reports the presence of a stone for the next player. In a two-player game, both inputs for sensing an opponent are activated when an opponent's stone is present, and neither of the inputs are active otherwise. In all cases the presence of a stone is indicated by a value of 1.0, and a lack thereof is indicated by 0.0.

Although networks that play in two-player games could be evolved with one fewer input for each space on the board, keeping the number of inputs consistent for all players allows for comparison between players of two and three player games, and allows networks evolved by these two separate methods to play directly against each other. Furthermore, networks evolved in two-player games with two inputs per space did not demonstrate performance significantly different from networks evolved with three inputs per space. All results presented in this paper used networks with three inputs per space.

The networks also have one output for every space on the board. These outputs range from 0.0 to 1.0, where each output indicates the utility of placing a stone on the given space.

A value of 1.0 indicates that the move is good, and 0.0 indicates that the move is bad. The network outputs values for all spaces, even if the space is already occupied. Therefore when making a move, the best legal move is chosen, meaning the move with maximum utility that is not occupied. In case of a tie in maximum utility values, the first of the available moves found is chosen.

The one exception to the above movement rules comes in the case where placing a stone on a given space results in five-in-a-row and thus wins the game for the neural network. Because there is absolutely no ambiguity as to the utility of this move, it will always be chosen over all others, regardless of the network output.

There is also one hidden layer between the input and output layers that has not yet been mentioned, because it is what is evolved. The composition of the hidden layer is evolved by the Enforced Sub-Populations (ESP) algorithm, described below.

3.2 Enforced Sub-Populations

ESP is a technique for neuro-evolution that evolves separate sub-populations of neurons to occupy the hidden layer of a neural network. For each hidden unit in the network there exists a separate sub-population, and members of a sub-population can only be recombined with members of the same sub-population.

A neural network is formed by taking a random member from each sub-population and putting it in the hidden layer of the network. Then the neural network is evaluated and awarded a fitness score. Within a given generation, networks are formed and evaluated in this manner until every neuron has participated in a minimum number of networks. The final fitness score of each neuron is the average score from all networks that the neuron took part in. Then crossover and mutation are performed on the fittest members of each sub-population, and the resulting neurons replace the least fit members of that sub-population. For more information on ESP, see Gomez and Miikkulainen (1997).

ESP also allows the evolution of recurrent networks, which have proven particularly useful in control tasks, such as finless rocket control (Gomez and Miikkulainen 2003). However, ESP's ability to evolve recurrent networks has not yet been applied to board games, although agents to play Go have been developed using feed forward networks in ESP (Perez-Bergquist 2001) and also using ESP's direct predecessor Symbiotic Adaptive Neuro-Evolution (SANE; Lubberts and Miikkulainen 2001; Richards et al. 1998). Therefore there is evidence that ESP should perform well in evolving feed forward networks to play Pente, but the usefulness of recurrent networks for playing board games has not yet been addressed.

ESP allows for the evolution of simple recurrent networks, second order recurrent networks and fully recurrent networks, but all of the recurrent networks evolved in this study are simple recurrent networks. A simple recurrent network remembers the activation values of its hidden layer from the previous time step. These values become additional inputs to the network on the current time step. Other types of recurrent networks were not used because their potential complexity seemed to greatly slow down the evolutionary process.

4 Experimental Setup

4.1 Fitness Function

The fitness function is based on how quickly the neural network wins and, if it loses, how long it lasts before failing to win. Initial experiments showed that using only the number of wins and losses did not exert enough evolutionary pressure on the population, so the following fitness function for a neural network n was developed:

$$Fitness_n = \sum_{t \in \mathcal{T}} \sum_{m=1}^{M} \sum_{p=1}^{P} G(n,p,t)$$

where $\mathcal{T}$ is a set of tests, each consisting of a group of opponents that the neural networks play against on a single evaluation, M is the number of matches played, where a match consists of as many games as there are players in the game, P is the number of players in the game, and G is a function for the fitness gain from a single game in which the network n is the p-th player against the opponents specified by test t. G is defined as follows:

$$G(n,p,t) = \begin{cases} moves_{max} - moves(n,p,t) & \text{: n won} \\ 0.5 \times moves(n,p,t) & \text{: otherwise} \end{cases}$$

where $moves_{max}$ is the maximum number of moves possible in a game and $moves(n,p,t)$ is the number of actual moves that occurred when the network n played the opponents from test t as player number p. The maximum fitness is achieved when the network wins as quickly as possible, but in order to have some selective pressure exerted on the players that lose, networks are awarded for preventing their opponents from winning as long as possible. The coefficient of 0.5 is meant to assure that the fitness gain from winning is ultimately greater than what can be achieved by losing, though no claims are made as to the optimality of the value 0.5. A smaller value may be more appropriate.

Notice in the definition of $Fitness_n$ that every network plays as the first, second and, in a three-player game, third player against the same opponents. Such a set of games against the same opponents is considered a *match*. Changing the position of the neural network in the turn order requires the network to come up with a general strategy capable of coping with different starting points. This is considerably harder than evolving the networks to always have the same position in the turn order, but was felt necessary because turn order conveys a large advantage or disadvantage depending. For more on the fairness of k-in-a-row games (two-player only) see Wu and Huang (2005).

The contents of the set $\mathcal{T}$ are different in each of the experiments below. The justification for playing against the opponents from each test on every evaluation is that by playing against different strategies all at once, a more general strategy should emerge. This increases the time it takes to perform a single evaluation, but the resulting players should be able to defeat all tests and not just a subset thereof. All test opponents for this study were drawn from a group of five simple hand-coded players.

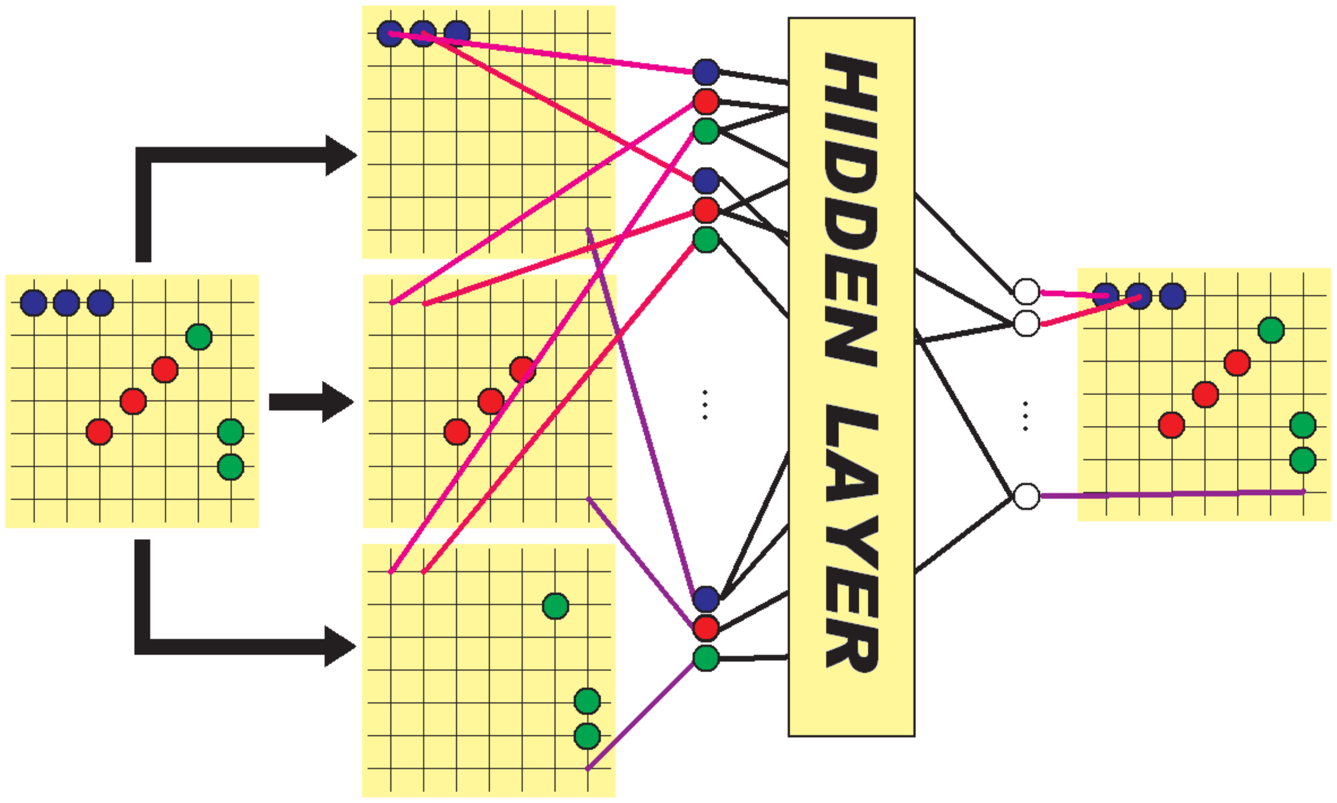

Figure 4: **Network Architecture.** The 147 inputs consist of three inputs for each space on the board, where each of the three inputs corresponds to one of the players. There are 49 outputs: one for each space on the board. The hidden layer is evolved by ESP.

4.2 Hand-Coded Players

The five hand-coded players used as opponents for the neural networks were

1. *Random*
2. *Blocker*
3. *Capturer*
4. *Simple*
5. *Better*

The *Random* player is the simplest, and makes a random move every turn. However, a random trainer is useful because evolution often finds ways to exploit quirks against deterministic opponents.

The *Blocker* plays like a random player until its opponent creates a formation that it decides to block. Highest priority is given to blocking formations of four-in-a-row, and secondary priority is given to blocking formations of three-in-a-row. This simple *Blocker* will not block cleverer formations, such as two pairs of two with a gap between, so it is ultimately easy for a human player to beat, especially since it does not explicitly attempt to win for itself.

The *Capturer* always tries to position itself to capture its opponent. If it can make a capture on its turn, it will do so. If a capture is not possible, it will place a piece adjacent to an opponent's piece such that if the opponent's next move is along the same line, it will be able to make a capture. Only in rare circumstances is one of these two types of moves not possible, but when this is the case the *Capturer* makes a random move.

Moving on to players that actually attempt to make five-in-a-row, the *Simple* player will first attempt to make a horizontal row of five one piece at a time until it finds itself confined on both sides either by the edge of the board or an enemy piece. At this point the *Simple* player then changes its goal to forming a vertical column of five-in-a-row. Whenever it finds itself boxed in both horizontally and vertically it makes a random move and starts over.

The strategy of the *Better* player is similar except that it also considers making five-in-a-row along the diagonals, and it also aborts its attempts prematurely if it senses that it is no longer possible to make five-in-a-row along the current line. In other words, it does not wait until the pieces on either side of its formation are directly adjacent to obstacles before switching lines the way the *Simple* player does. The *Better* player realizes in advance that it cannot make five-in-a-row, even though there are still blank spaces along which it could extend its current line.

Both the *Simple* and *Better* players attempt to win as quickly as possible with very simple strategies. They make no attempts to block, and therefore are easily beaten by a human player.

These players are used to train the neural networks in both two and three player games. In three-player games these players make no distinction between the neural network and other hand-coded players. In particular, the *Blocker* and the *Capturer* may sometimes focus on the other hand-coded player instead of the neural network. This makes fitness evaluation in three-player games noisy, and is yet another reason why three-player games are so complicated. Thus the first experiment is done with two-player games, and the rest are done with three-player games.

5 Experiments

5.1 Two-Player Pente

For the two-player experiments the test set $\mathcal{T}$ consisted of all five hand-coded players, and the neural networks played 10 matches against each opponent. The networks played as both the first and second player in each match, making for a total of 20 games against each opponent and a total of 100 games per evaluation. The networks were evolved for 2000 generations. Feed forward networks were used.

5.2 Three-Player Pente

The first three-player experiment also makes use of all the hand-coded players, but the neural networks played against combinations of these players instead of individuals. A single test consisted of a pair of two different opponents from the set of hand-coded players. Teams with two of the same player were excluded for the sake of simplicity, and to reduce the number of games per evaluation. As it is there are still a total of 10 tests in the set. The number of tests would be twice this, except it is assumed that the ordering of the two opponents in the test does not make a significant difference, which is not necessarily true, but greatly reduces the number of necessary tests.

The number of matches against each opponent is still 10, but because there are three players now there are 3 games per match, making for 30 games against each pair of opponents. With 10 pairs of opponents to face, this adds up to 300 games per evaluation. The networks were only evolved for 900 generations due to time constraints. Feed forward networks were also used in this experiment.

5.3 Effects of Paying Attention

The final experiment consisted of a single test, but this test was faced by three different types of networks: feed forward networks, simple recurrent networks and simple recurrent networks that *pay attention* by taking inputs from the game board on each player's turn.

The single test they played against was a pair of two *Better* players, and the number of matches was still 10. Because there was only one test, the evolution proceeded fairly quickly, requiring only 30 games per evaluation. Each of the three types of networks being compared were evolved for 2000 generations.

6 Results

6.1 Two-Player Pente

At the end of the 2000 generations the champion of the population was able to consistently beat all five hand-coded players in all 20 games. Its behavior generalized enough that it could win regardless of whether it went first or second. The champion fitness level across generations shows that evolution reaches a plateau very early on, but eventually a jump in fitness occurs (Figure 5) that coincides with the networks finally being able to consistently beat their hardest opponent, the *Better* player (Figure 6). The *Random* player, *Blocker* and *Capturer* were all beaten very early on, and the defeat of the *Simple* player coincides with the defeat of the *Better* player.

Although these results show that evolution eventually overcomes all tests, other trials not shown here make it clear that the fortuitous jump in fitness that allows the networks to always beat the *Better* players does not always occur, at least not always within 2000 generations.

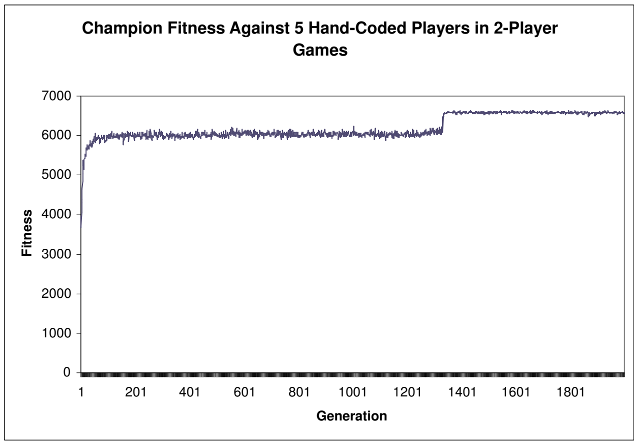

Figure 5: **Fitness Against 5 Hand-Coded Players.** The fitness raises quickly and levels out for a long period. In this trial there is a fortuitous jump in fitness, which demonstrates the potential to evolve successful players in this manner, but other trials have shown that this jump does not always occur.

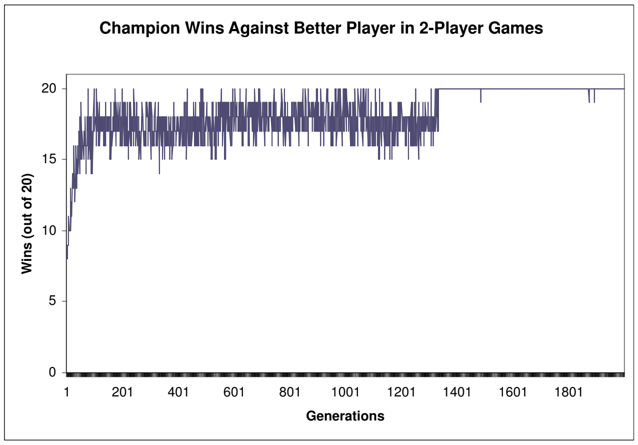

Figure 6: **Wins Against *Better* Player.** When evolved against all 5 hand-coded players at once, the champion networks took the longest to beat the *Better* player in all games. The other players were beaten considerably faster.

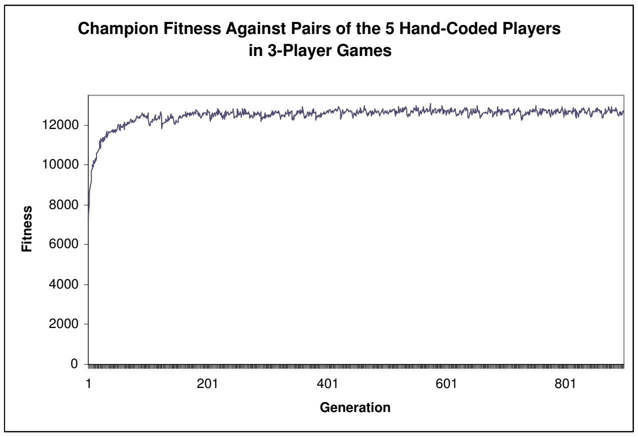

Figure 7: **Fitness Against Pairs of Hand-Coded Players.** The fitness of the networks levels out early in the simulation and does not improve within the allotted 900 generations, even though there is still clear room for improvement. Unlike in the two-player experiment, no fortuitous jump in fitness occurred.

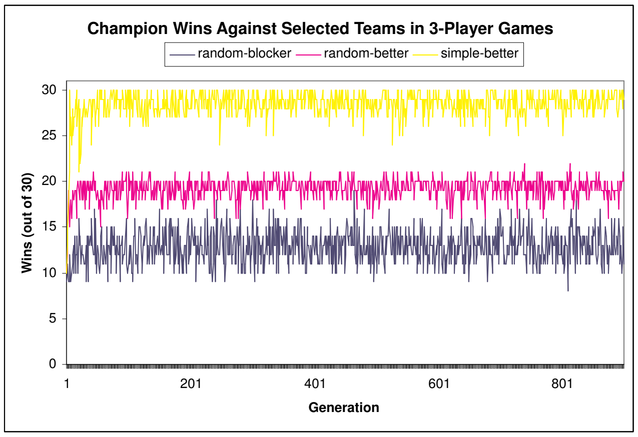

Figure 8: **Wins Against Selected Teams of Hand-Coded Players.** Number of wins against the *Random-Blocker*, *Random-Better* and *Simple-Better* teams. Surprisingly, the networks perform best against the *Simple-Better* team.

6.2 Three-Player Pente

Within the 900 generations allotted, the networks from the three-player games do not evolve to the point where they consistently win all the games they play. Even the 2000 generations afforded to the two-player experiment may not have been enough to evolve the three-player networks properly. The fitness level evens out early on (Figure 7), and it would likely take a fortuitous jump in fitness, as in the first experiment, to reach anything significantly higher. Of course, if and when such a jump would occur is uncertain.

The number of wins achieved by the three-player networks against selected teams is shown in figure 8. Surprisingly, the networks do best against the *Simple-Better* pair. Observation indicates that *Simple* player and the *Better* player interfere with each other slightly in three-player games because the board is very crowded and they end up blocking each other's paths. However, as can be seen, the networks do not perform as well against the *Random-Blocker* or *Random-Better* pairs, nor do they perform particularly well against any of the teams not shown. For all teams of opponents the number of wins hovers around the same value for most of the generations. There is some fluctuation, but it is clear that the number of wins in each case has leveled out.

6.3 Effects of Paying Attention

The simple recurrent networks performed poorly in comparison to the feed forward networks against the *Better* players. The recurrent networks with and without *attention* had erratic performance that fluctuated wildly throughout the course of evolution, whereas the performance of the feed forward networks raised quickly and remained stable at a point where the networks were consistently winning all 30 games against the *Better* players. Furthermore, there seems to be no significant difference between the recurrent networks with *attention* and those without (Figures 9, 10).

It is particularly surprising that the simple recurrent networks without *attention* quickly reach a point where they

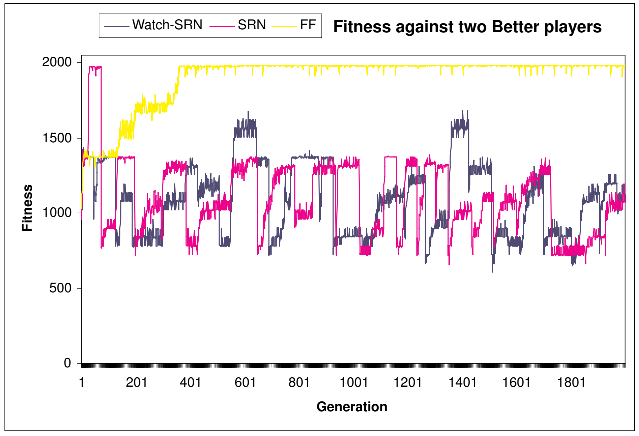

Figure 9: **Fitness Against Two *Better* Players.** The fitness of the feed forward networks (FF) shoots to the top and stays there. The fitness of the simple recurrent networks (SRN) also goes to the top, but then quickly drops and begins to fluctuate. The fitness of the simple recurrent networks with *attention* (Watch-SRN) fluctuates throughout the entire run. The performance of the simple recurrent networks with and without *attention* is inferior to that of the feed forward networks, and generally inconsistent.

can win all games against the *Better* players, but then just as quickly lose this ability and never regain it. The fluctuations in both fitness and number of wins demonstrated by both types of recurrent networks indicate that information from recurrent links is of dubious value, and is perhaps confusing and unnecessary when all information needed to play the game is available on the board.

7 Discussion and Future Work

While evolving networks to play against multiple combinations of opponents has proven exceedingly difficult, it is nonetheless possible to evolve networks that successfully handle a single test well, as in the experiment where two

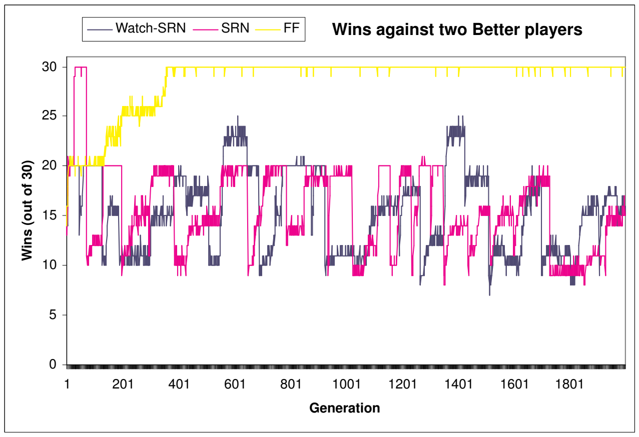

Figure 10: **Wins Against Two *Better* Players.** The feed forward networks (FF) demonstrate the ability to consistently beat two *Better* players. The simple recurrent networks without *attention* (SRN) also play well for a brief time, but then inexplicably start losing. Networks with *attention* (Watch-SRN) never reach a point where they are always winning.

Better players were the only opponents of the feed forward networks.

The fact that the feed forward networks outperformed the simple recurrent networks indicates that memory is not necessary, and in some cases actually confusing, when playing a game of perfect information. Of course, the purpose of the recurrent links was to maintain memory of the strategies being used by the opponents, which would have in theory allowed a network to detect whether or not its opponents were colluding against it. This may still be possible with recurrent links, but was not adequately tested in this experiment since the opponents the networks faced had such simple strategies, and did not in anyway collude with each other.

Evolving networks to play in two-player games has also proven possible, though the difficulty that the networks had in achieving this goal indicates that there is some problem in how the networks are being evaluated. It could be that the fitness function as defined gives too large an incentive to lose, so long as it takes a long time to lose. It could also be that expecting the networks to deal with different turn orders is too harsh, since according to the results, attaining over 10 wins was fairly easy for the networks. This indicates that the networks likely had an easy time winning when they were the first player, since 10 out of every 20 games were played as the first player, and had difficulties as the second player.

In the case of three-player games this problem became worse, since the networks also had to play as the third player. If being the third player is such a huge disadvantage, then perhaps the strategy required to be competitive as the third player is fundamentally different than the strategy used as the first or second player. This indicates that it might be better to evolve different populations of networks to specialize in playing with different turn orders.

Another reason that the networks may have had problems winning is that they had no knowledge of how many captures had been made by any player. The omission of this knowledge was an oversight, which could quite possibly have given the networks a blind spot when it comes to winning or losing as a result of captures. However, perhaps instead of pushing forward and adding to the networks the ability to sense captures, we should step back and study how well networks evolve to play multi-player Go-moku, which has no captures. Although Go-moku has already been extensively studied, multi-player Go-moku has yet to be considered. Success in the domain of multi-player Go-moku could serve as a stepping stone to success in future experiments involving Pente.

Other directions for future work include the following:

7.1 Co-Evolution

The five hand-coded players used in these experiments were very simplistic, and are easily beaten by human players. As a result, the networks evolved to beat them are also easy for humans to beat. In order to get better absolute performance from the networks, better opponents will be needed, since evolution only occurs when there is sufficient pressure to improve. Although better hand-coded players could be used, such an approach would still only produce networks that played well enough to beat the hand-coded players. In order for continued improvement to occur, a co-evolutionary method could be used.

Co-evolution is the evolution of two or more separate species that compete with each other in an evolutionary arms race. Many methods for co-evolution, such as “Hall of Fame” (Lubberts and Miikkulainen 2001; Rosin and Belew 1997) and “Layered Pareto Archive” (Monroy et al. 2006), have been proposed and applied to games, though more work needs to be done in applying co-evolution to multi-player games to see how evolution addresses the issue of collusion and other multi-player game issues.

7.2 Communication

In multi-player games played by humans, collusion often arises as the result of direct communication between players. Players may inform opponents of beneficial moves, so long as they also benefit as a result. Players may even make explicit agreements to collude against another opponent. Lying is also a possibility.

These social aspects of multi-player games have yet to be considered, and are worthy of study. One way to do so with neuro-evolution would be to have some of a network’s outputs be inputs to the networks of other players. It would be interesting to see if traits such as trust, loyalty and deception could evolve in such a setting.

7.3 Scalable Architecture

The setting for the experiments in this paper was a 7 by 7 board allowing up to three players. Even with such small boards and so few players, the number of inputs and outputs required per neural network was quite large. These restrictions on the size of the board and the number of players are problems that need to be overcome by more scalable architectures.

An architecture that deals with boards of large, and even varying size has already by proposed by Stanley and Miikkulainen (2004). The proposed networks play Go with a neural network that acts as a "roving eye". The network is of a fixed size, and can only view a portion of the board at a time, but is able to move around the board to consider different regions of it. Recurrent connections help maintain memory of the spaces previously viewed.

An architecture that scales to multiple players has, to this author's knowledge, not been proposed. Because so much previous work has focused on two-player games, the issue of varying numbers of players has not been seriously considered.

8 Conclusion

Multi-player gaming is an important area in need of further study. This study demonstrates that in spite of neuro-evolution's success in two-player games, neuro-evolution of players for three-player games is considerably more difficult. This study also demonstrates that simple recurrent networks are of little use in games of perfect information against simple opponents. The effects of *paying attention* by observing the board on other players' turns are negligible. When no information is hidden from the players, feed forward networks are sufficient, at least against simple opponents.

References

Allis, L., van den Herik, H., and Huntjens, M. (1993). Go-Moku and Threat-Space Search. Report CS 93-02, Department of Computer Science, University of Limburg, Maastricht, The Netherlands.

Fogel, D. B., and Chellapilla, K. (2002). Verifying Anaconda's Expert Rating by Competing Against Chinook: Experiments in Co-Evolving a Neural Checkers Player. *Neurocomputing*, 42(1-4):69–86.

Fogel, D. B., Hays, T. J., Hahn, S. L., and Quon, J. (2004). A Self-Learning Evolutionary Chess Program. *Proceedings of the IEEE*, 92(12):1947–1954.

Freisleben, B. (1995). A Neural Network that Learns to Play Five-in-a-Row. In *2nd New Zealand Two-Stream International Conference on Artificial Neural Networks and Expert Systems (ANNES-1995)*, 87–90. IEEE Computer Society.

Gabrel, G., and Braunlich, T. (2004). Pente. Rules to Pente included with the game.

Gomez, F., and Miikkulainen, R. (1997). Incremental Evolution of Complex General Behavior. *Adaptive Behavior*, 5:317–342. Also Available from http://nn.cs.utexas.edu/.

Gomez, F. J., and Miikkulainen, R. (2003). Active Guidance for a Finless Rocket Using Neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*. Also Available from http://nn.cs.utexas.edu/.

Katz, W. T., and Pham, S. (1991). Experience-Based Learning Experiments using Go-moku. In *Proceedings of the 1991 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, 1405–1410.

Lubberts, A., and Miikkulainen, R. (2001). Co-Evolving a Go-Playing Neural Network. In *2001 Genetic and Evolutionary Computation Conference Workshop Program (GECCO-2001)*, 14–19. San Francisco, CA: Kaufmann. Also Available from http://nn.cs.utexas.edu/.

Luckhardt, C., and Irani, K. (1986). An Algorithmic Solution of N-person games. In *Proceedings AAAI-86*, 158–162.

Monroy, G. A., Stanley, K. O., and Miikkulainen, R. (2006). Coevolution of Neural Networks using a Layered Pareto Archive. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO-2006)*, 329–336. New York, NY: ACM Press. Also Available from http://nn.cs.utexas.edu/.

Moriarty, D. E., and Miikkulainen, R. (1995). Discovering Complex Othello Strategies Through Evolutionary Neural Networks. *Connection Science*, 7:195–209. Also Available from http://nn.cs.utexas.edu/.

Perez-Bergquist, A. S. (2001). Applying ESP and Region Specialists to Neuro-Evolution for Go. Honors Thesis, Technical Report CSTR01-24, Department of Computer Sciences, University of Texas, Austin, TX.

Richards, N., Moriarty, D., and Miikkulainen, R. (1998). Evolving Neural Networks to Play Go. *Applied Intelligence*, 8:85–96. Also Available from http://nn.cs.utexas.edu/.

Rosin, C. D., and Belew, R. K. (1997). New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29.

Stanley, K. O., and Miikkulainen, R. (2004). Evolving a Roving Eye for Go. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*. New York, NY: Springer-Verlag. Also Available from http://nn.cs.utexas.edu/.

Sturtevant, N. (2004). Current Challenges in Multi-Player Game Search. In *Computers and Games*. Also Available from http://www.cs.ualberta.ca/∼nathanst/papers.html.

Tang, W. H., Moura, A., and da Rosa, A. (1999). Using Genetic Algorithms in the Game Five-in-Line (Gomoku). In *Proceedings of the Second International Symposium on Artificial Intelligence - Adaptive Systems (ISAS-1999)*, 167–173. Also Available from http://www.laseeb.org/publications/.

Wàgner, J., and Viràg, I. (2001). Solving Renju. *ICGA Journal*, 24(1):30–34.

Wu, I.-C., and Huang, D.-Y. (2005). A New Family of k-in-a-row Games. In *11th Advances in Computer Games Conference (ACG11)*. Also Available from http://java.csie.nctu.edu.tw/∼icwu/connect6/.