Evolving Neural Networks for the Capture Game

George Konidaris   Dylan Shell   Nir Oren

School of Computer Science
University of the Witwatersrand, Johannesburg
Private Bag 3, 2050 Wits, South Africa

Abstract

This paper proposes the use of a genetic algorithm to develop neural networks to play the Capture Game, a subgame of Go. The motivation for this is twofold: to evaluate and possibly improve upon current genetic algorithm variants in order to produce a good player and (more importantly) to use this process to examine the properties and processes that are present in evolutionary systems in an attempt to shed some light on the phenomena that are required for an evolutionary process to produce robust, perpetually improving individuals and avoid local minima without any outside interaction. A brief survey of related work in the area is given, which highlights some of the interesting research questions that remain. This is followed by an outline of a distributed system that has been developed for use in the experimental evaluation of some of the proposed ideas and some of the initial results generated by the system.

1 Introduction

Game playing has long been a fertile area for research into Artificial Intelligence. Popular board games such as Chess, Go, and Backgammon provide simple, elegant and conceptually clean environments for research into intelligent systems that have both surprising depth and much existing theory. From early on in the development of AI, board games were considered a good place to start thinking about intelligence [18], and right from the beginning, machine learning was used in an attempt to develop good players [17].

More recently, the use of machine learning has produced a world-class Backgammon program that has, in at least a few cases, produced genuinely novel strategies that improve on current human practice [20]. Despite the expensive computational process involved, machine learning procedures that can produce good players are particularly interesting and valuable because the players produced can be generated without human intervention, and then tested against human expertise. This often provides insight into the development process as well as the game itself.

This paper proposes that an evolutionary process can be used to develop neural networks to play the Capture Game, a subgame of Go [12], and that the use of various algorithms and methods can provide insight into the properties and processes that exist in evolutionary environments that are capable of evolving complex, dynamic, and robust individuals without falling into local minima.

2 The Capture Game

The Capture Game is a simplified but important version of Go where the first player to capture an opponent's piece wins. Although Go is strategically a very complex game, a thorough understanding of the tactical aspects of the game is usually a prerequisite for the mastery of its strategic subtleties. The Capture Game is often used to provide beginners with practice that strengthens their tactical abilities and is capable of producing didactic tactical situations quickly and often.

The Capture Game is usually played on a 9x9 board that is initially empty. The opposing players (black and white) take turns placing pieces, or *stones*, of their colour on the board, with the black player starting. The stones are placed at the intersections of the lines on the board, as shown in Figure 1, where each players has placed three stones.

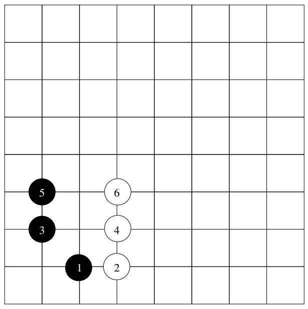

Figure 1: A Capture Game Board

In Figure 1, the black stones 3 and 5 form a *group*. Two stones are called *connected* if they are directly adjacent to each other horizontally or vertically (but not diagonally). A group is a set of one or more connected stones. In Figure 1, the white stones marked 2, 4 and 6 form a group. In the Capture Game (and in Go) groups must be captured as a whole, and cannot be split once they are connected.

In order to capture (or kill) a group, all of the group's *liberties* must be removed. A liberty is an open intersection horizontally or vertically adjacent to a stone in the group – for example, the black group consisting of the stones 1 and 3 in Figure 2 has 3 liberties (one beneath it and two to its left), and the white stone marked 2 can be captured if black places a stone on the intersection to its right.

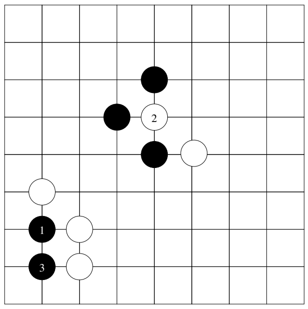

Figure 2: A Typical Capture Game

Stones placed on the edge of the board have fewer liberties than usual (since they have no liberties to one side), and suicidal moves (placing a stone where it has no liberties) lead to an immediate loss, unless placing the stone in that particular position results in the immediate capture of an opposing group.

Although the Capture Game omits much of the strategic subtlety of Go, it includes some of the concepts in the game that all players must master before they can become good Go players. For example, in Figure 3, the configuration on the left has an *eye*.

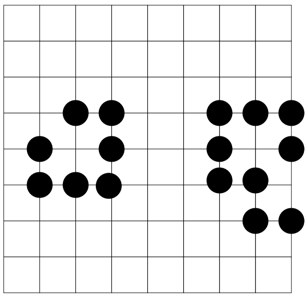

Figure 3: Groups with Eyes

An eye is an empty space within a group. The presence of an eye in the group on the left in Figure 3 makes it difficult to kill; white must first surround the entire group, and then place a stone inside the eye to kill the group – placing a stone inside the eye before surrounding the group would be a suicidal move. However, there is no way to kill the group on the right in Figure 3 because there is no way to close both eyes simultaneously. A group that cannot be killed is called an *alive* group. The fact that two eyes makes a group alive is one of the first lessons taught to new Go players, and the Capture Game is often used to reinforce it.

The Capture Game represents a suitable test domain for intelligent system approaches in general [5], and the evolution of neural networks in particular. While it is significantly simpler than Go (for which there is no known master-level program [16]) the Capture Game possesses some of the subtlety of full Go, and hopefully many of the skills useful in the Capture Game are also useful in Go. If so, a good Capture Game player could be considered an important component of a Go playing agent; or, alternatively, it could be considered the first step in the incremental development of such an agent.

3 Neuro-evolution

Neuro-evolution refers to the use of an evolutionary process (usually a standard genetic algorithm) to develop a neural network that performs a given task. Typically, the evolutionary algorithm is used to determine the connection weights in a fixed-architecture neural network, although in some more interesting cases the architecture itself or even the combination of architecture and weights have been evolved [4].

The combination of neural networks and genetic algorithms seems to be a natural fit, because although neural networks provide a robust and highly distributed form of knowledge representation that is capable of learning, they are usually not explicitly designed and often generated or learned [14]. Therefore, the use of a genetic algorithm to "breed" good neural networks would seem to be a natural way to obtain good networks.

Neural networks have been extensively used to learn to play games [7, 11, 16, 20], partially because of the natural fit between a fixed network input and a game board, but also because neural networks are known to be good at pattern matching [14], which is thought to be important in games, especially complex games such as Go.

However, the use of a genetic algorithm to develop neural networks comes with certain pitfalls, the foremost of which is that it is often difficult to provide a fitness rating for a network that plays a game, especially a game like Go, which is very difficult and has no known strong computer players [16]. Systems where an external, hand-written opponent is used are unlikely to develop players that are very much better than the given opponent [16] even in more direct learning situations [7]. On the other hand, systems where the individuals are evaluated directly against each other are prone to premature convergence [8] when a particularly strong individual dominates the population and eventually breeds other individuals out.

The problems encountered by researchers when using genetic algorithms to develop complex or adaptive behaviour are typically solved by a either a process of parameter variations, or the use of *ad hoc* methods that seem to work for the problem at hand. One of the issues that this paper proposes examining is whether or not there are fundamental principles that belong to evolutionary processes that render them capable of developing complex behaviour in general.

Several surveys of the work in this field, organised primarily around the representation scheme used, are available for those interested in reading further [3, 4, 21].

4 Related Results

It has already been noted that the use of game playing [18] and learning in game playing [17] is nearly as old as artificial intelligence research. Unsurprisingly, a considerable body of work exists on the subject of neuro-evolution. The following two sections present the application of both direct learning in neural networks and the use of neuro-evolution for game playing and related tasks.

4.1 Direct Learning

The use of more direct learning methods for playing games is perhaps a more intuitive approach to game playing than the use of a genetic algorithm. Game playing as an illustrative example is used in at least one major machine learning text [14], and one of the earliest known example of machine learning was for the game of checkers [17]. However, two recent results are of particular interest to the ideas examined in this paper.

The first details the use of reinforcement learning for neural networks learning to play Go-moku [7]. In this particular case, the networks used external teachers for learning (a random player, a simple program, and a strong public-domain program). The results indicated that although learning of this form is feasible, the strength of the learned networks are dependent upon the strength of the opponents used, and in particular, that "if the opponent of the network selected as the trainer is too weak, the network does not learn at all, if it is too strong, the network will not be able to reach best playing quality" [7]. It is clear that removing the need for an external opponent would benefit the process by removing both a parameter and an external interaction bias factor from the system.

In another recent result, a reinforcement learning method called *temporal difference learning* was used to develop a Backgammon-playing neural network that performs near the level of the best players in the world [20]. Of particular interest is the fact that the network was developed *without the use of an external opponent*, and hence was able to contribute genuinely novel strategies at the highest level of human play. This case makes it clear that although most attempts to use neural networks to play games are not entirely successful, this is more likely due to weaknesses in the methods used than the use of neural networks as a learning mechanism.

4.2 Evolutionary Algorithms

There have been several research efforts aimed at using evolutionary methods to produce game playing, and several methods that, although not directly concerning game playing, have potential application to the problem.

A direct use of neuro-evolution in an attempt to evolve Go playing neural networks [16] found that neural networks could be evolved to defeat a simple external opponent. Unfortunately, the external opponent again imposed a limit on the level to which the networks could play. However, the extent to which the external opponent influenced play was more pronounced than usual – random noise had to be added to the opponent in the evolutionary process to prevent the evolved networks from learning completely specific sets of moves that were able to beat it. This illustrates the tendency that evolutionary processes display towards the discovery of simple, mechanical solutions which are not capable of generalisation [22], unless there is significant selection pressure otherwise. However, the results of the process with the addition of random noise was promising, even though the process was computationally expensive [16].

A more interesting approach was the use of competitive co-evolution to evolve Go playing neural networks [11]. Here, the need for an external opponent is removed, because two populations are maintained: a *host* population that attempts to learn to play Go, and a *parasite* population that attempts to learn to foil individuals of the host population. The results of this approach are extremely promising. In particular, the resulting evolutionary process appears to be open-ended: the population does converge, but only temporarily, and an "arms race" occurs between the two populations, where each improvement in a population is followed by a counter-improvement in the rival population. This ensures that the process does not stagnate, and that it continues to produce promising individuals.

An interesting modification of this idea is the use of *enforced subpopulation* (ESP) neuro-evolution, where several populations of neurons (one for each neuron position in a neural network architecture) are evolved separately [9]. Here, the populations of neurons evolve independently but are evaluated together, encouraging the evolution of co-operative behaviour. Similar methods have been used for the evolution of multi-agent systems [22].

Another interesting aspect of evolutionary systems is the use of *incremental evolution* [9] in genetic algorithms. In incremental evolution (sometimes called "shaping") individuals are evaluated on a number of simple subgoals, as well as the overall goal, with the aim of rewarding behaviour that brings the agent closer to the overall goal. Incremental evolution is often useful when the goal behaviour is much too complex to evolve directly, and has been found to be successful in several applications [1, 15, 22]. As an example of the possible use of incremental evolution for game playing, the Capture Game could be considered a subgoal of the overall goal of learning to play Go.

The final methodology that will be examined in this paper is the use of the "culture" present in a neuro-evolution population through the use of *culling* and *teaching* [13]. The individuals that form a population in an evolutionary process carry a significant amount of information about the kinds of solutions that are viable for the given task. This information can be considered a "culture" of problem solving behaviour. In culling, a large number of offspring are produced from each crossover operation and compared to the current population. Those that seem to behave too differently to the current population are "culled", hopefully resulting in fewer degenerate solutions as a result of crossover. In teaching, new offspring are trained to behave in the same way as their parents before they are evaluated. Both of these operations are designed for use in an environment where fitness evaluations are expected to be expensive. An interesting side-effect of teaching is that networks that are good at learning how to perform well at the given task are evolved, rather than networks that are good at the given task directly [13].

From the above overview it is clear that a great deal of interesting work has been done relating evolutionary processes and neuro-evolution. It is also clear that a great deal of work remains to be done – while many of the interesting methods given above show promise, the reasons behind their success and their place within the general framework of evolutionary processes are not well understood.

5 Research Questions

The primary aim of this paper is to establish a basis for future research into several key "question areas" where there is potential for improvement in current practice. The areas in which future research will take place are examined in the following sections, starting with those that are directly examinable, and moving towards higher level phenomena which are more complex and may present observational difficulties.

5.1 Evolutionary Parameters

Typical experiments in evolutionary processes center around the variation of basic parameters (such as form of crossover, mutation frequency, replacement rates, network size, etc.) and the effects they have on the resulting process. Although this is a widely covered area, the question of whether or not varying these parameters *during* the evolutionary run can introduce some of the higher level phenomena observed in evolutionary systems (such as arms races or punctuated equilibria) does not appear to have been extensively explored.

5.2 Genotypic Representation

The representation of neural networks is fundamental to any attempt to understand the process underlying neuro-evolution and the results it produces. It has been suggested that a less direct way of encoding a neural network representation than simply a concatenation of weights may be the most successful approach [3].

There are two important aspects to the problem of representation. The first is the distinction between phenotype and genotype [19], which is the distinction between a genome instance and the individual that results from the combination of the genome and a development process. In nature, a genome is more akin to a *recipe*, providing a set of instructions on how to build a phenotype, than a *blueprint* of a phenotype. Various conditions have been proposed for the evaluation of novel representation schemes [2].

Perhaps more immediately important is the second aspect of representation, which is that of scalability [2]. In any evolutionary system aimed at continuous development the genomes used should be able to represent an unlimited number of phenotypes, without restricting size or form [19]. This ability is important both in terms of the differentiation of species which may occur in some evolutionary systems, but also in terms of the flexibility of the process. A process which is stuck with a single, rigid phenotype form can never do better than is possible in that particular form, and since experimentation is time consuming, many experiments only consider a small range of the network sizes and architectures available.

5.3 Lifetime Learning

One of the more interesting areas of research in neuro-evolution is that of *lifetime learning*. Since neural networks are capable of learning, it makes sense to use their ability to learn within the evolutionary process. Such hybrid systems, where the broad global genetic algorithm search is complemented by local network learning are a natural and interesting fit [3]. It is possible that selection pressure that favours networks that are capable of learning in the environment could result in networks that possess the ability to adapt to a relatively dynamic environment.

Another interesting aspect of lifetime learning was demonstrated in an experiment involving teaching and culling where the network resulting from a crossover between two individuals was trained to perform similarly to its parents [13]. The resulting network population was no better than random before training, but was able to adapt to parental training very quickly, yielding a fast and efficient evolutionary process. There is some potential for the use of similar techniques in simulation, when the resulting individuals must eventually be used in a real environment.

5.4 Arms Races

An arms race is a situation where two competing species are forced to continually adapt because improvements in one species create selection pressure for counter-improvements in the other [6]. Such a situation creates an ongoing evolutionary process that avoids stagnating in suboptimal solutions for extended periods of time. In natural systems, arms races are responsible for some of the most striking examples of the specialisation of predator and prey [6].

Similar ideas have been explored in artificial evolutionary systems [11, 22], although often several species are evolved in cooperation, rather than competition. However, it is often difficult to frame a problem in terms of competing species without introducing bias into the process, and therefore this idea has not been widely explored.

5.5 Incremental Evolution

A more general idea is that of *incremental evolution* [9]. Here, individuals are not ranked solely on performance at a single task. Rather, they are ranked using several sub-tasks or abilities that may prove useful in eventually achieving the goal task. This provides selection pressure for partial achievement of the goal, which is important in situations where the target behaviour is complex and unlikely to appear soon.

The artificial addition of sub-goals is problematic in most domains because the target tasks are typically not well understood or not even well defined. Therefore, the use of incremental evolution has not been explored in the context of game playing. It is also interesting to examine whether or not such a reward system can arise naturally in an evolutionary system, perhaps evolving out of individual mating preferences, rather than being artificially included in the system.

It would also be interesting to evaluate the relative importance of incremental evolution and arms races. It may be true that arms races are the mechanism by which incremental evolution comes about in natural system. This is particularly interesting with respect to the idea of *punctuated equilibria* [10], where a species converges to a point and then stays there until some change in the environment forces rapid adaptation. This suggest that the punctuated equilibria found in nature are symptomatic of incremental evolution forced by the environment.

5.6 Open Ended Evolution

An open-ended evolutionary process is one in which individuals evolve continuously, rather than prematurely converging on a sub-optimal solution and staying there [19].

The achievement of an open-ended evolutionary system is not an easy goal. No known artificial evolutionary system has exhibited the capacity for entirely open-ended evolution, although this is not surprising given the rigid genotype representations usually employed. However, working towards an open-ended system would be a way to avoid the problem of premature convergence, which has always plagued artificial evolutionary systems.

There seem to be several methods employed in nature to facilitate open-ended evolution. The arms race phenomenon is an obvious example of the kind of condition that would preclude either of the two competing species from stagnating for too long. The dynamic environments typically encountered in the real world coupled with gradual climate change are another, and the introduction of adaptive behaviour is yet another. It would seem that an approach that coupled incremental evolution, differentiation and competition, lifetime learning, and a dynamic and complex environment (or evaluation function) would be the natural way to facilitate open-ended evolution. Whether or not these are requirements, and whether or not others remain, seems to be an open question.

6 A Distributed Implementation

An implementation is a precursor for any serious experimental investigation of the questions outlined above. The initial implementation that is intended to form the basis of future experimentation is discussed in the following sections.

Simulating the evolutionary processes of nature on a serial computational device puts us at an immediate disadvantage since the process of evolution is naturally distributed. The sheer number of evaluations that need to occur in a genetic algorithm that seeks to find a complete (or approximate) ranking can be very large. Taking nature's lead, our current implementation seeks to utilise the natural parallelisability of the problem in making evolutionary experiments feasible.

6.1 The Distributed Architecture

In an attempt distribute the computation as widely as possible we considered the problem of dealing with the large number of evaluations that need to occur to obtain a ranking of the individuals. A natural way to handle this is to represent each individual as a separate process on a separate computational node.However, even if a small population is used, the number of evaluations[1] needed in ranking may be prohibitively large (for example in a round robin tournament the evaluations are $\Theta(n^2)$ where n is the number of individuals) in terms of the inter-node communication required. Such evaluations would need to occur for each generation.

In order to exploit the possibility of using more machines than individuals it was decided to distribute the evaluations over the computing nodes. Each node thus solves a subset of the number of evaluations per epoch that are required in ranking.

The ranking of individuals is controlled by a central node, called the *Tournament Server*. Each of the computational nodes request blocks of games that need to be played. In reality *genome identifiers* specify which individuals need to be played. The node then requests the individual from the *population server* by using the genome identifier. The individuals are cached at the node since unique identifiers are given for each genome. The tournament server recognises when it has received all the results for the current generation and then requests that the population server perform breeding (cross-over) and mutation. When genomes are mutated they are assigned new identifiers, in order to avoid cache consistency issues.

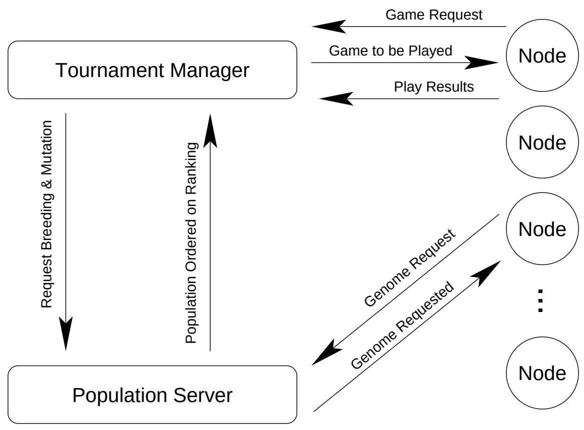

Figure 4: The Distributed Architecture

This "pull" methodology employed allows for the certain degree of robustness especially since the computational nodes are in our case diskless workstations in undergraduate laboratories, and are thus prone to being reset. The tournament server caters for such situations by allocating duplicates games to be played if results are not returned within a reasonable interval.

The Tournament and Population Servers run on a dedicated machine so they do not have to cater for such perilous conditions. The population is, however, written to disk at the end of each generation

[1] An evaluation equates to playing two individuals against one another, each playing once as black as once as white.

as a precaution, and it allows the system to be restarted at any given generation for the purposes of analysis.

Figure 4 gives an overview of the system. It illustrates that the actual population, kept by the population server, has been separated from the fitness evaluation. Our working implementation, however, has both population server and tournament manager running on a single node since a single machine is capable of running both.

Although the architecture as described is significantly more complex than a standard approach, the inherent parallelisability of genetic algorithms [14] means that with the increase of nodes there is almost linear speed up.

6.2 Neural Network and Genetic Algorithm Parameters

There are numerous parameters in any neuro-evolution process that need to be carefully considered in order to obtain optimal performance. Since the aim of the early experiments performed were for evaluation of the distributed architecture, some of the choices made may not be optimal. This should not detract from the usefulness of the experiments.

One difficult question that arises early is that of neural network structure. Since a large number of network evaluations need to occur, a feed-forward network has an advantage over other structures which are likely to make evaluation prohibitively slow. Neural networks in existing work with games of similar complexity have tended to use the standard three-layer feed-forward structure [16, 11], which is a network with a single hidden layer. It has, however, been suggested that genetic algorithms seem to perform better with deeper networks [3]. Hence it was decided that a two hidden layers should be used.

The 81 intersections on the 9x9 board were mapped directly as input to the network. Figure 5 shows the network structure in our current implementation. Note that the second hidden layer feeds into a single output. During play the network is evaluated for each possible position a stone could be placed (i.e. all open intersections), and a stone is played where the output reacts strongest. In this way the output represents the networks preference for placing a stone in a particular location.

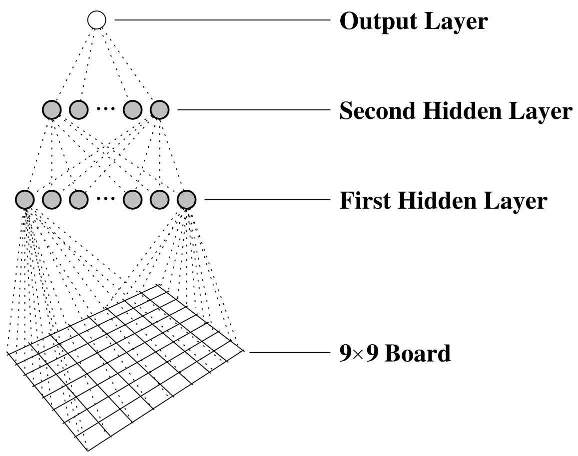

Figure 5: A Neural Network with Two Hidden Layers.

Simple concatenation was used to encode the network weights as a genome. This results in the weights of a particular unit being placed closely together. It has been suggested that keeping functional units localised in the genome string produces better results using a standard cross-over operator than dispersing them[3]. The implementation uses standard cross-over, where better individuals have a higher chance of mating. Individuals were chosen for mutation without any bias.

6.3 Implementation Details

Each of the components were implemented using Java. All interprocess communication was achieved by using the Remote Method Invocation (RMI) feature of the language. Various problems that arose were conveniently solved using constructs in the language – for example the possibility of dead-lock was avoided using synchronised functions.

Hidden Layer Sizes	Total Chromosome Size	Mean Time for a Game	Mean Time for an Evaluation
(10,10)	920 genes	$0.3907s$	$1.1912 \times 10^{-4}s$
(80,30)	8910 genes	$1.8322s$	$5.5860 \times 10^{-4}s$
(80,80)	12960 genes	$1.8793s$	$5.7260 \times 10^{-4}s$
(100,100)	18200 genes	$1.9006s$	$5.7945 \times 10^{-4}s$

Table 1: Fitness Evaluation Timing Information

Table 1 shows some timing data collected from the current implementation. The timing details are for neural networks of various sizes, and they show (as one might expect) that increasing network size increases time required to play a game. In practice it was found that actual time required to complete a generation varied greatly from the predicted values. This is most likely because the figures presented are for only a single node, and hence do not consider the degradation in performance due to increased network traffic as more computational nodes are added. The network performance is a major factor when using around 100 machines.

In order to fully utilise the network, requests are granted in blocks. Increasing the block size decreases the frequency with which nodes return with results since the larger the block size, the more computation needs to occur between network access. It was found that with a small block size (around 50 games) the network soon became a major bottleneck. Since our machines are diskless workstations, the saturated network resulted in undergraduate students rebooting machines in an attempt to get faster responses. Block sizes of between 200 and 350 are currently in use, and performance is much closer to that which was predicted.

7 Initial Results

Although the structure of the neural network has been discussed, the actual size of the layers in the network has not yet been mentioned. Table 1 indicates that four different sizes were considered.

Initially a network with ten units in each hidden layer was used. Although it had a short evaluation time, it was found that even after more than a hundred generations little progress towards a reasonable solution had been made. The network's play was still close to random, and an analysis of the resulting individuals showed that the strongest networks were often beaten by fresh mutations, which in turn were beaten by newer mutations. Additionally, when observing the individuals, it was found that they played very close to a fixed sequence of moves, and moves made by the opposition were for the most part completely ignored.

It was expected that since the Capture Game is a conceptually simpler game than Go, the number of neurons required would be significantly less than the number used in existing implementations that play Go. Additionally it was hoped that the use of a second hidden layer would make a large difference. It was clear that more neurons than those provided by the two layers of ten would be required. After considering the network sizes in existing Go playing networks [11], the layers were increased to include 100 neurons in each hidden layer.

Since the number of connections had been increased by a factor of about 20, there was a significant increase in the computation required. After 70 generations the networks were demonstrating that the primary problem with the previous representation had in fact been the small number of hidden nodes. The larger networks responded to their opponent's moves, although still not necessarily in the most desirable way.

The computation time required for the larger networks meant that the number of generations had to be kept small. In an attempt to allow for more generations the number of neurons in the layers was reduced. This resulted in the two intermediate size networks, neither of which seem to perform markedly worse than the larger networks after around 50 generations. Figure 6 shows the penultimate board state during a game between two networks.

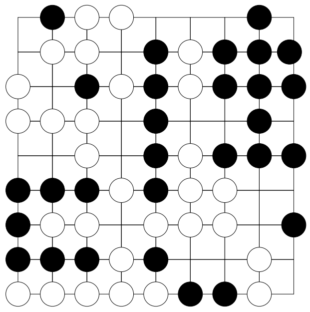

Figure 6: Two networks playing a game

Although none of the networks yet have shown great promise as a Capture Game player, it has been successfully demonstrated that the current implementation of the distributed architecture is useful for testing questions about evolutionary methods. Also the Capture Game has been shown to be of sufficient complexity to make it an interesting domain for further experimental investigation.

8 Summary

This paper examines a number of research areas that are the subject of much ongoing research, and promise interesting results, centered around the use of genetic algorithms to represent and train neural networks for game playing. In addition the Capture Game was proposed as a useful test-bed for future research. We are confident that the relative representational simplicity of the Capture Game, coupled with its difficulty to master, will provide us with a framework that will be useful in attempting to answer some of the important remaining questions in evolutionary computing.

The paper also highlighted some interesting research areas for which it is believed that the Capture Game is a suitable domain for further investigation. It is our belief that the research into the areas presented will prove fruitful in terms of results and providing insight into evolutionary processes and how they are able to produce complex, adaptive behaviour.

Finally, a distributed architecture that will make further computational experiments feasible was presented, along with some early initial results demonstrating that the current implementation of the architecture functions as desired and is capable of speeding up computational experiments involving evolutionary processes.

References

[1] David Andre and Astro Teller. Evolving Team Darwin United. In *RoboCup*, pages 346–351, 1998.

[2] K. Balakrishnan and V. Honavar. Evolutionary and neural synthesis of intelligent agents. In M.J. Patel, V. Honavar, and K. Balakrishnan, editors, *Advances in the Evolutionary Synthesis of Intelligent Agents*, chapter 1, pages 1 – 27. MIT Press, 2001.

[3] R.K. Belew, J. McInerney, and N.N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.

[4] J. Branke. Evolutionary algorithms for neural network design and training. In J. T. Alander, editor, *Proceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applications*, number 95-1, pages 145–163, Vaasa, Finland, 1995.

[5] T. Cazenave. Abstract proof search. In T.A. Marsland and I. Frank, editors, *Computers and Games*, pages 39–54, Hamamatsu, Japan, October 2000. Springer.

[6] R. Dawkins. *The Blind Watchmaker*. W.W. Norton & Company, 1996.

[7] B. Freisleben and H. Luttermann. Learning to play the game of Go-Moku: A neural network approach. *Australian Journal of Intelligent Information Processing Systems*, 3(2):52–60, 1996.

[8] D.E. Goldberg and K. Deb. A comparison of selection schemes used in genetic algorithms. In G.J.E. Rawlings, editor, *Foundations of Genetic Algorithms*, pages 69–93. Addison-Wesley, Redwood City, CA, 1991.

[9] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[10] S.J. Gould and N. Eldredge. Punctuated equilibria: The tempo and mode of evolution reconsidered. *Paleobiology*, 3:115–151, 1977.

[11] A. Lubberts and R. Miikkulainen. Co-evolving a Go-playing neural network. In *2001 Genetic and Evolutionary Computation Conference Workshop Program*, pages 14–19, San Francisco, California, USA, 2001. Kaufmann.

[12] C. Matthews. *Teach yourself Go*. Hodder & Stoughton Educational, London, 1999.

[13] Paul McQuesten and Risto Miikkulainen. Culling and teaching in neuro-evolution. In *Proceedings of the 7th International Conference on Genetic Algorithms*, San Francisco, California, USA, 1997. Morgan Kaufmann.

[14] T.M. Mitchell. *Machine Learning*. McGraw-Hill, Singapore, 1997.

[15] S. Nolfi. Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems*, 22:187–198, 1997.

[16] N. Richards, D.E. Moriarty, and R. Miikkulainen. Evolving neural networks to play Go. In *Proceedings of the 7th International Conference on Genetic Algorithms*, East Lansing, NI, USA, 1997.

[17] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[18] C.E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:265–275, 1950.

[19] T.J. Taylor. *From Artificial Evolution to Artificial Life*. PhD thesis, Division of Informatics, University of Edinburgh, 1999.

[20] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), March 1995.

[21] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, September 1999.

[22] C.H. Yong and R. Miikkulainen. Cooperative coevolution of multi-agent systems. Technical Report AI01-287, The University of Texas at Austin, 2001.