An Empirical Comparison of Evolution and Coevolution for Designing Artificial Neural Network Game Players

Min Shi
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
minshi@idi.ntnu.no

ABSTRACT
In this paper, we compare two neuroevolutionary algorithms, namely standard NeuroEvolution (NE) and NeuroEvolution of Augmenting Topologies (NEAT), with three neurocoevolutionary algorithms, namely Symbiotic Adaptive Neuro-Evolution (SANE), Enforced Sub-Populations (ESP) and Evolving Efficient Connections (EEC). EEC is a novel neurocoevolutionary algorithm that we propose in this work, where the connection weights and the connection paths of networks are evolved separately. All these methods are applied to evolve players of two different board games. The results of this study indicate that neurocoevolutionary algorithms outperform neuroevolutionary algorithms for both domains. Our new method, especially, demonstrates that fully connected networks could generate noise which results in inefficient learning. The performance of standard NE model has been improved significantly through evolving connection weights and efficient connection paths in parallel in our method.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms
Algorithms, Measurement, Performance, Design, Experimentation.

Keywords
Neuroevolution, Neurocoevolution, NE, NEAT, ESP, SANE, EEC, TTT, Gobang.

1. INTRODUCTION
As summarized by Yao [10], the design of artificial neural networks (ANNs) using evolutionary techniques can be roughly classified into three levels: evolving connection weights; evolving architectures and evolving learning rules. In the past decade, more and more approaches have focused on simultaneously evolving connection weights and architectures of ANNs. Both standard evolutionary techniques and coevolutionary techniques have been introduced into ANNs design. The resultant techniques are so-called neuroevolutionary or neurocoevolutionary algorithms.

Neuroevolutionary algorithms evolve ANNs using standard evolutionary techniques. Conventionally, each individual in the population represents a complete network. The fitness of each individual is evaluated independently of other individuals.

Citing techniques of cooperative coevolution, neurocoevolutionary algorithms evolve solutions by decomposing complete networks. Each individual of a population represents a partial network. A complete network is composed through building cooperative relationships among individuals.

Symbiotic Adaptive Neuro-Evolution (SANE) [3], Enforced Sub-Populations (ESP) [1] and NeuroEvolution of Augmenting Topologies (NEAT) [9] are three popular methods for evolving ANNs. NEAT is a neuroevolutionary algorithm in which every individual presents a complete network, while SANE and ESP are neurocoevolutionary algorithms where every individual represents a neuron instead of a complete solution. In addition, a new neurocoevolutionary algorithm, called Evolving Efficient Connections (EEC), has been developed in this work. EEC cooperatively evolves connection weights and connection paths that are respectively represented in their own populations.

The objective of this paper is to empirically compare two neuroevolutionary algorithms, namely standard NeuroEvolution (NE) and NEAT, versus three neurocoevolutionary algorithms, namely SANE, ESP and EEC, for evolving board game players.

A few similar comparisons were previously performed. Moriarty and Miikkulainen [3] compared SANE with standard NE in the Khepera robot simulator. In their analysis, SANE performed more efficiently and maintained a higher level of diversity than standard NE. In [9] Stanley and Mikkulainen compared NEAT with published results of standard NE, SANE and ESP on the double pole balancing with velocity problem. Their results showed that NEAT performed as well as ESP while finding more minimal solutions and using fewer evaluations.

The contributions of our work are twofold: 1) we propose a novel neurocoevolutionary algorithm, EEC, based on standard NE model, borrowing ideas from SANE and ESP; 2) an empirical

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

study is facilitated to compare different methods under as fair as possible conditions, in which each method evolves networks within the same number of generations and in each generation the same number of evaluations are performed. Two board games, Tic-Tac-Toe (TTT) and Gobang, are chosen as our benchmark tasks. The performance of these methods is measured according to their players' strength against hand-coded opponents, learning speed, network complexity and time consumption of evolution.

The body of this paper is organized as follows. Section 2 and Section 3 briefly introduce two neuroevolutionary algorithms and three neurocoevolutionary algorithms. Our experimental setup is presented in section 4. Section 5 illustrates the empirical evaluations for comparing different methods. Our observations and future work are given at the end.

2. NEUROEVOLUTION

2.1 Standard NE

Standard Neuro-Evolution (NE) evolves connection weights of complete neural networks, where each individual in the population represents a vector with all connection weights of a network, and each gene of an individual specifies a weight value between two neurons.

The fact that standard Evolutionary Algorithms (EAs) continually select and breed the best individuals in one population, therefore, losing diversity and eventually converging around a single "type" of individual is a major limitation. In standard EAs, the approaches of maintaining diversity are mainly to decrease selection pressure or increase mutation rate. Therefore, similar problems are also encountered in standard NE, especially when a standard NE is applied to solve complex and dynamic problems [3].

2.2 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) [9] is a new neuroevolutionary technique that evolves both the connection weights and connection topologies of ANNs simultaneously. However, since NEAT does not represent connection weights and topologies in separate populations, an individual of the population still represents a complete network.

A NEAT individual consists of a series of connection status genes. Each gene represents the connection status of two neurons. It contains five elements: 1) in-node neuron; 2) out-node neuron; 3) connection weight; 4) connection status; and 5) innovation number. The length of an individual is flexible and expandable so that evolution is able to both complexify and simplify network topologies.

The first NEAT model was proposed by Stanley and Miikulainen [9], where networks were evolved from no hidden layer structure and complexified by using mutation operators. NEAT overcomes diversity loss by using unstandard NE mutation operators. In NEAT, the mutation operator plays two roles: (1) changing connection weights; and (2) complexifying network topologies. Changing weights via mutation is the same as in standard NE algorithms, where a weight of a network is randomly selected to change. Topological mutation complexifies network structure in two ways, either inserting a new node between an existing connection or creating a connection between two unconnected nodes. The connection status could be therefore changed due to mutation operation. Innovation numbers are used when recombining networks with differing topologies.

James and Tucker added a new topological mutation operator into this model [2]. A delete connection mutation operator was used to remove an existing connection between two nodes. The blended mutations have been shown to perform better than either complexifying or simplifying networks alone [2]. The new approach is therefore used in the NEAT experiments of this work.

3. NEUROCOEVOLUTION

3.1 SANE

Symbiotic Adaptive Neuro-Evolution (SANE) [3] evolves three-layer networks, where the number of hidden nodes of the network is predefined and fixed, but the network connection topology is not. There are two populations evolved in SANE, a population of neurons and a population of network blueprints.

Each individual in the neuron population represents the connection paths and weights of a hidden neuron from the input layer and to the output layer. Each gene of an individual contains two parts: one part specifies the connection path (that is which neuron to connect to) and the other specifies the weight of that connection. All the hidden neurons have the same number of connections, but could have different connection paths from the input layer and to the output layer.

Networks are constructed by combining selected individuals from the neuron population. The information of combination is saved in the blueprint population. Each individual of the blueprint population represents a combination of selected individuals from the neuron population. At the beginning of blueprint evolution, combinations are created randomly. Effective combinations can be maintained and new combination forms can be explored by evolving the blueprint population. A well-contributing neuron does not always cooperate well with any other neurons. Therefore, through maintaining a blueprint population, well-contributing neurons are protected from being eliminated due to ineffective cooperation with some other neurons.

SANE is an intra-population neurocoevolutionary algorithm, that is all the cooperative neurons come from the same population. Each individual of neuron population represents a partial solution instead of a complete network. A complete network is formed by a collection of cooperative neuron individuals. The fitness of a neuron individual is not evaluated independently of other individuals in the neuron populations, but is based on its cooperation. After evaluating all the networks built by blueprints, besides assigning the fitness to each blueprint, each neuron also obtains a fitness value that equals the fitness sum of the best five networks in which the neuron participates. Cooperation only happens when we evaluate individuals, while two populations perform recombination and mutation process independently.

3.2 ESP

Enforced Sub-Populations (ESP) [1] is also a neurocoevolutionary algorithm, where complete networks are evolved by decomposing them into sub-populations of neurons. Different from SANE, ESP is an inter-population neurocoevolutionary algorithm, because all the cooperative individuals come from different populations.

ESP evolves three-layer fully connected networks; it creates a sub-population for each hidden neuron. Each individual of the sub-population represents a weight vector of a hidden neuron that is fully connected with input neurons and output neurons. A complete network is constructed through randomly selecting an individual from each sub-population. A number of networks are created during each generation; each one is evaluated in turn. Each individual of the sub-population obtains average fitness of the networks that it participated in. Neurons are evolved in their own sub-populations, but cooperatively adapt to the problem domain.

At the beginning, the evolution of ESP runs mainly to search the weight space of the network. However, the topology of the network will change when stagnation happens (that is the best fitness of networks have not improved after predefined generations). A hidden neuron will be removed if the neuron does not make enough contribution to the whole solution. Or a new hidden neuron will be added if no neuron can be removed. A delta-coding technique will be applied if no new neuron is added. It selects the best network evolved so far and uses each neuron of the network as a seed of each sub-population. New sub-populations will be generated by perturbation operations of the selected seeds. In this way, both optimal weights and network topologies can be explored simultaneously.

ESP decomposes a network search space into several neuron sub-spaces, which is especially helpful in complex problem domains.

3.3 EEC

Borrowing many ideas from both SANE and ESP, we have developed a novel neurocoevolutionary algorithm, called Evolving Efficient Connections (EEC). EEC separates the search space of a network into two sub-spaces: connection weights space and connection paths space. Thus, it contains two populations.

The population of connection weights evolves weights vectors for fully connected networks as does standard NE. Each individual represents connection weights of a network, in which each neuron hypothetically connects with all neurons in the next layer. While the population of connection paths evolves switches of these connections; each individual represents a series of binary bits to specify the status of the connections, where "0" indicates disconnection between two nodes and "1" indicates connection between two nodes. Figure 1 shows how the connection-weights, individuals, and the connection-paths individuals are related.

Below we provide a brief description of the EEC algorithm.

1. Initialization. The number of hidden neurons of the networks has to be specified at the beginning. Each chromosome of connection weights encodes fully-connected weights vectors with random real numbers. The chromosome of connection paths encodes connection status with a random string of binary bits, corresponding to the connection weights.

2. Evaluation. A complete network is formed through respectively sampling an individual from both the connection-weights population and the connection-paths population. To achieve the best performance of our method, we suggest evaluating every individual of both populations. The top *n* individuals are regarded as *elite* based on their previous evaluation. An e*lite-rate* percent of the cooperative individuals are selected from the *elite* to cooperate with the current evaluated individuals, while the rest are selected randomly from the cooperative population. The resulting network is evaluated on the task and assigned a fitness score. Each individual is awarded an average of the cumulative fitness of the networks in which it participated during one generation.

3. Recombination. The same recombination process of ESP is employed in our algorithm. All the individuals will be ranked according to their fitness within each population. The top 1/4 individuals are breeding members. The lowest ranking half of the individuals will be replaced by offspring of the breeding pairs. One-point crossover and one-point mutation are employed.

4. Iteration. During the generation process, the stagnation-handling strategy of ESP is also used in our method. A similar delta-coding technique of ESP will be employed when the best fitness of networks has not improved after a specified number of generations. In our delta-coding process, the connection-weights individual and the connection-paths individual from the current best network are regarded as seeds of each population. We replace all the individuals of each population with new individuals generated by perturbations of the selected seed. The generation is iterated until an optimal solution is found or the maximum generation is reached.

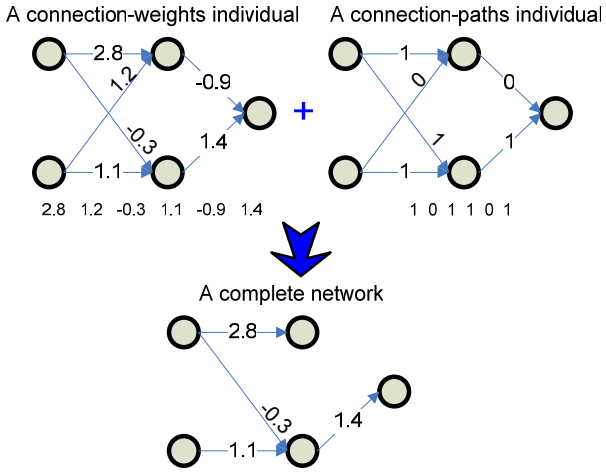

Figure 1. In EEC, connection-weights individual represent weights vectors of fully connected networks, complete networks cut a partial connections specified by connection-paths individuals.

Besides evolving weights, EEC evolves efficient connection topologies of networks. Although we evolve a three-layer feedforward network using EEC in our experiments, this neurocoevolutionary algorithm is able to be applied to other types of networks, such as recurrent networks, multi-hidden layer networks and so on.

4. EXPERIMENTAL SETUP

All of our experiments carried out in this work were implemented in Java[1].

Two domains used to compare the neuroevolutionary algorithms and neurocoevolutionary algorithms described above are Tic-Tac-Toe (TTT) and Gobang. The reasons that these two games are chosen as our test domains are twofold. First, games are widely used for evaluating neuroevolutionary methods, especially board games, and are easily implemented. Second, TTT can be regarded as an abbreviated version of Gobang since the two games have similar game rules but different board sizes. Although we have divided each problem into several increasingly difficult levels, we may be able to analyze how these methods keep their robustness to solve higher level problems though expanding the problem search space.

For the comparison purpose, networks were evolved to play with hand-coded strategies in both games in our experiments.

4.1 Tic-Tac-Toe

TTT is a classic game that is commonly used to evaluate neuroevolutionary algorithms [2, 6]. In TTT two players mark their symbols on a 3×3 board in turn. The one who first obtains three in a row horizontally, vertically, or diagonally wins. The game ties if all grids are filled with symbols and no one wins. The objective of two players is to win or tie the games, and block their opponent's winning moves as well.

James and Tucker [2] evolved networks with three NEAT dynamics, simplification, complexification and blended, to play with five hand-coded strategies of TTT, called BEST, FORKABLE, CENTER, RANDOM and BAD. The same five strategies were employed in our TTT experiments[2].

4.2 Gobang

Gobang is also known as GoMoku, or 5-in-a-row, which was originated in ancient China. This game has some similar game rules to TTT, but plays on a bigger size board, normally 14×14. Similar to TTT, the two players of Gobang take turns to mark their symbols, black and white stones, on the board. The one who achieves five in a row wins. Gobang is as easy to learn as TTT but much more difficult to master even by humans, because the size of the board is bigger than the goal number of symbols in a row.

To both reduce the time consumption for evolving networks and simplify the structure of networks, we chose to do the experiments of Gobang on a 7×7 size board. Even though the board size has been reduced quite a bit, however, it is still difficult to force a win, especially when one plays with sophisticated hand-coded computer strategies.

The hand-coded strategy of Gobang used in our experiments was developed by Vladimir Shashin[3]. 106 patterns composed of four symbols, "*", "-", "o" and "x", with a length of five each (such as "-*ooo", "-x*x-") were created in his program. These patterns were used to make a decision of the best move for each next step after scanning the board state. We divided the hand-coded strategy into three increasingly difficult levels through disabling partial patterns:

1) BEST strategy employed all the 106 patterns.

2) MIDDLE strategy disabled six patterns that include four same symbols, such as "o*ooo" and "*xxxx". It contained 100 patterns.

3) BAD strategy disabled 30 patterns that included four or three same symbols. It employed 76 patterns.

4.3 Networks Representation and Evaluation

All the different neuroevolutionary and neurocoevolutionary algorithms represent the problem domains and were evaluated in the same way.

In both games, the board states are represented to the networks through mapping each grid to both an input neuron and an output neuron. An input neuron is: 1) 1 if play-1 marks its symbol in the corresponding grid of the board, 2) -1 if the corresponding grid is occupied by player-2, and 3) 0 denoting a blank grid that is one of legal moves for both players. After performing the activation computation, a move decision is made by the output neurons. The one with the highest output value corresponding to a legal move is chosen as the move decision of the network player.

To evaluate the performances of a network, 100 matches are played between the network player and a predefined hand-coded player. Each player takes turns going first. The network player is awarded 5 points for a win, 2 points for a tie and 0 for a loss. The fitness of the network, of course, is the cumulation of the points from the 100 matches.

The fitness evaluation described above is used to evaluate networks playing against all types of hand-coded players of TTT and MIDDLE and BAD hand-coded players of Gobang. The BEST hand-coded strategy of Gobang, however, is too strong. No one network is able to evolve the ability to force a win or even a tie using the above fitness evaluation. So a bonus is awarded additionally to the network player in each play when the network plays against the BEST hand-coded player of Gobang. Instead of forcing a win or a tie first, the network player at least attempts to play with its opponents as long as possible. The bonus is, therefore, awarded with the number of filled board grids divided by the total number of the board grids.

4.4 Parameter Setting

The system parameters used in our experiments were not set arbitrarily. Besides using some related work [2, 4, 5] in the same

[1] SANE, ESP and NEAT (ANJI) experiments of our work were implemented based on the relative software developed by Neural Networks Research Group at the University of Texas, see http://nn.cs.utexas.edu/.

[2] The TTT with five hand-coded strategies is included in ANJI package that is an implementation of NEAT developed by James and Tucker, see http://anji.sourceforge.net/. The behaviors of the five hand-coded TTT strategies were described in [2].

[3] The Java game of Gobang developed by Vladimir Shashin can be download at http://down1.tech.sina.com.cn/download/downContent/2004-03-16/9313.shtml.

or similar problem domains, a number of experiments were carried out at the beginning to search effective parameters as well. A comprehensive evaluation for all the parameters of different methods is impractical, because each method has at least a dozen adjustable parameters. Our preliminary experiments mainly focused on the population size, the number of hidden neurons, elite rate, and mutation rate. Table 1 summarizes the parameters chosen for both TTT and Gobang experiments.

Table 1. Parameter settings for TTT and Gobang experiments

Methods	Parameters	Value
All	Number of generations	200
	Number of evaluation per generation	200
NE	Population size	200
	Hidden neurons	10
	Mutation rate of population	0.2
	Breeding rate of population	0.25
NEAT	Population size	200
	Initialized hidden neurons	0
	Weight mutation rate	0.75
	Survival rate	0.2
	Excess gene compatibility coefficient	1.0
	Disjoint gene compatibility coefficient	1.0
	Common weight compatibility coefficient	0.4
	Speciation threshold	0.9
	Add connection mutation rate	0.2
	Add neuron mutation rate	0.2
	Delete connection mutation rate	0.02
SANE	Population size of neurons	1000
	Population size of blueprints	200
	Number of connections for TTT	36
	Number of connections for Gobang	196
	Hidden neurons	10
	Breeding neurons	250
	Elite neurons	250
	Mutation rate	0.02
	Number of top network	100
	Number of top network breedings	20
ESP	Sub-population size	100
	Number of trial networks	200
	Initialized hidden neurons	10
	Mutation rate	0.2
	Generation of stagnation	20
	Breeding rate	0.25
EEC	Connection-weights population size	200
	Connection-paths population size	200
	Number of evaluated networks	200
	Hidden neurons	10
	Mutation rate	0.2
	Generation of stagnation	20
	Breeding rate	0.25
	Elite of connection weights	10
	Elite rate of connection weights selection	0.2
	Elite of connection paths	10
	Elite rate of connection paths selection	0.2

It's impossible to compare these different methods under completely fair conditions, because each method has its own way to represent networks and evolve the network structures. Thus, in order to perform the comparisons as fairly as possible, we evolved the solutions with the same number of generations and built the same number of networks in each generation for all the methods. The activation function of all experiments was sigmoid.

Recommended by some related work using NEAT, the initialized hidden neuron was set to 0 in our experiments, because NEAT has been demonstrated to be a powerful neuroevolutionary algorithm that has the capacity of evolving solutions of minimal complexity [2, 7, 9]. In their work, the adding-connection mutation rate and adding-neuron mutation rate were quite small, generally, smaller than 0.03. In our experiments of NEAT, however, both mutation rates were set with a slightly large value (that is 0.2), because we found that NEAT tended to evolve minimal topologies of networks when very small mutation rates were used, but the final solutions might not be as optimal as those evolved by other methods. Therefore, a slightly high mutation rate was used in our experiments of NEAT to allow the evolution of more robust players through sacrificing the minimal complexity of networks in some way.

As we have presented, EEC performs full-evaluation on all the individuals. For the comparison purpose, however, only 200 evaluations were carried out in each generation. So a half-evaluated ECC was performed in our experiment, in which only every individual in the connection-weights population was evaluated actively. The individuals of connection-paths population were evaluated passively only if they were selected to participate in the cooperation. Using the half-evaluation, thus, some very good connection paths could be lost due to a failure to participate in the cooperation.

NE, SANE, ESP and EEC worked with the same breeding rate and started searching with the same number of hidden neurons. Except in SANE, the same mutation rate, 0.2, was used in NE, ESP and EEC experiments, because big mutation rates normally result in worse performances in SANE. Despite having the capacity of evolving connection paths, SANE was run to evolve fully connected feedforward networks in our experiments.

5. Results

We carried out 20 runs for each type of evaluation. These methods were compared based on the average results of our experiments.

5.1 Tic-Tac-Toe

At first, we evaluated the performance of the different algorithms to evolve network players for the TTT domain.

Table 2 lists the average results of wins, ties, losses and fitness for the 20 best solutions from each type of performance. From these results we can see that, after 200 generations the average final results came out from NEAT, SANE, ESP and half-evaluated EEC were quite close to each other and appreciably better than standard NE for the FORKABLE strategy of TTT and the BEST strategy of TTT. TTT is somewhat simple domain. The difference was not statistically significant among these methods.

Table 2. Average Results of the network players play with five hand-coded players of TTT (20 runs each)

		NE	NEAT	SANE	ESP	EEC
BA	W	100	100	100	100	100
	T	0	0	0	0	0
	L	0	0	0	0	0
	F	500	500	500	500	500
RA	W	94.35	95.8	93.45	95.95	95.65
	T	2	1.05	1.85	1.8	1.2
	L	3.65	3.15	4.7	2.25	3.15
	F	475.75	481.1	470.95	483.35	480.65
CE	W	95.3	96.95	96.55	97.6	96.55
	T	2.3	1.65	1.7	1.15	1.75
	L	2.4	1.4	1.75	1.25	1.7
	F	481.1	488.05	486.15	490.3	490.45
FO	W	39.6	56.15	54.7	47.6	50.45
	T	38.95	25.65	28.15	38.3	30.3
	L	21.45	18.2	17.15	14.1	19.25
	F	275.9	332.05	329.8	314.6	312.85
BE	W	0	0	0	0	0
	T	91.05	98.1	97.7	97.7	96.9
	L	8.95	1.9	2.3	2.3	3.1
	F	182.1	196.2	195.4	195.4	193.8

BA: BAD strategy, RA: RANDOM strategy, CE: CENTER strategy, FO: FORKABLE strategy, BE: BEST strategy.
W: win times, T: tie times, L: loss times, F: fitness

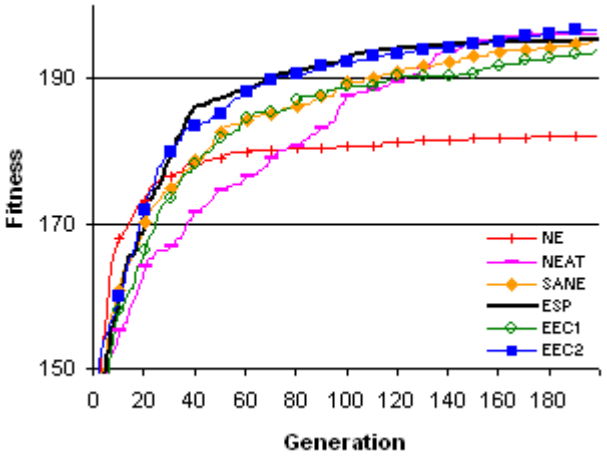

Figure 2. Comparison of average learning speeds of different methods for the BEST strategy of TTT. EEC1 is half-evaluated EEC, EEC2 is full-evaluated EEC. (20 runs each)

To facilitate a more explicit comparison, figure 2 shows the average learning curve for the BEST strategy of TTT from 20 runs. As a whole, ESP learned faster than all of the others. The half-evaluated EEC (line EEC1) performed as well as SANE; the two curves of SANE and half-evaluated EEC almost overlapped each other. There was a big leap at the beginning of the learning curve of standard NE, but soon the premature convergence was encountered after around 60 generations. It's not surprising that the learning speed of NEAT was slower than all of other methods, since NEAT started its search from null hidden neurons. Noticeably, the final results of NEAT exceeded all other methods in the end for the TTT domain.

A significant advantage of NEAT could be in evolving minimal complexity of networks rather than in finding optimal solutions. Table 3 lists the average complexity of the best networks evolved by different methods for the BEST strategy of TTT. NEAT found the most compact networks that, on average, contained 3.35 hidden neurons and 61.3 connections. ESP removed neurons that did not contribute to the solutions, and eventually, found simpler networks than the initialized ones, which, on average, contained 7.25 hidden neurons and 130.5 connections. The networks evolved by EEC, on average, contained 90.3 efficient connections. SANE evolved fully-connected networks in our experiments, so the networks of SANE contained the same number neurons and connections as those of standard NE.

Table 3. Average complexity of best networks found by different methods for the BEST strategy of TTT (20 runs each)

	NE	NEAT	SANE	ESP	EEC
Hidden neurons	10	3.35	10	7.25	10
connections	180	61.3	180	130.5	90.3

5.2 Gobang

As we have described, Gobang is a more complex domain than TTT, more detail analysis was therefore drawn from Gobang.

The experimental results of Gobang (table 4 and figure 3-5) show that ESP defended the champion of the fastest system for all three tasks. Interestingly, SANE found a small quantity of networks that were able to occasionally beat the BEST hand-coded strategy of Gobang, although SANE encountered more losses than ESP. The average learning speed of half-evaluated EEC was faster than that of SANE for the BAD and MIDDLE strategy of Gobang, while the best solutions and learning speed of half-evaluated EEC were worse than those of SANE for the hardest task. The behavior of NEAT for the BAD strategy of Gobang looked similar to that in figure 2. The fitness of NEAT slowly went up during generations and eventually exceeded standard NE after 150 generations on average (figure 3). For the other two tasks, however, the performance of NEAT was even worse than standard NE. The evolutionary process of NEAT nearly stagnated after a few generations. Almost no hidden neurons were added during 200 generations when NEAT evolved networks to play against the BEST strategy of Gobang. As we have presented, our experiments of NEAT carried out a blended mutation to implement both complexification and simplification dynamics search. We also implemented complexification dynamics search through turning off simplification in NEAT for the BEST strategy of Gobang. However, the improvement was slight as shown in figure 5.

Remarkablely, all three neurocoevolutionary algorithms outperformed the two neuroevolutionary algorithms for the three tasks, especially, for the hardest task, the BEST strategy of Gobang. It seems that the two neuroevolutionary algorithms inevitably encountered premature convergence for complex domain, although NEAT employed innovative mutation operators to maintain diversity. The learning speed of the two neuroevolutionary algorithms was far lagging behind those of the neurocoevolutionary algorithms. The three neurocoevolutionary

algorithms, moreover, were able to keep the growth of learning curve during the whole generations for the hardest problem.

Table 4. Average Results of the network players play with three hand-coded players of Gobang (20 runs each)

		NE	NEAT	SANE	ESP	EEC
BA	W	96.3	97.95	100	100	100
	T	0.3	0.1	0	0	0
	L	3.4	1.95	0	0	0
	F	482.1	489.95	500	500	500
MI	W	77.6	61.4	97.4	99.5	96.1
	T	0.05	0.1	0.05	0	0
	L	22.35	38.5	2.55	0.05	3.9
	F	388.3	307.2	487.1	499.75	480.5
BE	W	0	0	1.65	0	0
	T	9.4	2.42	60.5	69.3	56.05
	L	90.6	97.58	37.85	30.7	43.95
	F	65.33	45.51	210.25	218.04	181.73

BA: BAD strategy, MI: MIDDLE strategy, BE: BEST strategy.
W: win times, T: tie times, L: loss times, F: fitness

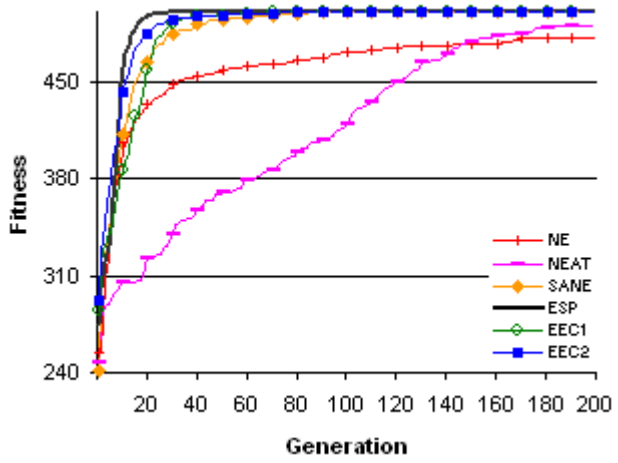

Figure 3. Comparison of average learning speeds for the BAD strategy of Gobang. EEC1 is half-evaluated EEC, EEC2 is full-evaluated EEC (20 runs each).

Evolutionary algorithms themselves are time consuming, thus, the time consumption of different algorithms for evolving networks is an important consideration. The evaluation of time consumption was carried out on a desktop PC with a 3.40 GHz Pentium(R) D CPU, 3 GB of RAM, and the Windows XP operating system. The average time consumption for Gobang from 3 runs each is given in table 5. The execution time of NEAT is very long, more than 10 times longer than the others. Although the neurocoevolutionary algorithms evolve multi-populations simultaneously, the time consumption is merely a little longer than standard NE.

The performance of original EEC that implements full-evaluation of individuals has also been exhibited in figure 2-5 (line EEC2). We can see that the performance of full-evaluated EEC was even better than SANE for all tasks of Gobang. Because EEC evolves two populations, the time consumption of full-evaluated EEC is at most twice as high as standard NE.

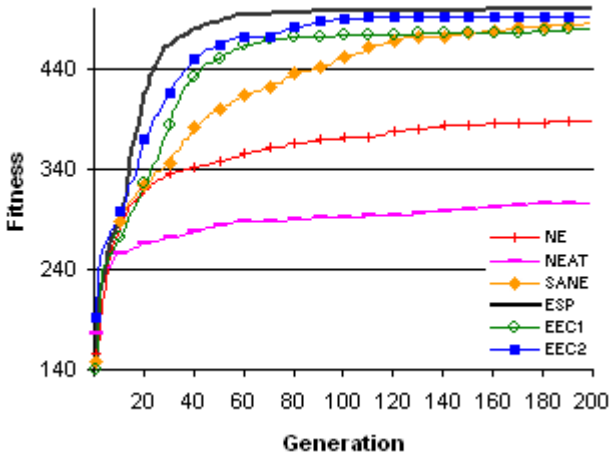

Figure 4. Comparison of average learning speeds for the MIDDLE strategy of Gobang. EEC1 is half-evaluated EEC, EEC2 is full-evaluated EEC (20 runs each).

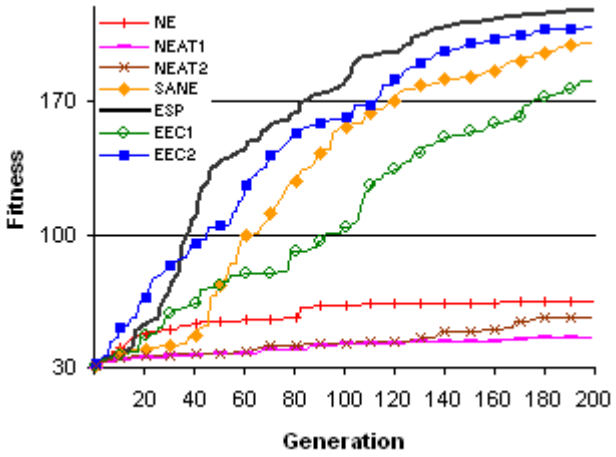

Figure 5. Comparison of average learning speeds for the BEST strategy of Gobang. EEC1 is half-evaluated EEC, EEC2 is full-evaluated EEC. NEAT1 is blended dynamics search of NEAT and NEAT2 is complexification dynamics search of NEAT (20 runs each).

Table 5. The comparison of average time consumption for Gobang from 3 runs each

	NE	NEAT	SANE	ESP	EEC
Avg. hours per run	0.59	9.58	0.63	0.78	0.70

6. DISCUSSION AND FUTURE WORK

Our study shows that neurocoevolutionary algorithms are highly robust compared to neuroevolutionary algorithms. In our experiments all the three neurocoevolutionary algorithms were able to keep their vigor for solving problems from simple domains such as TTT to similar but more complex domains such as Gobang.

NEAT has been demonstrated to be efficient in many different domains, such as Go, Pole Balancing, Robot Duel and so forth [7, 8, 9]. In our study, however, NEAT encountered difficulties when it evolved game players against sophisticated hand-coded opponents of Gobang. NEAT evolves networks based on the principle of searching from a minimal complexity. New neurons and new connections are added only when beneficial. However, benefits were almost never found when NEAT started from no hidden neuron topology to play with the hardest strategy of Gobang. On average, only 0.14 and 0.25 hidden neurons were respectively added using blended dynamics search and complexification dynamics search during 200 generations from 20 runs. Learning cannot be performed efficiently without enough hidden neurons. This could explain the fatal failure when NEAT is forced into a very complex domain. An incremental evolution could be helpful to turn the tables. Through decomposing a very difficult task into several increasingly difficult sub-tasks, incremental evolution evolves networks to achieve the sub-tasks one by one during the evolutionary process.

Our method, EEC, was developed based on a standard NE model. An additional connection-paths population is evolved simultaneously in order to cooperate with connection weights to build complete networks with efficient connections. The results have demonstrated that evolving connection weights along with connection paths can significantly enhance the performance of standard NE.

A fully-connected network could generate noise. As we have known, a neuron will be activated when its input signal reaches a threshold value, where that signal is the sum of weighted output signals from upstream neighbor neurons. Redundant products that come from inefficient connection could result in incorrect activation of neurons. Standard NE restricts inefficient connections by evolving their connection weights toward 0. However, a holistic search served by standard NE will be inefficient when the search space is large. Neurocoevolutionary algorithms benefit from decomposing the holistic search space into sub-spaces.

In neurocoevolutionary algorithms, there are two basic steps to perform efficient search: 1) decomposing the genotype space of a complete network and the partial solutions of the network are evolved in the sub-genotype spaces; and 2) recombining the sub-genotypes to build complete networks. The purpose of recombination is to find cooperative sub-genotypes that achieve optimal solutions.

Both ESP and SANE decompose the genotype space of networks into the sub-genotype space of neurons. Each sub-genotype only represents a weights vector connected with one hidden neuron. Conversely, EEC decomposes the genotype space of networks into two sub-spaces: one for weights and one for connections. No matter which one is employed, decomposition can simplify the search problem. Thus, neurocoevolutionary algorithms are able to implement search more efficiently by the decomposition and combination processes.

Two main future works are proposed at present. First, EEC has demonstrated that evolving connection weights and evolving connection paths are both important for the search of optimal networks. To further demonstrate the importance of efficient network connections, an analysis of efficient connections will be carried out on other methods, such as NE, ESP and SANE. Second, one limitation of SANE and ESP is that they can only evolve three-layer networks. EEC, however, has more flexible representation to evolve other types of networks, including multi-hidden layer networks. Our EEC method can be improved further to have flexible hidden neurons. The effective hidden neurons should be evolved to adapt to different problem domains.

The empirical study in this paper compares two neuroevolutionary algorithms and three neurocoevolutionary algorithms for evolving two board-games players. Through decomposing the search space of networks, neurocoevolutionary algorithms find stronger game players. Our results suggest that neurocoevolutionary algorithms perform more efficient search than neuroevolutionary algorithms for large and complex search spaces.

7. ACKNOWLEDGMENTS

The author would like to thank Keith Downing and all the members of the SOS group for their support. The author also would like to thank reviewers for their valuable comments.

8. REFERENCES

[1] Gomez, F. and Miikkulainen, R., *Robust Non-Linear Control through Neuroevolution*, Technical Report AI-TR-03-303, The University of Texas at Austin Department of Computer Sciences August 2003.

[2] James, D. and Tucker, P., A Comparative Analysis of Simplification and Complexification in the Evolution of Neural Network Topologies, in *GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 2004.

[3] Moriarty, D. E. and Miikkulainen, R., Forming Neural Networks through Efficient and Adaptive Coevolution, *Evolutionary Computation*, vol. 5, pp. 373-399, 1997.

[4] Perez-Bergquist, A. S., *Applying ESP and Region Specialists to Neuro-Evolution for Go*, Technical Report CSTR01-24 May 2001.

[5] Richards, N., Moriarty, D., McQuesten, P., and Miikkulainen, R., Evolving Neural Networks to Play Go, *Applied Intelligence*, vol. 8, pp. 85-96, 1998.

[6] Rosin, C. D. and Belew, R. K., Methods for Competitive Co-evolution: Finding Opponents Worth Beating in *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Francisco, CA, 1995.

[7] Stanley, K. O. and Miikkulainen, R., Competitive Coevolution Through Evolutionary Complexification *Journal of Artificial Intelligence Research*, vol. 21, pp. 63-100, 2004.

[8] Stanley, K. O. and Miikkulainen, R., Evolving a Roving Eye for Go, in *Proceedinngs of the Genetic and Evolutionary Computation Conference*, 2004.

[9] Stanley, K. O. and Miikkulainen, R., Evolving Neural Networks through Augmenting Topologies, *Evolutionary Computation*, vol. 10 (2), pp. 99-127, 2002.

[10] Yao, X., Evolving Artificial Neural Networks, *Proceedings of the IEEE*, vol. 87, pp. 1423-1477, 1999.