A Comparative Analysis of Simplification and Complexification in the Evolution of Neural Network Topologies

Derek James and Philip Tucker

djames@gmail.com
ptucker@gmail.com

Abstract. Approaches to evolving the architectures of artificial neural networks have involved incrementally adding topological features (*complexification*), removing features (*simplification*), or both. We will present a comparative study of these dynamics, focusing on the domains of XOR and Tic-Tac-Toe, using NEAT (NeuroEvolution of Augmenting Topologies) as the starting point. Experimental comparisons are done using complexification, simplification, and a blend of both. Analysis of the effects of each approach on the variation, complexity, and fitness of the evolving populations demonstrates that algorithms employing both complexification and simplification dynamics search more efficiently and produce more compact solutions.

1 Introduction

The past decade has seen an increase in the number of algorithms for producing topological and weight evolved artificial neural networks (TWEANNs). This subset of evolutionary artificial neural networks seeks to find optimal architectures along with optimal weight configurations using an evolutionary algorithm.

As summarized in Yao's overview of TWEANNs, algorithms for finding optimal architectures fall into two broad categories, *constructive* and *destructive* [1]. For the purposes of this paper, we refer to these dynamics as *complexification* and *simplification*, defined in the following ways:

A *complexification* algorithm is one that begins with a minimal architecture and includes mutation operators that incrementally add topological features, such as hidden nodes and connections.

A *simplification* algorithm generally begins with an overabundance of topological features and incrementally removes features.

These dynamics are not unique to artificial evolution. Biological populations experience both dynamics, depending on their genetic composition and consequent evolutionary pressures. Examples of complexification are ubiquitous throughout nature, from wings to eyes to intricate behavioral modifications such as mating rituals.

But simplification as an evolutionary dynamic is widespread as well. Parasites that become more and more dependent upon their hosts through coevolution often lose features such as eyes and limbs, or experience dramatic simplification in their circulatory or digestive systems. Cave-dwelling organisms often exhibit the evolutionary loss of pigmentation or eyes [2]. In the relative absence of predators due to geographic isolation, many species of birds revert to flightless forms [3]. As Darwin himself noted in *Origin of Species*, there is a cost associated with the production and upkeep of such morphological features, and individuals expending fewer resources on the production and upkeep of unnecessary features are often more fit than their rivals.

In the context of evolutionary algorithms there are inherent problems with each of these approaches taken alone. As Angeline is quoted in Yao's overview: "Such structural hill climbing methods are susceptible to becoming trapped at structural local optima." In addition, they "only investigate restricted topological subsets rather than the complete class of network architectures".

There are ways to mitigate these problems. One complexifying TWEANN approach in particular, NEAT (NeuroEvolution of Augmenting Topologies), retains variation (and hence simpler structures) in the population through the use of speciation [5]. This approach allows for simpler individuals in the population to act as starting points for reinvestigation of the solution space through complexification.

However, though niching techniques such as speciation avoid the susceptibility of being trapped at local optima, they do not address Angeline's second objection, that the full range of architectures is not being investigated.

A problem arises, in spite of these dynamics, in the choice of starting topology. NEAT defines a minimal architecture as "a uniform population of networks with zero hidden nodes (i.e., all inputs connect directly to outputs)." This concept of starting minimally, though, makes assumptions about ideal topologies for solutions to given domains. In purely complexifying algorithms there is no way to remove unnecessary features. If those features exist in the starting topology they represent an impediment to finding the optimal architecture.

With simplifying algorithms, even if great care is taken to insure variation and efficiency in search, to find an optimal architecture for a given problem the initial topologies must be a superset of that optimal architecture. In the vast majority of cases, the optimal architecture for a given problem's solution is not known (hence the use of TWEANNs in the first place). Also, as Stanley, et al noted in [11], a purely simplifying algorithm must begin the search in a higher dimensionality than the solution, resulting in an inherently inefficient search.

To confront these concerns, some algorithms (GNARL [7], EPNet [8]) have employed both simplification and complexification, enabling a broader search throughout the range of possible architectures.

2 NEAT

For the basis of our experiments, we used NEAT (NeuroEvolution of Augmenting Topologies). NEAT has proven to be a powerful and effective system in a wide array of experimental domains [5, 6, 11]. The NEAT framework, specifically an open-source Java-based implementation of NEAT written by the authors called ANJI [http://anji.sourceforge.net/], was used for all experiments.

A detailed explanation of NEAT can be found in [5], but a summary of the primary features, and important particulars where our implementation differs, are included in this section.

NEAT is a system for evolving both the connection weights and topology of ANNs simultaneously. It does so by means of crossover and three types of mutation:

1) Modify connection weight mutation
2) Add connection mutation
3) Add neuron mutation

The first type of mutation uniformly perturbs the weight of an existing connection. The second type adds connections between unconnected neurons (or self-connections to neurons in the case of recurrent networks). The third replaces an existing connection with a neuron and a single incoming and outgoing connection.

To implement simplification, a fourth mutation was added to the NEAT framework:

4) Delete connection mutation

The delete connection mutation rate determines the percentage of existing connections to be removed. Connection genes are sorted, and then deleted, in ascending order. Neurons and associated substructures stranded due to this mutation are removed from the topology. This methodology was chosen under the assumption that connections with weights closer to zero are less influential, and thus better candidates for deletion.

Although GNARL and EPNet employ both complexifying and simplifying mutations, they avoid the use of crossover because of the problem of competing conventions. NEAT is able to mitigate the competing conventions problem through the use of historical markings. Historical markings are unique identifiers for each topological structure, assigned upon creation or mutation, easily and quickly allowing the comparison of two individuals' genetic history, facilitating an efficient method for crossover.

NEAT also uses speciation, a niching mechanism which reproductively isolates sections of the population based on a measure of their topological differences. Because species share fitness, speciation acts to protect innovation, but also serves an important role in maintaining variation in the evolving population.

When a new neuron is added through mutation, the connection it replaces is not removed, but disabled. The disabled connection has a given probability of being re-enabled in later generations. However, ANJI simply removes the connection permanently. NEAT proposes normalizing each term of the speciation compatibility calculation by dividing each term by the number of genes in the larger of the genomes being compared, although in practice this normalization is not used in NEAT research to date. ANJI does employ this normalization. NEAT also clamps the number of species allowed to coexist in a given population (e.g. 8-10), while in ANJI the number of species is not clamped. A simple form of elitism was used in all experiments. The fittest individual from each species was copied unchanged into the next generation. In NEAT, the constraint on elites is that its species must contain at least 5 individuals in the next generation to be copied unchanged. NEAT can be used to evolve recurrent networks, but all experiments for this paper used only feed-forward networks.

3 Experimental Setup

3.1 XOR

XOR is a simple verification domain able to demonstrate the basic functionality of a neuroevolutionary approach. The minimal number of hidden nodes required to solve XOR is one. Since NEAT begins with a bias node in the input layer, fully-connected to the output node, the initial topology, as seen in Figure 1a, has 7 genes, two input nodes, a bias node, an output node, and three forward connections.

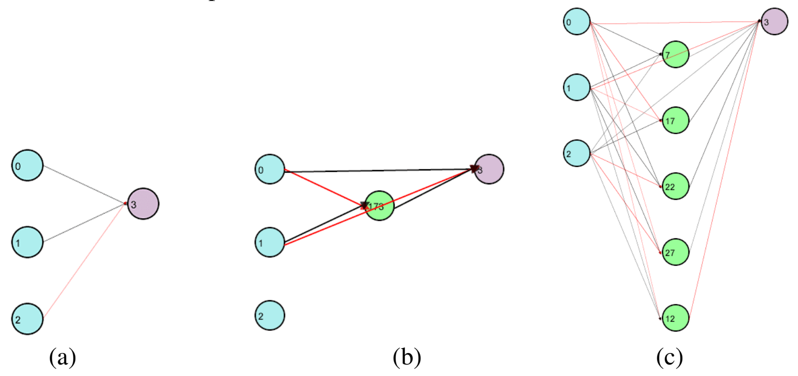

Fig. 1. (a) Initial topology for standard NEAT. (b) Minimal solution for XOR (Note that the bias node is not necessary for a solution.) (c) Initial topology for simplifying XOR

Starting with this topology, the minimal number of genes required to solve XOR is 10 (the bias is not required to solve XOR). The optimal topology is shown in Figure 1b.

Three sets of 100 runs each were performed. The first used only complexifying mutations, the second only simplifying mutations, and the third set used both. Specific parameter settings are listed in Section 3.3.

The initial topologies for both the complexifying and blended runs followed the original NEAT methodology. The input and output layers were fully connected with forward connections, and there were no hidden nodes. Initial connections weights were randomly, uniformly distributed between –1.0 and 1.0. The initial topologies for simplifying runs contained an additional 5 hidden nodes, with all layers fully connected via forward connections (Figure 1c).

Since the activation function was sigmoid, all outputs fell between 0.0 and 1.0. Each input pattern had an associated target range, 0.0-0.2 for false and 0.8-1.0 for true. An output in the target range had an error of 0.0. Otherwise, its error was the distance to the inner edge of the range (0.2 for false and 0.8 for true). The total error of the network was the sum of errors for the 4 XOR input-target pairs. To calculate fitness, this error was subtracted from the maximum possible total error, and the result was then squared. The success of a network could have been determined by a different analysis of the outputs (i.e., each output is correct if it is on the same side of the midpoint as the target), but to be consistent with more complex domains it was decided to have the error function and success criteria be the same. A network was considered to have solved XOR when it had reached 90% of maximum fitness.

3.2 Tic-TacToe

The experimental domain of Tic-Tac-Toe was chosen for the second set of experiments due to the relative ease of identifying a range of behavioral objectives. Tic-Tac-Toe, also known as “Naughts and Crosses,” is a simple, well-known two-player game in which players take turns placing tokens on a 3-by-3 grid. The first player to place three tokens in a row wins.

For even such a simple game, there exist identifiable strategies, such as forking, or playing in such a way to set up two possible paths to a win:

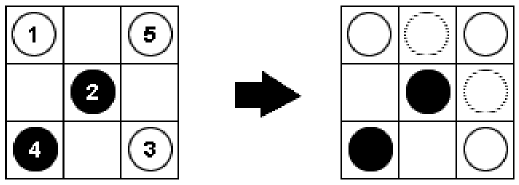

Fig. 2. The sequence of moves on the left results in a fork, or two possible winning states, for white, ensuring a win

EANNs have been applied to the domain of Tic-Tac-Toe, with modest results [8]. A generalized, robust player should be capable of avoiding loss to an optimal player, while exploiting the weakness of a suboptimal player. Coevolutionary approaches have demonstrated the ability to evolve fairly robust strategies [10, 11]. However, for the purposes of this study, we have chosen to explicitly represent strategic objectives in the form of hand-coded strategies and evolve the TWEANNs directly against them. In more complex domains this approach is likely not feasible, but our goal here is to demonstrate the efficacy of simplification and complexification dynamics, and evolving and evaluating against explicitly-represented strategies serves that purpose well.

Table 1 describes the behavior of the hand-coded strategies used for evolution and evaluation.

Table 1. Description of the behavior of hand-coded Tic-Tac-Toe strategies

Name	Behavior
BEST	1) If the board is empty, move randomly 2) If winning move is available, make it 3) If opponent has winning move, block it 4) Try to fork opponent 5) Try to block forking by opponent 6) Play in center if open 7) Play randomly in open corner 8) Play randomly
FORKABLE	Same as BEST, excluding rule 5
CENTER	Plays in center if open, otherwise plays randomly
RANDOM	Plays randomly
BAD	Plays randomly in first open side space. If no sides are open, plays randomly in first open corner. If no corners are open, plays in center.

This approach is similar to that taken by Angeline, et al [10], who coevolved modular LISP programs and tested them against three hand-coded strategies, analogous to our BEST, FORKABLE, and RANDOM. To these we have added CENTER (one of the first lessons a human child learns is the strategic importance of the center) and BAD (this purposefully poor player enables its opponents very quickly to learn to complete a sequence of winning moves).

For comparison purposes, Table 2 shows head-to-head results for the hand-coded players, with each player taking turns going first.

Table 2. Results for head-to-head match-ups between hand-coded strategies

		Vs. Best	Vs. Forkable	Vs. Center	Vs. Random	Vs. Bad	**Avg**
Best	W	0.0%	25.4%	79.9%	88.7%	100%	**58.8%**
	L	0.0%	0.0%	0.0%	0.0%	0.0%	**0.0%**
	T	100%	74.6%	20.1%	11.3%	0.0%	**41.2%**
Forkable	W	0.0%	26.3%	81.5%	90.0%	100%	**59.6%**
	L	25.4%	26.3%	0.0%	0.5%	0.0%	**10.4%**
	T	74.6%	47.4%	18.5%	9.5%	0.0%	**30.0%**
Center	W	0.0%	0.0%	44.0%	53.8%	86.1%	**36.8%**
	L	79.9%	81.5%	44.0%	33.4%	9.1%	**49.6%**
	T	20.1%	18.5%	12.0%	12.8%	4.8%	**13.6%**
Random	W	0.0%	0.5%	33.4%	44.0%	74.0%	**30.4%**
	L	88.7%	90.0%	53.8%	44.0%	18.7%	**59.0%**
	T	11.3%	9.5%	12.8%	12.0%	7.3%	**10.6%**
Bad	W	0.0%	0.0%	9.1%	18.7%	39.0%	**13.4%**
	L	100%	100%	86.1%	74.0%	39.0%	**79.8%**
	T	0.0%	0.0%	4.8%	7.3%	22.0%	**6.8%**

Representation for the TWEANNs was as follows: Nine input nodes each received input corresponding to the nine spaces on the board. An empty space was input as 0, friendly tokens as 1, and enemy tokens as –1. A tenth input node was clamped with an input of 1, for use as a bias. Output from the nine output nodes corresponded to potential moves. For each move the network was fully activated with the board state as input. The output node with the highest value corresponding to a legal move was taken as the move of the network player. Illegal moves suggested by the network were ignored. To attempt to evolve robust players, opponents always played an even number of games, with each player taking turns moving first. Players were awarded 5 points for a win, 2 for a tie, and 0 for a loss.

3.3 Experimental Parameters

Table 3 lists parameters for each of the three XOR experiments. The first section lists parameters common to all runs. The only parameters differing between runs were mutation rates and initial topologies. Complexifying and Blended runs began with the standard NEAT starting topology, an input layer and output layer fully connected with feed-forward connections and no hidden nodes (for a total of 7 genes). The Simplifying run began with an input layer, a hidden layer with five nodes, and an output layer, all fully connected to one another with feed-forward connections (for a total of 32 genes).

Table 3. Parameter settings for XOR experiments

Type of Run	Parameters	Value
All	Population size	150
	Number of generations	150
	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	0.2
Complexifying	Add connection mutation rate	0.03
	Add neuron mutation rate	0.01
	Delete connection mutation rate	0.0
Simplifying	Add connection mutation rate	0.0
	Add neuron mutation rate	0.0
	Delete connection mutation rate	0.04
Blended	Add connection mutation rate	0.03
	Add neuron mutation rate	0.01
	Delete connection mutation rate	0.02

The activation function for all nodes in the XOR experiments was a modified sigmoid, while the activation function for all nodes in the Tic-Tac-Toe experiments was a standard hyperbolic tangent.

Table 4 lists parameters for each of the three Tic-Tac-Toe experiments. As above, the first section lists parameters common to all runs. The only parameters differing between runs were mutation rates and initial topologies. Complexifying and Blended runs began with the standard NEAT starting topology, an input layer (9 input nodes plus bias node) and output layer (9 nodes) fully connected with feed-forward connections and no hidden nodes (for a total of 109 genes). The Simplifying run began with an input layer, a hidden layer with 15 nodes, and an output layer, all fully connected to one another with feed-forward connections (for a total of 409 genes). This topology is a superset of the best performing ANNs from the Complexifying and Blended experiments, which were carried out first. It also provides more raw structure to search than the experiments in [9], which used a single hidden layer in which the number of nodes mutated between 1 and 10.

Parameters were chosen after a modest amount of ad hoc exploration. Complexifying mutation rates were relatively small for the larger genotypes in Tic-Tac-Toe due to their exponential effect. Simplifying mutation rates were found to perform best at rates slightly higher than complexifying ones, while mutation parameters for blended runs functioned best when complexifying and simplifying rates were reasonably balanced, producing a more even search through both more and less complex architectures.

Table 4. Parameter settings for TTT experiments

Type of Run	Parameters	Value
All	Population size	200
	Number of generations	200
	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
Complexifying	Add connection mutation rate	0.01
	Add neuron mutation rate	0.005
	Delete connection mutation rate	0.0
Simplifying	Add connection mutation rate	0.0
	Add neuron mutation rate	0.0
	Delete connection mutation rate	0.03
Blended	Add connection mutation rate	0.01
	Add neuron mutation rate	0.005
	Delete connection mutation rate	0.02

4 Results

4.1 XOR

Table 5 indicates results for the three sets of XOR experiments.

Table 5. Averaged results for XOR runs (100 runs each)

Type	Avg. # of Genes of Best Solution	Avg. # of connections	Avg. # of hidden nodes	Avg. # of generation to find solution
COMPLEXIFYING	18.84	11.72	3.12	44.08
SIMPLIFYING	10.47	5.36	1.09	14.45
BLENDED	11.55	6.10	1.45	49.32

XOR experiments in [5] resulted in an average of 7.48 connections and 2.35 hidden nodes, while finding a solution within an average of 32 generations. Differences between the results in [5] and the ones here may be due in part to differences in the implementation of the NEAT algorithm, or more likely differences in XOR methodology, specifically in calculating error and in determining whether or not a given ANN has found a solution.

In this domain, simplifying dynamics alone produce the best results, finding more compact solutions more quickly than either of the other two methods. This suggests that when the optimal architecture is known and very compact, simplifying dynamics alone may be most desirable.

However, XOR is an extremely simple, somewhat artificial domain, in that it has no local optima. More general conclusions may be drawn from a more complex domain.

4.2 Tic-Tac-Toe

To evaluate the performance of evolved networks for the Tic-Tac-Toe domain, the best performers from the final generation of each run played 10,000 games against each type of hand-coded player, alternating playing first. Percentages of wins/losses/ties were then averaged for each of the 10 champions from each type of run. Results are listed in Table 6.

Table 6. Averaged results for Tic-Tac-Toe runs (10 runs each)

		Vs. Best	Vs. Forkable	Vs. Center	Vs. Random	Vs. Bad	**Avg**
Complexifying	W	0.00%	2.27%	79.23%	75.88%	99.53%	**51.38%**
	L	24.45%	24.60%	10.10%	16.44%	0.47%	**15.21%**
	T	75.55%	73.13%	10.67%	7.68%	0.00%	**33.41%**
Simplifying	W	0.00%	0.56%	76.77%	72.93%	98.75%	**49.80%**
	L	25.89%	25.14%	11.59%	16.77%	0.86%	**16.05%**
	T	74.11%	74.64%	11.64%	10.30%	0.40%	**34.22%**
Blended	W	0.00%	4.50%	75.59%	74.61%	100%	**50.94%**
	L	21.89%	22.02%	10.57%	15.36%	0.00%	**13.97%**
	T	77.99%	73.48%	13.84%	10.03%	0.00%	**35.07%**

Champions from the simplifying runs on average play weaker games than champions from either the complexifying or blended runs. Their win rate is the lowest and their loss rate is the highest. Between the complexifying and blended runs, the complexifying champions have a slightly higher win rate, but their loss rate is higher, and their tie rate is lower. Table 7 shows the complexity of these champions.

Table 7. Average complexity of champions of TTT runs (10 runs each)

Type	Avg. # of genes	Avg. # of connections	Avg. # of hidden nodes
COMPLEXIFYING	216.6	179.3	18.3
SIMPLIFYING	109.9	80.2	10.7
BLENDED	96.9	70.3	7.6

The average complexity of champions from the complexifying runs is over twice that of the blended runs. Also, it is important to note that, as in the XOR domain, champions from the blended Tic-Tac-Toe runs on average exhibit topologies less complex than the initial topologies (109 genes).

Figures 3, 4, and 5 show typical complexity dynamics for Tic-Tac-Toe complexifying, simplifying, and blended runs, respectively. The x-axis indicates the generation, the y-axis the number of genes. Each vertical line connects the maximum and minimum complexity for that generation. Each black circle indicates the average complexity of the population for that generation, and the triangles represent the complexity of the fittest individual for each generation.

In the complexifying run, though the complexity of the champion for a given generation varies, there is a steady increase in both the maximum and average complexity of the evolving population. The minimum complexity of the population also increases. This rate of increase is not as large as the increase in the maximum complexity, but it does demonstrate a gradual loss of simpler individuals as the run proceeds.

The dynamics of the simplifying run are just the opposite. Maximum, minimum, and average complexity of the population all *decrease* as the run proceeds. As with the complexifying run, the search is a unidirectional hill-climbing algorithm.

However, in the blended run, maximum complexity increases while minimum complexity decreases. The search is bi-directional. Meanwhile, the average complexity remains fairly constant.

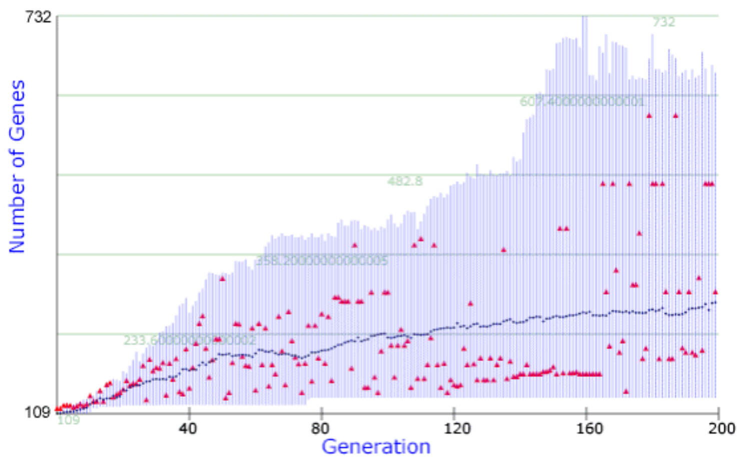

Fig. 3. Complexity of evolving population from typical Tic-Tac-Toe Complexification (Standard NEAT) run

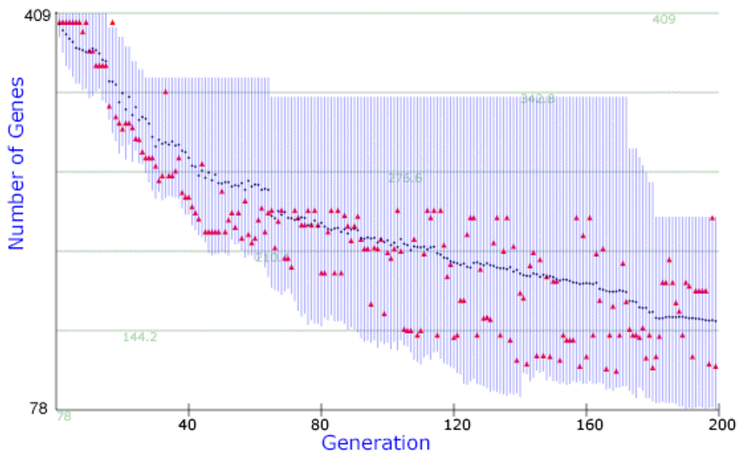

Fig. 4. Complexity of evolving population from typical Tic-Tac-Toe Simplifying run

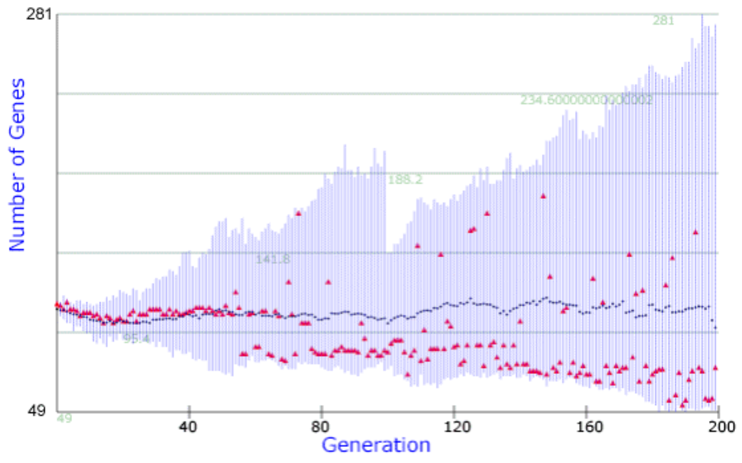

Fig. 5. Complexity of evolving population from typical Tic-Tac-Toe Blended run

Another important consideration is the time spent searching. All experiments were carried out on a desktop PC with a 1.00 GHz AMD Athlon processor, 504 MB of RAM, running the Windows XP operating system. Table 7 lists average times for the 10 Tic-Tac-Toe runs of each type.

Table 7. Averaged computation time for Tic-Tac-Toe runs (10 runs each)

Type	Avg. minutes per run (rounded to nearest minute)
COMPLEXIFYING	216
SIMPLIFYING	210
BLENDED	170

Because there is a random element in the play of all hand-coded players, there is some variance in the number of moves in each game. However, because of the large number of games played (500 against each ANN), the variance in the number of moves due to randomness should be consistent across runs.

In complexifying runs, because the average complexity of the population increases, the search will only become more computationally intensive as the run proceeds. Simplifying runs begin with large networks, so they are very computationally intensive at the outset. Because the blended runs started with fairly minimal topologies and searched bi-directionally, they tended to be more efficient in their search.

5 Discussion and Future Work

In attempting to evolve optimal or near-optimal neural network topologies to solve problems, the two important decisions to be made are (1) what set of topologies should comprise the initial population, and (2) what dynamics should act upon those topologies as the algorithm searches for a solution.

The decision of starting topologies is still a difficult one. NEAT's approach of trying to start minimally is appealing, though systems that begin with random initial topologies, such as GNARL, can still perform reasonably well.

But the decision to include both complexifying and simplifying topological mutations in the algorithm mitigates the impact of choosing sub-optimal initial topologies. A blended approach allows for a broader, more flexible search of potential topologies, so that the starting point for such a search is not quite as crucial.

This bi-directional search favors solutions that are the necessary complexity for a given task.

Thus, a very good approach to evolving a neural network to solve a given problem seems to be to start with relatively minimal architectures while including both complexifying and simplifying mutations, their rates balanced in such a way to produce an even search in both directions through the range of possible architectures. This approach should yield both more efficient searches and more efficient solutions than either complexifying or simplifying algorithms alone. However, work remains to be done to verify these conclusions across a wide range of domains of varying difficulty.

In some cases, it may be preferable to favor one dynamic over the other. Depending on the actual mutation rates, a blended approach may still be biased toward a particular trend. These experiments demonstrate that a balanced approach works well, and proceeds under the assumption that the optimal architecture may be either more complex or simpler than one might assume. However, if it is clear that the target behavior is extremely complex, a higher rate of complexification, along with a lower rate of simplification, may be preferred. This may provide an interesting avenue for further research.

Comparisons were not performed with fixed topologies, or with neuroevolution in which other factors (such as the activation function) are mutated.

Investigation into the relative impact of complexifying and simplifying dynamics on indirect encoding methods is another important direction for future research in this area.

6 Conclusion

The experiments in this paper have demonstrated that while complexifying and simplifying dynamics are both important in the search for optimal neural network topologies, these dynamics work better together than either dynamic working alone.

Algorithms using only complexification tend to produce overly complex solutions, and become much more computationally intensive as the search progresses. They are also unable to eliminate unnecessary topological features that either evolve during the run or were present in the initial topologies. Algorithms using only simplification are more computationally intensive to begin with, and are susceptible to removing topological features early on that could prove beneficial later.

Algorithms using both dynamics simultaneously tend to be more computationally efficient, produce more efficient solutions, and do not suffer from the problem of "overstepping" topological features that either dynamic used exclusively poses.

The results suggest that a blended approach leads to both a more robust and more efficient search for optimal topologies.

References

1. Yao, X., ``Evolving artificial neural networks," *Proceedings of the IEEE*, 87(9):1423-1447, September 1999.
2. Culver, D.C., Kane, T.C., Fong, D.W.: *Adaptation and Natural Selection in Caves*. Harvard University Press, Cambridge London (1995) 4–30.
3. Chatterjee, S. *The Rise of the Birds*. The Johns Hopkins University Press, Baltimore London (1997) 189-230.
4. Darwin, C.: On the Origin of Species. 6th edn. John Murray, London (1872)
5. Stanley K.O., and Miikkulainen, R., "Evolving Neural Networks Through Augmenting Topologies," *Evolutionary Computation, 10* (2), 99-127.
6. Stanley K.O., and Miikkulainen, R., "Competitive Coevolution Through Evolutionary Complexification," *Journal of Artificial Intelligence Research, 21*, 63-100.
7. Angeline, P.J., Saunders, G.M., and Pollack, J.B., "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," *IEEE Transactions on Neural Networks*, 5(1):54-65, January 1994.
8. Yao, X., "A New Evolutionary System for Evolving Artificial Neural Networks," *IEEE Transactions on Neural Networks*, 8(3):694-713, May 1997.
9. Chellapilla, K. and Fogel, D.B., "Evolution, neural networks, games, and intelligence," *Proceedings of the IEEE*, 87(9), 1471-1496, September 1999.
10. Angeline, P. J. and Pollack, J. B., "Competitive Environments Evolve Better Solutions for Complex Tasks," *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed San Mateo, California: Morgan Kaufmann, 264-270.
11. Stanley K.O., and Miikkulainen, R., "Competitive Coevolution through Evolutionary Complexification," *Journal of Artificial Intelligence Research, 21* (2004), 63-100.