Evolving Neural Networks through Augmenting Topologies

Kenneth O. Stanley and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 USA
{kstanley, risto}@cs.utexas.edu

Technical Report TR-AI-01-290

June 28, 2001

Abstract

An important question in neuroevolution is how to gain an advantage from evolving neural network topologies along with weights. We present a method, NeuroEvolution of Augmenting Topologies (NEAT) that outperforms the best fixed-topology method on a challenging benchmark reinforcement learning task. We claim that the increased efficiency is due to (1) employing a principled method of crossover of different topologies, (2) protecting structural innovation using speciation, and (3) incrementally growing from minimal structure. We test this claim through a series of ablation studies that demonstrate that each component is necessary to the system as a whole and to each other. What results is significantly faster learning. NEAT is also an important contribution to GAs because it shows how it is possible for evolution to both optimize *and complexify* solutions simultaneously, offering the possibility of evolving increasingly complex solutions over generations, and strengthening the analogy with biological evolution.

1 Introduction

Neuroevolution (NE), the artificial evolution of neural networks using genetic algorithms, has shown great promise in complex reinforcement learning tasks (Gomez and Miikkulainen 1999; Gruau et al. 1996; Moriarty and Miikkulainen 1997; Potter et al. 1995; Whitley et al. 1993). Neuroevolution searches through the space of behaviors for a network that performs well at a given task. This approach to solving complex control problems represents an alternative to statistical techniques that attempt to estimate the utility of particular actions in particular states of the world (Kaelbling *et al.* 1996). NE is a promising approach to solving reinforcement learning problems for several reasons. Past studies have shown NE to be faster and more efficient than reinforcement learning methods such as Adaptive Heuristic Critic and Q-Learning on single pole balancing and robot arm control (Moriarty 1997; Moriarty and Miikkulainen 1996). Because NE searches for a behavior, it is effective in problems with large state spaces. In addition, memory is easily represented through recurrent connections in neural networks, making the method a natural choice for learning non-Markovian tasks (Gomez and Miikkulainen 1999).

In traditional NE approaches, a topology is chosen for the evolving networks before the experiment begins. Usually, the network topology is a single hidden layer of neurons, with each hidden neuron connected

to every network input and every network output. Evolution searches the space of connection weights of this fully-connected topology by allowing high-performing networks to reproduce. The weight space is explored through the crossover of network weight vectors and through the mutation of single networks' weights. Thus, the goal of fixed-topology NE is to optimize the connection weights that determine the functionality of a network.

However, connection weights are not the only aspect of neural networks that contribute to their behavior. The topology, or *structure*, of neural networks also affects their functionality. There has been a great deal of interest in the evolution of both topologies and connection weights over the last decade (Angeline et al. 1993; Branke 1995; Gruau et al. 1996; Yao 1999). The basic question, however, remains: Can evolving topologies along with weights provide an advantage over evolving weights on a fixed-topology? A fully connected network can in principle approximate any continuous function (Cybenko 1989). So why waste valuable effort permuting over different topologies?

The answers provided thus far are inconclusive. Some have argued that network complexity can affect the speed and accuracy of learning (Zhang and Muhlenbein 1993). Although this assertion is true for backpropagation, backpropagation does not matter when weights are being optimized by evolution and *not* backpropagation. On sparse reinforcement problems, backpropagation does not even apply, since target outputs are not known.

A persuasive argument for the evolution of both topology and weights was put forward by Gruau *et al.* (1996), who claimed that evolving structure saves the time wasted by humans trying to decide on the topology of networks for a particular NE problem. Although almost all fixed-topology NE systems use a fully connected hidden layer, deciding how many hidden nodes are needed is a trial-and-error process. Gruau *et al.* supported their argument by evolving the topology and weights of an artificial neural network that solved the hardest pole-balancing benchmark problem to date. However, later results suggested that structure was not necessary to solve the difficult problem. A fixed-topology method called Enforced Subpopulations was able to solve the same problem 5 times faster simply by restarting with a random number of hidden neurons whenever it became stuck (Gomez and Miikkulainen 1999).

This article aims to demonstrate the opposite conclusion: if done right, evolving structure along with connection weights can significantly enhance the performance of NE. We present a novel NE method called NeuroEvolution of Augmenting Topologies (NEAT) that is designed to take advantage of structure as a way of minimizing the dimensionality of the search space of connection weights. If structure is evolved such that topologies are minimized and grown incrementally, significant gains in learning speed result. Improved efficiency results from topologies being minimized *throughout* evolution, rather than only at the very end.

Evolving structure incrementally presents several technical challenges: (1) Is there a genetic representation that allows disparate topologies to cross over in a meaningful way? (2) How can topological innovation that needs a few generations to be optimized be protected so that it does not disappear from the population prematurely? (3) How can topologies be minimized *throughout evolution* without the need for a specially contrived fitness function that measures complexity?

The NEAT method consists of solutions to each of these problems as will be described below. The method is validated on pole balancing tasks, where NEAT performs 25 times faster than Cellular Encoding and 5 times faster than ESP. The results show that structure is a powerful resource in NE when appropriately utilized. NEAT is unique because structures become increasingly more complex as they become more optimal, strengthening the analogy between GAs and natural evolution.

2 Background

Many systems have been developed over the last decade that evolve both neural network topologies and weights (Angeline et al. 1993; Braun and Weisbrod 1993; Dasgupta and McGregor 1992; Fullmer and Miikkulainen 1992; Gruau et al. 1996; Krishnan and Ciesielski 1994; Lee and Kim 1996; Mandischer 1993; Maniezzo 1994; Opitz and Shavlik 1997; Pujol and Poli 1998; Yao and Liu 1996; Zhang and Muhlenbein 1993). These methods encompass a range of ideas about how Topology and Weight Evolving Artificial Neural Networks (TWEANNs) should be implemented. In this section, we address some of the ideas and assumptions about the design of TWEANNs, and offer solutions to some unsolved problems. Our goal is to find how a neuroevolution method can use the evolution of topology to increase its efficiency.

2.1 TWEANN Encoding

The question of how to encode networks using an efficient genetic representation must be addressed by all TWEANNs. We will discuss several prototypical representational schemes.

TWEANNs can be divided between those that use a direct encoding, and those that use an indirect one. Direct encoding schemes, employed by most TWEANNs, specify in the genome every connection and node that will appear in the phenotype (Angeline et al. 1993; Braun and Weisbrod 1993; Dasgupta and McGregor 1992; Fullmer and Miikkulainen 1992; Krishnan and Ciesielski 1994; Lee and Kim 1996; Maniezzo 1994; Opitz and Shavlik 1997; Pujol and Poli 1998; Yao and Liu 1996; Zhang and Muhlenbein 1993). In contrast, indirect encodings usually only specify rules for constructing a phenotype (Gruau 1993; Mandischer 1993). These rules can be layer specifications or growth rules through cell division. Indirect encoding allows a more compact representation than direct encoding, because every connection and node are not specified in the genome, although they can be derived from it.

2.1.1 Binary Encoding

Direct encodings usually require simpler implementations than indirect encodings. The simplest implementation is based on the traditional bit string representation used by GAs. For example, Dasgupta and McGregor (1992) use such an encoding in their method, called sGA (Structured Genetic Algorithm) where a bit string represents the connection matrix of a network. sGA is notable for its simplicity, allowing it to operate almost like a standard GA. However, there are several limitations as well. First, the size of the connectivity matrix is the square of the number of nodes. Thus, the representation blows up for a large number of nodes. Second, because the size of the bit string must be the same for all organisms, the maximum number of nodes (and hence connections as well) must be chosen by a human running the system, and if the maximum is not sufficient, the experiment must be repeated. Third, using a linear string of bits to represent a graph structure makes it difficult to ensure that crossover will yield useful combinations.

2.1.2 Graph Encoding

Because bit strings are not the most natural representation for networks, most TWEANNs use encodings that represent graph structures more explicitly. Pujol and Poli (1997) use a dual representation scheme to allow different kinds of crossover in their Parallel Distributed Genetic Programming (PDGP) system. The first representation is a graph structure. The second is a linear genome of node definitions specifying incoming and outgoing connections. The idea is that different representations are appropriate for different kinds of

operators. Subgraph-swapping crossovers and topological mutations use the grid, while point crossovers and connection parameter mutations use the linear representation.

As in sGA, PDGP has a finite limit on the number of nodes in the network, corresponding to the number of nodes in the two-dimensional grid that represents the graph version of the genome. PDGP uses graph encoding so that subgraphs can be swapped in crossover. Subgraph swapping is representative of a prevailing philosophy in TWEANNs that subgraphs are functional units and therefore swapping them makes sense because it preserves the structure of functional components. However, we cannot be sure whether the particular subgraphs being combined in PDGP are the right ones to create a functional offspring.

2.1.3 Non-mating

Because crossover of networks with different topologies can frequently lead to a loss of functionality, some researchers have given up on crossover altogether in what is called Evolutionary Programming (Yao and Liu 1996). Angeline *et al.* (1993) implemented a system called GNARL (GeNeralized Acquisition of Recurrent Links), commenting that "the prospect of evolving connectionist networks with crossover appears limited in general." Although GNARL uses a graph encoding, it is fundamentally different from PDGP in that it sidesteps the issue of crossover entirely. GNARL demonstrates that a TWEANN does not need crossover to work, leaving the problem of demonstrating the advantages of crossover to other methods.

2.1.4 Indirect Encoding

Gruau's Cellular Encoding method (CE; Gruau 1993) is an example of a system that utilizes indirect encoding of network structures. In CE, genomes are programs written in a specialized graph transformation language. The transformations are motivated by nature in that they specify *cell divisions*. Different kinds of connectivities can result from a division, so there are several kinds of cell divisions possible. A major advantage of CE is that its genetic representations are compact. Genes in CE can be reused multiple times during the development of a network, each time requesting a cell division at a different location. CE shows that cell divisions can encode the development of networks from a single cell, much as organisms in nature begin as a single cell that differentiates as it splits into more cells.

Although CE demonstrates that it is possible to evolve developmental systems, we chose direct encoding for NEAT because, as Braun and Weisbrod (1993) argue, indirect encoding requires "more detailed knowledge of genetic and neural mechanisms." In other words, because indirect encodings do not map directly to their phenotypes, they implicitly restrict the search to the class of topologies to which they can be expanded. For the sake of efficiency, we need to be sure that indirect encodings do not restrict phenotype networks to some suboptimal class of topologies. Further, experimental results suggest that CE is not necessarily more efficient than direct encoding methods. (Section 4.3.3).

We now turn to several specific problems with TWEANNs and address each in turn.

2.2 Competing Conventions

One of the main problems for NE is the *Competing Conventions Problem*, also known as the *Permutations Problem* (Radcliffe 1993). Competing conventions means having more than one way to express a solution to a weight optimization problem with a neural network. When genomes representing the same solution do not have the same encoding, crossover is likely to produce damaged offspring.

Figure 1 depicts the problem for a simple 3-hidden-unit network. The three hidden neurons A, B, and C,