NEUROEVOLUTION

Presenter: Vlad Chiriacescu

Contents

Evolutionary Computation

- Evolutionary Computation (EC) is a class of algorithms that can be applied to open-ended learning problems in Artificial Intelligence
- Traditionally, these algorithms evolve fixed-length genomes under the assumption that the space of the genome is sufficient to encode the solution
- A genome containing genes encodes a single point in an - dimensional search space.
- In many cases, a solution is known to exist somewhere in that space.

Neuroevolution

- **Neuroevolution** is a form of machine learning that uses evolutionary algorithms to train artificial neural networks
- Useful in **applications where is difficult to create correct input-output pairs** (where supervised learning algorithms are not suitable). Notable examples: games and robot motor control
- It is classified in the **category of reinforcement learning** techniques

Types of neuro-evolution algorithms

- Some algorithms evolve only the weights of a neural network and others evolve both the weights and the topology of the network (Topology & Weight Evolving Artificial Neural Networks or TWEANNs)
- Some methods evolve the structure in parallel with network parameters such as the network's weights

Types of neuro-evolution algorithms

- When the genotype contains descriptions of neurons and connections inside the network, the neuro-evolution technique uses a **direct encoding scheme**.
- Through evolution, a new network (genotype) is determined. The observable trait (or phenotype) is the new network and thus the genotype is the same with the phenotype.
- When the genotype specifies rules or other structures that describe how to generate a network, the neuro-evolution technique uses an **indirect encoding scheme**.
- Through evolution, a new set of rules/structures (genotype) is determined. The observable trait (or phenotype) is the newly generated network and thus the phenotype is not the same with the genotype.

Issues in Evolutionary Computation

- Many common structures are defined by an indefinite number of parameters – these can contain a variable number of parts that can be represented by any number of parameters above some minimum.
- As an example, two neural networks with different numbers of connections and nodes can represent the same function .
- Therefore, it is not clear what number of genes is appropriate for solving a particular problem.
- In the case of fixed-length genotypes it is necessary to use heuristics to estimate *a priori* the appropriate number of genes to encode such structures.

Issues in Evolutionary Computation

- Unfortunately, for very complex problems, **heuristically determining the appropriate number of genes becomes impossible**
- For example, how many nodes and connections are necessary for a neural network that controls a poker playing robot?
- Because little is known about the solutions, answers to such questions can hardly be based on empirical experience
- One approach is to make the genome extremely large, so that the space it encodes is extremely large and a solution is likely to lie somewhere within -> but the larger the genome, the higher dimensional the space that evolution needs to search

Issues in Evolutionary Computation

- Another problem appears in open-ended problems where **phenotypes are meant to improve indefinitely** and there is no known final solution.
- For example, when a good behavior of a poker playing robot is found through evolution, the researcher doesn't know if there are better players out there -> therefore **continual evolution has to take place**

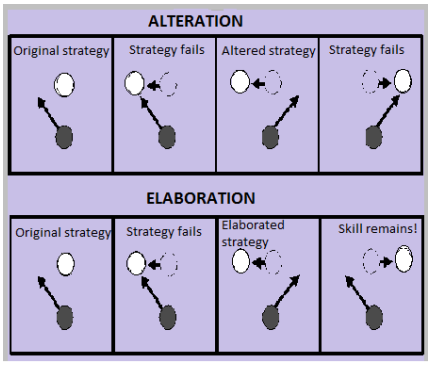
Issues in Evolutionary Computation

- Such continual evolution is difficult with a fixed genome because:

(1) The entire representational space of the genome is used to encode a good solution and improving it means to *alter* the strategy, thereby sacrificing some of the functionality learned over previous generations

(2) Fixing the size of the genome in such domains arbitrarily fixes the maximum complexity of evolved creatures, defeating the purpose of the experiment.

Issues in Evolutionary Computation



Evolving Neural Network Topologies (NEAT method)

- It's a method that evolves both network topology and synaptic weights; uses a direct encoding scheme
- Combines the usual search for appropriate network weights with complexification of the network structure
- Generally, methods that include evolution of topologies have a bound on the complexity of networks that can be evolved; they also begin evolution with random topologies
- The advantage of NEAT is that it **can evolve networks of unbounded complexity** from a minimal starting point

Major NEAT goals

- **Continual coevolution**
  Successful competitive coevolution can use the evolution of topologies to continuously elaborate strategies.
- **Evolution of Adaptive Networks**
  The evolution of topologies allows neuro-evolution to evolve adaptive networks with plastic synapses by designating which connections should be adaptive and in what ways.
- **Combining Expert Networks**
  Separate expert neural networks can be fused through the evolution of connecting neurons between them.

Challenges addressed by NEAT

- NEAT addresses 3 main challenges in evolving neural network topology:

(1) What kind of genetic representation would allow disparate topologies to crossover in a meaningful way?

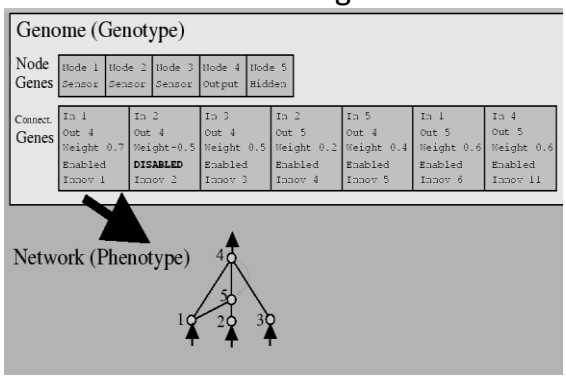
(2) How can topological innovation that needs a few generations to optimize be protected so that it does not disappear from the population prematurely?

(3) How can topologies be minimized *throughout* evolution so the most efficient solutions will be discovered?

Genetic encoding in NEAT

- Evolving structure requires a flexible genetic encoding. In order to allow structures to complexify, their representations must be dynamic and expandable. Each genome in NEAT includes a list of *connection genes*, each of which refers to two *node genes* being connected.
- Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an *innovation number*, which allows finding corresponding genes during crossover.

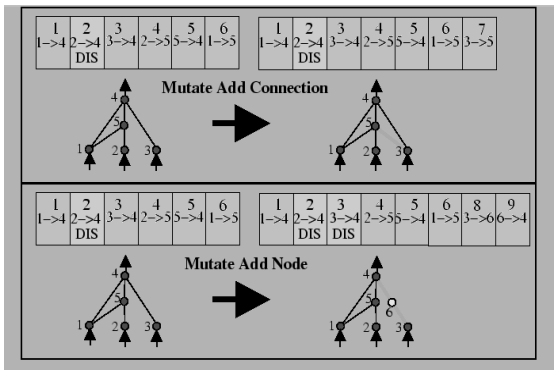
Genetic encoding in NEAT



NEAT Mutation

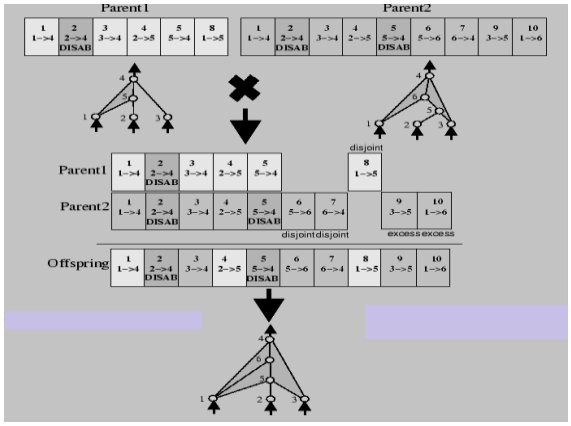
- **Mutation** in NEAT can **change both connection weights and network structures**.
- Structural mutations form the basis of complexification and occur in two ways:
  (1) In the ***add connection*** **mutation**, a single new connection gene is added and it connects two previously unconnected nodes.
  (2) In the ***add node*** **mutation**, an existing connection is split and the new node placed where the old connection used to be. The old connection is disabled and two new connections are added to the genome.
- The connection between the first node in the chain and the new node is given a weight of one, and the **connection between the new node and the last node in the chain is given the same weight as the connection being split**.

Structural mutation in NEAT



Crossover in NEAT

- Uses historical markings to line up genes with the same origin.
- Historical origin of each gene can be used to tell us exactly which genes match up between *any* individuals in the population.
- Two genes with the same historical origin represent the same structure (although possibly with different weights), since they were both derived from the same ancestral gene at some point in the past. Thus, all a system needs to do is to keep track of the historical origin of every gene in the system.
- Whenever a new gene appears (through structural mutation), a *global innovation number* is incremented and assigned to that gene.
- The innovation numbers thus represent a chronology of every gene in the system.



Crossover in NEAT

- Historical markings allow NEAT to perform crossover without the need for expensive topological analysis.
- Thus, the problem of comparing different topologies is essentially avoided and allows NEAT to complexify structure while still maintaining genetic compatibility.
- However, it turns out that a population of varying complexities cannot maintain topological innovations on its own.
- The solution is to protect innovation by speciating the population.

Protecting Innovation through Speciation

- NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population.
- Speciation also prevents bloating of genomes: Species with smaller genomes survive as long as their fitness is competitive, ensuring that small networks are not replaced by larger ones unnecessarily.
- The rationale for this is that new ideas must be given time to reach their potential before they are eliminated.

Protecting Innovation through Speciation

- The distance between two network encodings is computed as a linear combination of the number of excess () and disjoint () genes, as well as the average weight differences of matching genes.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}.$$

- The coefficients c_1, c_2 and c_3 adjust the importance of the three factors, and the factor N represents the number of genes in the larger genome; it normalizes for genome size

Protecting Innovation through Speciation

- Genomes are tested one at a time; **if a genome's distance** to a randomly chosen member of the species is **less than a compatibility threshold, it is placed into this species**. Each genome is placed into the first species from the previous generation where this condition is satisfied, so that no genome is in more than one species.
- **If a genome is not compatible with any existing species, a new species is created**. The problem of choosing the best value for the threshold can be avoided by making it dynamic; that is, given a target number of species, the system can raise if there are too many species, and lower if there are too few.

Protecting Innovation through Speciation

- As a reproduction mechanism, NEAT uses explicit fitness sharing, where organisms in the same species must share the fitness of their niche. Thus, a species cannot afford to become too big even if many of its organisms perform well.
- Therefore, any one species is unlikely to take over the entire population, which is crucial for speciated evolution to maintain topological diversity. The adjusted fitness for the organism is calculated according to its distance from every other organism in the population:

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} \mathrm{sh}(\delta(i,j))}.$$

- The sharing function sh is set to 0 when distance is above the threshold ; otherwise, is set to 1. Thus, the denominator is reduced to the number of organisms in the same species as organism.

Protecting Innovation through Speciation

- Every species is assigned a potentially **different number of offspring in proportion to the sum of adjusted fitnesses of its member organisms**.
- Species **reproduce by first eliminating the lowest performing members** from the population. The entire population is then replaced by the offspring of the remaining organisms in each species.
- **The net effect of speciating the population is that structural innovation is protected.**

Minimizing Dimensionality through Complexification

- NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights.
- **Speciation protects new innovations, allowing topological diversity to be gradually introduced** over evolution. Because of this preservation, NEAT can start minimally, and grow new structure over generations.
- **New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations**. This way, NEAT searches through a minimal number of weight dimensions, and ensures that networks become no more complex than necessary. **This gradual production of increasingly complex structures constitutes complexification.**

Coevolution

- **Coevolution:** In natural ecosystems, organisms of one species compete and/or cooperate with many other different species in their struggle for resources and survival. The fitness of each individual changes over time because it is coupled to that of other individuals inhabiting the environment. As species evolve they specialize and co-adapt their survival strategies to those of other species.
- **Competitive coevolution:** Coevolution system in which the emphasis is on competition between species. The idea consists in establishing an 'arms race' where each species produces stronger and stronger strategies in order to defeat the other. This is a natural approach for problems such as game-playing where often an optimal opponent is not available.

Coevolution

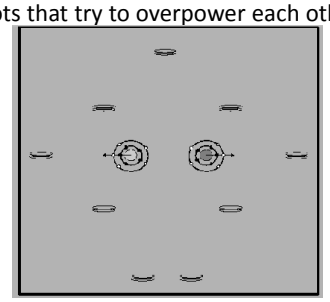
- A very different kind of coevolutionary model emphasizes **cooperation**. Cooperative coevolution is motivated, in part, by the recognition that the complexity of difficult problems can be reduced through modularization.
- In cooperative coevolutionary algorithms the species represent solution components. Each individual forms a part of a complete solution but need not represent anything meaningful on its own. The components are evolved by measuring their contribution to complete solutions and recombining those that are most beneficial to solving the task.

Complexification in competitive coevolution

- NEAT hypothesis: the complexification process allows discovering more sophisticated strategies, i.e. strategies that are more effective, flexible, and general, and include more components and variations than do strategies obtained through search in a fixed space
- To demonstrate this hypothesis, it is required a domain where it is possible to develop a wide range of increasingly sophisticated strategies, and where sophistication can be readily measured.
- Solution: A competitive coevolution domain is particularly appropriate because a sustained 'arms race' should lead to increased sophistication.

The Robot Duel Domain

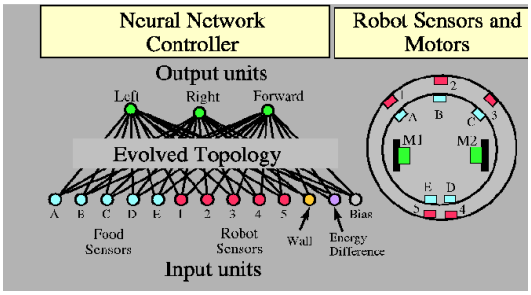
- The domain used to show the effects of complexification consists of two simulated robots that try to overpower each other



The Robot Duel Domain

- The robot duel task supports a broad range of sophisticated strategies that are easy to observe and interpret.
- The competitors must become proficient at foraging, prey capture, and escaping predators. In addition, they must be able to quickly switch from one behavior to another.
- The task is well-suited to competitive coevolution because naive strategies such as forage-then-attack can be complexified into more sophisticated strategies such as luring the opponent to waste its energy before attacking.

The Robot Duel Domain



The Robot Duel Domain

- Each has two wheels controlled by separate motors. Five rangefinder sensors can sense food and another five can sense the other robot. Finally, each robot has an energy-difference sensor, and a single wall sensor.
- The robots are controlled with neural networks evolved with NEAT. The networks receive all of the robot sensors as inputs, as well as a constant bias that NEAT can use to change the activation thresholds of neurons.
- They produce three motor outputs: Two to encode rotation either right or left, and a third to indicate forward motion power. These three values are then translated into forces to be applied to the left and right wheels of the robot.

Competitive Coevolution Setup

- The robot duel domain supports highly sophisticated strategies. Thus, the question in such a domain is whether **continual coevolution** will take place, i.e. whether increasingly sophisticated strategies will appear over the course of evolution.
- In competitive coevolution, every network should play a sufficient number of games to establish a good measure of fitness. To encourage interesting and sophisticated strategies, networks should play a diverse and high quality sample of possible opponents.
- A solution is to evolve two separate populations, one for each robot. In each generation, each population is evaluated against an intelligently chosen sample of networks from the other population.

Competitive Coevolution Setup

- The population currently being evaluated is called the **host** population, and the population from which opponents are chosen is called the **parasite** population . The parasites are chosen for their quality and diversity, making host/parasite evolution more efficient and more reliable than one based on random or round robin tournament.
- Each host is evaluated against four highest species champions from the parasite population. Other eight opponents are chosen randomly from a **Hall of Fame composed of all generation champions**. This exploits speciation and fitness sharing that occurs in NEAT. The Hall of Fame ensures that existing abilities need to be maintained to obtain a high fitness.
- Together, speciation, fitness sharing, and Hall of Fame comprise an effective competitive coevolution methodology.

Monitoring Progress in Competitive Coevolution

- A competitive coevolution run returns a record of every generation champion from both populations.
- How can a sequence of increasingly sophisticated strategies be identified in this data, if one exists?
- The answer is the **dominance tournament** method for monitoring progress in competitive coevolution
- First of all, it is necessary to come up with a way to make individual comparisons, i.e. whether one strategy is superior to another one; the two champion networks (one from each population) to be compared compete in 288 games; those consist in various starting and food positions so that the advantages and disadvantages are balanced between the two individuals.
- A network A **is superior** to a network B if it wins more games than B out of the 288 total games.

Monitoring Progress in Competitive Coevolution

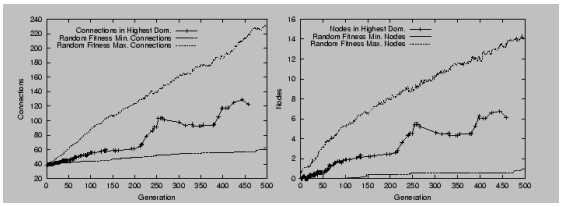
- In order to decisively track strategic innovation, we need to identify **dominant strategies**, i.e. those that defeat **all previous** dominant strategies. This way, we can make sure that evolution proceeds by developing a progression of strictly more powerful strategies, instead of e.g. switching between alternative ones.
- This is accomplished by the **dominance tournament** method
- A **generation champion** is the winner of a 288 game comparison between the host and parasite champions of a single generation.
- d_j is the jth dominant strategy to appear over evolution. Dominance is defined recursively starting from the first generation and progressing to the last:

Monitoring Progress in Competitive Coevolution

- d_j is the jth dominant strategy to appear over evolution. Dominance is defined recursively starting from the first generation and progressing to the last:
- The first dominant strategy d_1 is the generation champion of the first generation;
- Dominant strategy d_j, where j > 1, is a generation champion such that for all I < j, d_j is superior to d_i (i.e. wins the 288 game comparison with it).
- This definition of dominance prohibits circularities.

Evolution of Complexity

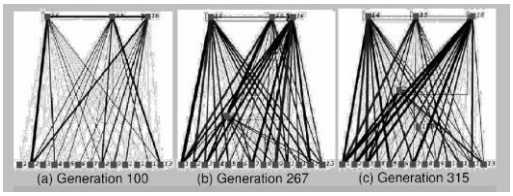
- Armed with the appropriate coevolution methodology and a measure of success, the question that remains is : Does complexification result in more successful competitive coevolution?
- In order to analyze the results, **complexity** is defined as the number of nodes and connections in a network.



Evolution of Complexity

- There are two underlying forces of progress: The building of new structures, and the continual optimization of prior structures in the background. The product of these two trends is a gradual stepped progression towards increasing complexity.
- An important question is: does the complexity always increase whether it helps in finding good solutions or not?
- To see how complexification contributes to evolution, the development over time of a sample dominant strategy is observed

Sophistication through Complexification



Sophistication through Complexification

- In some cases, weight optimization alone can produce improved strategies.
- However, when those strategies need to be extended, adding new structure allows the new behaviors to coexist with old strategies.
- Also, in some cases it is necessary to add complexity to make the timing or execution of the behavior more accurate. Results obtained show how complexification can be utilized to produce increasing sophistication.
- Complexifying coevolution was also compared to two alternatives: standard coevolution in a fixed search space, and simplifying coevolution from a complex starting point. The results obtained show the net superiority of complexification.

Discussion

- What makes complexification such a powerful search method?

- Whereas in fixed-topology coevolution, as well as in simplifying coevolution, the good structures must be optimized in the high-dimensional space of the solutions themselves, complexifying coevolution only searches high-dimensional structures that are elaborations of known good lower-dimensional structures.

Discussion

- Before adding a new dimension, the values of the existing genes have already been optimized over preceding generations. Thus, after a new gene is added, the genome is already in a promising part of the new, higher-dimensional space.

- Thus**, the search in the higher-dimensional space is not starting blindly as it would if evolution began searching in that space**. It is for this reason that complexification can find high-dimensional solutions that fixed-topology coevolution and simplifying coevolution cannot.

References

- 1. Kenneth O. Stanley, Competitive Coevolution through Evolutionary Complexification, Department of Computer Sciences, The University of Texas at Austin (http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html)

- 2. Wikipedia, Neuroevolution http://en.wikipedia.org/wiki/Neuroevolution

- 3. Faustino Gomez, Jurgen Schmidhuber, Risto Miikkulainen, Accelerated Neural Evolution through Cooperatively Coevolved Synapses