

Introduction to Evolutionary Computation and Evolutionary Computation Module Tutorial 6

V. Landassuri-Moreno

v.landassuri-moreno@cs.bham.ac.uk

School of Computer Science
University of Birmingham

December 4, 2009

Outline

Comments from previous EA with real representation in Matlab

Exercise 7

Exercise 8

EPNet algorithm

Neat algorithm

Summary

Implement the simple EA with real representation

- ▶ In tutorial 5 it was presented a Matlab implementation of a simple EA with real representation.
- ▶ It is was commented two ways to calculate the roulette wheel selection, one of them does not work OK with negative numbers.
- ▶ Another option that was raised was to take the 'abs' of the min fitness and sum it to all the fitness:

Implement the simple EA with real representation

- ▶ In tutorial 5 it was presented a Matlab implementation of a simple EA with real representation.
- ▶ It is was commented two ways to calculate the roulette wheel selection, one of them does not work OK with negative numbers.
- ▶ Another option that was raised was to take the 'abs' of the min fitness and sum it to all the fitness:
- ▶ e.g.

Having 5 individuals with fitness:

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

Implement the simple EA with real representation

- ▶ In tutorial 5 it was presented a Matlab implementation of a simple EA with real representation.
- ▶ It is was commented two ways to calculate the roulette wheel selection, one of them does not work OK with negative numbers.
- ▶ Another option that was raised was to take the 'abs' of the min fitness and sum it to all the fitness:

- ▶ e.g.

Having 5 individuals with fitness:

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

$$\sum f_i = 10 \quad \text{which is incorrect}$$

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 2, f_2 = 97, f_3 = 107, f_4 = 112, f_5 = 202;$$

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 2, f_2 = 97, f_3 = 107, f_4 = 112, f_5 = 202;$$

Calculating the inverse to give more priority to best individuals

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 2, f_2 = 97, f_3 = 107, f_4 = 112, f_5 = 202;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 0.50, f_2 = 0.0103, f_3 = 0.0093, f_4 = 0.0089, f_5 = 0.0050; \sum f_i = 0.6262$$

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 2, f_2 = 97, f_3 = 107, f_4 = 112, f_5 = 202;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 0.50, f_2 = 0.0103, f_3 = 0.0093, f_4 = 0.0089, f_5 = 0.0050; \sum f_i = 0.6262$$

Calculating $\frac{f_i}{\sum f_i}$

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 2, f_2 = 97, f_3 = 107, f_4 = 112, f_5 = 202;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 0.50, f_2 = 0.0103, f_3 = 0.0093, f_4 = 0.0089, f_5 = 0.0050; \sum f_i = 0.6262$$

Calculating $\frac{f_i}{\sum f_i}$

$$p_1 = 0.7985, p_2 = 0.1645, p_3 = 0.0149, p_4 = 0.0142, p_5 = 0.0080$$

Implement the simple EA with real representation

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 2 to the previous calculation to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 2, f_2 = 97, f_3 = 107, f_4 = 112, f_5 = 202;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 0.50, f_2 = 0.0103, f_3 = 0.0093, f_4 = 0.0089, f_5 = 0.0050; \sum f_i = 0.6262$$

Calculating $\frac{f_i}{\sum f_i}$

$$p_1 = 0.7985, p_2 = 0.1645, p_3 = 0.0149, p_4 = 0.0142, p_5 = 0.0080$$

- ▶ As it can be seen, the best individual has a big probability to be selected, so, the Rank base selection may be useful here



Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:



Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Calculating the inverse to give more priority to best individuals

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 10, f_2 = 0.0105, f_3 = 0.0095, f_4 = 0.0091, f_5 = 0.0050; \sum f_i = 10.1286$$

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 10, f_2 = 0.0105, f_3 = 0.0095, f_4 = 0.0091, f_5 = 0.0050; \sum f_i = 10.1286$$

Calculating $\frac{f_i}{\sum f_i}$

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 10, f_2 = 0.0105, f_3 = 0.0095, f_4 = 0.0091, f_5 = 0.0050; \sum f_i = 10.1286$$

Calculating $\frac{f_i}{\sum f_i}$

$$p_1 = 0.9873, p_2 = 0.0010, p_3 = 0.00093794, p_4 = 0.00089845, \\ p_5 = 0.00049365$$

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 10, f_2 = 0.0105, f_3 = 0.0095, f_4 = 0.0091, f_5 = 0.0050; \sum f_i = 10.1286$$

Calculating $\frac{f_i}{\sum f_i}$

$$p_1 = 0.9873, p_2 = 0.0010, p_3 = 0.00093794, p_4 = 0.00089845, \\ p_5 = 0.00049365$$

- ▶ In this case it was worst as the previous case, then you need to think very well each part of the algorithm.

Same scenario as the previous, but adding 0.1 instead of 2

$$f_1 = -100, f_2 = -5, f_3 = 5, f_4 = 10, f_5 = 100$$

- ▶ Now, taking the $\text{abs}(\min(f_i)) = 100$, then after we add this value to each fitness we have:

$$f_1 = 0, f_2 = 95, f_3 = 105, f_4 = 110, f_5 = 200$$

But usually we want the minimum to be in 0, so we can add a value of 0.1 to the previous to stop the algorithm when we reach a value of 0, new fitness:

$$f_1 = 0.1, f_2 = 95.1, f_3 = 105.1, f_4 = 110.1, f_5 = 200.1;$$

Calculating the inverse to give more priority to best individuals

$$f_1 = 10, f_2 = 0.0105, f_3 = 0.0095, f_4 = 0.0091, f_5 = 0.0050; \sum f_i = 10.1286$$

Calculating $\frac{f_i}{\sum f_i}$

$$p_1 = 0.9873, p_2 = 0.0010, p_3 = 0.00093794, p_4 = 0.00089845, \\ p_5 = 0.00049365$$

- ▶ In this case it was worst as the previous case, then you need to think very well each part of the algorithm.
- ▶ Now, instead to take the inverse, if you calculate $(1 - p_i)$, do you think it may be better?

Q1. Difference between Fitness sharing, Crowding and Speciation

Fitness sharing:

- ▶ Is a method for niching
- ▶ Limit the fitness of solutions based in a distance measure, (similar individuals share fitness). Then transform the raw fitness of an individual into the shared one
- ▶ Prevent clustering at a peak
- ▶ Maintain diversity and the algorithm explore the full search space
- ▶ Individuals will never actually converge on the optimum
- ▶ It can be applied in the genotype or phenotype level

Q1.

Crowding:

- ▶ Crowding techniques do not modify fitness values
- ▶ Maintain diversity by replacing similar solutions
- ▶ It takes place in the replacement phase of the EA

Speciation:

- ▶ Speciation methods occur in the recombination phase and its objective is to maintain diversity
- ▶ Focuses on finding the peaks in some search space by applying mating restriction (limiting recombination)
- ▶ Encouraging the exploitation of fit solutions

Q2. *Functional and categorical modularization from paper Speciation as automatic categorical modularization*

- ▶ In functional modularization, the components perform very different tasks, such as subroutines of a large software project.
- ▶ In categorical modularization, the components perform different versions of basically the same task, such as antibodies in the immune system against certain type of antigen.

Example given by a student

For example, take a car showroom.

- ▶ Functional modularization would have a module/person in charge of welcoming a customer, one in charge of taking a customer for a test drive, and one in charge of negotiating the purchase/contract, etc..
- ▶ On the other hand, categorical modularization would have a module/person in charge of selling porsches, one in charge of selling renaults, one in charge of selling volkswagons, etc..

Q3. Can you explain what is Genetic programming?

- ▶ Create / develop programs by artificial evolution that perform a user-defined task
- ▶ Then an individual is a program
- ▶ It is not limited for a unique representation (there are diverse representations)
- ▶ From another point of view, it is a way to automatically develop programs

Q4. Which kind of representation can be used in GP?

- ▶ Trees (mathematical operators/symbols/functions)
- ▶ Linear genetic programming
- ▶ Network representation (graphs, ANNs or routing problems)

Q5. Is it possible to use Crossover and Mutation in a GP? Can you give an example?

- ▶ Yes, it is possible but you need to take care of some aspects
- ▶ e.g. in Crossover the algorithm can switch nodes, but it is possible to obtain bigger representations (bloat problem)
- ▶ In tree representation: replacing a node means replacing the whole branch
- ▶ In Mutation there are some nodes that need to be mutated systematically to obtain a desired result, but that is a disadvantage
 - ▶ Then, in Mutation is deleted a random branch and replaced with a new random branch

Q6. What is bloat in a GP? and How can we manage it in a tree-base representation?

- ▶ Bloat is the uncontrolled growth of the average size of an individual in the population
- ▶ Or in other words, uncontrolled growth of program length
- ▶ The most common approach to dealing with bloat in tree-based genetic programming individuals, is to limit their maximal allowed depth
 - ▶ Pruning
 - ▶ Recombination that only swaps subtrees of similar structure (which means that there is not possible to change the size of the tree)

Q1. No-Free Lunch (NFL) theorem

- ▶ In general the No-Free Lunch theorem says that any search / optimization method will have an average performance in a given number of problems to solve
- ▶ i.e. if you have an algorithm A that performs well in a given number of problems (P1), then there will exist other different problems (P2) where algorithm A will have a poor performance. On the other hand, there could be an algorithm B that performs better than A in P2 but worst in P1
- ▶ From another point of view, it can be said that there is not a general or universal algorithm for optimization

Q2. Selection pressure

Definition 1:

- ▶ The intensity with which an environment tends to eliminate an organism, and thus its genes, or to give it an adaptive advantage. ¹

Definition 2:

- ▶ Degree to which selection emphasizes the better individuals (definition from lectures).

Comments:

- ▶ Then an algorithm with low selection pressure will have a big amount of average or bad individuals
- ▶ In this way, a high selection pressure is focused to improve the actual solutions and low selection pressure will be more focused to discover new solutions
- ▶ e.g. if we look at the *elitism*, automatically it can be said that it has a high selection pressure

¹ISCID Encyclopedia of Science and Philosophy, accessed Oct 2009, url: <http://www.iscid.org/>

Q4. Could you explain what is the *Michigan* and *Pitt* approach?

Michigan and *Pittsburgh* approach are concepts used in Evolutionary learning.

Michigan

- ▶ In the Michigan approach one individual represent a rule
- ▶ In this way all the population is the complete learning system
- ▶ **Drawback:** the credit assignment problem for each part, i.e. how well it can be evaluated the fitness of each part
- ▶ Note that the Back-Propagation algorithm could be a similar approach that has manage that issue (but only apply to ANNs's training)

Pittsburgh

- ▶ On the other hand, in the Pittsburgh approach, one individual is a complete system (set of rules)

Q5. What are the parts of Neural Networks that can be evolve with EAs?

Training

- ▶ Evolve the weights, however it could inefficient, i.e. if the weights are change randomly
- ▶ One approach could be the use of Simulated Annealing like the used in the EPNet algorithm

Topology

- ▶ Evolve Layers
- ▶ Nodes
- ▶ Connections

Learning rules

- ▶ Evolve the way in which ANNs learn

Node transfer functions

- ▶ Evolve the type of transfer function selecting one for each node/layer: linear, step, ramping, sigmoid, tan-sigmoid or softmax functions
- ▶ Evolve the transfer function and use the evolved solution as the function for nodes/layers

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons
- ▶ There are dependencies between the position of a neuron and its neighbors

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons
- ▶ There are dependencies between the position of a neuron and its neighbors
- ▶ If it is applied a crossover operator to evolve them, usually the information carried by neurons is lost

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons
- ▶ There are dependencies between the position of a neuron and its neighbors
- ▶ If it is applied a crossover operator to evolve them, usually the information carried by neurons is lost
- ▶ Putting a group of *good* neurons from different ANN's together may not produce a better ANN

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons
- ▶ There are dependencies between the position of a neuron and its neighbors
- ▶ If it is applied a crossover operator to evolve them, usually the information carried by neurons is lost
- ▶ Putting a group of *good* neurons from different ANN's together may not produce a better ANN
- ▶ If it is used a recombination operator it could be presented the permutation problem (do you agree?)

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons
- ▶ There are dependencies between the position of a neuron and its neighbors
- ▶ If it is applied a crossover operator to evolve them, usually the information carried by neurons is lost
- ▶ Putting a group of *good* neurons from different ANN's together may not produce a better ANN
- ▶ If it is used a recombination operator it could be presented the permutation problem (do you agree?)
- ▶ For this reason the EPNet algorithm [YL97b] use only mutations to modify the architecture and weights of the networks

Q6. Crossover to evolve ANNs?

- ▶ The information in a Neural network is spread over all neurons
- ▶ There are dependencies between the position of a neuron and its neighbors
- ▶ If it is applied a crossover operator to evolve them, usually the information carried by neurons is lost
- ▶ Putting a group of *good* neurons from different ANN's together may not produce a better ANN
- ▶ If it is used a recombination operator it could be presented the permutation problem (do you agree?)
- ▶ For this reason the EPNet algorithm [YL97b] use only mutations to modify the architecture and weights of the networks
- ▶ Nevertheless, there are other algorithms that use crossover to evolve them, like the NEAT algorithm (NeuroEvolution of Augmenting Topologies)

EPNet algorithm

In this section it is going to be presented an EA to Evolve Artificial Neural Networks called EPNet [YL97b, Yao99, YL97a]

- ▶ Based in the Evolutionary Programming approach

EPNet algorithm

In this section it is going to be presented an EA to Evolve Artificial Neural Networks called EPNet [YL97b, Yao99, YL97a]

- ▶ Based in the Evolutionary Programming approach
- ▶ Thus, only mutations are used to evolve the ANNs

EPNet algorithm

In this section it is going to be presented an EA to Evolve Artificial Neural Networks called EPNet [YL97b, Yao99, YL97a]

- ▶ Based in the Evolutionary Programming approach
- ▶ Thus, only mutations are used to evolve the ANNs
 1. Hybrid training formed with Modified BackPropagation algorithm and Simulated Annealing
 2. Node deletion
 3. Connection deletion
 4. Node addition
 5. Connection addition

EPNet algorithm

In this section it is going to be presented an EA to Evolve Artificial Neural Networks called EPNet [YL97b, Yao99, YL97a]

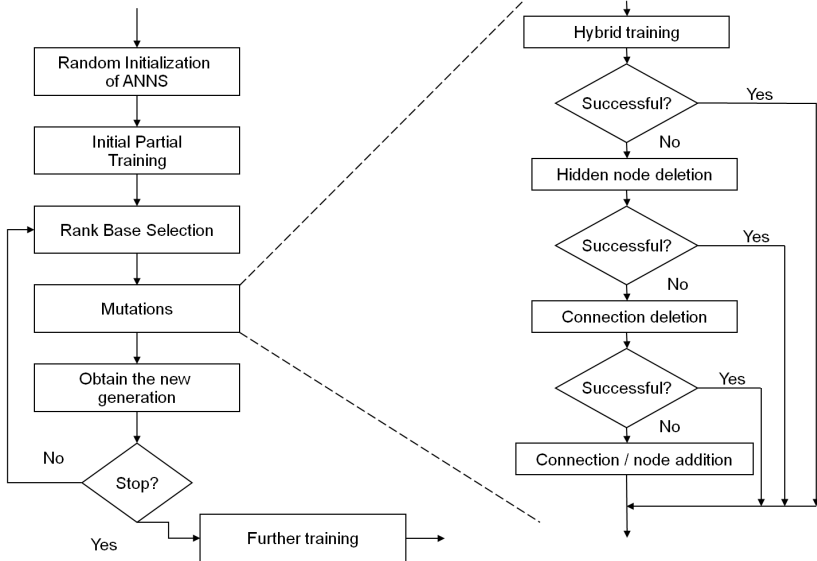
- ▶ Based in the Evolutionary Programming approach
- ▶ Thus, only mutations are used to evolve the ANNs
 1. Hybrid training formed with Modified BackPropagation algorithm and Simulated Annealing
 2. Node deletion
 3. Connection deletion
 4. Node addition
 5. Connection addition
- ▶ Developed to evolve Architectures and Learning at the same time (Lamarckian inheritance)

EPNet algorithm

In this section it is going to be presented an EA to Evolve Artificial Neural Networks called EPNet [YL97b, Yao99, YL97a]

- ▶ Based in the Evolutionary Programming approach
- ▶ Thus, only mutations are used to evolve the ANNs
 1. Hybrid training formed with Modified BackPropagation algorithm and Simulated Annealing
 2. Node deletion
 3. Connection deletion
 4. Node addition
 5. Connection addition
- ▶ Developed to evolve Architectures and Learning at the same time (Lamarckian inheritance)
 - ▶ Through partial training

EPNet algorithm



Neat algorithm

In [SM01] is presented another algorithm to Evolve Artificial Neural Networks called NEAT (NeuroEvolution Augmenting Topologies)

- ▶ Claim to be an efficient system capable of evolving complex structures

Neat algorithm

In [SM01] is presented another algorithm to Evolve Artificial Neural Networks called NEAT (NeuroEvolution Augmenting Topologies)

- ▶ Claim to be an efficient system capable of evolving complex structures
- ▶ Historical marking are used to know the precedence of genes (Crossover)

Neat algorithm

In [SM01] is presented another algorithm to Evolve Artificial Neural Networks called NEAT (NeuroEvolution Augmenting Topologies)

- ▶ Claim to be an efficient system capable of evolving complex structures
- ▶ Historical marking are used to know the precedence of genes (Crossover)
- ▶ It uses speciation for innovation

Neat algorithm

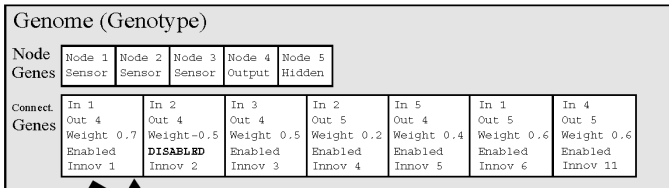
In [SM01] is presented another algorithm to Evolve Artificial Neural Networks called NEAT (NeuroEvolution Augmenting Topologies)

- ▶ Claim to be an efficient system capable of evolving complex structures
- ▶ Historical marking are used to know the precedence of genes (Crossover)
- ▶ It uses speciation for innovation
- ▶ Initialize the networks with minimal configurations

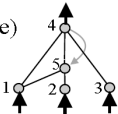
Neat algorithm

In [SM01] is presented another algorithm to Evolve Artificial Neural Networks called NEAT (NeuroEvolution Augmenting Topologies)

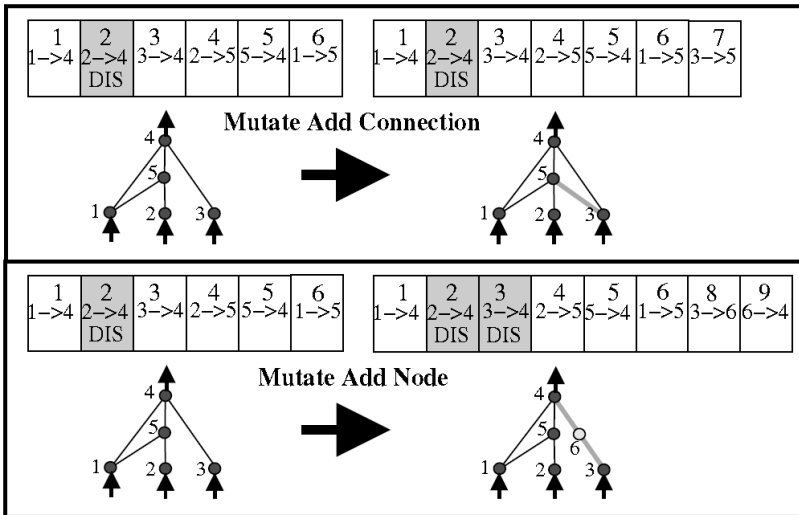
- ▶ Claim to be an efficient system capable of evolving complex structures
- ▶ Historical marking are used to know the precedence of genes (Crossover)
- ▶ It uses speciation for innovation
- ▶ Initialize the networks with minimal configurations

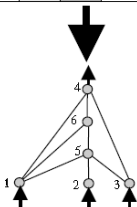
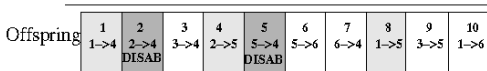
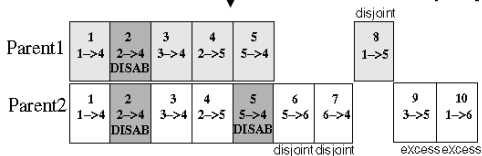
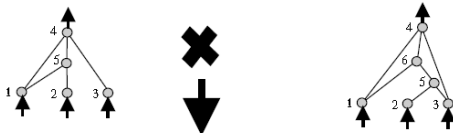
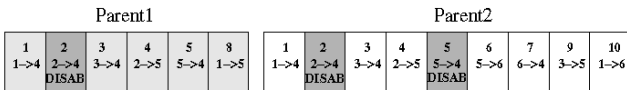


Network (Phenotype)



Neat algorithm





Summary

- ▶ It has been presented the exercise 7 and 8
- ▶ Reviewed some concepts from fitness sharing, GP, NFL and ANNs
- ▶ We saw an explanation and comparison of the EPNet and algorithm based in EP and the NEAT algorithm to evolve ANNs

References



[Kenneth O. Stanley and Risto Miikkulainen.](#)

Evolving neural networks through augmenting topologies.

Evolutionary Computation, 10:2002, 2001.



[Xin Yao.](#)

Evolving artificial neural networks.

Proceedings of the IEEE, 87(9):1423–1447, 1999.



[Xin Yao and Yong Liu.](#)

EPNet for chaotic time-series prediction.

In *SEAL'96: Selected papers from the First Asia-Pacific Conference on Simulated Evolution and Learning*, pages 146–156, London, UK, 1997. Springer-Verlag.



[Xin Yao and Yong Liu.](#)

A new evolutionary system for evolving artificial neural networks.

IEEE Transactions on Neural Networks, 8(3):694–713, 1997.