**Joni Töyrylä**

# Building NeuroSearch - Intelligent Evolutionary Search Algorithm For Peer-To-Peer Environment

Master's Thesis
in Information Technology
3rd September 2004

**University of Jyväskylä**

**Department of Mathematical Information Technology**

**Jyväskylä**

**Author:** Joni Töyrylä
**Contact information:** Taitoniekantie 9b 304,
40740 Jyväskylä
email: jtoyr@cc.jyu.fi
phone: 014-607176
**Title:** Building NeuroSearch - Intelligent Evolutionary Search Algorithm For Peer-To-Peer Environment
**Työn nimi:** NeuroSearch hakualgoritmi vertaisverkko ympäristössä
**Project:** Master's Thesis in Information Technology
**Page count:** 72
**Abstract:** This thesis describes the results of a process where evolutionary artificial Neural Networks (EANN) were developed for finding an efficient search algorithm for Peer-To-Peer (P2P) networks. In this research questions like how structural changes of neural network affect the search and how the selection of fitness function or input values affects the resulting neural network are considered. The thesis also analyzes the different behavior that neural networks algorithms can exhibit and sketchs the limitations inherent in this kind of search algorithm design.
**Suomenkielinen tiivistelmä:** Tutkimuksessa kuvataan evolutionaarisen hakualgoritmin kehittymisprosessi ja sen tehokas käyttö vertaisverkossa. Tutkimuksessa pohditaan kuinka neuroverkon rakenteelliset muutokset, erilaiset sisääntuloarvot ja hyvyysfunktion muuttuminen vaikuttavat hakualgoritmin kehittymiseen ja toimintaan. Tutkimuksessa selvitetään minkälaisia eri luonteenpiirteitä neuroverkkohakualgoritmeilla voi esiintyä ja hahmotellaan rajoituksia, jotka tämän tyyppisessä hakualgoritmin suunnittelussa on olemassa.
**Keywords:** Peer-to-Peer, P2P, Neural Network, resource discovery, evolutionary computing, search algorithm
**Avainsanat:** Vertaisverkko, P2P, Neuroverkko, evoluutiolaskenta, hakualgoritmi, resurssien löytäminen

# Glossary

| | |
|---|---|
| Activation total | Value of all weighted connections summed together (in summation units). See: net sum |
| ANN | Artificial Neural Network. |
| BFS | Breadth First Search. Peer-To-Peer resource query type. Query will be forwarded to all neighbors except the one where it came from. Proceeding of the query is stopped with Time-To-Live value. BFS-2 means that time-to-live value is set to 2. |
| Bias | Unzero value which is added to neurons activation total. Without bias the output of neuron would also always zero if all inputs would be zero. Bias may also be weighted as other connections. |
| DFS | Depth First Search. Peer-To-Peer resource query type. Query is routed to one neighbor at the time. |
| DNS | Domain Name Server. Server translates computer ip's to domain names and vice versa. |
| FFNN | Feed-Forward Neural network. The most common, basic neural network architecture. FFNN is used in NeuroSearch. |
| FreeNet | Decentralized, structured P2P-System. |
| Gnutella | Decentralized, unstructured P2P-System. |
| HDS | Highest Degree Search. Peer-To-Peer resource query type. Query is forwarded to one neighbor at time, as DFS, but largest node is selected to be the target. Largestnode is node with most neighbors. |
| Hops | Times of links the message has passed in P2P-Network. |
| Input value | Neural network is fed with input values. Properly trained NN will generate correct output from these values. |
| Net sum | Value of all weighted connections summed together. See: activation total. |

| | |
|---|---|
| NeuroSearch | Intelligent resource discovery algorithm in P2P-Network. Main subject of this thesis. |
| P2P, P2P-Network | Peer-To-Peer. Nodes forming the network are all functioning as clients and servers. |
| Power-law network | Network where few nodes have lots of connections and lots of nodes have few connections. |
| Product unit | Neuron that multiplies all weighted connections together for activation total. See: summation unit. |
| Random network | Network where nodes are randomly connected to each others. |
| Resource discovery problem | How to minimize use of sent messages and maximize found resources in P2P-Network. |
| Scale-free networks | See: Power-law network. |
| Small world network | Distance between nodes is small related to size of the network. |
| Structured P2P-Topology | Nodes form the network using strict rules. There is a reason why this node is in that place in the network. Also called as Distributed Hash Table P2P-Topology. |
| Summation unit | Neuron that sums all weighted connections together for activation total. See: product unit. |
| Time-To-Live | Measures how long query message may travel in P2P-Network. This value is decreased every time the message reaches new node. |
| Unstructured P2P-Topology | Nodes form the network without rules. Nodes may join the network at any time and to any place. |
| Weight | Every connection between neurons have weight value which modifies the connection strength. |

# Contents

# 1 Introduction

In P2P-Networks discovering resources is a central problem because nodes forming the network do not before hand know what kind of structure network has or how the resources are distributed. Different algorithms have been suggested and they have advantages and disadvantages. In this thesis is presented one search algorithm, NeuroSearch. As in all P2P-Network query algorithms also in NeuroSearch the routing of messages is based on local knowledge.

When P2P-Node receives a message it routes it forward using some rules. It may forward it further if some value is fullfilled or it may drop the message for the same reason. NeuroSearch is intelligent algorithm. The decision of routing is based on output of neural network. Neural networks are one major research area at this time in artificial intelligence.

Neural network cannot make right decisions in the beginning. It needs to be trained. NeuroSearch is trained using evolutionary computing, more precisely using a combination of evolutionary programming and evolutionary strategies. In the beginning there are some population of neural network candidates. These all are tested in P2P-Network and their performance is measured. The best half of population are chosen to stay while the rest is discarded. Remaining neural networks are used to build new half of the population and then these are tested again. This way good abilities of neural networks survive and are multiplied and mutated.

Constructing NeuroSearch is complicated and time consuming process. It contains many parts and beforehand it is not known how different parts modify the outcome. Changing the architecture of neural network affects the outcome as well as many parameters in training. These changes affect the behavior, performance and training time of the algorithm. In this thesis is examined how these changes influence e.g., which change lowers the performance, which rises it and how the behavior of the algorithm changes.

## 1.1 Resource Discovery Problem

Resource discovery is essential problem in Peer-To-Peer (P2P) [19] networks. P2P-Network is formed by nodes connecting each other. Nodes contain resources. Nodes also want to communicate with other nodes, locate and grab resources other nodes have. Because in P2P there is no central points containing global information of the network this is not that simple task. There is no list of all nodes, connections nor resources in the network. When trying to locate resources on the network the node can only send the resource query to its neighbors and hope that after sometime it will be informed by some of its neighbors where the wanted resource was found. But with what information the node makes its decision - to send or not to send? If nodes send queries with too loose principles the network will eventually be flooded with packets. If nodes are too strict of sending query further the query stops too early. How to minimize the use of packets but still maximize founded resources?

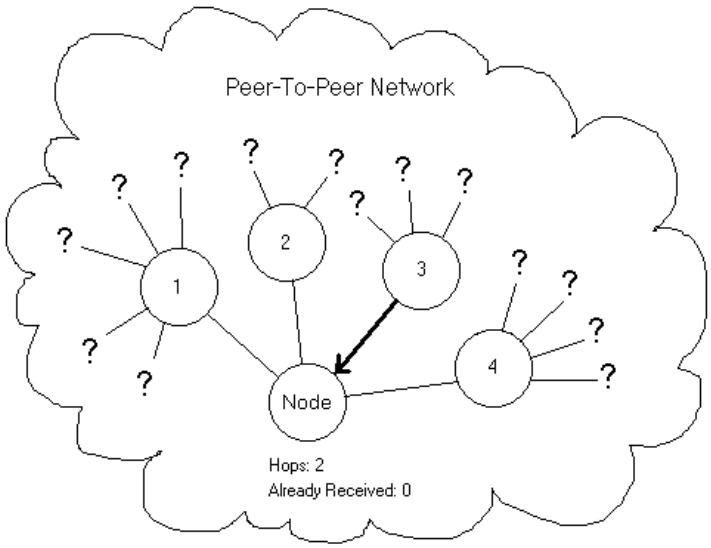


Figure 1.1: P2P-Node receives resource query from its neighbor, node number 3. Now, based on local knowledge it has to decide which neighbors would be the best targets (if any) for the query to be successful globally.

In the figure 1.1 local routing situation in P2P-Network is shown. Node receives resource query from node 3. It has 4 neighbors which it knows but the rest of the network is unknown. It knows that this message has travelled 2 hops, it knows that this message has not been here earlier and it knows its neighbor's neighbor amounts.

There is no list of resources and no global view of topology to optimize search route. Every node who receives queries will make the decision based on conditions on that moment in that local situation.

Gnutella's [12] Breadth First Search (BFS) [16] is one widely used P2P search algorithm. The idea of this algorithm is to flood the whole network with queries so that every node will be queried. This search will always find all the resources of the network but the backside is that it will use huge amount of packets. Gnutella uses Time-To-Live (TTL) value in messages, so every time message is routed further TTL decreases by one. When TTL is zero the message is dropped. Even while using TTL the algorithms which behaviors this way will eventually slow the network down. When size of the network grows there are too many simultaneous messages and thus latency will grow and eventyally when buffers become full packets will be dropped. It has been shown that Gnutella [12] is not scalable thus it cannot work efficiently when network grows. So better search strategies needs to be found.

One solution for the problem is to use Artificial Neural Network (ANN) [9] as search algorithm. Every P2P-Node will base its routing decision on neural network's output. Output value is revealed when the neural network is calculated through with chosen input values. Before neural network can be calculated it needs to be trained and methods of evolutionary computing can be used to do this. This thesis describes how evolutionary neural network search algorithm, named as NeuroSearch, was created and how well it performs contrasted with other search algorithms.

NeuroSearch needs to be trained to solve the resource query problem. This is done by using evolutionary computing e.g., selecting strongest neural networks and discarding the weakest ones. There is problem that neural network overlearns the settings of environment and it cannot manage same quality if environment changes. This is described more closely in chapter 5.

## 1.2 Cheese Factory Project

This work is part of the Cheese Factory research project in Agora Center [2]. Agora Center is part of the University of Jyväskylä and contains multiple research projects. Cheese Factory project's interest are for example Peer-To-Peer network architectures, message routing, P2P in mobile environment, distributed data fusion and neural net-

works.

## 1.3 Related Work

Michael Iles and Dwight Deugo have researched how Gnutella network can be constructed and maintained more efficiently [14]. They used genetic programming to find the optimal rules for Gnutella's network protocol. Connection amount was found to be six instead of commonly used seven and new rule for establishing new connections was determined also. In the result, their network was more than 10% efficient compared to original Gnutella network. All the information for managing the network can be done locally so the protocol is a direct substitute to the one used by Gnutella. Although managing and constructing P2P-Networks is not subject of this thesis this article gives a clue what computationally intelligent methods can accomplish when used properly for complex problems.

In [26] one NeuroSearch algorithm is shown. Six input values, scaling, activation and fitness functions are revealed. After 100000 generations of 30 Neural Networks the best was chosen and tested against Gnutella BFS. NeuroSearch has found to be more stabile i.e., resource query succeeded whether it was started by edge or central node. Efficiency of BFS search depends where querying node is located thus queries started from edge were significantly poorer than those started from center.

# 2 Peer-To-Peer Network

In this chapter brief history, different architectures and search algorithms of Peer-To-Peer networks is reviewed. Two decentralized Peer-To-Peer systems are introduced and their performance is analyzed.

## 2.1 Brief History

The early Internet, established at late 1960s was Peer-To-Peer (P2P) system. The objective of the ARPANET project was to share computing resources around the United States. Users of the network were researchers who generally knew each others [19, p.4].

When Internet started to grow it also started to change. Slow speed modem users used FTP and Telnet programs to connect bigger and faster machines. Corporations started to build firewalls to protect their internal networks from malicious users. World wide web and simple client/server protocol was introduced. Web users do not need continuous connection to Internet for using web. User just connects to web server, downloads the web page and disconnects. [19, p.9]. Internet was changed from peer-to-peer system where all machines were generally equal peers to client/server system where bigger and better connected servers are used by weaker clients.

In 1999, an 18-year old college student created Napster [18] file sharing program which success reshaped our thoughts of distributed computing and peer-to-peer systems. The Record Industry Association of America (RIAA) sued Napster in December of 1999 for copyright infringement [23].

Napster was peer-to-peer program because addresses of Napster nodes bypass DNS and because control of file transfer is shifted to nodes [19, p.25]. Napster was hybrid peer-to-peer network because there was centralized index of users and files on the network. Because of the centralized index Napster did not scale well since central directory was constantly updated [13]. Central server also made it possible to actually shut down the Napster network.

Publicity of Napster inspired more peer-to-peer development over the globe. Early in 2000 Nullsoft company introduced new P2P file sharing system - Gnutella [23]. Gnutella is pure P2P-System [22] e.g., there is no central points containing information of global view of the network. It is ad hoc network where peers may join or leave at will and users may decide what files they share to the network.

By March 2000 the first version of Freenet was released. Freenet is as well decentralised peer-to-peer system for file sharing. Purpose of the Freenet is that people may anonymously publish and retrieve any kind of information without fear of cencorship or punishment.

Different P2P-Systems and applications have been released e.g. KaZaA, file sharing system with two kind of P2P-Nodes, normal and supernodes [15], Publius - anonymous distributed publishing system [19, p.145] [20] and Free Haven - anonymous P2P storage system [19, p.159] [11]. These systems are not described in this thesis, but more information can be found from references.

## 2.2 Architectures

Peer-To-Peer networks are decentralized opposite to client/server networks where server is the central point of the network. Resources in the P2P-Network are stored in a distributed manner to the nodes opposite to client/server where resources are located in the servers. Peer-To-Peer is a way for decentralizing costs and administration because resources are distributed all over the network [19, p.23]. All nodes in the Peer-To-Peer network are connected to neighbors which in turn are connected to their neighbors.

In client/server network the client nodes are connected to server nodes. When client wants to locate a specific resource it sends query to server which replies if resource is found on its local data store. This kind of querying is simple and efficient which is one reason why it has been so widely used. However, there are drawbacks in client/server architecture. This system has weak point on its server because if server does not function properly all the clients will be out of service. Clients cannot share information between themselves because they are dependant on server. For this kind of network malfunction Peer-to-Peer network becomes invulnerable.

In client/server model content is located in the center. In Peer-To-Peer system content is located where it is created and used, in the common PC's of the network. There is no center and therefore there is no need to push data to central servers. Also, bandwidth usage and CPU usage are distributed all over the network [19, p.29]. "The Network is the Computer" is very true [19, p.56]. For example SETI@home distributed computing program has performance of 48,26 TeraFLOPs/second while having almost 5 million users all around the world in total and over thousand users every day [24].

Peer-To-Peer systems can be divided to two different architectures, hybrid P2P-Networks and pure P2P-Networks. Hybrid networks have global data index of the network containing information of the nodes and resources in the network. Pure networks do not have any centralized points in it and there is no global information of the peers nor resources. In pure P2P-Network any peer can be removed from the network without having any loss of network service [22]. Gnutella and Freenet are pure Peer-To-Peer networks and Napster is hybrid P2P-Network.

Pure P2P-Networks can be divided also to two based on their topology. These are unstructured and structured topologies. Unstructured topologies are created without any rules. Topology management is easy because peers may leave or join to network at any time and there is no restrictions which neighbors should be contacted. However, finding resources is more difficult because peers do not know where queried resources are located. Gnutella is unstructured pure Peer-To-Peer system.

In structured topologies there are strict rules how the topology is created. These rules determine how the neighbors are selected and therefore more efficient search may be used. However, these topologies are harder to keep stable in dynamic environments. Structured topologies include e.g., FreeNet, Chord, CAN, Pastry, Tapestry [13].

Peer-To-Peer networks may be constructed using whatever topology structure because it usually is based on TCP-connections. Thus Peer-To-Peer topology lays over physical topology and links between peers are not related how the computers are located in physical network [13, p.4].

It has been proven that Gnutella network forms power-law network [21]. In power-law network there are some nodes which have more neighbor nodes than others. These nodes form the core of the network and tie the network together thus making the net-

work diameter ultra-small [7].

## 2.3 Algorithms

In this section two different search algorithms are revealed and their performance is analyzed. Other algorithms are also available, e.g., Iterative deepening, Directed BFS, [27], Random Walkers, Highest Degree Search (HDS) [1], and many more [13, p.16] but these algorithms are not subject of this thesis.

In figure 2.1 is simple classification of few search algorithms. Breadth first search can be divided to normal BFS and directed BFS. Depth first search may be divided as highest degree search e.g., message will be routed to one neighbor only or none. This way the messages do not flood the network so easily but it may have difficulties to find resources on edges of P2P-Network. Random walkers are quite equal to HDS but the decision where the message is routed is chosen randomly. In HDS the target is chosen based on the number of neighbors. Gnutella is BFS-algorithm while Freenet is Depth First Search.

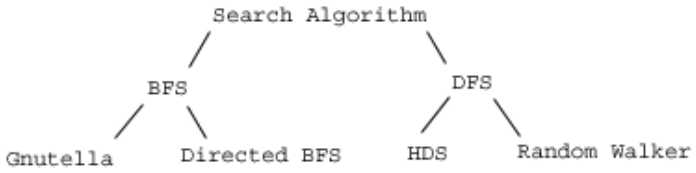


Figure 2.1: Few search algorithms

### 2.3.1 Gnutella

When Gnutella node wants to find something from the network it sends query message to all the neighbors it has. Those nodes check if they have the queried resource. They send the message further to all their neighbors except to the node who sent the message to them. If they have the resource they send a reply back towards the query originator but also forward the search message further.

Communication is message based TCP broadcasting. Each message has unique identifier (UUID). Every host that this message passes through memorizes it. When message comes to node it checks if the same message has already been here. If the message is memorized it is not retransmitted. Also, to prevent the network from flooding forever Gnutella uses Time To Live (TTL) number. TTL is decreased everytime message reaches new node in the network and when it reaches zero the query is dropped [19, p.105].

Gnutella does not scale because of the flooding. When size of the network grows the queries will either flood the whole network or queries can reach only some parts of the network because messages are dropped when TTL reaches zero [13, p.6]. Because Gnutella uses Breadth First Search it does not benefit the possible advantages of the network topology, e.g. power-law structure. Search will continue node by node without thinking would it be better to find central nodes where more resources are located, e.g., as in Highest Degree Search (HDS).

### 2.3.2 Freenet

FreeNet is also a decentralized Peer-To-Peer system but differs from Gnutella that the network is structured. Every document in the network has unique hash key value. Every node stores data, corresponding hash value and host address from where this data was received. These values are stored in a list where recently requested documents are located first. This list of nodes, keys and data is basically equal to Gnutella's neighbor list and thus these lists form the Freenet network. Keys that are associated to the data itself are made so that they can be compared. If keys are close each other the data is more likely to be found from that direction [19, p.127]. Because in Gnutella the topology is ad-hoc e.g., peers may join anywhere in the network or leave at any time Gnutella is message based network. The topology of Freenet is based on data lists managed by peers, and so it is storage based network.

When node starts to search for document it forwards the query to node that is found on the list and is more likely to have the searched document. Search continues until messages are dropped or requested document is found. When document is found it is replied back through the same nodes that previously forwarded it. Every node in the chain stores the document, the key and the host where it came from to the data list. This allows two benefits - documents are moving toward the searchers and thus they

are more easily reached and more commonly requested data is multiplied all over the network.

Data store that peer maintains is continuously changing. The data that is least requested is eventually removed from peers data list. However key and the host are kept because possible future requests could now be delivered to node which most likely will have it [19, p.128]. Bigger documents remain more bottom of the data lists and so they are removed more easily than small documents. This guides the users to spare space as well as compress the documents before inserting them to the network [19, p.126]. The network is all the time changing as the data moves and new references to the new nodes is established and unused data is removed from the network. This differs from Gnutella network where location of data does not affect the network structure. Nodes with more data will eventually get more links in Freenet and thus form the core of the network. This is because more requests are done to those nodes which have more data and more links.

### 2.3.3 Performance

Freenet network simulation where amount of nodes and links remain same all the time is able to shorten the path length, e.g., shortest route between any node in the network, as more data requests and transfers are processed [19, p.224]. Freenet's local routing cannot choose globally optimal path all the time but it manages to get close. Even when network grows Freenet is able to keep path length short. This is possible because new nodes that join the network tend to add links to nodes that have more links already. Nodes which have more links see more requests and thus will get more links. In Gnutella network the search locates shortest paths to results and therefore is quick. It also uses many nodes for search and therefore consumes bandwidth like explained earlier. Fault tolerance of Freenet is significantly better when network is under failure e.g., nodes are randomly removed. Targeted attack which removes largest nodes will lower the performance much faster. In randomly generated Gnutella network this is just opposite because all the nodes have quite equal number of links/neighbors [19, p.219-241]. In such network targeted attack and random failure scenarios have quite similar behavior resulting both in a critical point where network divides into disjoint clusters [4].

Both decentralized systems have advantages. Gnutella is simple and easy to imple-

ment, it has powerful search which locates all the queried resources if time-to-live value is set high enough. Backside is that these algorithms will flood the whole network full of messages and eventually prevents the network from functioning. Freenet has more complicated, dynamic structure which allows the search algorithm to scale very well. However, users of Freenet are forced to store anonymous data which does not actually belong to them, because it is stored as requests go by. Also, files that are not requested are eventually removed from the Freenet [19, p.385].

Later in this thesis new search algorithm, NeuroSearch, is introduced. This algorithm uses computational intelligence to solve message routing problem. It is more like Gnutella than Freenet because there is no hash values for data, network is message based and it is unstructured while decentralized. However, this algorithm solves message routing problem more intelligently than Gnutella. It chooses, based on local knowledge to which neighbors it will route the message. So, NeuroSearch takes good sides of BFS and DFS, it is unstructured, it can spread like BFS but it can also route messages like DFS. It locates resources using fewer packets and thus spares network bandwidth significantly.

# 3 Neural Networks

In this chapter history of Artificial Neural Networks (ANN) is described. Architecture of one of the most common neural network, Feed forward neural network (FFNN) is shown, and training methods are analyzed.

## 3.1 Brief History and Introduction

The Brain is complex computer capable to perform task such as pattern recognition, perception and motor control whereas an Artificial Neural Network is crude model of brain. An Artificial Neuron (AN) is a model of biological neuron (BN) which is a complicated system. BN has myriad of parts, sub-systems and control mechanisms. There are hundreds of different BNs. Together these neurons and their connections form process which is not binary, not stable and not synchronous. An ANN tries to replicate most basic systems of the brain [8].

Modern theories of neural processing and development of digital computer occured at the same time, during the late 1940s. Development of artificial neural networks is inspired by understanding of animal brain. At late 1950s first artificial neural networks appeared. Frank Rosenblatt developed *perceptron*, father of the complex neural networks used nowadays. *Perceptron* had input layer and output layer and as an oldest neural network it is still in use. Bernard Widrow developed *adaline* and *madaline*. *Madaline* was first neural network which was used to solve real world problem [8].

In 1969 Minsky and Papert published book called *Perceptrons* [17] where they concluded that *perceptron*, developed by Rosenblatt, could not solve even simple problems. Their conclusion was that "despite the fact that perceptrons were interesting to study, ultimately perceptrons and their possible extensions were a 'sterile' direction of research" [25]. This caused major set-back to ANN development. In 1980s the research of NNs started again and is today one of the largest research areas in computer science [9, p.12].

### 3.1.1 Artificial Neuron

An AN is a nonlinear mapping $F_{AN}$ from $R^I$ to $O$ where $O$ is value depending on the activation function used. I is the number of input signals to the AN.

$$F_{AN} : R^I \rightarrow O$$

Each input signal $x_i$ is modified by weight $w_i$ to strength or deplete the input signal. The AN sums all incoming modified signals.

$$net_S = \sum_{i=1}^{I} x_i w_i$$

The AN that computes weighted sum is referred to *summation unit* (SU) (see figure 3.1). Alternatively *product unit* (PU) can be used. Product units provide higher-order combinations of inputs and so have increased information capacity [9, p.18]. Product units are not subject of this thesis.

Net value is modified by threshold value $\theta$, also referred as *bias*. Threshold value can be any real value which is added to total or it can be input/weight pair where $x_0 = 1$ and $w_{th}$ is the weight value for bias. Without bias the value of neuron is zero if all inputs are zero. This means that hyperplane that neuron produces in space has to pass through zero. In many problems it would be much useful that hyperplane could be elsewhere [3].

After calculation of the net sum activation function $F_{AN}$ is used (also called transfer function) to compute the output signal $y$. Any activation function may be used. Frequently used functions are Linear function, Step function (Threshold function), Ramp function, Sigmoid function, Hyberbolic tangent function and Gaussian function [9, p.19]. When using threshold activation function the AN (also called as *Threshold Logic Unit*) will fire when *net value* e.g., sum or product of the modified input signals, is larger than threshold value:

$$f_{AN}(net - \theta) = \begin{cases} a & \text{if net} \geq \theta \\ b & \text{if net} < \theta \end{cases}$$

where $a$ is usually 1 and $b$ is 0.

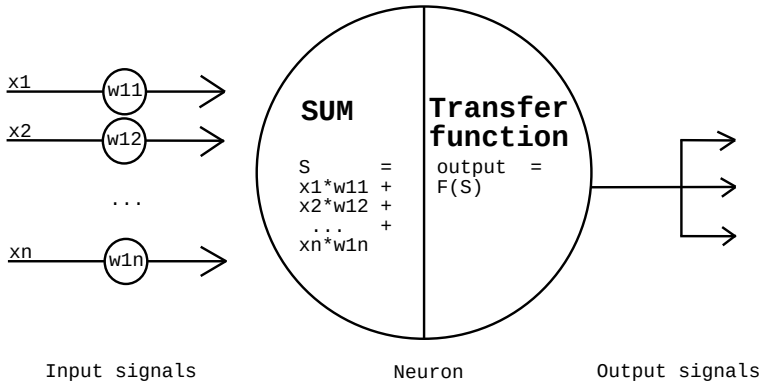


Figure 3.1: Internal representation of one neuron. After weight values and connection values are multiplied these values are sent to neuron. Neuron adds all these values together and uses this new value on transfer function which will then generate the output of the neuron.

Sigmoid is often used as activation function because it is continuous, monotonous and differential but still quite linear in presense of origo. This means that there is no point in the function output that would not react to changes in weights or input values. This makes teaching neural networks easier.

$$f(u) = (1 + e^{-u})^{-1}$$

The final output signal O of the AN is

$$O = f_{AN}(\sum_{i=1}^{I} x_i w_i + x_{I+1} w_{I+1}),$$

where $f_{AN}$ is the chosen activation function, I is the number of signals, $x_i$ is the input signal, $w_i$ is the corresponding weight and $x_{I+1} w_{I+1} = \theta$. $x_{I+1}$ is referred to *bias unit* and it is usually 1 or -1. $w_{I+1}$ is adaptive weight for bias and as all weights, it also will change when neuron is trained [9, p.20-22].

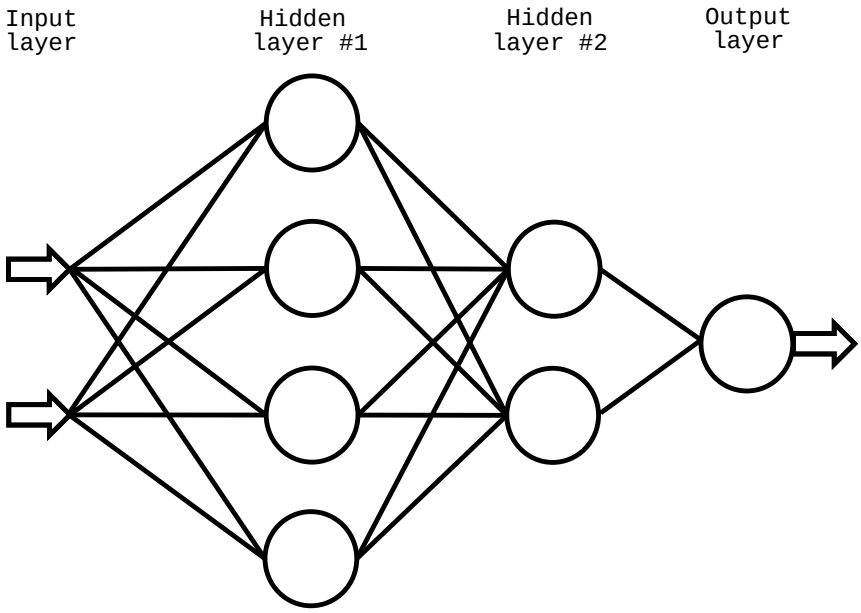


Figure 3.2: Internal representation of feedforward neural network which has 2 hidden layers. Neural network is calculated from left to right. First layer is input layer where the raw data is located. Input values are connected to first hidden layer. All connections are weighted connections. Neurons on first hidden layer are connected to second hidden layer which in turn are connected to output neuron. After all calculations are made the output neuron will reveal final output.

## 3.2 Feed-Forward Neural Networks

Feed-forward neural network or Multi-layer Perceptron (MLP) contains neurons assigned to layers. Between neurons there are weighted connections. Neurons are connected to neurons on another layer but they are never connected to neurons on the same layer [9]. Internal representation of Neural Network is described in figure 3.2.

First layer of the neural network is input layer and it contains all the input values that are used for calculation. Last layer is output layer and it contains one or more output neurons which specifies the result. All the remaining layers are named as hidden layers. Usually there is one or two hidden layers in a neural network. Main idea is to use as few neurons as possible but no fewer to minimize CPU time usage consumed by calculations.

Feed forward neural network is calculated through by summing all incoming modified connections in each neuron. This sum is handled as previously described. After all layers have been computed through the final output value can be read from output layer. Output value can also be modified by sigmoid or any other function

$$Output = f(net),$$

where $f$ is the activation function used in output neuron.

FFNN with one hidden layer, multiple output units and multiple input patterns is formulated as

$$O_{k,p} = f_{o_k}(\sum_{j=1}^{J+1} w_{kj} f_{y_j}(\sum_{i=1}^{I+1} v_{ji} z_{i,p})),$$

where $z_p$ is input pattern and $z_{i,p}$ is $i$'s input value. $v_{ji}$ is the weight value for connection between input value $i$ and hidden unit $j$. $I$ is the amount of input units, $I+1$ is bias and thus $v_{j(i+1)}$ is weight value for bias. $f_{y_j}$ and $f_{o_k}$ are activation functions for hidden unit $y_j$ and output unit $o_k$. $w_{kj}$ is a weight value between hidden unit $j$ and output unit $k$. $J$ is the amount of hidden units.

## 3.3 Training Neural Networks

### 3.3.1 Supervised Learning

In supervised learning the input values and output values are known. This means that for specific inputs we are expecting specific output also. Training means that if the output differs from desired value neural network's weight values are adjusted. Error value between desired output and neural network's output is counted and weights are correspondingly adjusted until a certain stop criteria is satisfied. Supervised learning is mostly used for training Feed forward neural networks [9, p.27-54].

It is crucial in this learning method that desired output is known. In NeuroSearch the decision to send or reject packet in certain situation in P2P-Network is unknown beforehand. If the node decides not to send query further there is no way to tell if this decision is good or bad because it depends on the decision made by other nodes. So, this kind of training is not possible for neural network based resource query algorithms until we have good resource discovery algorithm to use as training data for supervised learning method.

### 3.3.2 Other Learning Methods

Unsupervised learning is learning method that requires no supervisor or external guidance. Objective of unsupervised learning is to find specific patterns from input data by performing clustering of input space. Reinforcement Learning in the other hand is based on psychology of animal learning. Neural network is awarded for correct actions and punished for wrong actions [9, p.55,56,79]. These learning methods are not subject of this thesis.

Neural network used in NeuroSearch is trained using Evolutionary Computing. This method is explained in next chapter.

# 4 Evolutionary Computing

In this chapter Evolutionary Computing (EC) is described and some Evolutionary Algorithms (EA) are revealed.

## 4.1 Introduction to Evolutionary Computing

*"THE AFFINITIES of all the beings of the same class have sometimes been represented by a great tree. I believe this simile largely speaks the truth. The green and budding twigs may represent existing species; and those produced during each former year may represent the long succession of extinct species... The limbs divided into great branches, and these into lesser and lesser branches, were themselves once, when the tree was small, budding twigs; and this connexion of the former and present buds by ramifying branches may well represent the classification of all extinct and living species in groups subordinate to groups... From the first growth of the tree, many a limb and branch has decayed and dropped off, and these lost branches of various sizes may represent those whole orders, families, and genera which have now no living representatives, and which are known to us only from having been found in a fossil state... As buds give rise by growth to fresh buds, and these, if vigorous, branch out and overtop on all a feebler branch, so by generation I believe it has been with the Tree of Life, which fills with its dead and broken branches the crust of the earth, and covers the surface with its ever branching and beautiful ramifications." (Darwin, 1859)*

Because the environment of the world is constantly changing the individuals habiting it need to adapt dynamic conditions. Failure in this race means that eventually all individuals will die and species extinct. Those individuals that survive will form the species that carry on life to next generations, to future. Evolution is the process to improve survival capabilities of the life on space.

Evolutionary Computing (EC) is the emulation of this process. It is an emulation of natural selection, survival of the fittest, reproduction and mutation. Evolutionary Algorithm (EA) is a search for an optimal solution to a given problem e.g., how the life

can survive as time goes by and environments on earth changes or how neural network can learn to approximate given function [9, p.124].

Evolutionary Algorithm's main components are chromosomes, fitness function, initial population, selection operators and reproduction.

### 4.1.1 Chromosome

Each individual in population is candidate solution and it has charasteristics which are represented by chromosome. Chromosome determines how well the individual solves the given problem. A chromosome consists of a number of genes. Each gene represents one charasteristic of the individual e.g. each gene represents one parameter of the optimization problem [9, p.125].

### 4.1.2 Fitness Function

Fitness function tests how well the individual e.g. chromosome can solve the problem. It maps chromosome representation into a scalar value:

$$F_{EA} : C^I \rightarrow \Re,$$

where $F_{EA}$ is fitness function and $C^I$ is $I$-dimensional chromosome. Because fitness value informs the quality of the solution e.g. individual it is important that this function models the optimization problem. Twisted fitness function will give twisted solutions [9, p.125].

### 4.1.3 Initial Population

Evolution needs some population of individuals to begin. If the initial population is small it covers small part of the search space. Too large population will need more time on each generation but it also covers more search space [9, p.126].

### 4.1.4 Selection Operators

At each generation EA produces new set of individuals. New generation is reproduced by individuals on earlier generation. Selection operator finds those individuals on population that may reproduce. Different operators are random selection, proportional selection, tournament selection, rank-based selection and elitism [9, p.126-129].

### 4.1.5 Reproduction Operators

Reproduction operators produce new offsprings from previously selected individuals. This is done either by cross-over e.g., creating new individual by combining genes between parent chromosomes or mutation e.g., creating new individual by altering genes of parent chromosome [9, p.130].

## 4.2 Cross-over

Cross-over method produces new individuals from two existing inviduals. It is a way to mix genes of two chromosomes to make one. In population cross-over happens on certain probability. In real world mating partners are chosen by some criteria, weakest individuals are closed out to prevent weak gene transfer to the future. Pseudocode algorithm to illustrate cross-over between individuals $C_{n1}$ and $C_{n2}$ is shown below [9, p.137-138]:

1. Randomly determine if cross-over takes place. If no, parents are returned. Else continue.
2. Copy the genes of parents to offspring e.g. $\alpha = C_{n1}$ and $\beta = C_{n2}$.
3. Compute mask, $m$ which specifies which genes of the parents should be swapped. Several cross-over operators have been developed: uniform cross-over, one-point cross-over and two-point cross-over.
4. For $i = 1, ..., I$ if $m_i = 1$ swap genes. $\alpha_i = C_{n2,i}$ and $\beta_i = C_{n1,i}$.
5. Return $\alpha$ and $\beta$.

## 4.3 Mutation

Mutation changes genes of an individual and thus brings new genetic material to population. Mutation occurs with certain probability, referred to *mutation rate*. Mutation rate should be larger when evolution starts to cover larger area of search space. When better solution candidates are found the mutation rate should be decreased and when candidates are approaching the optimal mutation rate should be significantly smaller [9, p.138].

## 4.4 Evolutionary Algorithms

Genetic Algorithms (GA) model genetic evolution. Offsprings are generated by crossover and mutation. GAs may be evolved using more than one population at the same time. Selection and reproduction may occur between subpopulations. Individuals can also immigrate other populations [9, p.133].

Genetic Programming is specialization of genetic algorithms. GP is also a model of genetic evolution where GP represents individuals as executable programs constructed as trees [9, p.147].

Evolutionary Programming (EP) models behavioral evolution and genetic evolution. In reproduction only mutation is used [9, p.155].

Evolutionary Strategies (ES) is evolution of evolution. Each individual is represented as genetic material and set of strategy parameters. Changes due the mutation are only accepted in case of success e.g. if mutated individual has improved fitness value [9, p.161].

Differential Evolution (DE) is similar to basic EAs but differs in reproduction. DE introduces new way to mutate offspring. DE uses arithmetic operator which depends on the differences between randomly selected individuals. If mutation occurs the offspring contains combination of genes of three different chromosomes [9, p.167].

EP methods are used in training of NeuroSearch added with strategy parameter as in ES.

# 5 NeuroSearch

In this chapter the NeuroSearch search algorithm is described.

## 5.1 Introduction

NeuroSearch is intelligent resource query algorithm. Every peer in the P2P-Network will decide based on local information if it will send query to its neighbor. Local information contains such elements as how long the message has travelled and how many neighbors the target neighbor has. As a total of six different local information pieces are used to make routing decision. Peer may want to send query further if it has travelled only few hops or it may drop the query even if it has travelled few hops but the target has too few neighbors. The answer for the problem - should the peer send it or not is revealed when input values are computed through neural network. Good neural network will make wise decisions and so lowers the bandwidth usage on P2P-Network while maximises the probability of finding the wanted resources. Neural network cannot make good decisions if it is not trained properly. NeuroSearch is trained using evolutionary programming and evolutionary strategies, described earlier.

## 5.2 Inner Structure

NeuroSearch is feedforward neural network (FFNN). It has input layer, one or more hidden layers and output layer. Different amount of neurons are used and their behavior and success are analyzed. If neural network is too large it consumes more CPU time for calculations and it may overlearn the training environment e.g., P2P-Network. NeuroSearch differs from all other search algorithms used in P2P-Networks because it can adapt its functionality by combining multiple input types to make a wise routing decision.

## 5.3 Input Values

Six input values are used:

- *hops* is the number of links the message has gone this far. Neural network can decide to drop packets which have gone too long or maybe send those which have been routed only few times.
- *neighbors* is the amount of neighbor nodes this node has. It can decide to send or drop message by knowing whether it is located in center or in the edge of P2P-Network.
- *target's neighbors* is the amount of neighbor nodes the message's target has. This tells to the neural network if the target is central or edge node.
- *neighbor rank* tells target's neighbor amount related to this node's other neighbors e.g., if this value is 0 this target node is the node which has more neighbors than any other neighbor.
- *sent* is a flag for telling if this message has already been forwarded to the target node by this node.
- *received* is also a flag and will be 1 if this node got this message from target node.

Input values are scaled so that all inputs are between 0 and 1. *Sent* and *received* are either 0 or 1. *Hops* and *neighbor rank* are scaled with:

$$f(x) = \frac{1}{x+1}, \tag{5.1}$$

where $x \geq 0$.

*Neighbors* and *target's neighbors* are scaled with

$$f(x) = 1 - \frac{1}{x}, \tag{5.2}$$

where $x \geq 1$. *Neighbors* and *target's neighbors* are always over 1 and thus they can be dividers.

NeuroSearch can imitate BFS if all other inputs are dropped but *hops* and *received* which informs if the query came from this target node. So, best neural networks with only this input and bias will send the query further but will not send it to the same target where it came from. NeuroSearch with inputs *hops*, and *random value* will imitate Random Walker algorithm. This NeuroSearch will send or drop query based on

random value but also on hop count. If NeuroSearch is filled with inputs containing heuristic information of the network i.e. P2P-Node reply rates or average latency of the replies it may imitate Directed BFS algorithm [27].

## 5.4 Activation Functions

*Hyperbolic tangent* (Tanh) is used as an activation function in the neurons on hidden layers:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1,$$

where $x$ is the *net* sum of all input values to this neuron

$$net = \sum_{i=1}^{I} w_i z_i,$$

where I is total number of connections to this neuron. $w_i$ is the weight value for input value $z_i$. The output of hyperbolic tangent is between -1 and 1.

## 5.5 Training

The idea of training is to find correct weight values for neural network. Evolutionary Programming and Evolutionary Strategies are used to train NeuroSearch. Training steps are the following:

1. Load P2P-Network from the file.
2. Set initial population of ANN. Every individual in the population is prepared by randomly setting weight values and self adaptive parameters. Weight values vary randomly between values -0.2 and 0.2 and self adaptive parameters is set to 0.05.
3. Test the ANN-population in P2P-Network to obtain fitness value for each ANN.
   - Take ANN from population
   - Multiple times choose randomly one peer and one resource and execute query in P2P-Network starting from the selected peer. Determine fitness by counting how many resources are found and how many packets were used:

$$fitness_{ANN} = \sum_{j=1}^{n} score_j,$$

where $n$ is the number of resource queries. Score is calculated using following principles.

$$score = \begin{cases} Min(R,T) * R_{value} - P & R > 0, T > 0, R_{value} > 0, P > 0 \\ 1 - \frac{1}{P+1} & if R = 0 \\ 0 & if P > P_{limit} \end{cases}$$

where $R$ is found resources, $P$ is used packets used in query, $T$ is target amount of resources e.g., half of the available resources, $R_{value}$ tells how much one found resource is worth compared to packets e.g., every resource is worth of 50 points as every consumed packet is one minus point and $P_{limit}$ is maximum amount of packets that may be consumed in one query.

First formulae tells that score is counted by taking founded resources, multiplying them by some number and decreasing from this number the amount of used packets. If more than limit amount of resources are found the limit amount is used. This way NeuroSearch gets best score by founding wanted amount of resources and using minimal amount of packets.

Second formulae states that score is very small if no resources are found, however using fewer packets NeuroSearch gets better fitness - even very small one.

Third one states that if amount of used packets grow over packet limit the algorithm floods the network and its score is thus set to 0.

- If all individuals have been tested go further, otherwise go back and test new ANN.

4. Use selection methods to separate the most fit individuals from the population to breed new generation. Remove weakest half of the population.

5. Reproduce new half by cloning and mutating each survived network. Descendant is an ANN that has slightly changed weight values than its parent. Mutation is done by random variation using normal distribution. Scaling factor is used for each weight value. Scaling factor will determine how much the weight is being changed [10]. At the beginning mutation should be aggressive but when the

fitness grows and algorithm comes closer to the optimal solution the mutation should be very small. The random variation functions are given as:

$$\sigma'(j) = \sigma(j)exp(\tau N_j(0,1)), j = 1, ..., N_w$$

$$w_i'(j) = w_i(j) + \sigma'(j)N_j(0,1), j = 1, ..., N_w$$

Finally we have exactly the same amount of neural networks than initially.

6. Increase generation amount by one. Stop if chosen generation time has been reached, else go back to step 3.

After the process the best neural network is found e.g., neural network with highest fitness score. Behavior of this ANN may be analysed further with NeuralSee and P2PStudio programs.

# 6 Research Environment

In this chapter is reported the tools with what this study is made. Computer programs are shown and their function is explained. Computers used is listed with statistic and network used in calculations is explained.

## 6.1 P2PRealm

P2PRealm - the simulator of P2P-Environment as well as generator of NeuroSearch algorithm is the main program used in this thesis. P2PRealm generates virtual P2P-Network, initiates chosen neural network population, tests and mutates neural networks, writes statistics of the process and prints specific files for P2PStudio.

P2PRealm may be connected by telnet and commands to modify running process may be given. Software has advanced configuration system for running multiple research cases and the computing may be distributed so that processing slave computers do not use any storage space - all generated data is transferred back to the master computer by network. The distribution of P2PRealm is built upon Chedar P2P-Platform [5]. These features are crucial because evolutionary programming consumes huge amount of computing power and thus time. Configuration system allows user to set up all needed cases at once for processing and thus computing is uninterrupted. Distributed system allows these cases to be computed on separate computers decreasing computing time.

P2PRealm is coded with Java 1.4.1 and can be run both on Unix or Windows machines. P2PRealm is programmed by Niko Kotilainen and Joni Töyrylä.

### 6.1.1 Java Classes

Netsimulator.Java is the main class of the program. Simulation variables is read from config file and evaluator is created with these parameters.

Evaluator.Java is the component for running training of neural network. NeuroSearch-

Generator and Network are initialized. After this the main loop of simulation is started i.e., generation loop. In each generation all neural networks from population are tested and then variated.

NeuroSearchGenerator.Java creates and holds the population of neural networks. It contains methods for calculating average fitness of population and finding neural network with best fitness value.

NeuralNet.Java is the class of neural network. It contains weight values, scaling factors and method for weight variation as well as used packets and found resources in last generation. Method `private void variateGaussian(NeuralNet father, int variationRound)` is used to variate neural network.

NeuroSearch.Java is the class for making decision should NeuroSearch send packet further or not. Method `public static Vector handleMessage(Node node, Message msg, Vector decisions)` is called from every P2P-Node when decision is needed to be made. `public static double calculateOutput(double data[], double weights[])` is the method for calculating neural network output. Data contains scaled input data and weights are the array of NN weights.

Network.Java is the class for P2P-network. It contains Node, queried resource and querier arrays and main method for calculating the route of a query. It also loads the network from file or creates it dynamically.

Node.Java is the class for one node in P2P-network. It contains array of its neighbors and resources.

Message.Java is the class of the routed message. It contains information of weight values of corresponding algorithm, amount of hops used and from what node it came from.

## 6.2 P2PStudio

P2PStudio was created for analyzing query routing in P2P-Networks. With P2PStudio researchers may see the graphical representation of P2P-Network - nodes and links but also the resource query routes for the loaded NeuroSearch. This is useful when analyz-

ing behavior of NeuroSearch. For example is it acting like Breadth-First-Search or is it more like Depth-First-Search. P2PStudio can be used in real situation with TCP/IP to monitor Chedar network or it can be fed with proper data and used offline. In figure 6.1 the interface of the P2PStudio and topology of P2P-Network used in all research cases in this thesis are shown.

First version of P2PStudio was created in student project of University of Jyväskylä in autumn 2002. Further development of the software has been done by Niko Kotilainen.

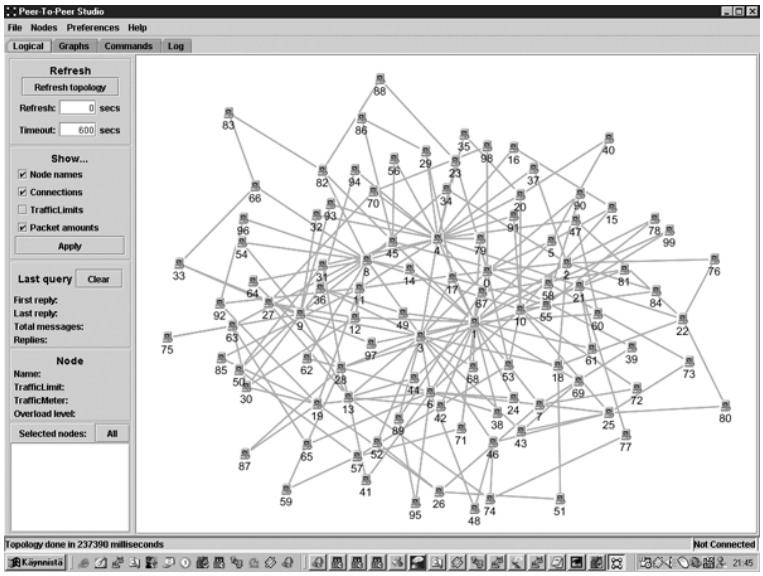


Figure 6.1: Interface of the P2PStudio

## 6.3 Internal Representation of P2P-Network and Queries

P2P-Network used in simulations is created using Barabasi-Albert model [6] and thus is power-law distributed. Every node has as many resources as it has neighbors - central nodes have more resources than nodes on the edge. Parameters of the used network are:

- 100 peers
- 394 links
- 394 resources

- Highest degree node has 25 resources and links
- Every node has at least 2 resources and links
- Connections between nodes are power-law distributed

Each NeuroSearch algorithm in certain generation is tested $n$ times in the P2P-Network. All tests are done with randomly chosen queriers to make fitness value represent overall performance in the network rather than only certain individual query patterns. The queriers are the same to make fitness values comparable between different NeuroSearch algorithms inside one generation.

# 7 Research Cases

In this chapter different research cases are described. First is researched how initial population affects the development of NeuroSearch. Secondly the size of neural network is analyzed. Third case reveals information of input values, where some inputs are more important than others and some combinations work better than others. Fourth case is about fitness function, we analyze what kind of NeuroSearches are developed with different amount of required resources to be found.

## 7.1 General Information

In all experiments, the same fixed values are used when possible. Those research cases which do not concentrate on topology will use same topology [i.e., network.xml], initial population, variation amount and neuron amounts are used. In the beginning it is not sure which values or value combinations are better than others and so at first some values need to be set more or less randomly. The following variables are used:

| Variable | Explanation |
|---|---|
| Peers | Number of P2P-Nodes in the network |
| Links | Number of links between nodes in the network |
| Resources | Number of resources in the network |
| Topology | Topology of the network |
| Variation | Variation function used when generating new weight values for new neural network |
| Fitness | Percent amount of target resources to be found in the query |
| Neurons | Neuron amount in the hidden layers of the ANN. Different layers are separated by ":" |
| Hidden activation | Activation function used in neurons on hidden layers |
| Output activation | Activation function used in output neuron |
| Inputs | Amount of inputs used |
| Generations | Amount of generations in evolution |
| Population | Amount of neural networks in the population |

In almost all graphs there are fitness values of best neural network and average fitness of better half of the population in the last generation [1]. Hops and Age is shown same way e.g., amount of best evolved ANN and average value of the better half of the last generation are shown in the graph.

Most of the figures presented in these cases are made so that curves are an average of 50 last generations. Without this moving average the curve would be too messy to be analysed. If other average value is used it is written in the caption.

### 7.1.1 Fitness

Fitness value determines how good the neural network is compared to others. Even smallest and simplest neural networks manage to have fitness value over 10000. In table 7.1 are shown statistics of poor and good neural network. The poor network founds 239 resources after 50 queries and consumes total of 1290 packets. Because

[1] Population of 30 neural networks will have after every generation 15 networks which have proven to be better than other half. These 15 are mutated to get new breed of neural networks which have proven nothing, there is a high probability that these 15 will not succeed. This is the reason why average fitness is calculated from better half of neural networks

fitness is calculated so that each found resource is worth 50 points to total score minus used packets, we can calculate the fitness value $239 * 50 - 1290 = 10660$. This neural network finds in average $239/50 = 4.78$ resources every query and consumes $1290/50 = 25.8$ packets for locating them. This is not very good algorithm though. Better algorithms are listed below this one - good NeuroSearch, Highest Degree Search and steiner Algorithm. Steiner algorithm finds the best possible search path for given querying node to locate specific resource in that P2P-Environment but it uses global knowledge to solve problem and therefore cannot be used as local search algorithm. Efficiency is counted by dividing replies by packets.

| Name | Fitness | Fitness/Query | Packets | Replies | Efficiency |
|---|---|---|---|---|---|
| Poor NS | 10660.0 | 213.2 | 1290 | 239 | 0.19 |
| Good NS | 17905.0 | 358.1 | 1995 | 398 | 0.20 |
| HDS | 18891.0 | 377.0 | 1209 | 402 | 0.33 |
| Steiner | 18371.0 | 366.4 | 379 | 375 | 0.99 |

Table 7.1: Statistic of different algorithms. As can be seen, steiner algorithm where global knowledge is used has enormous advantage compared to rest of the algorithms.

## 7.2 Population

### 7.2.1 Setup

Initial population is the first research case because the results may prove better and makes it faster to get results for later cases. If the amount of ANN is too small it may lose information in the training and if the amount is too large there may be unnecessary computing when variating new breed of ANNs.

In this case the values that were changed were ANN amount and generation amount. Generation amount is changed to set variation amount to be the same for all ANN amounts. The amount of neural nets and generations used in this case are shown in table 7.2.1. Only half of the population is variated, so total variation amount is counted as:

$$(initialpopulation/2) * generationamount$$

| Neural Nets | Generations | Total Variations |
|---|---|---|
| 2 | 300000 | 300000 |
| 16 | 37500 | 300000 |
| 30 | 20000 | 300000 |

Table 7.2: The amount of neural nets, generations and variations used in this case

Variables used in this case are:

- 100 peers
- 394 links
- 394 resources
- Power-Law distributed topology
- gaussian variation
- 50 percentage of resources
- 16:4 neurons
- sigmoid hidden activation
- threshold output activation
- 6 inputs

### 7.2.2 Results

In figure 7.1 the population contains only 2 neural networks which is the minimum amount of population. It can easily be seen that the evolution of NeuroSearch is not stable. It may sometimes find reasonable fitnesses but in general its performance is not admirable because the fitness drops multiple times during the training. There are huge leaps in the fitness values. These leaps mean that population had got some intelligence about solving the problem but it was lost. This happens because population size is so small that it cannot store the level of performance. Because P2P-Nodes starting the queries are randomly selected there is possibility that those nodes are very bad nodes for starting to find resources e.g., they may be on the edge of the network with few

links. This may give a chance to weaker networks to get slightly better fitnesses. Also, if the query originators are set so that finding resources is very easy also poor networks manage to attain good fitness values. This can be crucial, stable network (e.g., a neural network that keeps fitness values inside about 800 points range) can be dropped by network which counts only on luck.

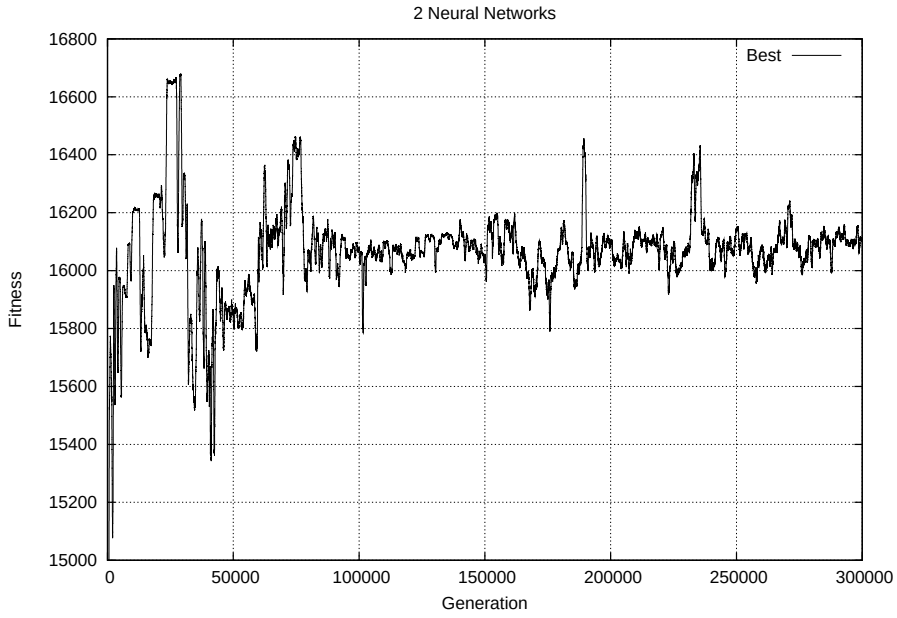


Figure 7.1: Evolution of 2 Neural Networks and 300000 generations. There may appear nets which contains more intelligence but because population is too small it cannot hold the information. The curve is calculated as an average of 500 last generations.

Because the queriers are randomly chosen the querier may be on the edge of the network or in the center. Query started from the edge is different than query started from center, usually is thought that queries started from edge are "hard" problems while central queries are "easy" problems because most of the resources are located in the centre of the P2P-Network. This randomness leads to situation that some tests are easier to neural network to solve than others and because of this the difference between fitnesses of neural networks could be very small. The difference between fitnesses can be only few packets after 50 resource queries.

However, even if fitness difference is very small, the better network will survive and live to the next generation. Down leap will come when weak survivor cannot maintain the fitness. It has no proper intelligence to solve resource discovery problem in all ran-

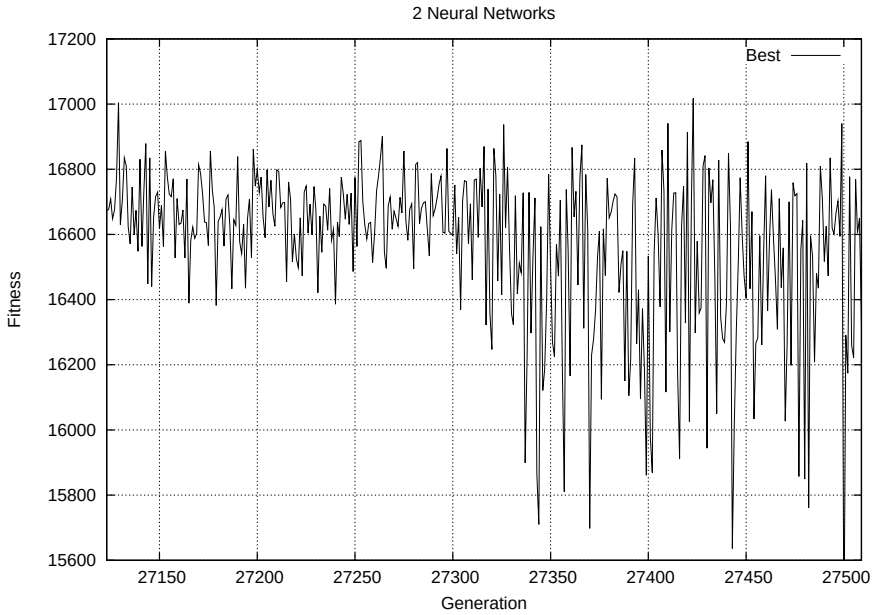


Figure 7.2: Two different behaviors of neural networks. First half of the graph demonsrates stable network and second half shows unstable network. Stable network tries to do good job in every environment, unstable tries to do good job when environment is easy for it but drops down when the environment is difficult. Stable network is beaten up by unstable at generation 27316. Stable network got fitness value of 16738 and unstable got 16870.

domly chosen situations, and thus the fitness will drop. It manages to get good fitness time to time but as often it performs very poorly. Because fitness curve is average value those poor values will push the curve down.

Even though the initial population was smallest possible the evolution could reach over 16000 fitness value and it did not drop down from there. It managed to find some intelligence and could keep that information but it could not reach higher. Every time it found new solution it also lost it.

It seems that on some level the life of the neural network is dangerous. In a figure 7.2 we can see a more close situation of the evolution of 2 neural networks. This is the same situation as shown in figure 7.1 at generations 27000-27500. There is huge rising of the fitness and few thousands generations of stableness. This is second highest fitness value over the whole evolution. But what happened after that? Terrible downhill. Stable network is beaten by unstable network. Unstable network behavior differs greatly from stable network. Unstable gets sometimes good values but sometimes very poor values depending on the environment e.g., on the nodes that send resource queries.

In the figure 7.3 fitness curve of 16 neural networks population is presented and it is shown that it is capable of keeping the reached intelligence much better. Fitness droppings caused by small population do not exist. Minor varying of the curve is caused by different environment and different behavior of search algorithm. Environment can be easy or hard and some networks are more stable than others. Stable network is generally better than unstable because it can get reasonable fitness in all different situations.

At time of 20000 generations there is major rising of the fitness curve. This is normal in the training and will eventually happen to all neural networks when there is reasonable amount of generations. This means that after random amount of generations a new neural network will evolve that can solve the problem better than its friends or its parent. With initial population amount of 16 neural networks the evolution could reach fitness of 17200 which is a lot better than what 2 neural networks can attain.

In the last figure 7.4 same curve but with population of 30 neural networks is shown. Curve is even more stable than with population of 16 neural networks. Average fitness curve follows best neural network but more farther than in figure of 16 neural networks. This population size surely can hold the information but it may also be

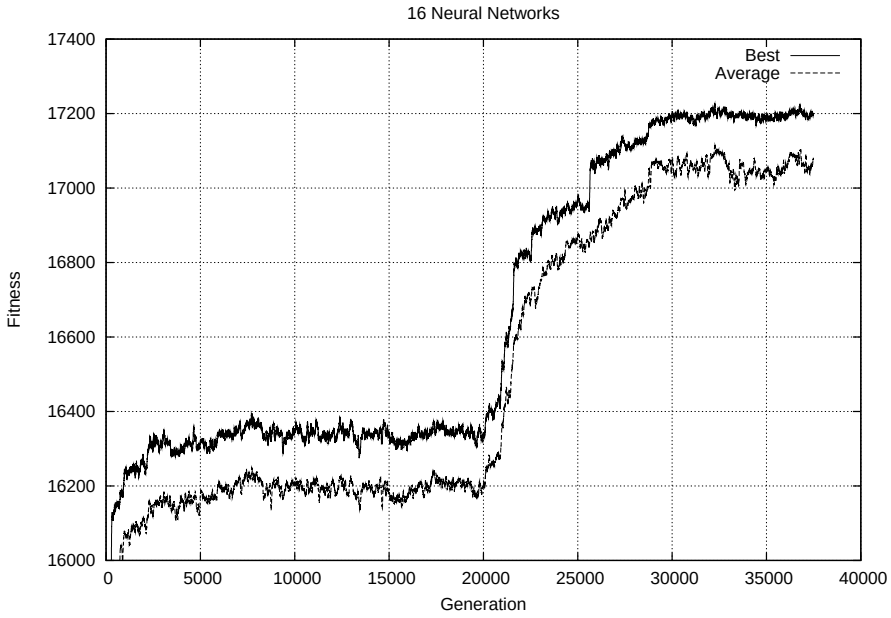


Figure 7.3: Evolution of 16 neural networks and 37500 generations. Moving average of 50 generations were used.

too big because 16 neural networks can also sustain the good solutions at reasonable probability. Larger population will use more computation time and so it is important to find population amount which can hold the information but which does not waste processing time.

### 7.2.3 Conclusion

Too small population cannot keep information and the evolution can make huge drops randomly. Very big population will keep the information but also uses much more computing time. It was shown that population of 16 networks can keep the information it has found with high probability. In later research cases initial neural network amount is risen by one third to 24 to be sure that evolution will not lose intelligence. The probability to lose best network will grow if the generation amount grows.

Also must be noticed that all evolutions could reach up to 16000 fitness values and they did not drop from there. So with this setup the fitness values below 16000 are rare as well as values above 17000. The fact that all evolutions could reach 16000 at very little time (e.g. 100-1000 generations) proves also that values near 16000 are common in fitness space. To get neural network that manages to have fitness value over

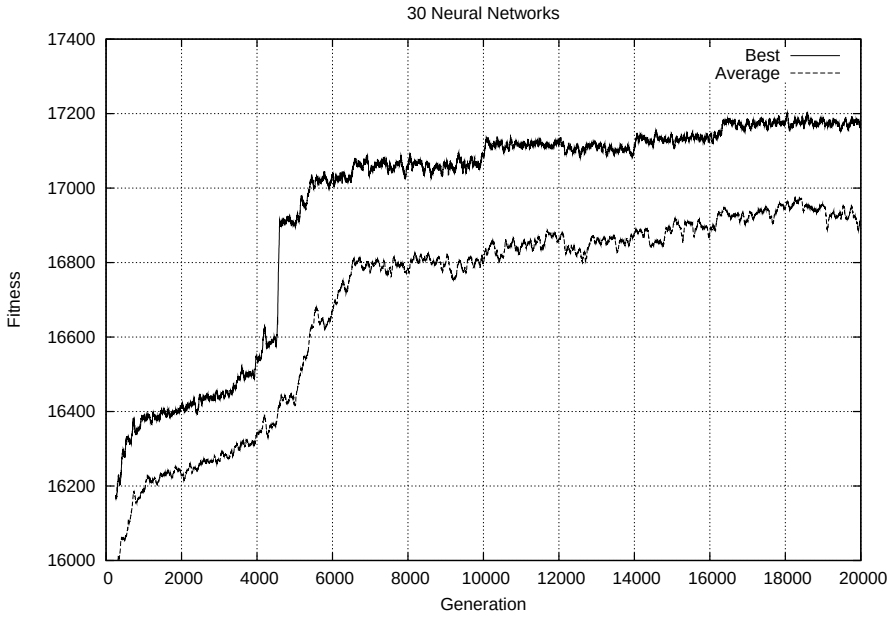


Figure 7.4: Evolution of 30 neural networks and 20000 generations. Moving average of 50 generations were used.

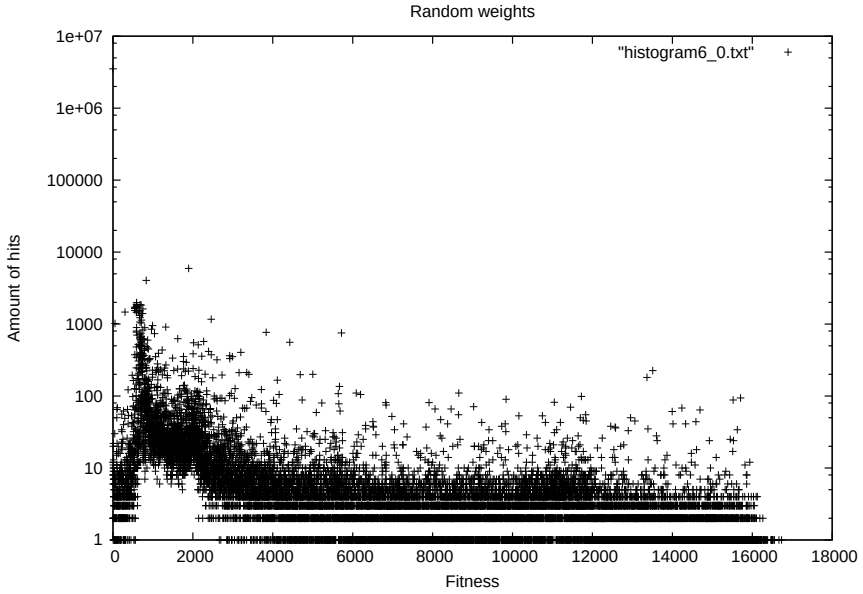


Figure 7.5: Fitness hits using neural networks with randomly chosen weight values. The value 0 has been hit $1 * 10^7$ times, other fitness values are found almost equally frequently. Fitness values near 18000 are almost impossible to find.

16000 without using evolution is very difficult. In figure 7.5 is shown fitness values of neural networks with randomly chosen weight values. Most of the random neural network have fitness value 0. Only few thousands of 10 million randomly chosen networks manage to have fitness over zero. Few hundred manages to have fitness close to 16000 and few single have fitness over 16000 and no one has near 18000. So, probability to have fitness over 16000 randomly is very small, near 0,0001 %.

## 7.3 Inputs

### 7.3.1 Setup

Neural network's input values are only way for network to get information from enviroment. In this thesis six input values are used as described earlier. In this research case every input is tested alone to get a clue how important is the input's information. Also, some input values are tested together and their performance is analyzed. In this case neuron amount was dropped to 10:5 because larger networks used too much computing power and valuable time.

Variables used in this case are:

- 100 peers
- 394 links
- 394 resources
- Power-Law distributed topology
- gaussian variation
- 50 percentage of resources to find
- 10000 generations (sent input was tested with 50000 generations)
- initial population 24
- sigmoid hidden activation
- threshold output activation
- 10:5 neurons

### 7.3.2 Results

In figure 7.6 the evolution of six different inputs is shown. Using only hops as input NeuroSearch knows the distance how long the query has travelled in the network. It may decide to drop the message if it has gone too far or it may send it further if it has gone only few hops. With this information NeuroSearch does not perform very well, its fitness remains below 13500.

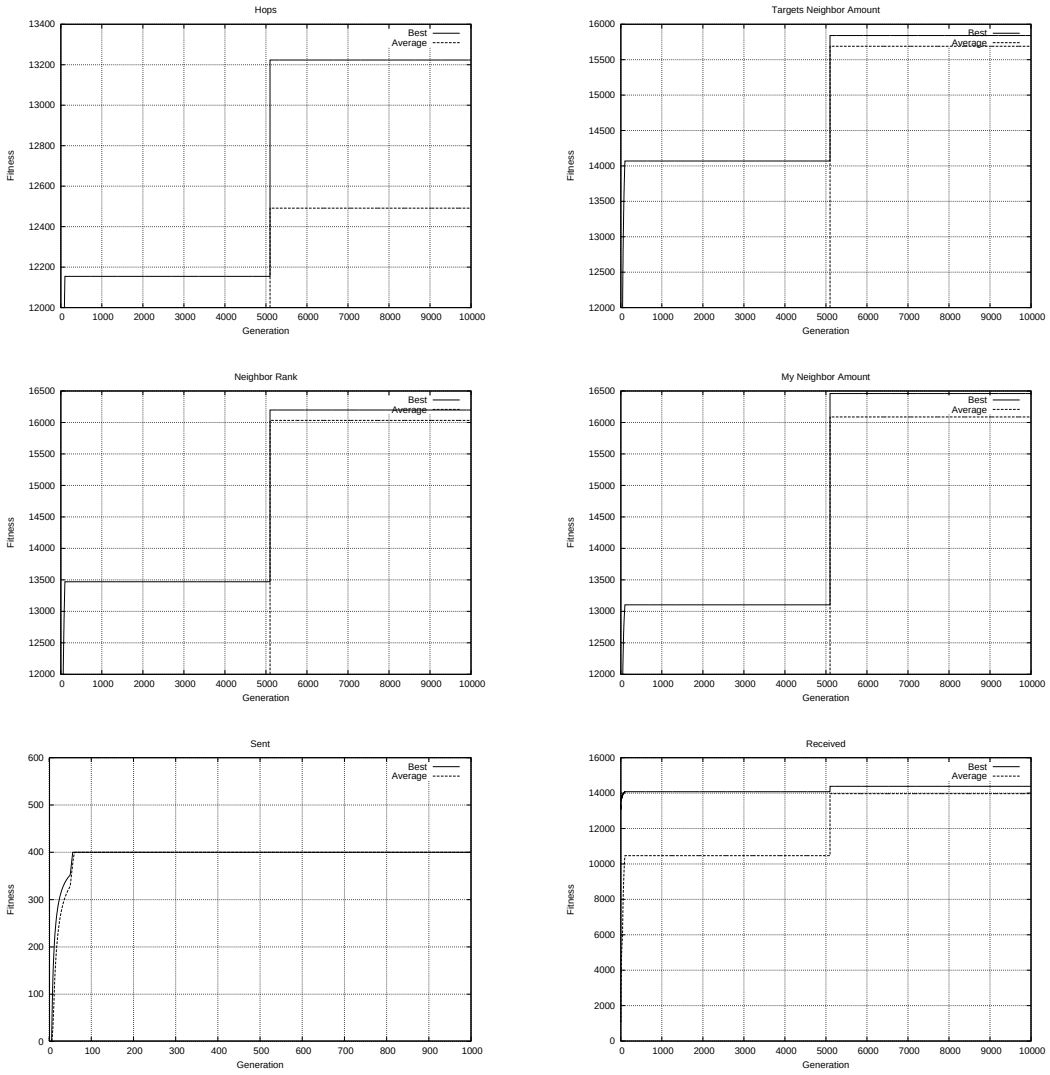


Figure 7.6: Fitness values of NeuroSearch using only one input value. Hops can reach 13K fitness but all inputs related to neighbors can reach near 16K fitness. Sent alone cannot do anything but flood the P2P-Network. Received is also able to reach fitness of 16K.

When using information of *target's neighbor amount* NeuroSearch's search capability improves significantly reaching fitness near 16000. With this input NeuroSearch knows if the target node is central node with many neighbors and thus has resources or if it is node on the edge of the network with few neighbors and resources. This information is more meaningful than information of the travelling distance because

P2P-Network used in this test is rather small, so the distance to travel to central nodes is only 2-3 hops. This may be a reason why NeuroSearch using only hops is rather poor compared to others (except *sent*).

*Neighbor rank* is counted for every target node and is calculated using target node's neighbor amount. Target node with most neighbors will have rank 0, second 1 and so on. Nodes with same amount of neighbors will get different rank numbers such the other will get better rank. This ensures that two similar neighbors do not have the same input values and thus enables NeuroSearch to send only to one neighbor. Using this information only NeuroSearch will get very good fitness value, over 16000. Fitness is even better than using exact value of neighbor amount. So, ranking the neighbors in order of their neighbor amounts is more important than counting how many neighbors they have. This may be caused by reason that many nodes have same amount of neighbors. Near the center all nodes have quite many neighbors and in the edge the nodes have few.

Using information of sender's neighbor amount, *my neighbor amount*, NeuroSearch knows how near the center e.g., well connected this node is. It does not know anything about the nodes where it is sending the query. It may think "I am central node. I have 10 neighbors - I will send the query further" or "I am node with 4 neighbors - I will reject the query". With this information NeuroSearch performs the most efficiently reaching fitness near 16500.

*Sent* input tells if this node, sender node, has already sent this message to that target node. Using this information NeuroSearch floods the P2P-Network. Every new node on query path will send the message to all its neighbors because it has not sent it earlier. So, every node in the P2P-Network will get the resource query, and all will send it at least once to all neighbors. It finds all resources but uses huge amount of packets. *Sent* reaches fitness value of 400.0 using 19700 packets and finds all 777 resources but is rewarded only from the half e.g., 402 resources.

Received input manages to do much better having fitness over 14000. Received contains the information if this node has already received the query. NeuroSearch drops query if this is not the first time when it gets the query. At start, it sends query to all its neighbors. These neighbors will send query to all neighbors also, even back to the previous senders. These previous nodes drop the queries because they know that they

already have sent the message further. Resource query is like a wave, it keeps going until all nodes have been reached but it does not use link more than two times.

In figure 7.7 three different fitness curves are shown. They all are related to neighbor amounts and describe the fitness curve of NeuroSearch with two inputs only. First two are figures with information of sender's neighbor amount and information of target's neighbors - first one with amount and second one with rank. Both of these curves are relatively equally good, NeuroSearch manages to reach fitness of near 12000. This fitness value is much smaller than with NeuroSearch using any of these inputs alone. This cannot be explained by simply saying that generation amount or neuron amount were too small because using all inputs NeuroSearch manages a lot better with same generation and neuron amount. Information of these inputs may be somehow overlapping and thus depletes the performance. Third figure is fitness curve of inputs *target's neighbor amount* and *neighbor rank* and this fitness is significantly smaller than the previous two. The information of these inputs is quite same, both contain information related to target node's neighbors. *Target's neighbor amount* contains the scaled value of neighbor amount. *Neighbor rank* contains value how many neighbors this target has related to other targets.

In figure 7.8 also 3 different fitness curves are shown. All the information in these inputs is related to query. First curve is evolution of NeuroSearch with two inputs, *Hops* and *Received*. This fitness curve is almost identical to next one, NeuroSearch with inputs *Hops* and *Sent*. Both of them manages to reach fitness value over 14000. In the last figure NeuroSearch with inputs *Received* and *Sent* are shown. Fitness of the last NeuroSearch is significantly poorer than previous two. In the last figure NeuroSearch knows only if the query has already been received by this node and has this node sent the query already to target node. There may be some overlapping information on these inputs. If the query has been in this node it is more likely sent also further. So, information got from input *received* may already contain much of the information that is contained in input *sent*. In the table 7.3 is shown relations between these inputs.

| sent | received |
|---|---|
| 1 | 1 |
| 0 | 0 or 1 |
| **received** | **sent** |
| 1 | 0 or 1 |
| 0 | 0 |

Table 7.3: Table of relations between inputs *sent* and *received*.

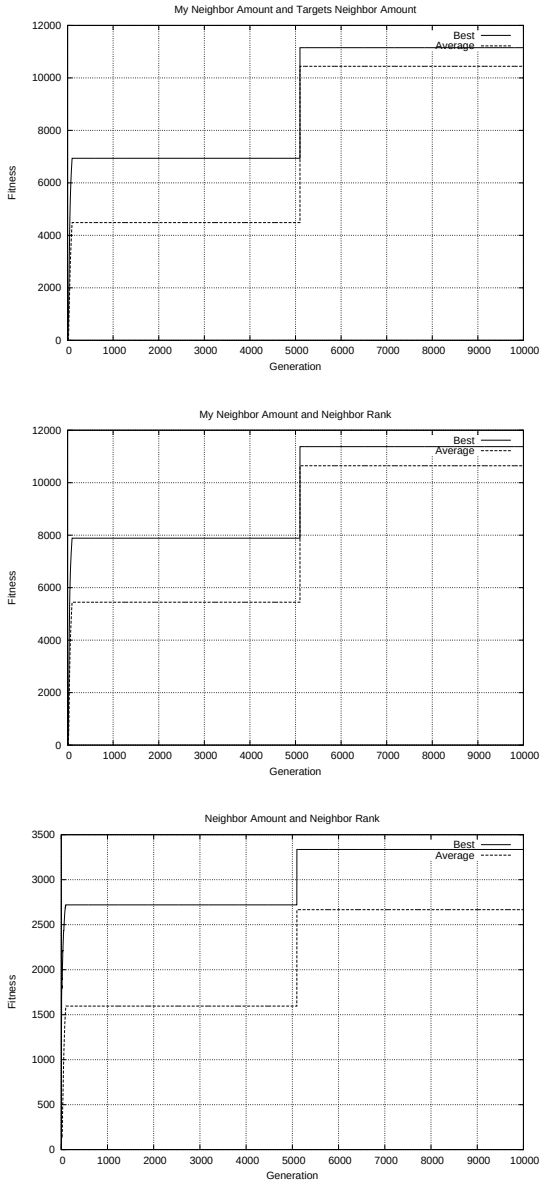


Figure 7.7: Three different fitness curves are shown. All related input values contained information on neighbors. In each figure two input values are used. First one is fitness curve of inputs *my neighbor amount* and *neighbor rank*. Second is calculated using inputs *targets neighbor amount* and *neighbor rank*. Third one is fitness curve of *target neighbor amount* and *my neighbor amount*. As can be seen the last figure is significantly poorer than previous two.

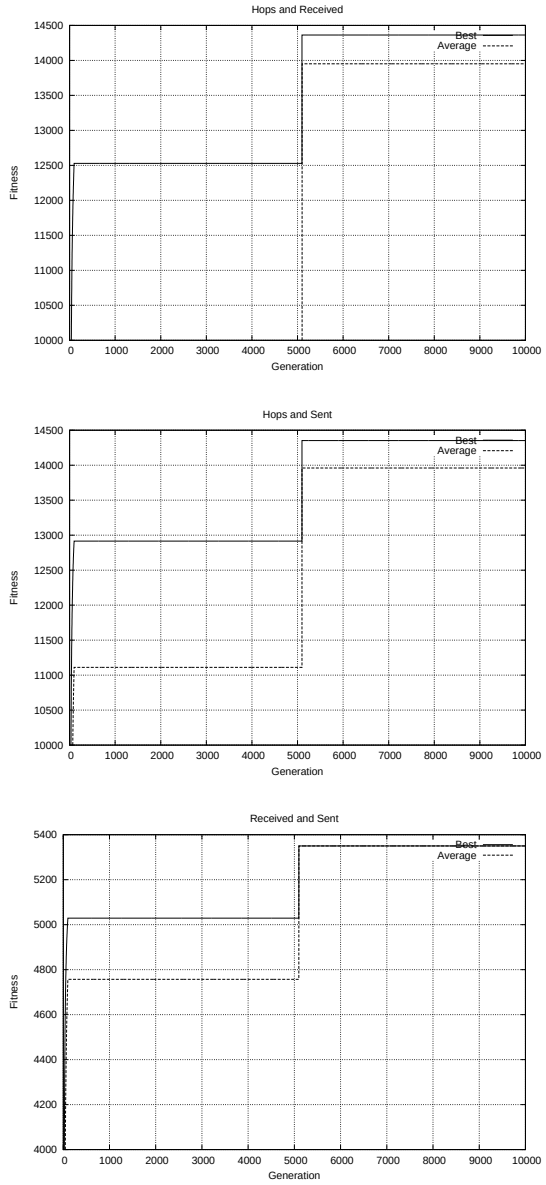


Figure 7.8: Three different fitness figures containing inputs not related to neighbors. In each figure two input values are used. *Hops* is tested with *received* and *sent* and finally *received* and *sent* are tested together. First and second fitness curves are almost identical reaching value 14000. Last one is much poorer, information contained in *sent* and *received* is not enough to make intelligent routing decisions.

### 7.3.3 Conclusion

Using inputs *hops*, *my neighbor amount*, *sent* or *received* alone the NeuroSearch will always send or reject the query for all targets of sender node. The information fed to neural network is independent of target node and thus neural network will give same output for all the targets. Target node's neighbor amount and neighbor rank of targets is related to target and thus neural network will give different output values for every target node and thus P2P-Node may send or reject queries.

It is noticeable that fitness curves of few inputs are much more straight than curves using all six input values. New generation of neural networks have almost same functionality and fitness values than their parents. Weight variation generates identical neural networks which keep getting almost same fitness values.

## 7.4 Resources

### 7.4.1 Setup

In this case is measured how much packets NeuroSearch uses when different amount of resources are needed to find. Target amount of resources varies from 10 percent to 100 percent with steps of 10 percent. Results are compared to other algorithms, BFS, HDS and steiner.

Variables used in this case are:

- 100 peers
- 394 links
- 394 resources
- Power-Law distributed topology
- Gaussian variation
- 10000 generations
- Population 24
- Sigmoid hidden activation

- Threshold output activation
- 10:5 neurons

### 7.4.2 Results

Figure 7.9 shows curves used in this case. NeuroSearch consumes much more packets than steiner but this is quite obvious. Steiner has global information of nodes, links and resources and with this knowledge steiner algorithm can make smallest possible route to locate all wanted resources. HDS uses also fewer packets than NeuroSearch but they both need lots of packets when amount of needed resources increases.

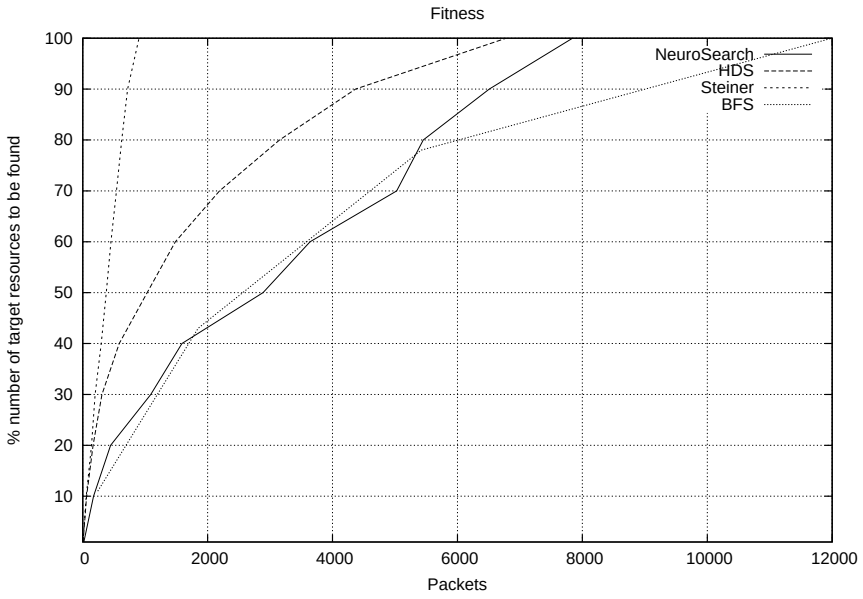


Figure 7.9: Usage of packets with different amount of target resources.

### 7.4.3 Conclusion

NeuroSearch uses more packets than steiner or HDS. BFS uses few packets when target amount of resources is kept small but when this amount rises over 70% BFS uses more packets than any other algorithm.

## 7.5 Queriers

### 7.5.1 Setup

In this case the effect of lowering amount of queriers per generation to calculate fitness value of neural network is examined. Other values are kept the same but amount of queries varies. If the fitness value of NeuroSearch is unaffected while querier amount is dropped in later evolutions smaller querier amount should be used because of smaller time usage.

Variables used in this case are:

- 100 peers
- 394 links
- 394 resources
- Power-Law distributed topology
- gaussian variation
- 10000 generations
- initial population 24
- sigmoid hidden activation
- threshold output activation
- 10:5 neurons

### 7.5.2 Results

In table 7.4 is shown amount of queriers and corresponding fitness per query. As can be seen, the values are quite near each other. 10 queries got best fitness value and thus it is suggested amount to use in future evolutions. Using smaller than 10 queries NeuroSearch's performance however drops. It does not get enough data from P2P-Network to figure out how to solve resource query problem.

| Querier amount | Fitness/query |
|---|---|
| 50 | 352.6 |
| 40 | 347.1 |
| 30 | 354.8 |
| 25 | 338.1 |
| 20 | 355.8 |
| 15 | 343.6 |
| 10 | 364.4 |
| 5 | 321.0 |
| 1 | 222.0 |

Table 7.4: Fitness per query for different query amounts per generation for the best neural network in last generation i.e., generation 10000 of evolution.

### 7.5.3 Conclusion

In this case is shown that using smaller amount of queriers while training NeuroSearch does not affect algorithm's performance. Only training using smallest query amounts lowers the fitness value. It is suggested that 10 queriers are used in future instead of 50, because this change fasten's calculations i.e., evolution significantly.

## 7.6 Brain Size

### 7.6.1 Setup

Neural network's intelligence capability can be modified by setting different amount of hidden layers and different amount of neurons on those layers. If the amount of neurons is small neural network cannot approximate input relations correctly. If there are too many neurons in neural network it wastes precious computing time.

Variables used in this case are:

- 100 peers
- 394 links
- 394 resources
- Power-Law distributed topology

| # | Neurons | Fitness | Packets | R | RR | Parents | Born | Hops | Age | Efficiency |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5:0 | 17500.0 | 2550 | 401 | 469 | 44185 | 99997 | 6 | 3 | 0.18 |
| 2 | 20:10 | 17905.0 | 1995 | 398 | 473 | 103 | 91855 | 24 | 8145 | 0.24 |
| 3 | 25:10 | 18091.0 | 1709 | 396 | 434 | 109 | 90715 | 5 | 9285 | 0.25 |

Table 7.5: Statistics of three different neural networks. Even the efficiency of the last two networks is almost the same. By taking a look at the real replies it can be seen that ANN 25:10 does not "overfind" resources that much as 20:10. It finds 32 resources too much while 20:10 finds 71, over twice as much (maximal possible amount of rewarded resources per 50 queries is 402). R is amount of resources found and awarded. RR is the amount of total found resources.

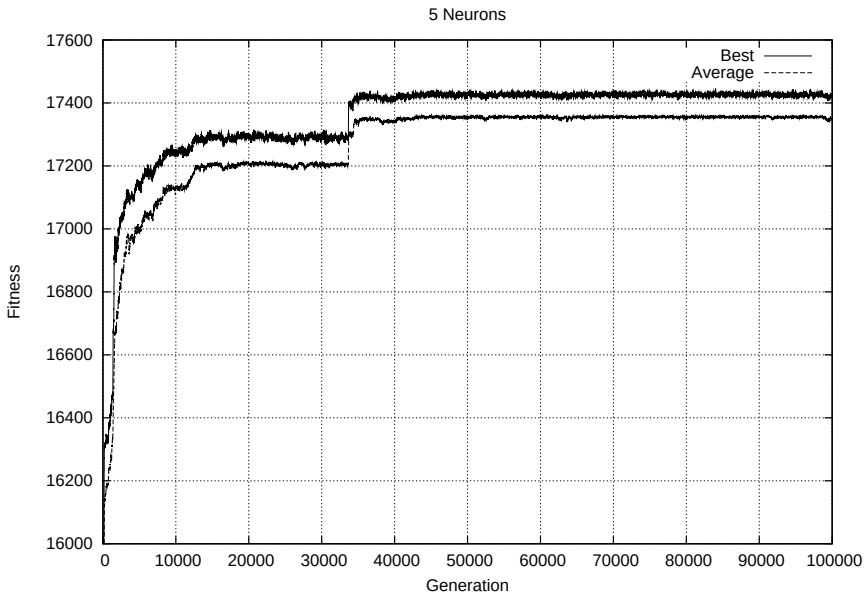


Figure 7.10: Evolution of neural network with one hidden layer and 5 neurons assigned on it. Even this small neural network reaches moderate fitness value, 17500.0. Average best fitness on generation 100000 is 17421.62. Fitness jumps suddenly near generation 33000 but after that this neural network cannot find new solutions to solve resource querying problem.

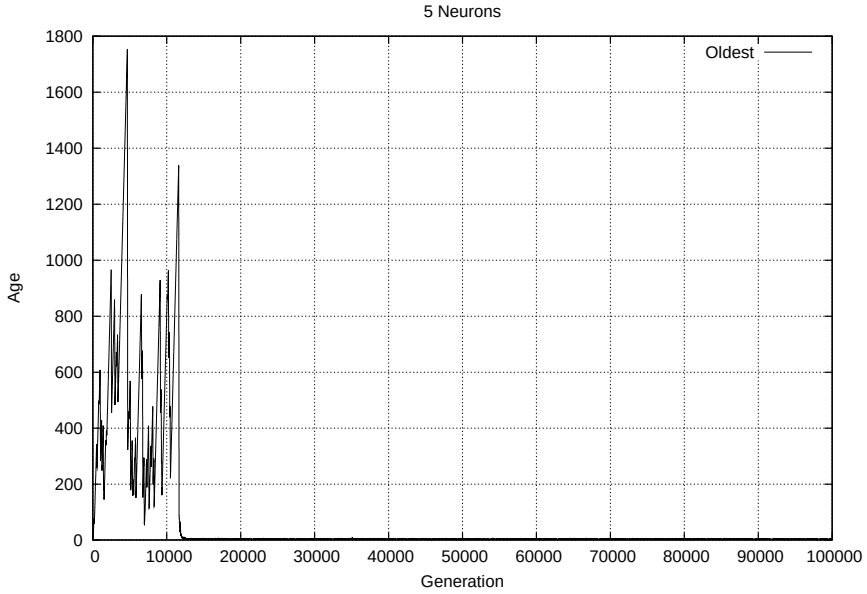


Figure 7.11: Age graph of the same network as in figure 7.10. After generation 13000 the population contains individuals whose life lasts only few generations.

- gaussian variation
- 50 percentage of resources
- 100000 generations
- initial population 24
- sigmoid hidden activation
- threshold output activation
- 6 inputs

Population amount of 24 were used because in earlier case it was found to be large enough to hold found intelligence as evolution proceeds. Variation amount was risen to 1.2 million and so generation amount was set to 100000. The varying values in this case are neuron amount and hidden layer amount. Different amount of neurons are tested both with one and two hidden layers.

Tests are calculated using either one or two hidden layers. One hidden layer is enough

to FFNN to approximate given function but it needs more neurons than FFNN using two hidden layers. Using more than two hidden layers does not improve the approximation performance of FFNN [9].

### 7.6.2 Results

In table 7.5 is the statistics of last generation (generation 100000) of neural networks used in this research case. First one is very small network which manages to reach good fitness. Second one is much bigger neural network which was able to find slightly better solution to resource query problem. First neural network is BFS algorithm because it uses only few hops thus sends queries fast to all neighbors. Second is more like DFS using fewer nodes but sends queries using long paths. Third NN is little larger than the second one. It is the best NeuroSearch algorithm shown in this thesis being the only one that could reach fitness over 18000. Surprisingly it uses only 5 hops and thus acts as BFS.

*Packets* and *replies* are the amount of packets used and amount of resources found in total of 50 queries. First network used about 50 packets/query and found almost 10 resources/query. Difference between *replies* and *real replies* is that *replies* is the amount of resources that NN is awarded and *real replies* is the real amount of resources it found. Every NN locates more resources than the target amount but none can still find maximum amount of resources e.g., maximum amount of rewarded resources in all 50 queries is 402. This means that algorithms find some resource type more than is needed but other resource type is found less than needed. Thus the total amount of found resources is over target amount but total rewarded amount is not the maximal. *Parents* is the amount of successful variations to reach this network. The huge difference between parents value of first and two last networks is explained later. *Born* is the generation number when the neural network was generated. *Hops* is the amount of nodes the longest message route has trespassed in the last query round of 50 queries. *Age* is the number of generations since ANN was born.

Smallest tested brain was neural network with one hidden layer and 5 input neurons assigned on it. Both layers included also bias. In figure 7.10 we see the evolution and fitness curve of very small neural network. There is nothing very dramatic on the evolution. In the generation 33000 there can be seen nice rising of the fitness but in all this evolution is quite stable. Even though this ANN is very small it can solve

the problem quite well. However, when we look at the figure 7.11 of age curve of the same network we see something quite interesting. At the beginning the age of the oldest network in the population stays quite high but in generation 11618 the neural network which has born in generation 10253 lost the competition and after this all neural networks in the population stayed very young. Average age of the oldest network after generation 13000 is below 6. This means that some of the scaling factors which control the strength of mutation is gone so small that changes in the weights of siblings are very small. Optimization process has come to its end. After generation 13000 the evolution could have been stopped. But if we look at the fitness curve again we see that in about generation 33000 there happens a fitness jump. Evolution of evolution e.g., size of scaling value has lots of variation. Some scaling values are very small while some values are huge. It may be that those weight values which have huge scaling value have minimal effect on performance of neural network. In evolution they change all the time but cannot find any value which would be better than others, so it may be that it is almost same to the neural network which these weight values are.

In appendix A (p.66) weight values and scaling values for this neural network are listed. There are 41 weight values in the network and for every weight there is scaling factor. $w$ means weight value and $s$ means scaling value. After this term there is in parantheses three numbers. First one tells the order number of the gap between layers where this connection is located e.g., #1 means that connection is between first gap, between input layer and hidden layer. #2 is gap between first hidden layer and second hidden layer or output layer, depending if the neural network has one or two hidden layers. #3 means gap between second hidden layer and output layer. This neural network has two hidden layers. Second number inside parentheses tells which neuron in the first layer has the connection. Third number is the number of neuron in the second layer. So, term *w(1,2,3)* means weight value between second input neuron and third neuron on first hidden layer.

In figure 7.12 evolution of much larger neural network is shown. This ANN has 20 neurons on its first hidden layer and 10 on second, we mark this neural network as 20:10. In charra.it.jyu.fi evolution of this network took 6 days and 12 hours. In contrast, the evolution of previous network took only 30 hours. This network reaches almost the fitness of 18000 (best network on generation 100000 has fitness of 17905).

In figure 7.13 the age graph is shown. This differs strongly from smaller ANN's age

graph. Here the best network is very old, over 3000 generations all the time. More detailed information can be found on table 7.6 where are some statistics of all neural networks in last two generations listed. There can be seen that oldest network is not necessarily the best one. The network with id -1504970559 is oldest but not the best. Average age of strongest half of the population is also high. Descendants do not survive in the contest of getting better fitness because only few times in 10000 generations new neural network manages to reach fitness value enough to stay on living.

| Generation: 99999 | | | | |
|---|---|---|---|---|
| Fitness | Parents | Born | Hops | ID |
| 17651.0 | 104 | 99844 | 13 | -941428707 |
| 17715.0 | 103 | 95684 | 12 | 2124936363 |
| 17725.0 | 104 | 67804 | 9 | -1504970559 |
| 17744.0 | 102 | 84107 | 8 | 600025180 |
| 17753.0 | 103 | 97300 | 10 | -63942598 |
| 17787.0 | 103 | 96959 | 6 | -2070662978 |
| 17798.0 | 103 | 90849 | 10 | 1705618726 |
| 17807.0 | 102 | 70843 | 15 | 359425110 |
| 17816.0 | 104 | 99347 | 14 | -646429487 |
| 17819.0 | 102 | 84450 | 12 | -914979540 |
| 17858.0 | 103 | 91855 | 16 | 140784151 |
| 17912.0 | 104 | 88289 | 17 | 1710203044 |
| Generation: 100000 | | | | |
| Fitness | Parents | Born | Hops | ID |
| 17664.0 | 104 | 99844 | 15 | -941428707 |
| 17738.0 | 102 | 84107 | 13 | 600025180 |
| 17739.0 | 102 | 70843 | 15 | 359425110 |
| 17770.0 | 103 | 97300 | 9 | -63942598 |
| 17782.0 | 104 | 99347 | 18 | -646429487 |
| 17821.0 | 103 | 96959 | 13 | -2070662978 |
| 17831.0 | 102 | 84450 | 14 | -914979540 |
| 17837.0 | 103 | 90849 | 11 | 1705618726 |
| 17872.0 | 104 | 88289 | 12 | 1710203044 |
| 17884.0 | 103 | 95684 | 14 | 2124936363 |
| 17897.0 | 104 | 67804 | 8 | -1504970559 |
| 17905.0 | 103 | 91855 | 24 | 140784151 |

Table 7.6: Statistics of all NeuroSearch algorithms (20:10 neurons) on two last generations of evolution. The oldest networks are not necessarily the fittest. All networks have similar amount of parents, little over 100.

Figure 7.14 shows how different amount of neurons on different input layers will

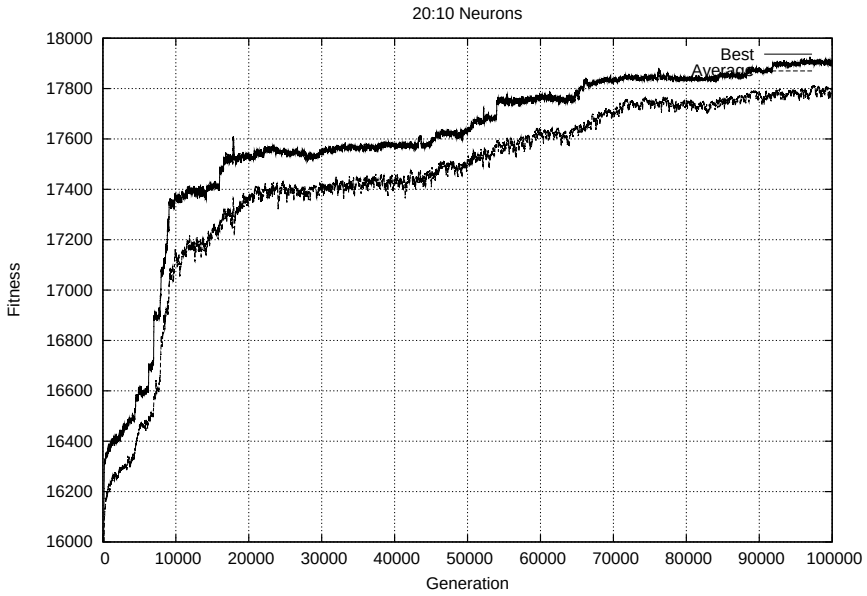


Figure 7.12: Figure of the evolution of 20:10 neural network. Fitness is continuously rising and reaches almost 18000.

affect the evolution of NeuroSearch. Small hops value means that algorithm behaves as BFS, it sends queries to many neighbors but stops sending after some time. Bigger hops value means that NeuroSearch acts more like DFS, sending query only to few neighbors but more frequently. Neural network 20:10 used 17.46 hops as an average of 50 last generations. Same network used 24 hops in generation 100000. In contrast that the previous neural network 5:0 used 6 hops, behavioring more like BFS. Amount of hops decreases significantly as amount of neurons increases over 20:10. This can be explained by two ways. First, amount of generations has been too small for larger neural networks. When increasing the size of the neural network it also needs more training time. Another explanation is that too large neural network have more free parameters that actually may decrease it's performance [9, p.103].

However, the best NeuroSearch algorithm in this thesis had 25:10 neurons and fitness value (in last generation 100000) 18091.0. This algorithm used only 5 hops to reach all resources. This leaves question whether BFS or DFS style algorithm is the best one to solve resource discovery problem.

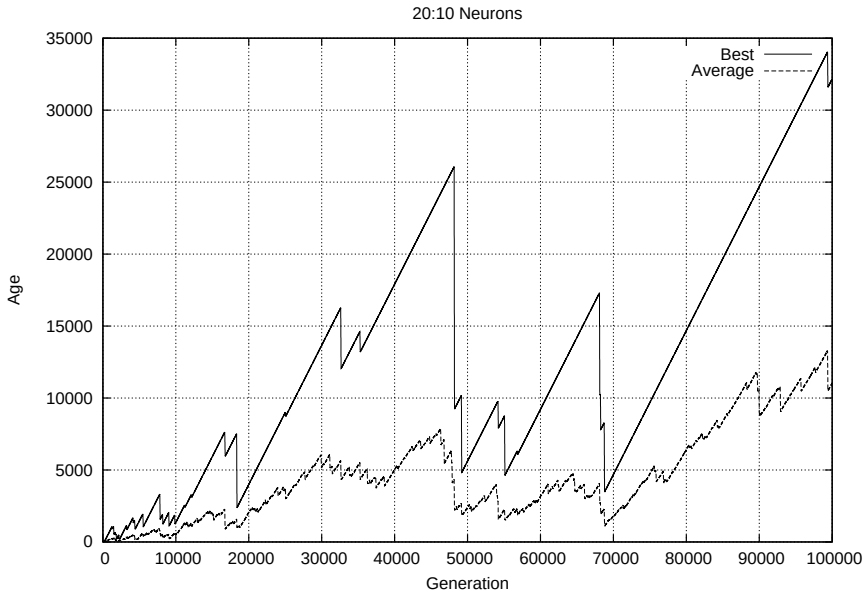


Figure 7.13: Age of the 20:10 neural networks. Neural networks in the population are much older than those in population of 6 neurons neural networks.

### 7.6.3 Conclusion

Larger neural networks consume more computing time but also manages to reach better fitness values. It is noticeable that fitness usually jumps powerfully up before generation 1000. This development of intelligence is purely random, it may happen at any time, but it usually happens somewhere between generations 100 and 1000. Another jump usually happens between fitness values 16000 and 18000 on random generation, usually after generation 10000.

ANN with 20:10 neurons found new ways to solve resource discovery problem. It used more hops but kept usage of packets still relatively small. Smaller and larger networks did not learn to use multiple hops. This may be caused by lack of generations in training of larger than 20:10 ANNs. Smaller neural network are not capable of finding any other than BFS stylish algorithms. ANN with 25:10 neurons could make better fitness value with using only 5 hops. This raises question is it possible that BFS style algorithm is the best one to solve this problem in this P2P-Network?

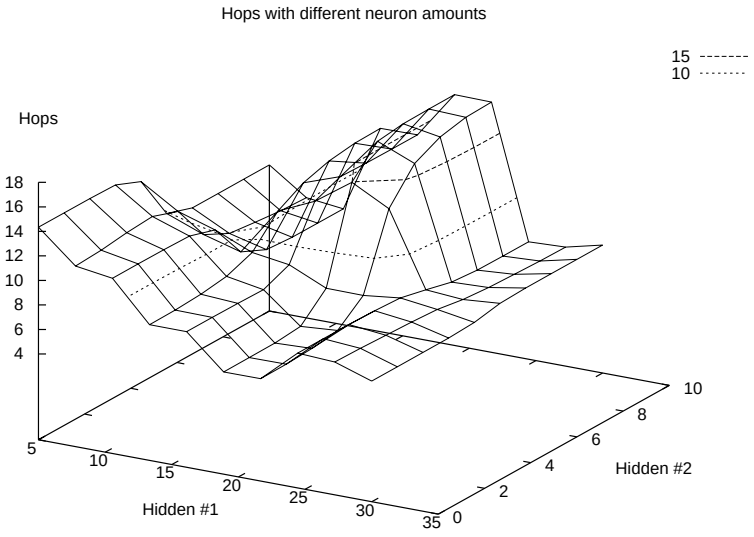


Figure 7.14: Different amount of neurons used in NeuroSearch. Total number of hops used in last query is shown in z-axis and number of neurons in hidden layer 1 on x-axis and hidden layer 2 on y-axis. ANN of 20:10 gets highest hops amount imitating DFS algorithm. It is future work to figure out what happens if neuron amount in hidden layer 2 increases e.g., will the hops drop. Values in this figure are counted as moving average of 50 last generations. Neural network 20:10 has the highest value of 17.46 hops counted as moving average.

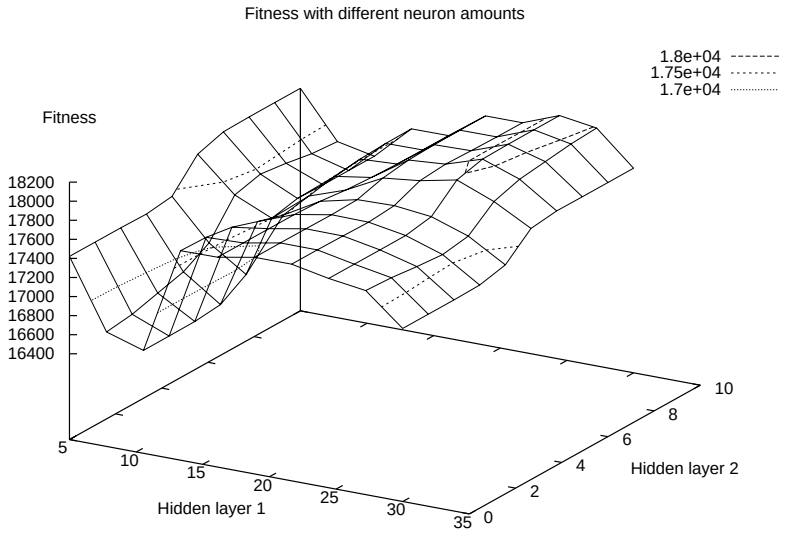


Figure 7.15: Fitness for different amount of neurons. ANN 25:10 had the greatest fitness value of 18053.62 calculated as moving average of 50 last generations. Hops value for this network was only 6.58 and thus this algorithm is of BFS style.

# 8 Conclusion

## 8.1 Summary

In this thesis has been shown five different aspects of building resource query algorithm using neural network, evolutionary programming and evolutionary strategies. In general, NeuroSearch has been found promising search algorithm for P2P-networks, intelligent routing uses less packets than BFS-stylish flooding algorithm. By choosing specific set of inputs researcher may train differently behavioring NeuroSearch i.e., NeuroSearch may imitate any existent search algorithm or it may behavior as combination any of those.

Five tests in this work were initial population, neural network size, inputs, query amounts and different target resource amounts. If initial population was set too small the population could lost its best individuals but too large population would consume computing time. It was found that 24 neural networks should be used as initial population. Amount of neurons in neural network has interesting results. Small neural networks could not learn to solve problem properly and thus had low fitness values. While size of the neural network was increased the amount of hops it used also increased, thus algorithm was behavioring more like depth first search. However, after 20:10 neural network the amount of hops significantly dropped and larger neural networks started to act like breadth first search but got still very good fitness values. This shows that intelligent BFS style search may be best solution to this problem (problem is different if amount of P2P-nodes increases). Inputs were also tested, alone and by groups. Interesting results showed that some inputs had better performance alone than grouped. This may be caused by overlapped information of the inputs. In query amounts test was found that previouly used 50 queries / fitness was too much for resolving valid fitness value. Instead, 10 queries was found to be enough. In resource amounts test different search algorithms performances were measured. Steiner algorithm using global knowledge was no doubt the most efficient. HDS and NeuroSearch came second while BFS managed poorly when amount of target resources increased.

## 8.2 Future

More input values need to be found and used e.g., random value, P2P-Networks heuristic inputs, output value as next peer's input value, neighbor amount as next peer,s input value and history based inputs e.g., number of received replies etc. Input that is 1 if this node has sent query further. With this input NeuroSearch may send message only to one neighbor in the beginning thus imitating DFS. Also input that would tell how many neighbor's neighbors are free ones (if more than one node is *grey node*, if none node is *black node*) should be tested as well as *decisionsLeft* i.e., how many neighbors node has unresolved and *forwardedDecisions* i.e., to how many of the resolved neighbors has been sent the query.

Neural network could be taught to modify its architecture e.g., neuron amounts and links between neurons. This could speed up the calculations (if same result is got with fewer amount of neurons or weights).

Is it possible to evaluate dynamic populations? Evaluation would start with 2 neural networks but the process would rise the population amount. After it would reach some state of balance it just listens certain events for optimizing population amount by dropping or rising new networks.

How about to count moving average value to the NEW descendants also. This means that every descendant need to be tested as many times as needed and after that the average value is calculated. In the present system at start of the new generation all nets are tested to get fitness. Average value can be calculated from older networks but new networks have only one value to count on. This problem is solved by testing the new networks so many times that counting the average value is possible. Of course the backside is that processing time will be many times larger.

Determine the probability of different amount of population to lose its best network on specific amount of generations e.g., how many generations should run on 30 neural networks that probability to loose best network grows to one percent?

Output of the neural network could be used as probability to send query further. Little research has been on using probability output but more work is still needed.

# 9 Bibliography


[1] Lada A. Adamic, Rajan M. Lukose, and Bernardo A. Huberman. Local search in unstructured networks. 2002.

[2] Agora-center,
http://www.jyu.fi/agora-center/.

[3] comp.ai.neural-nets faq,
http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-9.html.

[4] R. Albert, H. Jeong, and A.-L. Barabási. Attack and error tolerance in complex networks. *Nature 406*, 2000.

[5] Annemari Auvinen. Topologian hallinta -algoritmit chedar-vertaisverkko alustassa. Master's thesis, University of Jyväskylä, 2004.

[6] Albert Barabasi. *Linked.* Perseurs Publishing, 2002.

[7] Reuven Cohen and Shlomo Havlin. Scale-free networks are ultrasmall. *Phys. Rev. Lett. 90, 058701*, 2003.

[8] Defence software collaborators,
http://www.dacs.dtic.mil/techs/neural/neural4.html.

[9] Andries Engelbrecht. *Computational Intelligence - an Introduction.* Wiley, 2002.

[10] David B. Fogel. *Blondie24 - Playing at the Edge of AI.* Academic Press, 2002.

[11] Freehaven,
http://www.freehaven.net/.

[12] Gnutella,
www.gnutella.org.

[13] Hermanni Hyytiälä. Fenfire in peer-to-peer environment. Master's thesis, University of Jyväskylä, 2003.

[14] Michael Iles and Dwight Deugo. A search for routing strategies in a peer-to-peer network using genetic programming. *IEEE*, 2002.

[15] Kazaa network,
http://www.kazaa.com.

[16] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann publishers, 1997.

[17] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: extended edition*. MIT press, 1988.

[18] Napster,
www.napster.org.

[19] Andy Oram. *Peer-To-Peer - Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[20] Publius,
http://cs1.cs.nyu.edu/ waldman/publius/.

[21] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. 2002.

[22] Rüdiger Schollmeier. A definition of peer-to-peer networking towards a delimitation against classical client-server concepts. *EUNICE*, 2001.

[23] Website of sean mcmanus, a uk freelance journalist and internet consultant,
http://www.sean.co.uk/a/musicjournalism/var/historyoffilesharing.shtm.

[24] Seti@home,
http://setiathome.ssl.berkeley.edu/.

[25] University of louisiana at lafayette,
http://www.ucs.louisiana.edu/ isb9112/dept/phil341/histconn.html.

[26] Mikko Vapa, Niko Kotilainen, Annemari Auvinen, Joni Töyrylä, Hermanni Hyytiälä, and Jarkko Vuori. Neurosearch: Evolutionary neural network resource discovery algorithm for peer-to-peer networks. 2004.

[27] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. *International Conference on Distributed Computing Systems*, 2002.

# NeuroSearch Neural Network

Weight and Scaling values for 6 neuron neural network

| Weight value | Scaling value |
|---|---|
| $w(1,1,1) : 0.8$ | $s(1,1,1) : 9.3E-18$ |
| $w(1,1,2) : 0.2$ | $s(1,1,2) : 2.7E-19$ |
| $w(1,1,3) : 0.2$ | $s(1,1,3) : 1.1E-28$ |
| $w(1,1,4) : -9.3$ | $s(1,1,4) : 0.001$ |
| $w(1,1,5) : 0.9$ | $s(1,1,5) : 1.6E-21$ |
| $w(1,2,1) : 6.5E26$ | $s(1,2,1) : 5.9E21$ |
| $w(1,2,2) : -3.6E7$ | $s(1,2,2) : 1.8E-25$ |
| $w(1,2,3) : -2.8E22$ | $s(1,2,3) : 3.7E-5$ |
| $w(1,2,4) : -2.5E19$ | $s(1,2,4) : 1.04E-6$ |
| $w(1,2,5) : -1.3E33$ | $s(1,2,5) : 1.1E11$ |
| $w(1,3,1) : 8.0E16$ | $s(1,3,1) : 5.4E14$ |
| $w(1,3,2) : -5.9E10$ | $s(1,3,2) : 4.4E-10$ |
| $w(1,3,3) : 3.4E9$ | $s(1,3,3) : 4.9E-21$ |
| $w(1,3,4) : -1.5E57$ | $s(1,3,4) : 7.2E45$ |
| $w(1,3,5) : -14226.8$ | $s(1,3,5) : 8.8E-50$ |
| $w(1,4,1) : -18.4$ | $s(1,4,1) : 3.5E-36$ |
| $w(1,4,2) : -281276.5$ | $s(1,4,2) : 7.2E-10$ |
| $w(1,4,3) : -7954.8$ | $s(1,4,3) : 2.7E-23$ |
| $w(1,4,4) : 171342.0$ | $s(1,4,4) : 7.1E-13$ |
| $w(1,4,5) : -2.1E13$ | $s(1,4,5) : 317358.6$ |

| Weight value | Scaling value |
|---|---|
| $w(1,5,1) : 400.3$ | $s(1,5,1) : 0.1$ |
| $w(1,5,2) : -0.4$ | $s(1,5,2) : 1.3E-20$ |
| $w(1,5,3) : 0.1$ | $s(1,5,3) : 3.0E-35$ |
| $w(1,5,4) : 1.9$ | $s(1,5,4) : 3.3E-10$ |
| $w(1,5,5) : -1.5$ | $s(1,5,5) : 3.3E-42$ |
| $w(1,6,1) : 0.01$ | $s(1,6,1) : 4.6E-24$ |
| $w(1,6,2) : -2.4E13$ | $s(1,6,2) : 14191.7$ |
| $w(1,6,3) : -1.2E10$ | $s(1,6,3) : 1.3E7$ |
| $w(1,6,4) : 1.4E9$ | $s(1,6,4) : 13271.2$ |
| $w(1,6,5) : -0.3$ | $s(1,6,5) : 1.7E-26$ |
| $w(1,7,1) : -0.5$ | $s(1,7,1) : 2.3E-14$ |
| $w(1,7,2) : 0.01$ | $s(1,7,2) : 1.3E-40$ |
| $w(1,7,3) : 0.4$ | $s(1,7,3) : 4.8E-13$ |
| $w(1,7,4) : 0.6$ | $s(1,7,4) : 1.9E-46$ |
| $w(1,7,5) : -0.1$ | $s(1,7,5) : 1.6E-38$ |
| $w(2,1,1) : 6.9E18$ | $s(2,1,1) : 0.1$ |
| $w(2,2,1) : -3.1E10$ | $s(2,2,1) : 0.1E-11$ |
| $w(2,3,1) : -2.6E14$ | $s(2,3,1) : 2.9E-9$ |
| $w(2,4,1) : -1.2E17$ | $s(2,4,1) : 3.7E9$ |
| $w(2,5,1) : 3.1E55$ | $s(2,5,1) : 1.1E35$ |
| $w(2,6,1) : 1.3E8$ | $s(2,6,1) : 1.3E8$ |