NeuroSearch: evolutionary neural network resource discovery algorithm for peer-to-peer networks [⋆]

Mikko Vapa [a,*] Niko Kotilainen [a] Annemari Auvinen [a]
Joni Töyrylä [a] Hermanni Hyytiälä [a] Jarkko Vuori [b]

[a] *Agora Center, University of Jyväskylä, P.O.Box 35 FIN-40014, Finland*

[b] *Department of Mathematical Information Technology, University of Jyväskylä, P.O.Box 35 FIN-40014, Finland*

Abstract

Discovery of distributed resources is an essential problem in peer-to-peer (P2P) networks since there is no centralized index where to look for information about resources. One solution for the problem is to use a search algorithm that locates resources based on the local information about the network. Traditionally, the search algorithms have been designed to be based on few rules and designed completely by humans. The problem with these algorithms is that if the conditions like resource distribution and topology in the network change the algorithm becomes less efficient and won't have the flexibility to adapt to the new environment.

In this paper we describe the results of a process where evolutionary neural networks are used for finding an efficient search algorithm. By using this process we are able to define the network conditions and the quality of the search algorithm beforehand as an optimization problem and solve it computationally. As an end product we get a search algorithm that has adapted to the predefined conditions. The test results indicate that the approach is feasible and that an evolutionary optimization process can produce candidates that are competent compared to breadth-first search algorithm used in peer-to-peer networks.

Key words: peer-to-peer, resource discovery, search algorithm, neural networks, evolutionary computing

⋆ This work was funded by the Agora Center InBCT project.

* Corresponding author.
Email address: `mikvapa@jyu.fi, +358 40 7619696 (phone), +358 14 260 4400 (fax)` (Mikko Vapa).

1 Introduction

In the *resource discovery problem* any node in the network can possess resources and also query these resources from other nodes. The problem consists of graph with nodes, links and resources. Resources are identified by IDs and nodes may contain any number of resources. Also there can be duplicates of the same resource in different nodes. Any node in the graph can start a query which means that some of the links in the graph are traversed based on a local decision and whenever the query reaches the node with the queried ID, the node replies.

One solution for the resource discovery problem is the breadth-first search algorithm (BFS) [1] used in Gnutella peer-to-peer (P2P) protocol [2]. In BFS a node starts a query by sending the query to all its neighbors. When the neighbors receive the query, they send it further to all their neighbors except the one from which the query was received. Also, if the query has already arrived to the node earlier it is not sent further anymore. The algorithm ensures that if a resource is located in the network it can also be found from the network. Also, BFS has low latency because it locates the shortest path between the query originator and the resources. The drawback of the algorithm, however, is that it uses lots of query packets to find a specific resource thus making the network prone to congestion.

Resource discovery is multi-dimensional problem affected by the spatial distribution of resources, the network topology and the location of query originators. Also the quality of the search algorithm e.g., the number of results that needs to be located is heavily dependant on what kind of a resource is being queried. In some scenarios finding only one instance of a resource is sufficient when in other cases all instances of a resource might be needed for the query to be successful. Also in certain scenarios it might be beneficial for the search algorithm to locate only shortest paths to resources, but in presence of failures a better search strategy would be to use multiple paths to find the resources adding even more dimensions to the problem. When the number of dimensions increase the problem becomes complex and makes it increasingly difficult for the algorithm designer to take all the relevant aspects into account.

Fortunately, there are ways to cope with complex problems. Evolutionary computing and neural networks have become common in information technology to solve complex problems by approximating optimal solutions. Feedforward neural network [3] (also called as Multi-Layer Perceptron, MLP) is known as one example of an algorithm structure that can be used to find a solution to complex problems. In particular the property that MLPs are universal approximators [4] makes them interesting. Universal approximation means that if the algorithm structure is properly taught neural network can learn to map any function that is given as a training data correctly to output values. In resource discovery problem this function is the decision on which kind of input values the query should proceed to the neighboring node and on which kind of input values the querying should be stopped. Input values can be based for example on how many hops the query has travelled, how good the neighbor is at locating resources, in which kind of a node the query is currently being processed etc.

Neural network consists of weights and activation functions that needs to be tuned correctly before it can be used to calculate the output. In the case of resource discovery the right output values are not known beforehand, so the training process has to give feedback based on how well the neural network was able to locate resources. This can be done, for example, by counting the number of packets the algorithm used and the number of results the algorithm was able to retrieve for multiple queries.

Evolutionary algorithms [3] (such as genetic algorithm) use a process where initial population is instantiated as candidate solutions and then refined in forthcoming generations using mutation and crossover. Traditionally crossover has been the fundamental part of the variation algorithm, but in recent years also approaches that rely only on mutation operator has been used [5]. For neural network to be efficient the optimization method also needs to be able to avoid local maximums and provide ways for locating the global maximum of the fitness space. This can be achieved using only mutation operator if the variation range for mutation can evolve between each generation. Because of the variation sometimes the search makes huge leaps to different positions in the solution space thus avoiding to be stuck only in a local maximum.

The algorithm presented in this paper combines neural networks and evolution together forming a solution to resource discovery problem. This kind of a problem is present in P2P networks [6] where the location of a particular resource is not known beforehand. P2P networks are self-organizing decentralized systems having attended a lot of research activity in recent years. P2P networks exhibit dynamic and complex behavior when peers autonomously connect to each other and search resources using the topology they have constructed. Our approach takes advantage of this complex behavior by allowing neural network to adapt its function to estimate the optimal search algorithm in given network conditions.

The rest of this paper is organized as follows. The next section presents the references to related work done in P2P resource discovery. Section 3 describes NeuroSearch algorithm as a solution for the resource discovery problem. Section 4 describes the optimization process and Section 5 the test case used in the study. Section 6 analyzes the simulation results and in Section 7 the paper is concluded.

2 Related Work

A lot of research has been done regarding the resource discovery problem. Adamic et al. [7] and Kim et al. [8] propose a search strategy that utilizes the topological properties of a power-law network. The search strategy first proceeds towards highest-degree node e.g. the node that has the highest number of neighbors and then gradually moves to lower degree ones. Algorithm locates resources fast if they can be found from the network core, but the performance decreases when the central nodes are revisited in search for lower degree nodes.

Lv et al. [9] evaluate BFS, expanding ring and random walk search mechanisms with varying topologies including random graphs, power-law graphs and a snapshot of Gnutella network as obtained in October 2000. The authors find that BFS is not scalable and in particular on Gnutella and power-law graphs the effects of flooding are disastrous: the number of messages increases drastically when time-to-live is increased. Expanding ring where time-to-live is extended gradually for BFS is the first aid to the problem but because it forwards duplicate messages to the nodes that the query has already reached, a better solution to the problem using random walkers is proposed. Search initiates multiple walkers and forwards them based on random selection of neighbor. In addition to the time-to-live value as a termination condition for the walkers, Lv et al. use checking where the random walkers periodically check from the query originator whether walker should be terminated or not. While random walkers increase the number of hops and thus latency needed for the query, they decrease the total traffic because the search proceeds in depth-first manner.

Kalogeraki et al. [10] consider two search algorithms for the resource discovery problem. Modified Random BFS Search behaves like BFS, but the neighbors select only a random subset of neighbors for forwarding the query. This property reduces traffic, but adjusting the correct size of the subset for different networks may be difficult. Their work uses a random graph in which all the nodes have approximately similar degrees. Thus the performance of the algorithm in power-law graphs cannot be directly determined from the results. An another algorithm they present is called Intelligent Search Mechanism in which nodes keep track of recent query results provided by their neighbors. When a new query arrives the neighbors are sorted based on the similarity of the query to earlier replies from the neighbor. Because the nodes keep track of the earlier queries the performance of the algorithm improves as the network evolves.

Yang and Garcia-Molina [11] experiment with many types of directed search strategies based on different heuristics. These heuristics include the number of results returned, shortest average time to satisfaction, smallest average number of hops of received results, the highest number of results returned, shortest message queue, shortest latency and highest degree. The authors' suggest that for minimizing the time to satisfaction measure the best strategy is to pass the query to the neighbor that has had the shortest average time to satisfaction for last 10 queries. Also, when considering the bandwidth use most reliable measure is the smallest average number of hops of received results for last 10 queries. The heuristics used in the study are based on history data collected locally in each node.

Similar use of history information is found from the work by Tsoumakos and Roussopoulos [12,13]. In authors' proposal, Adaptive Probabilistic Search algorithm, neighbors keep track of the success rates of earlier queries and forward random walkers probabilistically based on the success rate. The algorithm is able to adapt to different query patterns and, therefore, performs better than random walkers.

There are certain limitations in all the approaches described above. First, all the algorithms use some control parameters (for example time-to-live, number of walkers or the proportion of neighbors to forward the query) that can be used to tune the algorithm. For a search algorithm, the number of control parameters should be kept as minimal as possible to allow zero configurability when applied to a real environment. Second, while some of these approaches have mechanisms to adapt to the environment they do not utilize the whole potential of the environment because they rely only on one strategy (for example the similarity of the query and earlier replies, shortest average time to satisfaction for last 10 queries or the success rate of earlier queries). In general, only one strategy cannot be efficient in all scenarios and therefore an efficient algorithm should be able to utilize many strategies at the same time.

To overcome these limitations a neural network based resource discovery algorithm called NeuroSearch was designed. NeuroSearch learns by itself the correct behavior in given network conditions and uses multiple strategies to locate resources. To authors' knowledge this is the first time when neural networks are being applied to resource discovery problem.

3 Neural Network Search Algorithm - NeuroSearch

NeuroSearch is an adaptive resource discovery algorithm that makes decision to whom of the node's neighbors the resource request message is forwarded. The decision is based on the output neuron of 3-layer perceptron neural network. The algorithm goes through all the node's neighbors (denoted as receivers) one by one and calculates the neural network output determining

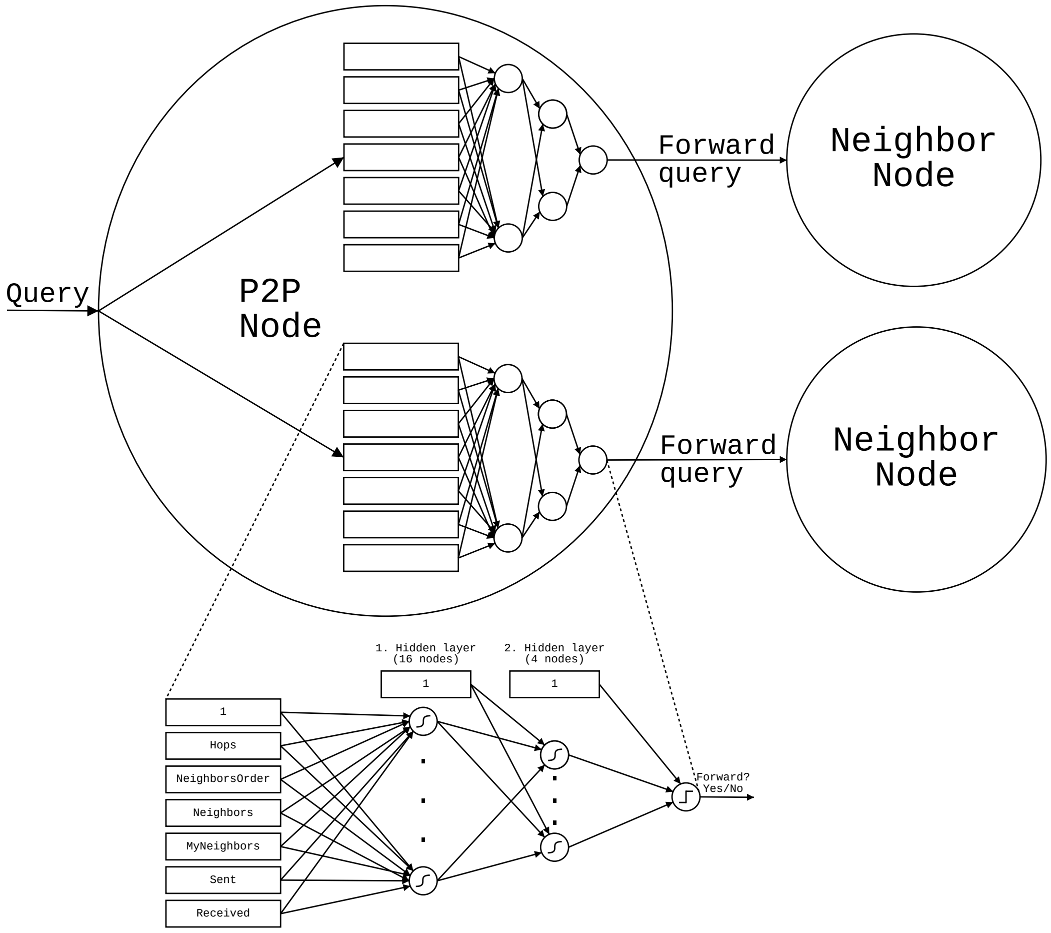

Fig. 1. Processing of NeuroSearch resource query and the NeuroSearch neural network.

which receivers will receive the query. This process is illustrated in Figure 1. If the query locates a matching resource in the node, reply message is forwarded back to the query originator via the path that the query has travelled. If the query has already been processed by the node the resource reply will not be sent.

The algorithm uses multiple input parameters for the neural network:

- *Bias = 1* is the bias term.
- *Hops* is the number of hops in the message.
- *NeighborsOrder* tells in which neighbor rank index this receiver is compared to other receivers. The receiver with the best rank has value of 0.
- *Neighbors* is the number of the receiver's neighbors.
- *MyNeighbors* is the number of node's neighbors.
- *Sent* has value 1 if the message has already been forwarded to the receiver using this link. Otherwise it has value of 0.
- *Received* has value 1 if the message has been received earlier, else it has value of 0.

Hops and *NeighborsOrder* are scaled with the function $f(x) = \frac{1}{x+1}$, and *Neighbors* and *MyNeighbors* with $f(x) = 1 - \frac{1}{x}$ before giving them to the neural network. Scaling is performed to ensure that all the inputs are between 0 and 1 and that the input values close to 1 (other than *Sent* and *Received*) are supposed to represent higher probability for the query to be forwarded.

There are two hidden layers in the neural network. In the first hidden layer there are 15 nodes + bias and in the second 3 + bias. *Tanh* is used as an activation function in the hidden layers:

$$t(a) = \frac{2}{1+e^{-2a}} - 1.$$

The activation function in the output node is the threshold function:

$$s(a) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}.$$

If output is 1, the message is forwarded to the receiver, otherwise it is dropped.

4 Neural Network Optimization Process using Gaussian Random Variation

In this section we define the optimization process for adjusting the correct weight values for the neural network. To find the appropriate weights we use methods of evolutionary computing [3] and a neural network population. The feedback, which guides the optimization process is obtained by counting the packets sent in the network and replies received.

The fitness for neural networks is defined in two parts. Each query j is scored for each neural network h and the fitness is calculated by summing up all the scores after n queries:

$$fitness_h = \sum_{j=1}^{n} score_j.$$

The score value is defined with the following rules:

(1) If $replies \geq \frac{availableResources}{2}$ then $score = 50 \times \frac{availableResources}{2} - packets$: when half of the available resource instances are found from the

network, the fitness value does not grow if neural network locates more resources.

(2) If $replies < \frac{availableResources}{2}$ and $replies > 0$ then $score = 50 \times replies - packets$: if the number of found resources is not enough then the neural network develops if it locates more resources.

(3) If $replies = 0$ then $score = 1 - \frac{1}{(packets+1)}$: if none of the resources are found then the neural network should increase the number of packets sent to the network.

(4) If $packets > 300$ then $score = 0$: an algorithm that eventually stops is always better than algorithm that does not.

In the equations *availableResources* is the maximum number of resources that can be located in the query, *replies* is the amount of replies that the neural network was able to locate for the query and *packets* is the amount of packets that the neural network used for the query. The constant value *50* was selected to be large enough to guide the training process towards neural networks that benefit of locating resources. Another constant value, *300* was set as a criteria when neural network is considered to forward the query infinitely.

The optimization process has an initial neural network population whose weights are randomly defined. Every neural network is tested in P2P simulation environment using multiple queries and the fitness value is calculated. When all neural networks have been tested the best half of them are selected for mutation and used to breed the new generation. As a result, the initial number of neural networks are available for testing the new generation. After a selected number of generations, the optimization process is stopped.

Mutating neural network means that the values of the weights are slightly changed from the original. This is done by random variation using normal distribution (also called as Gaussian distribution) [14]. The variation is similar to the one used by David Fogel and Kumar Chellapilla in their research [15–17] in which scaling factor is used for each weight. Scaling factor is assumed to improve the adaptability of the evolutionary search. The random variation function is given as:

$$\sigma_i'(j) = \sigma_i(j) exp(\tau N_j(0,1)), j = 1, ..., N_w \tag{1}$$

$$w_i'(j) = w_i(j) + \sigma_i'(j) N_j(0,1), j = 1, ..., N_w \tag{2}$$

where N_w is the total number of weights and bias terms in the neural network, $\tau = \frac{1}{sqrt(2sqrt(N_w))}$, $N_j(0,1)$ is a standard Gaussian random variable resampled for every j, σ is the self-adaptive parameter vector for defining the step size for finding the new weight and $w_i'(j)$ is the new weight value.

5 Test Case

As a simulation environment we use P2P network simulator that we have developed. The simulator can be used to simulate the behavior of a static peer-to-peer network and to train neural networks using Gaussian random variation.

For the test case topologies power-law graphs were generated using the Barabasi-Albert model [18,19]. Power-law networks' neighbor distribution follows the power-curve $P(k) = \frac{1}{k^{\gamma}}$, where γ for Barabasi-Albert graph is 3. In power-law networks there exists few hubs in the network that have many neighbors and lots of nodes that have only few links. Power-law graph was selected because some existing P2P networks have shown to express power-law dependencies [20]. The graphs being tested contained 100 nodes with the highest degree node having 25 neighbors.

The test case data was divided in three different data sets as described in [3]: training set, generalization set and validation set. Training set contained two power-law topologies with both being queried 50 times per generation for each neural network. Two topologies were used to have neural networks adapt to wider range of situations than one topology would have provided. Generalization set consisted of two power-law topologies with 50 queries and it was used to measure the point in which the neural network loses its ability to generalize. The neural network having highest fitness at that point was selected and as a validation set one topology with 100 queries was used to produce the final simulation results. This ensured that neural network's performance was being tested in new environment and thus measured the true generalization ability of the best neural network.

For each topology resource instances were allocated based on the number of neighbors each node has. There were 25 different resources in the test case and the number of different resources in a node was the same as the number of neighbors the node had. This means that the largest hub had one copy of all resources and the lower degree nodes only some of these randomly chosen from uniform distribution.

The queried resource and querying node were selected randomly for each query. All the nodes had equal probability to be query originators (randomly sampled from uniform distribution). The probability for a resource i to be selected from all resources as next queried resource was $P(i) = \frac{\Sigma i}{\Sigma n}$ where n is the total number of resources. This means that we had a test case where more common resources were queried more often and that the query originators changed every generation. This ensured that neural networks had to adapt to many different query distribution patterns.

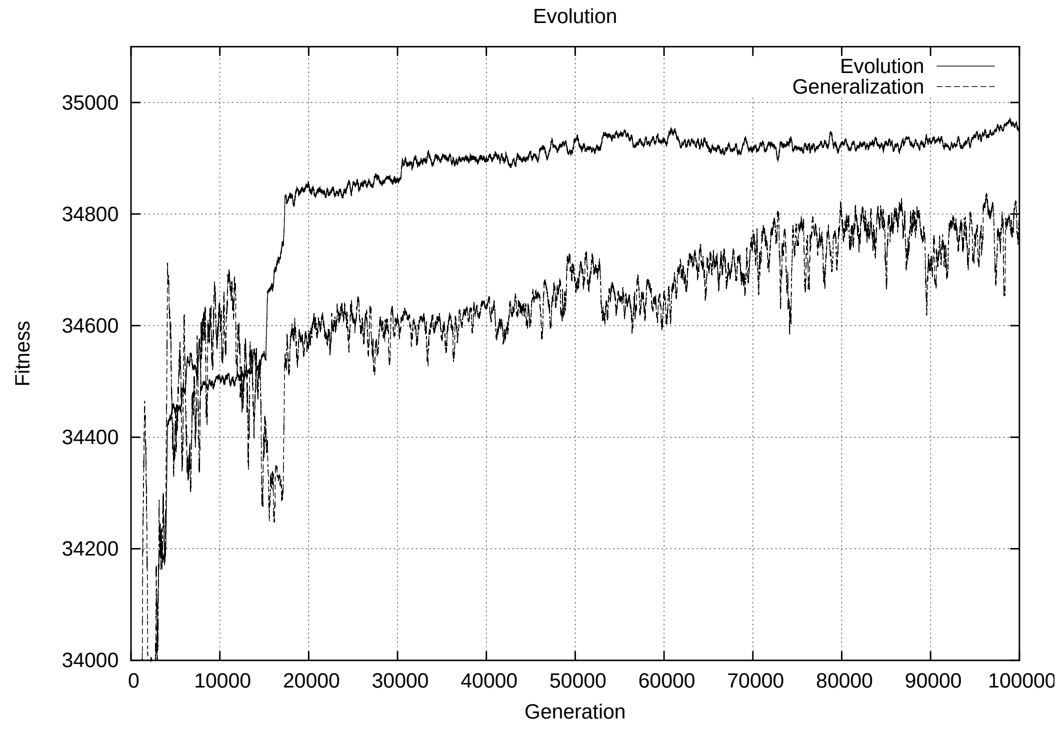

Fig. 2. Evolution of the best neural networks in each generation for training and generalization sets.

30 neural networks were used in the test case and 15 neural networks selected after each generation for next round. 15 new offsprings were created by using variation equations 1 and 2 for computing the new neural network weights. As a stopping criteria for the optimization process, 100.000 generations was set which seemed to be taking approximately two weeks on our desktop PC equipped with AMD Athlon XP 1800 processor. The evolution of best neural network in each generation is shown in Figure 2.

6 Simulation Results and Analysis

To evaluate the difference between BFS and NeuroSearch we selected the best algorithm at 85.736th generation and calculated the number of packets used and received replies for 100 different queries in a similar environment as where NeuroSearch was trained. 85.736th generation was selected because between 80.000 and 90.000 generations the best neural network had achieved steadily good results and in particularly at 85.736th generation the performance was highest. The results are presented in Figures 3, 4 and 5.

The results of Figure 3 show that the performance of NeuroSearch in number of packets is nearer to BFS with time-to-live value 2 than 3. In average

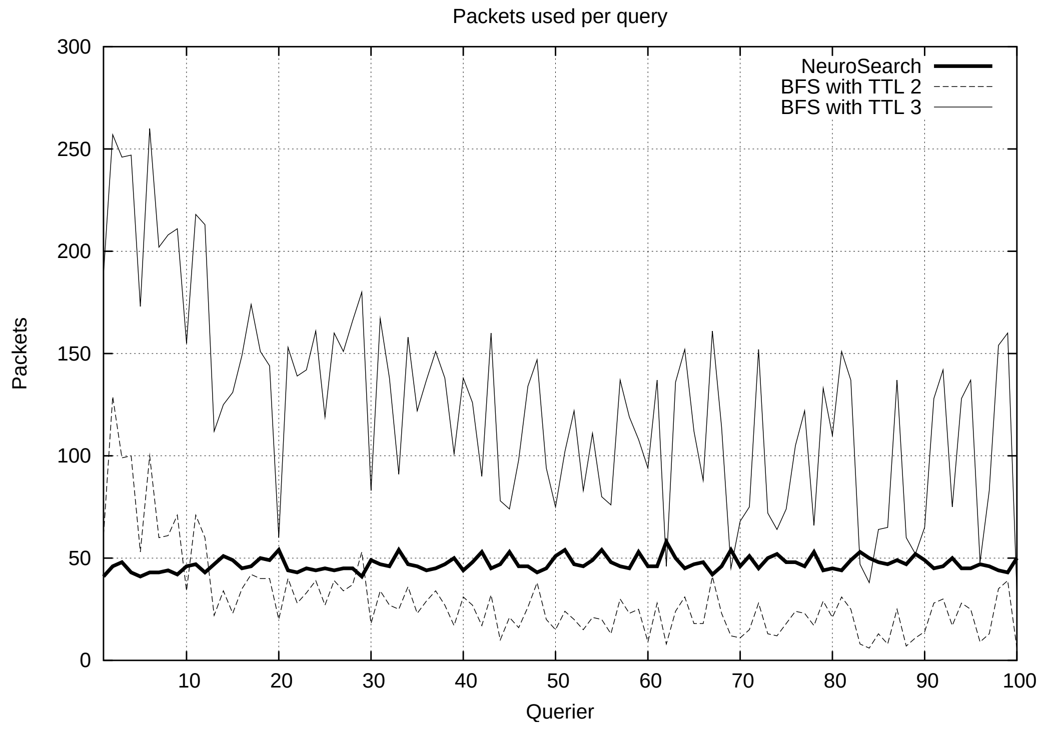

Fig. 3. Packets used by the algorithms.

NeuroSearch consumes 47.1 packets per query whereas BFS with TTL 2 consumes 30.0 and BFS with TTL 3 124.6 packets. The reason why there is some variation in the number of packets for successive BFS queries is that the distance where packets will be delivered depends on which node is querying. If the query starts from a central node (nodes 0-10) it will produce more packets than the same query started from an edge node (nodes 90-99) because the edge query has fewer connections where BFS can spread. In case of NeuroSearch the performance is stable and does not depend on what node is querying.

Figure 4 shows how many replies the algorithms are able to locate. NeuroSearch's performance in terms of located resources is quite similar to BFS with TTL 2 at central nodes but better in edge nodes. Compared to BFS with TTL 3 the performance of NeuroSearch is constantly lower reaching only at some edge nodes the same performance level. The reason why NeuroSearch is satisfied with this level of performance is that it has already reached the goal of finding half of available resources as defined in the fitness function and locating more resources would not be beneficial in its evolution.

To further analyze the performance difference of NeuroSearch and BFS algorithms we computed the ratio between number of replies received and number of packets used resembling the efficiency of the algorithms. These results are shown in Table 1.

The results show that NeuroSearch's efficiency is at the same level than BFS-

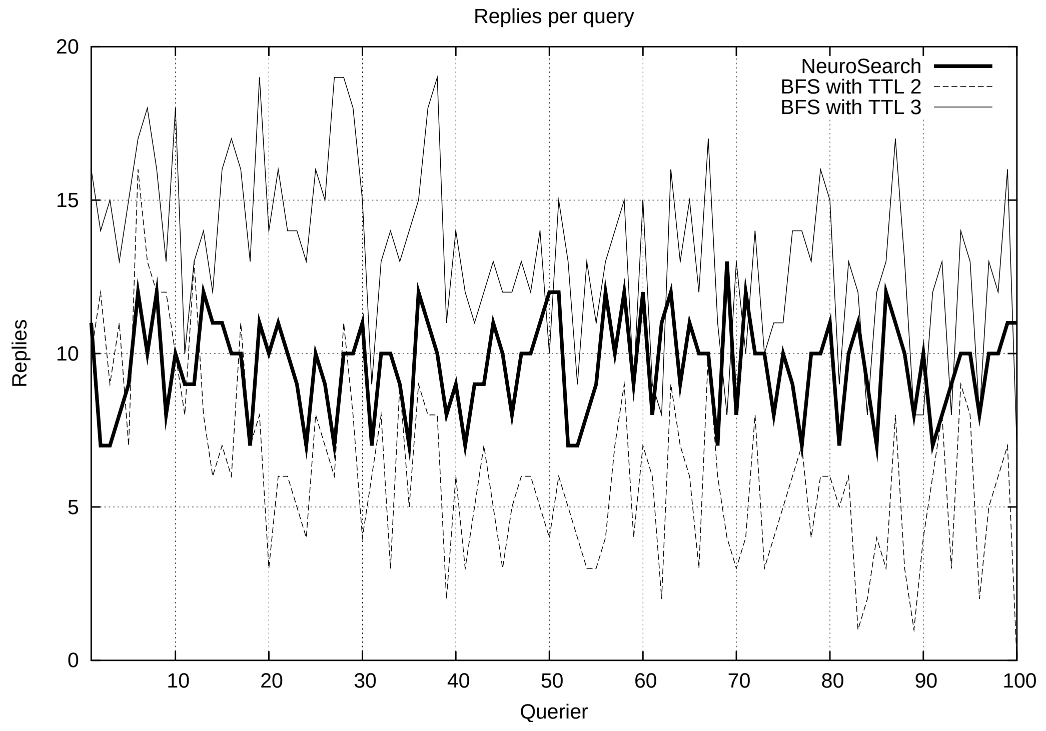

Fig. 4. Replies located by the algorithms.

Algorithm	*Replies to Packets ratio*
BFS-2	0.2063
BFS-3	0.1063
NeuroSearch	0.2040

Table 1
Replies to Packets ratio for BFS-2, BFS-3 and NeuroSearch

2 locating a new resource every 5th packet. BFS-3 locates a new resource approximately every 9th packet. Efficiency is easier to keep high when locating only few resources because usually those can be found from the central nodes alone. When the number of needed resources increases it becomes harder to locate more resources because then query has to travel more to the edges. Therefore the efficiency of BFS-3 decreases significantly.

For each query NeuroSearch locates approximately half of the resources which can be seen from Figure 5. There are 8 queries in which NeuroSearch misses the target to locate half of the resources. This variation results from the difference between the training set and the validation set datas. We verified that in queries 26 and 76 the queried resource was the same indicating that the particular resource being queried is harder to find than others for NeuroSearch. Still, however, the results indicate that optimization process has found an algorithm that is able to locate near half of the resources from the network with

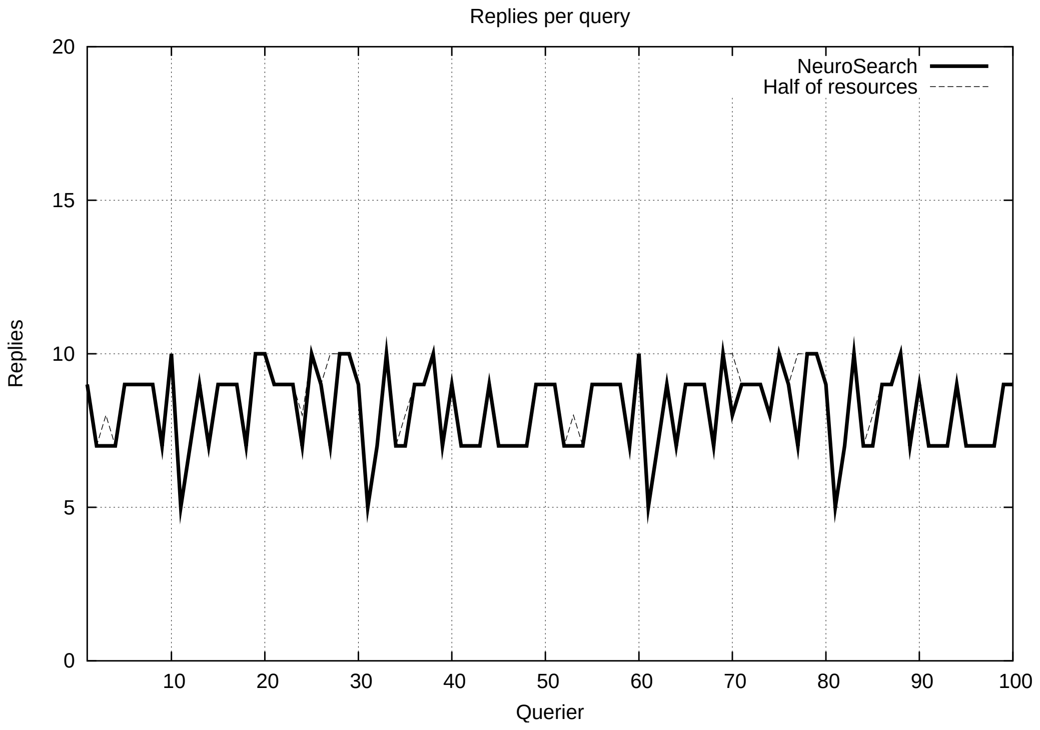

Fig. 5. Difference between half of the resources and NeuroSearch's replies.

high probability.

We analyzed the behavior of the best evolved neural network by tracking the path used by the queries. For searching in power-law graphs it has been suggested that a good strategy to locate resources is to first send the query to highest degree nodes and then gradually lower the degree when searching proceeds [7,8]. This ensures that if resources are likely to be found from the network's core the query finds them early in the search process.

However, the path NeuroSearch uses is not completely similar. NeuroSearch seems to prefer central nodes early in the query like expected, but after reaching the central nodes the spreading is stopped. The maximum number of hops is 5. This ensures that the central nodes are not revisited for the same query. Also at the beginning of query most of the neighbors will be used for querying ensuring that the query reaches the central nodes more likely than one depth-first search direction would. As a verification for this the behavior of a typical NeuroSearch query started from an edge node is illustrated in Figure 6.

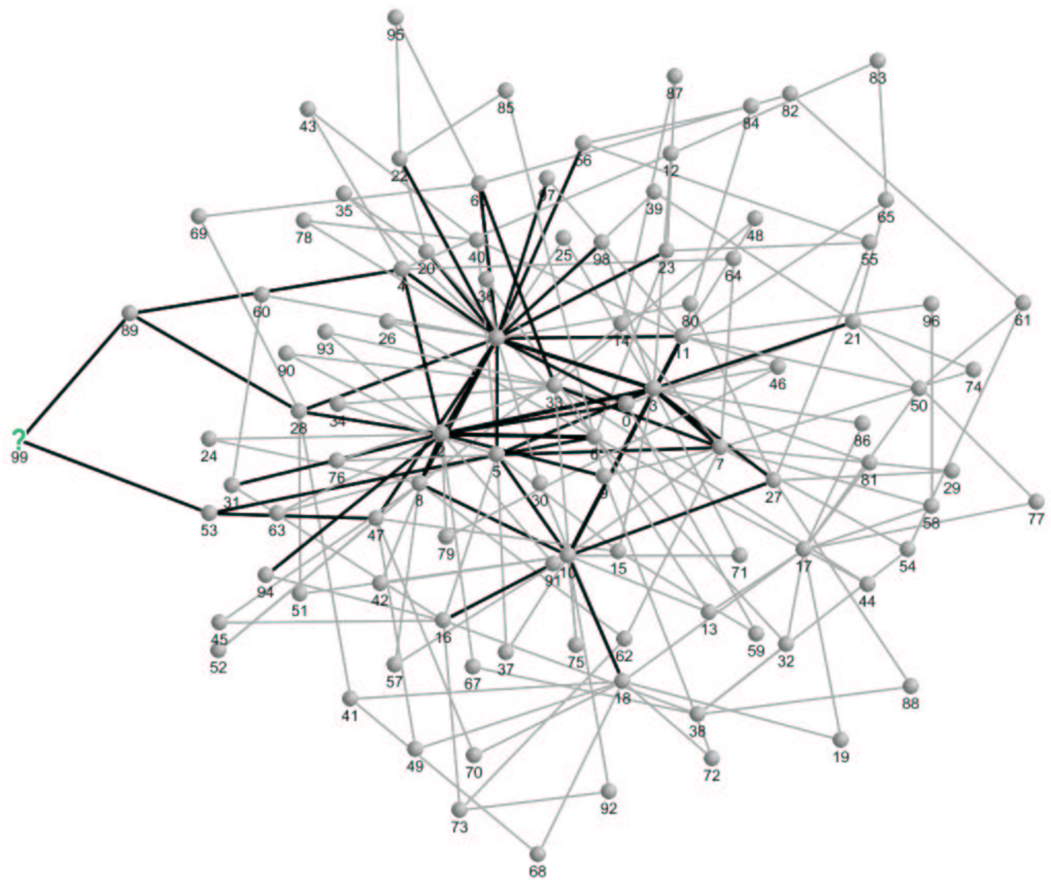

Fig. 6. Typical NeuroSearch resource query.

7 Conclusion

We have showed that a properly designed neural network can find a good search algorithm using simple optimization method. NeuroSearch algorithm takes into account the special characteristics of its environment and automatically adapts to different kind of P2P networks. At the same time the algorithm leaves out a behavior that is not beneficial to its evolution therefore making the algorithm robust to design errors. Also, since the process does not need human intervention after the input neurons have been defined the resulting algorithm is not limited to the designer's own ideas but only to the limitations of neural network's structure and its input neurons.

The behavior of the best evolved neural network was analyzed showing that the algorithm retains the effectiveness of BFS-2 and outperforms BFS-3. Also the algorithm provides a fine-grain control over the defined quality of search results and maintains a steady level of performance independent from the location where query is initiated. The analysis also showed some contradiction with the current research literature on searching in power-law networks and whether these results can be generalized to searching in all kinds of power-law networks will be a topic for further research.

While NeuroSearch performs well compared to BFS it is by no means yet designed to be optimal. For example, NeuroSearch does not yet include history-based inputs even though they would significantly improve the performance. Therefore, the research results obtained in [10–13] will be considered in forthcoming research on NeuroSearch. Also, there are other directions that were left out of this research. First, we are studying what improvements to the performance would be gained by varying the neural network's internal structure. Second, we are aiming to find out what are the scalability factors of NeuroSearch when the network size grows and third we are developing an optimal resource discovery algorithm using global knowledge to be able to measure the best efficiency a resource discovery algorithm can achieve. Also we are working on a solution to speed up the optimization process using parallel distributed computing and hybrid optimization method, which helps us determine more accurately what is the global performance maximum of NeuroSearch.

References

[1] N. A. Lynch, Distributed Algorithms, Morgan Kauffmann Publishers, 1996.

[2] Gnutella, http://gnutella.wego.com/.

[3] A. P. Engelbrecht, Computational Intelligence An Introduction, John Wiley & Sons, Ltd, 2002.

[4] K. Hornik, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.

[5] A. Jain, D. B. Fogel, Case studies in applying fitness distributions in evolutionary algorithms. ii. comparing the improvements from crossover and gaussian mutation on simple neural networks, in: Proceedings of the 2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, IEEE Press, 2000, pp. 91–97.

[6] A. Oram (Ed.), Harnessing the Power of Disruptive Technologies, O'Reilly, Sebastopol, CA, 2001.

[7] L. A. Adamic, R. M. Lukose, B. A. Huberman, Local search in unstructured networks, in: Handbook of Graphs and Networks: from the genome to the internet, Wiley-VCH, 2003, pp. 295–317.

[8] B. J. Kim, C. N. Yoon, S. K. Han, H. Jeong, Path finding strategies in scale-free networks, Physical Review E 65.

[9] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proceedings of the 16th International Conference on Supercomputing, ACM Press, 2002, pp. 84–95.

[10] V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yazti, A local search mechanism for peer-to-peer networks, in: Proceedings of the 11th International Conference on Information and Knowledge Management, ACM Press, 2002, pp. 300–307.

[11] B. Yang, H. Garcia-Molina, Improving search in peer-to-peer networks, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), 2002, pp. 5–14.

[12] D. Tsoumakos, N. Roussopoulos, Adaptive probabilistic search for peer-to-peer networks, in: Proceedings of the Third IEEE International Conference on P2P Computing (P2P2003), IEEE Press, 2003, pp. 102–109.

[13] D. Tsoumakos, N. Roussopoulos, A comparison of peer-to-peer search methods, in: Proceedings of the Sixth International Workshop on the Web and Databases (WebDB 2003), ACM Press, 2003, pp. 61–66.

[14] M. H. deGroot, Probability and Statistics, Addison Wesley Publishing Company, 1986.

[15] K. Chellapilla, D. Fogel, Evolving neural networks to play checkers without relying on expert knowledge, IEEE Transactions on Neural Networks 10 (6) (1999) 1382–1391.

[16] K. Chellapilla, D. Fogel, Evolving an expert checkers playing program without using human expertise, IEEE Transactions on Evolutionary Computation 5 (4) (2001) 422–428.

[17] D. B. Fogel, Blondie24: Playing at the edge of AI, Morgan Kaufmann, 2001.

[18] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512.

[19] A.-L. Barabási, Linked: The New Science of Networks, Perseus Publishing, 2002.

[20] M. A. Jovanovic, F. S. Annexstein, K. A. Berman, Scalability issues in large peer-to-peer networks - a case study of gnutella, Tech. rep., University of Cincinnati (2001).