Resource Discovery in P2P Networks Using Evolutionary Neural Networks

Mikko A. VAPA, Niko P. KOTILAINEN, Annemari K. AUVINEN,
Heikki M. KAINULAINEN, and Jarkko T. VUORI

***Abstract*-- Resource discovery is an essential problem in peer-to-peer networks since there is no centralized index in which to look for information about resources. One solution for the problem is to use a search algorithm that locates resources based on the local knowledge about the network. Traditionally, the search algorithms have been based on few simple rules, which often reduces the performance from optimal. In this paper, we describe the results of a process where evolutionary neural networks are used for finding an efficient search algorithm from a class of local search algorithms. The initial test results indicate that an evolutionary optimization process can produce search algorithm candidates that are competent compared to the breadth-first search algorithm (BFS) used in Gnutella peer-to-peer network.**

***Index Terms*-- resource discovery, peer-to-peer networks, multi-layer perceptrons, genetic algorithms.**

I. Introduction

IN the *resource discovery problem*, any node can possess resources and query these resources from other nodes in the network. The problem consists of graph with nodes, links and resources. Resources are identified by unique IDs and nodes may contain any number of resources. One node knows only the resources it is currently hosting. Any node in the graph can start a query, which means that some of the links are traversed based on a local decision in the graph. Whenever the query reaches the node with the queried ID, the node replies. The goal is to locate a predetermined amount of resource instances with a given ID using as few query packets as possible.

One possible solution for the resource discovery problem is the breadth-first search algorithm (BFS) [1]. In BFS a node that starts a query passes the query to all its neighbors. When the neighbors receive the query, they pass it further to all their neighbors except the one from which the query was received. Nodes cache the messages that they have received and if the query has already been received from other neighbor then query is dropped. Time-to-Live (TTL) value is used to limit the number of hops the query can take by reducing TTL value each time a query is received. When TTL decreases to zero the query is dropped. The BFS algorithm ensures that if a resource is located in the network it can be found from the network if TTL is high enough. The downside of the algorithm, however, is that it uses many query packets to find the needed resources. Thus, we propose an alternative algorithm that is more efficient in face of used query packets and evaluate it using peer-to-peer scenario with power-law distributed topology [2].

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

II. Related Work

Much research has been done regarding the resource discovery problem. Adamic et al. [3] and Kim et al. [4] 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. The algorithm locates resources efficiently if they can be found from the core of the network, but the performance decreases when the central nodes are revisited in search for lower degree nodes.

Lv et al. [5] evaluate BFS, expanding ring and random walk search mechanisms with varying topologies, including random graphs [2], power-law graphs and a snapshot of the Gnutella network obtained in October 2000. These researchers 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 TTL is increased. Expanding ring, where TTL is extended gradually for BFS, is the first aid to the problem. However, 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 by the researchers. A search initiates multiple walkers and forwards them based on a random selection of a neighbor. In addition to the TTL as a termination condition for the walkers, Lv et al. use checking, where the random walkers periodically check from the query originator whether the

Manuscript received September 2, 2004. This work was supported in part by the Graduate School in Electronics, Telecommunications and Automation (GETA) and Innovations in Business, Communication and Technology (InBCT) –project of Agora Center.

M. A. Vapa, A. K. Auvinen, and J. T. Vuori are with Department of Mathematical Information Technology, University of Jyväskylä, Finland (e-mail: firstname.lastname@jyu.fi).

N. P. Kotilainen is with Agora Center, University of Jyväskylä, Finland (e-mail: niko.kotilainen@jyu.fi).

H. M. Kainulainen is with WTS Networks, Jyväskylä, Finland (e-mail: heikki.kainulainen@wts.fi)

walker should be terminated or not. While random walkers increase the number of hops and thus latency, they decrease the total traffic because the search proceeds in a depth-first manner.

Kalogeraki et al. [6] consider two search algorithms for the resource discovery problem. The Modified Random BFS Search behaves like BFS, but the neighbors select only a random subset of neighbors for forwarding the query. This reduces traffic, but adjusting the correct size of the subset for various networks may be difficult. The researchers' 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. In another algorithm they present, called Intelligent Search Mechanism, the 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 [7] experimented with many types of directed search strategies based on various 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. Their work suggests that, to minimize 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 ten queries. Also, when considering the bandwidth use, the most reliable measure is the smallest average number of hops of received results for last ten queries. The heuristics used in the study are based on history data collected locally in each node.

Similar use of history data is found from the work by Tsoumakos and Roussopoulos [8]. In their proposal, called Adaptive Probabilistic Search algorithm, neighbors keep track of the success rates of earlier queries and forward random walkers probabilistically, based on the earlier 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, each of these algorithms uses some control parameters (for example time-to-live, the 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

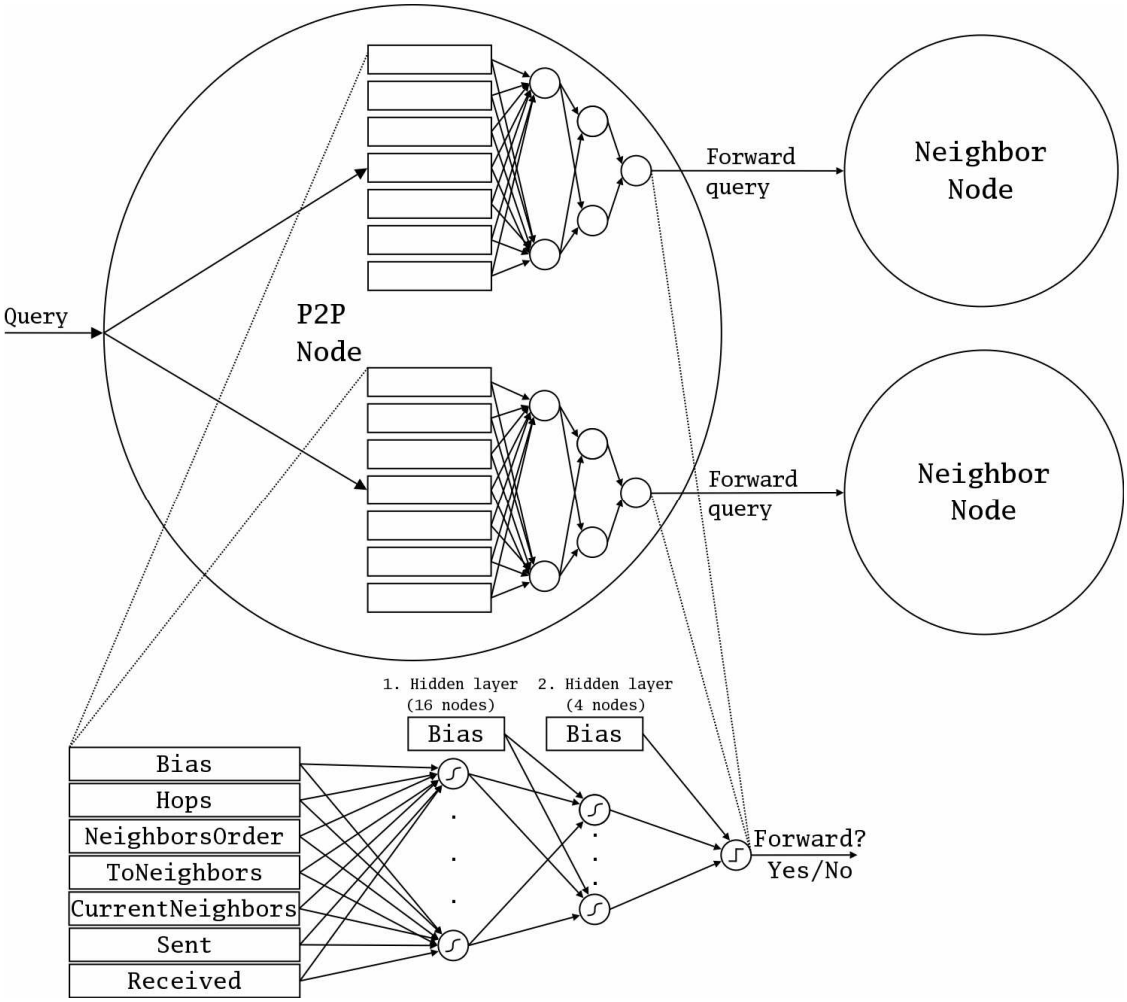

Fig. 1: Processing of NeuroSearch resource query and the NeuroSearch neural network

of control parameters should be kept to a minimal 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 entire 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 many combinations of strategies to locate resources. To authors' knowledge this is the first time when neural networks are being applied to resource discovery problem.

III. NeuroSearch Resource Discovery Algorithm

The proposed algorithm, called as *NeuroSearch*, makes decision to whom of the node's neighbors the resource request message is forwarded based on the output neuron of three-layer perceptron neural network. The algorithm is located inside a peer node as shown in Fig. 1 and is the same for all peers in the network. NeuroSearch can be represented as a function $O: I \rightarrow \{0,1\}$, where $I \in [0,1]^7$ is a 7-dimensional input vector representing the state of a resource discovery query. The output of *O* defines whether in a given state query should be dropped *O* = 0 or forwarded to a peer *O* = 1 and is evaluated for each neighbor peer separately.

When a resource request arrives to the algorithm it goes through all the node's neighbors (denoted as receivers) one by one with the neural network. The input parameters for the neural network are:

- *Bias* is the bias term and has value 1.
- *Hops* is the number of hops the message has travelled.
- *NeighborsOrder* indicates in which rank this receiver is in terms of number of neighbors compared to other neighbors. The connection with highest rank has the value of 0, second rank has the value of 1 and so on.
- *ToNeighbors* is the number of the receiver's neighbors.
- *CurrentNeighbors* is the number of node's neighbors.
- *Sent* has value 1 if the message has already been forwarded to the receiver. 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 *CurrentNeighbors* with $f(x) = \frac{1}{x}$ before giving them to the neural network. Scaling is performed to ensure that all the inputs are between 0 and 1.

There are two hidden layers in the network. In the first hidden layer there are 15 nodes + bias and in the second hidden layer 3 nodes + bias. Tanh is used as an activation function in the hidden layers: $t(a) = \frac{2}{1+e^{-2a}} - 1$, where *a* is the weighted sum of inputs to a neuron. Activation function in the output node is the threshold function $s(a) = \begin{cases} 0, a < 0 \\ 1, a \geq 0 \end{cases}$.

Combining all together, the output *O* of the neural network can be calculated with the following formula:

$$O = s(1 + \sum_{k=1}^{4} w_{3k} t(1 + \sum_{j=1}^{16} w_{2j} t(\sum_{i=1}^{7} w_{1i} f(I_i)))),$$

where I_i is the value of input parameter *i* and w_{xy} the neural network weights on layer *x* in position *y*.

Whenever the query locates a queried resource a reply message is sent back to the neighbor, which forwarded the request to the node. When all the nodes in the query path have forwarded the reply message backward, it is finally received by the query initiator.

IV. Neural Network Optimization

The weights w_{xy} are unknown and therefore they need to be adjusted to appropriate values. For doing this we use methods of evolutionary computing [9]. The decision, which neural networks are better than the others is done by counting the query packets traversed in the test network and found resources. The fitness for the neural network is defined in two parts. Each query *j* is scored for the 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* is defined with the following conditions:

1. If *packets* > 300 then *score* = 0
2. If *foundResources* = 0 then *score* = $1 - \frac{1}{packets + 1}$
3. If *foundResources* < *availableResources* / 2 and *foundResources* > *0* then *score* = 50 × *foundResources* – *packets*
4. If *foundResources* ≥ *availableResources* / 2 then *score* = 50 × *availableResources* / 2 – *packets*

In the equations *availableResources* is the maximum number of resource intances that can be located in the query, *foundResources* is the number of resource instances that the neural network was able to locate for the query, and *packets* is the number of query packets the neural network used for the query. The constant value 300 was set as criterion for determining when the neural network is considered to forward the query indefinitely and the query can be stopped. Another constant value, 50, was selected to be large enough to guide the training process towards neural networks that locate more resources than other neural networks. Now a neural network could spend 49 query packets more in a query to locate one additional resource compared to other neural network, which located one resource less.

The first rule ascertains that an algorithm that eventually

stops is always better than algorithm that does not. The goal of finding half of the available resource instances was set to demonstrate the algorithm's ability to balance on a predetermined quality of service level and not just on locating all resource instances or one resource instance. The second rule makes sure that if none of the resources are found then the neural network should increase the number of query packets sent to the network. The third rule states that if the number of found resources is not enough then the neural network develops only by locating more resources. Finally the last rule ensures that when half of the available resource instances are found from the network the fitness grows if neural network uses fewer query packets.

The optimization process had an initial population of 30 neural networks whose weights were randomly defined from interval [-0.2, 0.2]. Next, every neural network was tested in the peer-to-peer simulation environment and fitness value calculated. When all neural networks had been tested 15 best were chosen for mutation and used to breed the new generation of neural networks. As a result, 30 neural networks were available for testing the new generation.

Mutation was based on the Gaussian random variation and used weighted mutation parameter to improve the adaptability of the evolutionary search. The random variation function was similar to the one used by Fogel and Chellapilla in their research [10] and is given as:

$$\sigma_i'(j) = \sigma_i(j)\exp(\tau N_j(0,1)),\ j = 1,...,N_w,$$
$$w_i'(j) = w_i(j) + \sigma_i^j(j)N_j(0,1),\ j = 1,...,N_w,$$

where N_w = is the total number of weights and bias terms in the neural network, $\tau = \frac{1}{\sqrt{2\sqrt{N_w}}}$, $N_j(0,1)$ 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, $w_i'(j)$ is the new weight value and index $1 \le i \le 185$ denotes the number of neuron enumerated over all layers.

V. Simulation Environment

As a peer-to-peer simulation environment, we used Peer-to-Peer Realm (P2PRealm) network simulator [11] 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. P2PRealm has been implemented using Java.

In the test case we used power-law graphs generated using the Barabási-Albert model [12]. A power-law network's neighbor distribution follows the power-curve $P(k) = \frac{1}{k^{\gamma}}$, where $\gamma = 3$ for Barabási-Albert graph. Therefore in power-law networks there exist few hubs in the network that have many neighbors as well as many nodes that have only few neighbors. A power-law graph was selected because existing P2P networks have shown to express power-law dependencies [13]. The graphs tested contained 100 nodes with the highest degree node having 25 neighbors. Small network size was selected to allow visualisation of query paths in the network. Dynamic changes e.g., node failures were not taken into account to simplify the analysis. However, the approach can be applied in dynamic scenarios also as shown in [14].

The test case data was divided into three distinct data sets as described in [15]: a training set, a generalization set and a validation set. Training set is used for training the neural network. Generalization set is used to measure how well the trained neural network performs with a new data set indicating neural network's ability to generalize. When performance starts to decrease in generalization set the training can be stopped, because the neural network adapts only to the training set if training process is continued. Validation set is used as an objective measure to verify how well the algorithm performs with arbitrarily chosen new data set and ensures that the true generalization ability of the neural network is being measured.

The training set contained two power-law topologies with both being queried n = 50 times per generation for each neural network. Two topologies were used to have neural networks adapt to a wider range of situations than one topology would have provided. The generalization set consisted of two power-law topologies with 50 queries. When the performance started to decrease in the generalization set the neural network having highest fitness was selected and, as a validation set, one topology with 100 queries was used to produce the final simulation results.

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 instance of all resources and the lower degree nodes only some of these, randomly chosen from uniform distribution. The querying nodes and queried resources were selected also randomly from a uniform distribution for each query.

As stopping criteria for the optimization process, 100,000 generations were set. This seemed to take approximately two

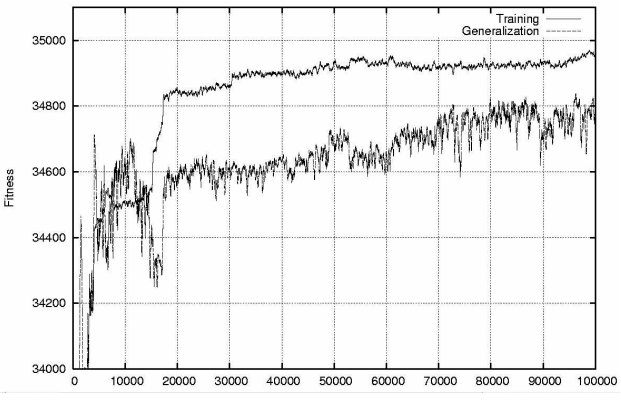

Fig. 2: Evolution of the best neural networks in each generation for training and generalization sets

weeks on our desktop PC equipped with an AMD Athlon XP 1800 processor. The evolution of the best neural network in each generation is shown in Fig. 2.

VI. Simulation Results

To evaluate the difference between BFS and NeuroSearch, we selected the best algorithm at the 85,736th generation and calculated the number of packets used and found resources for 100 different queries using validation set. The 85,736th generation was selected because between the 80,000 and 90,000 generations the neural networks had achieved steadily good results and, in particular, in the 85,736th generation, neural network had the best fitness. The results are presented in Fig. 3 and Fig. 4.

The results of Fig. 3 show that the performance of NeuroSearch regarding the number of packets is nearer to BFS with a time-to-live value 2 (BFS-2), rather than BFS with a time-to-live value 3 (BFS-3). In average NeuroSearch consumes 47.2 packets per query whereas BFS-2 consumes 30.0 and BFS-3 122.0 packets. The reason why there is some variation in the number of packets for successive BFS queries is that the number of delivered packets 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 which node is querying.

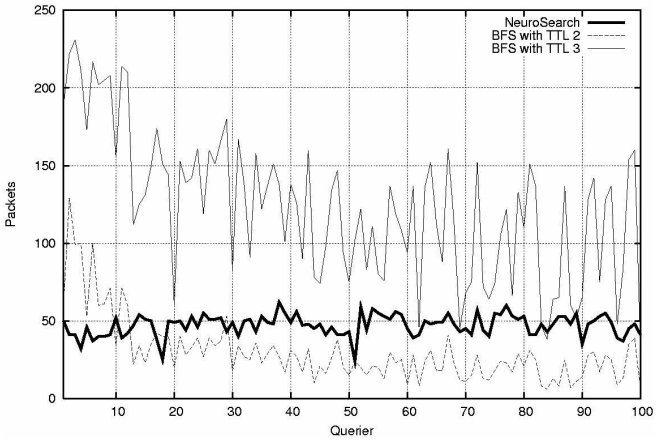

Fig. 3: Number of packets used by the algorithms

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

By calculating the ratio between the located resources and used query packets we can determine the efficiency of the algorithms. These values are shown in Table I. The results show that NeuroSearch's efficiency is at the same level as BFS-2's locating a new resource every fifth packet. BFS-3 locates a new resource approximately every ninth 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, query has to spread more to the edges to locate the additional resources. Therefore the efficiency of BFS-3 decreases significantly. BFS-2 and NeuroSearch achieve near similar efficiency indicating that NeuroSearch is able to sustain a good efficiency even though it needs to locate more resources than BFS-2.

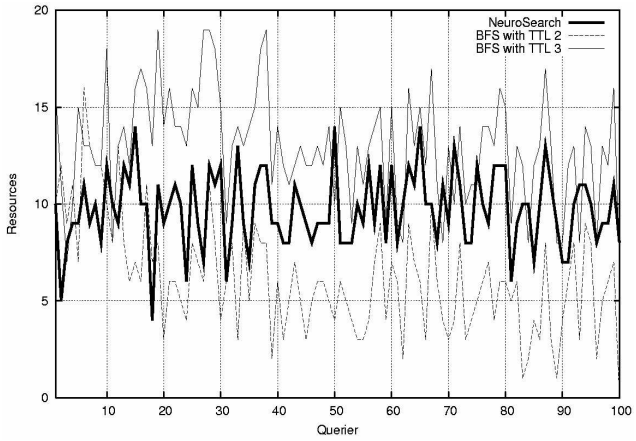

Fig. 4: Number of resources found by the algorithms

TABLE I
Efficiency of the Algorithms

Algorithm	Packets	Resources	Efficiency
BFS-2	3000	619	0.2063
BFS-3	12202	1295	0.1061
NeuroSearch	4719	975	0.2066

For each query, NeuroSearch locates approximately half of the resources or more, which can be seen in Fig. 5. There are six queries in which NeuroSearch misses the target to locate half of the resources. This variation results from the difference

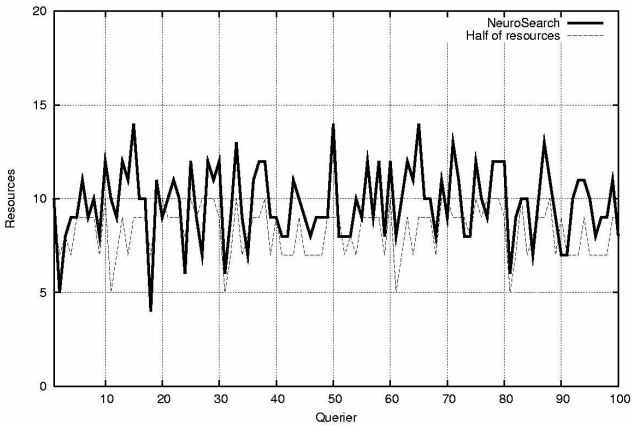

Fig. 5: Difference of located resources to half of resources

between the training set and the validation set. Nonetheless, the results indicate that the optimization process has found an algorithm that is able to locate nearly half of the resources from the network with high probability.

We analyzed the behavior of the best-evolved neural network by tracking the path used by the queries. NeuroSearch seems to prefer central nodes early in the query and uses multiple paths for doing this. After reaching central nodes or one hop later the spreading is stopped. The maximum number of hops is 5. As verification for this the behavior of a typical NeuroSearch query started from an edge node is illustrated in Fig. 5. In the figure the query travels through the connections denoted with a black line starting from node 99 with question mark (?). Nodes marked with an exclamation mark (!) contain the queried resource. In total the query uses 49 packets and locates 11 resources. Six connections are traversed from both directions, which is not shown in the figure.

VII. Conclusion

In this paper, a new resource discovery algorithm has been proposed. NeuroSearch algorithm takes into account the special characteristics of its environment and can be adjusted to different kind of P2P networks. The algorithm's performance is also stable and competitive compared to the BFS algorithm.

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 results obtained in [3]-[8] will be considered in forthcoming research on NeuroSearch. There are also 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 by parallelizing the evolutionary algorithm using distributed computing. This helps us to more accurately determine the performance maximum of NeuroSearch.

Acknowledgment

The authors would like to thank the co-designers of NeuroSearch Joni Töyrylä, Yevgeniy Ivanchenko, Matthieu Weber and Hermanni Hyytiälä. Also we thank Tommi Kärkkäinen for giving useful hints how to develop the algorithm further and Barbara Crawford for proofreading the article.

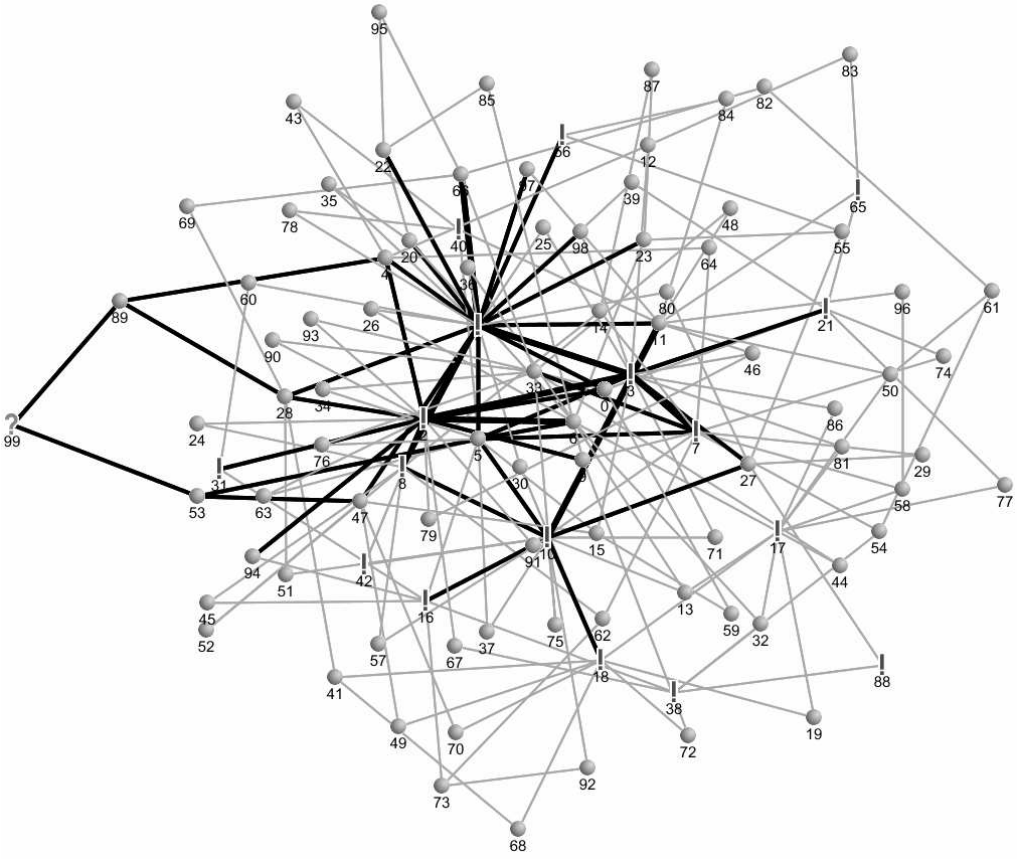

Fig. 5: Typical NeuroSearch resource query

REFERENCES

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

[2] A. Barabási, *Linked*, Perseus Publishing, 2002.

[3] L. A. Adamic, R. M. Lukose, and 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.

[4] B. J. Kim, C. N. Yoon, S. K. Han, and H. Jeong, "Path finding strategies in scale-free networks", *Physical Review E 65*, 2002.

[5] Q. Lv, P. Cao, E. Cohen, K. Li, and 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.

[6] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yatzi, "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.

[7] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[8] D. Tsoumakos and 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.

[9] K. Miettinen, M. Mäkelä, and P. Neittaanmäki and J. Périaux (eds.), *Evolutionary algorithms in engineering and computer science*, John Wiley & Sons, 1999.

[10] K. Chellapilla and D. Fogel, "Evolving neural networks to play checkers without relying on expert knowledge", *IEEE Trans. on Neural Networks*, 10 (6), pp. 1382-1391, 1999.

[11] J. Töyrylä, *Building NeuroSearch – Intelligent Evolutionary Search Algorithm For Peer-to-Peer Environment*, Master's Thesis, University of Jyväskylä, 2004.

[12] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks", *Science 286* (1999) 509-512.

[13] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman, *Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella*, Technical report, University of Cincinnati, 2001.

[14] Y. Ivanchenko, *Adaptation of Neural Nets For Resource Discovery Problem in Dynamic And Distributed P2P Environment*, Master's Thesis, University of Jyväskylä, 2004.

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