Resource Discovery in P2P Networks Using Evolutionary Neural Networks

Mikko Vapa, Niko Kotilainen, Heikki Kainulainen, Jarkko Vuori

Indexing terms: resource discovery, peer-to-peer networks, feed-forward neural networks, evolutionary computing

Resource discovery is an essential problem in peer-to-peer 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 knowledge about the network. Traditionally, these search algorithms have been based on few predetermined rules. The problem with these algorithms is that if the conditions in the network change the algorithm becomes less efficient and won't 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. The initial test results indicate that an evolutionary optimization process can produce search algorithm candidates that are more efficient compared to breadth-first search algorithm (BFS) used in Gnutella peer-to-peer network.

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 IDs and nodes can contain any number of resources. One node knows only the resources it is currently hosting. A 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 and whenever the query reaches the node with the queried ID, the node replies.

One possible solution for the resource discovery problem is the breadth-first search algorithm (BFS) [1]. In the BFS approach 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.

The BFS algorithm ensures that if a resource is located in the network it can also be found from the network. The downside of the algorithm, however, is that it uses lots of query packets to find the needed resources. Thus, we propose an alternative algorithm that is more efficient in face of used packets.

The 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 3-layer perceptron neural network.

When a resource request arrives to the algorithm it goes through all the node's neighbor connections one by one with the neural network. The input parameters for the neural network are:

- *Bias* is the bias term.
- *Hops* is the number of the hops in the message.
- *NeighborsOrder* tells in which rank this connection is in terms of number of neighbors compared to others. The connection with best rank has the value of 0.
- *Neighbors* is the number of the connection's neighbors.
- *MyNeighbors* is the number of node's neighbors.
- *Sent* has value 1 if the message has already been forwarded to the connection. Otherwise it has value of 0.
- *Received* has value 1 if the message came to the node from the connection, 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 made 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 layer 3 + bias. Tanh is used as an activation function $t(a) = \frac{2}{1+e^{-2a}} - 1$ in the hidden layers. Activation function in the output node is the threshold function $s(a) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$.

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

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.

Neural network optimization: To find the appropriate weights we use methods of evolutionary computing [2]. The decision, which neural nets are better than the others is done by counting the packets sent to the test network and replies received. 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 *replies* ≥ *availableResources* / 2 then *score* = 50 × *availableResources* / 2 – *packets*
2. If *replies* < *availableResources* / 2 and *replies* > *0* then *score* = 50 × *replies* – *packets*
3. If *replies* = 0 then *score* = 1 – 1 / (*packets* + 1)
4. If *packets* > 300 then *score* = 1 / (*packets* + 1)

The first rule ensures that when half of the resource instances are found from the network the fitness grows if neural network uses fewer packets. The second rule states

that if the number of found resources is not enough then the neural network develops if it gains more resources than the others. The third rule makes sure that if none of the resources are found then the neural network should increase the number of packets sent to the network. Finally the last rule ascertains that an algorithm that eventually stops is always better than algorithm that does not.

The optimization process had an initial population of 30 neural networks whose weights were randomly defined. 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 nets. 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 [3]. After 100.000 generations, the optimization process was stopped.

In the peer-to-peer simulation environment we used power-law graphs generated using the Barabasi-Albert model [4]. In power-law networks there exists few hubs in the network that have many neighbors and lots of nodes that have only few links. The power-law graph was selected because some existing P2P networks have shown to express power-law dependencies [5]. 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 [6]: 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 topologu 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 resources and querying nodes were selected also randomly from uniform distribution for each query.

Simulation results: To evaluate the difference between BFS and NeuroSearch the number of packets used and received replies for 100 different queries were calculated using validation set. The results are presented in Figure 1 and Figure 2.

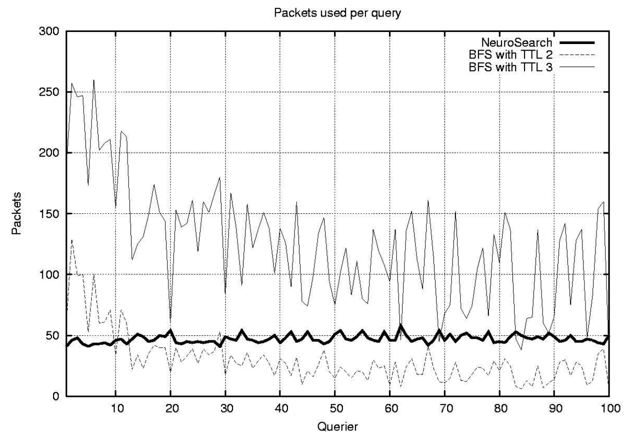

Figure 1: Number of packets used by the algorithms

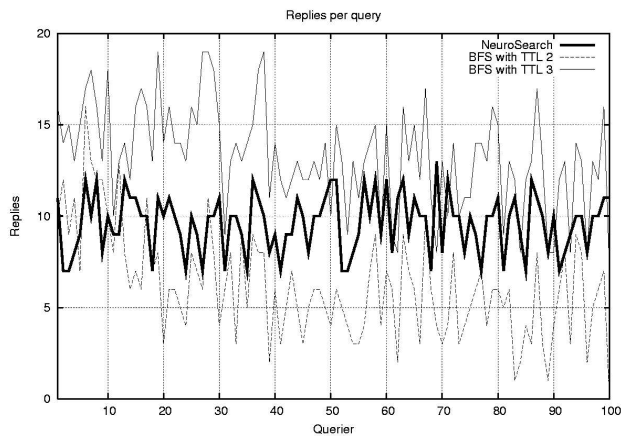

Figure 2: Number of replies used by the algorithms

The results of Figure 1 show that the performance of NeuroSearch in number of packets is nearer to BFS with time-to-live (TTL) value 2 than 3. In average 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 2 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.

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] 0. 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, but uses multiple paths for doing this. After reaching the central nodes the spreading is stopped and the maximum number of hops is 5. As a verification for this the behavior of a typical NeuroSearch query started from an edge node is illustrated in Figure 3.

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

References

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

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

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

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

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

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

[7] Lada A. Adamic, et. al, "Search in power-law networks", *Physical Review E*, 64, 2001

[8] B. J. Kim, C. N. Yoon, S. K. Han, H. Jeong, "Path finding strategies in scale-free networks", Physical Review E 64

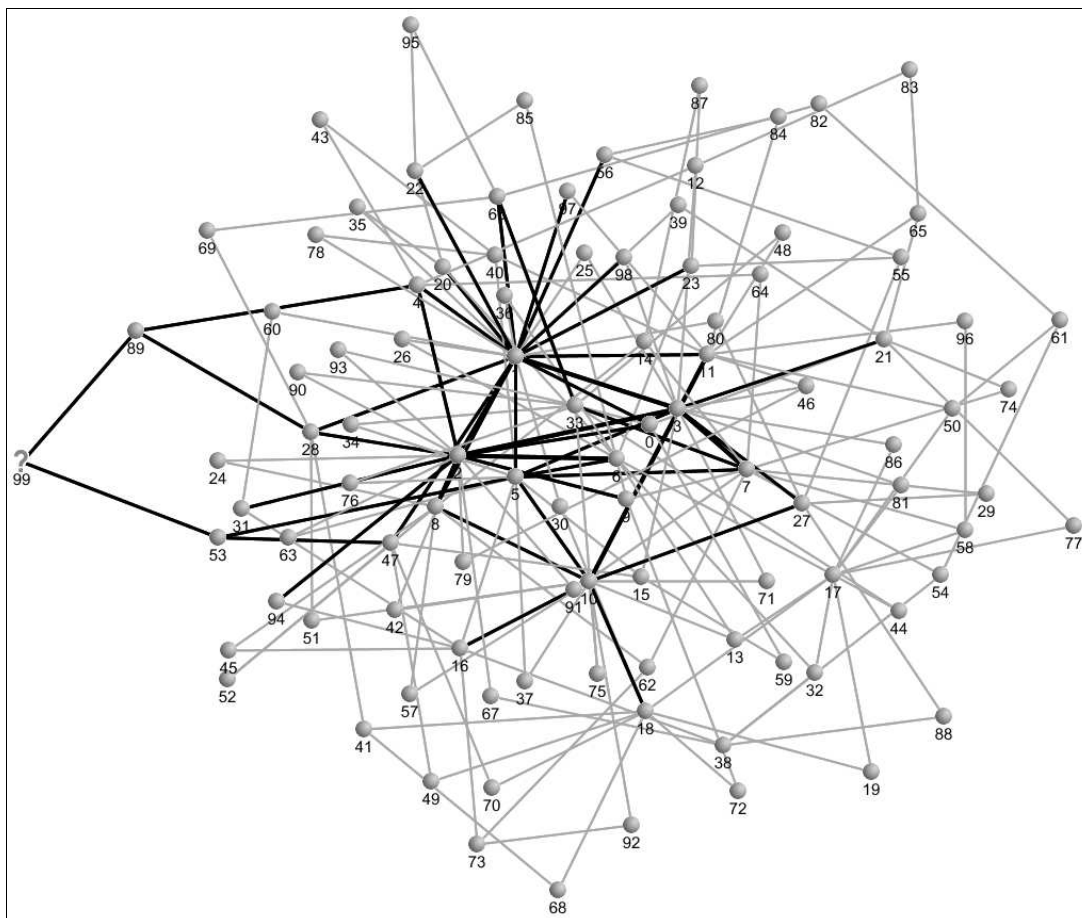

Figure 3: Typical NeuroSearch resource query