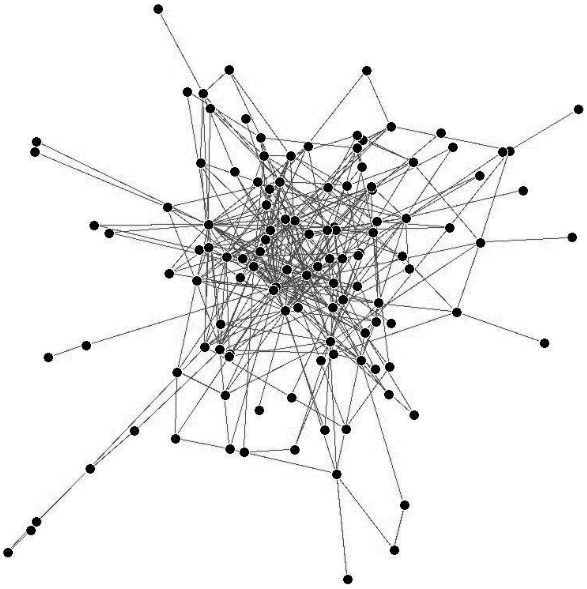
Parallelization of Evolutionary Algorithms in Peer-to-Peer Environment

Annemari Auvinen
Niko Kotilainen
Mikko Vapa

15.12.2003

Exercise Work for
TIE331 Parallel and Distributed Computing –course



Contents

1 Introduction to Peer-to-Peer Networks

Recently, the peer-to-peer (P2P) paradigm for building distributed applications has gained attention from both industry and the media. The classical definition of peer-to-peer states that a P2P system is composed of a distributed collection of peer nodes where each node acts both as a server and a client. Thus the nodes may provide services to other peers and may consume services from other peers. P2P is completely different from the client-server model, where few specialized servers provide services to a large number of clients.

Despite its poor reputation, P2P is extremely interesting from a technical point of view. Its completely decentralized model enables the development of applications with characteristics of **high-availability**, **fault-tolerance** and **scalability** that are previously unseen in Internet.

Peer-to-peer paradigm exploits what has been defined as the “dark matter” of Internet. Dark matter consists of the unused CPU and storage capacity that can be found from the idling PCs. With the means of peer-to-peer resource discovery this matter can be located and used efficiently.

File sharing has been the main application area for peer-to-peer. For example in 2003 KaZaA Media Desktop of Sharman Networks hit the most downloaded software record with 230 million downloads. Moreover, P2P is not limited to file sharing, but it can be applied to e.g., distributed computing and collaboration tools. Also some researchers see P2P as a robust information diffusion media in the future that can change the way content is distributed over the Internet.

2 NeuroSearch – Neural Network Search Algorithm

In peer-to-peer networks the main problem is how to locate resources when there is no centralized index to look for. This can be done using resource discovery search algorithms. One such algorithm is NeuroSearch [1], which has been developed in University of Jyväskylä in Cheese Factory research project [2]. Cheese Factory is part of Innovations in Business, Communication and Technology (InBCT) venture [3] in Agora Center and is funded by Nokia, TeliaSonera, Tietoenator, Metso Paper and Jyväskylä Science Park.

NeuroSearch uses multi-layer perceptron neural network as shown in Figure 1.

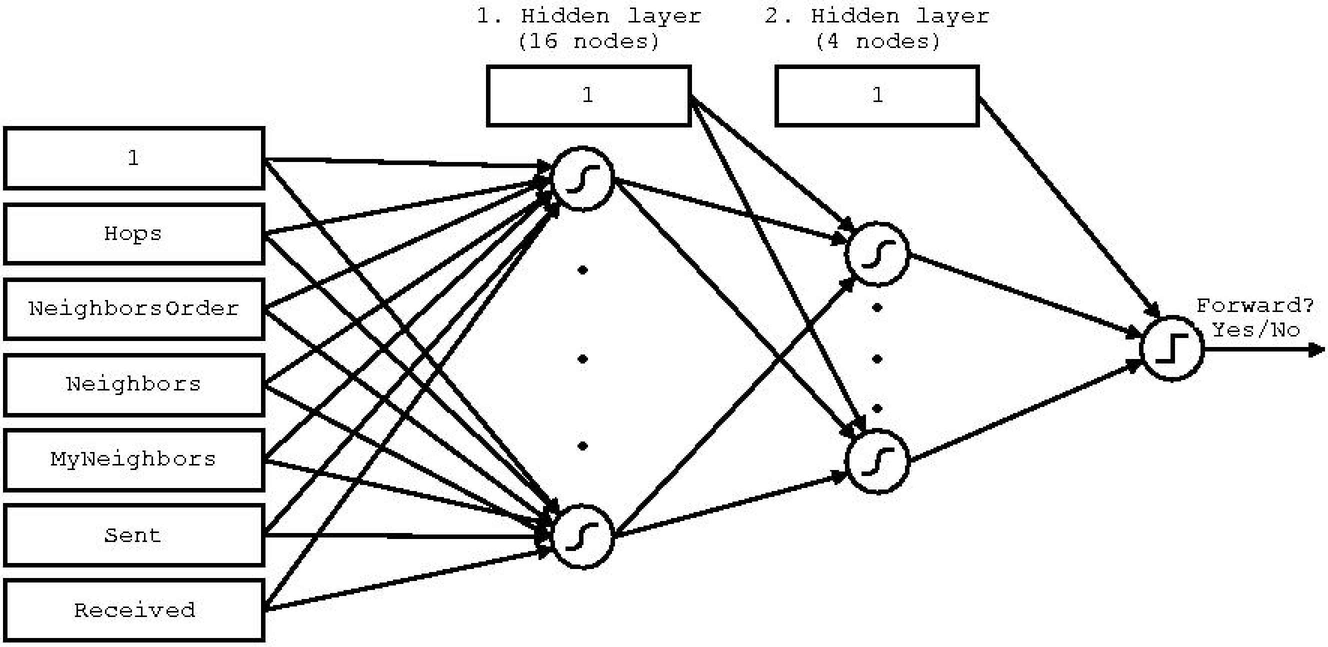


Figure 1. NeuroSearch neural network's structure.

Before NeuroSearch algorithm can be used the neural network's weights need to be optimized. This can be done using evolutionary computing.

3 Optimization of Neural Network

Neural network's structure is optimized using iterative optimization process based on evolutionary algorithm. This process is illustrated in Figure 2.

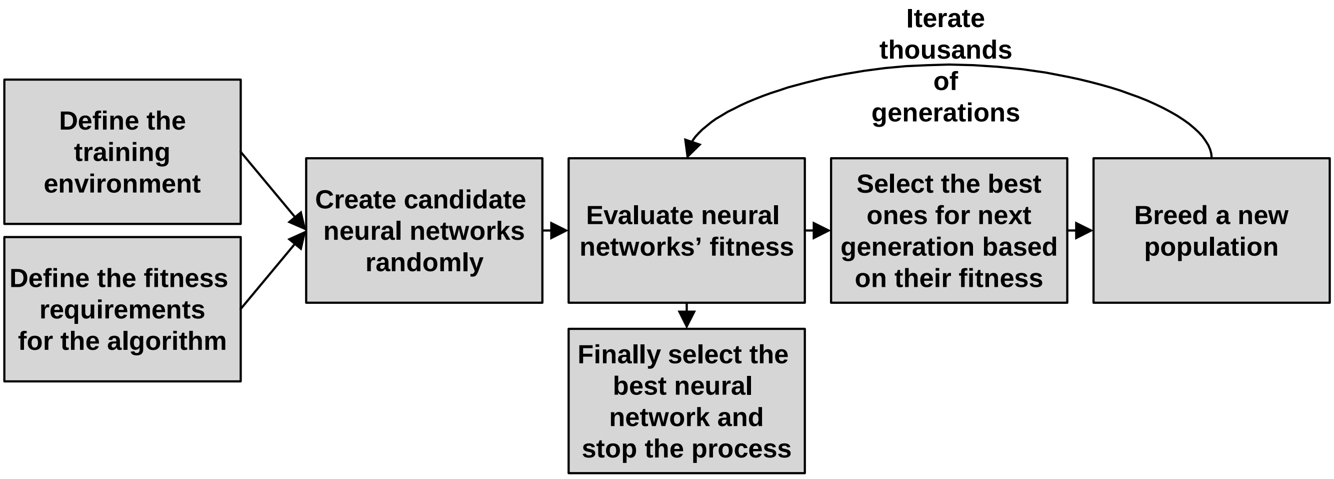

Define the training environment
Define the fitness requirements for the algorithm
Create candidate neural networks randomly
Evaluate neural networks' fitness
Select the best ones for next generation based on their fitness
Breed a new population
Iterate thousands of generations
Finally select the best neural network and stop the process

Figure 2. Evolutionary optimization process.

Evolving neural networks needs a lot of computing power and in NeuroSearch's case this may take about one week with a standard desktop PC. This is because the optimization process is sequential and thus can be run only on one computer. To study for example how the number of neurons in the neural network affect the neural network's performance multiplies the time with the number of different neuron settings. For a thorough investigation of neural network the total time scale shifts from days to months and perhaps years.

However, there are also ways to parallelize evolutionary algorithms because the population can be distributed and the fitness evaluation is independent from other population members. Therefore we sought for other alternatives to speed up the optimization process.

4 Parallel Evolutionary Algorithms

There are multiple ways how evolutionary algorithms can be parallelized. The most suitable ones for our case we found two alternatives that were presented in [4]: master-slave parallel evolutionary algorithm and multiple-population parallel evolutionary algorithm.

Master-slave parallel evolutionary algorithm as shown in Figure 3 uses one population and distributed fitness evaluation. This approach is efficient if fitness evaluation is computationally demanding but scales poorly because of master. The parallel version of the algorithm retains the converge properties of the sequential algorithm and therefore they are equivalent to each other in sense of optimization performance.

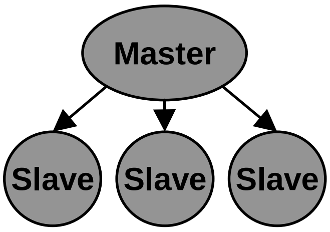


Figure 3. Master-slave parallel evolutionary algorithm.

Multiple-population parallel evolutionary algorithm presented in uses multiple populations and infrequent migration between populations. Algorithm is efficient if population can be divided and scales well on large population sizes. However, because the optimization proceeds without synchronization between populations the parallel version of the algorithm changes the converge properties compared to the sequential algorithm. Therefore a number of researchers have been able to measure super-linear speed-ups [5] when the parallel version of the algorithm converges better than the sequential version.

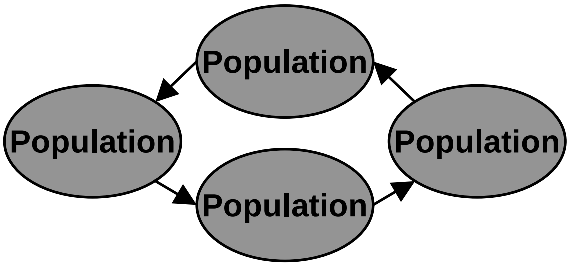


Figure 4. Multiple-population parallel evolutionary algorithm.

5 Solution Using Chedar P2P Platform

As computation resources we planned to use the publicly available desktop computers in the Agora building and thus defined two requirements for the software:

- The solution had to be as invisible as possible to the user of the neural network training program

- The computation should not interfere with the desktop use of the distributed computers

The solution we came up with is based on Chedar peer-to-peer platform developed also in the Cheese Factory –project. Chedar is a Peer-to-Peer (P2P) platform written in Java for searching resources from the distributed network and building P2P applications. Resources can be for example processor power or files. Because of the peer-to-peer design the distributed system does not have any central points.

Chedar is built over TCP-sockets and it uses the architecture shown in Figure 5 and Figure 6.

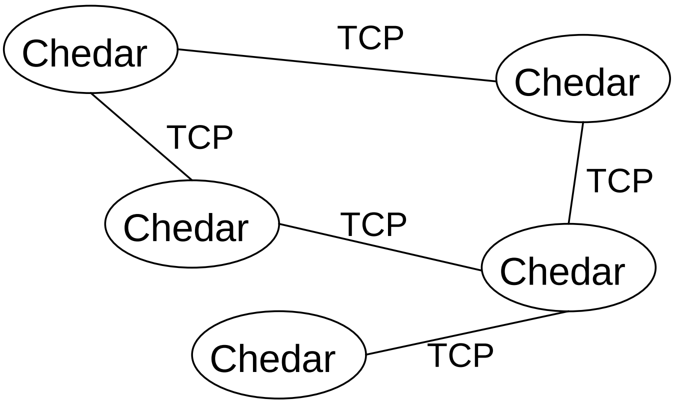


Figure 5. Chedar peer-to-peer network architecture.

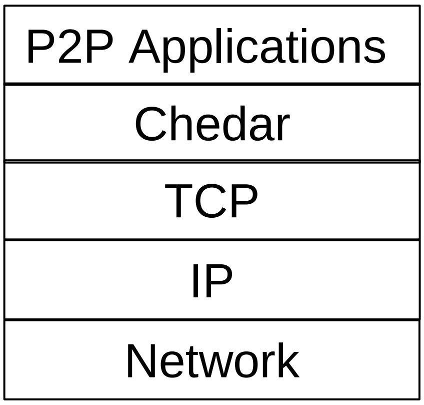
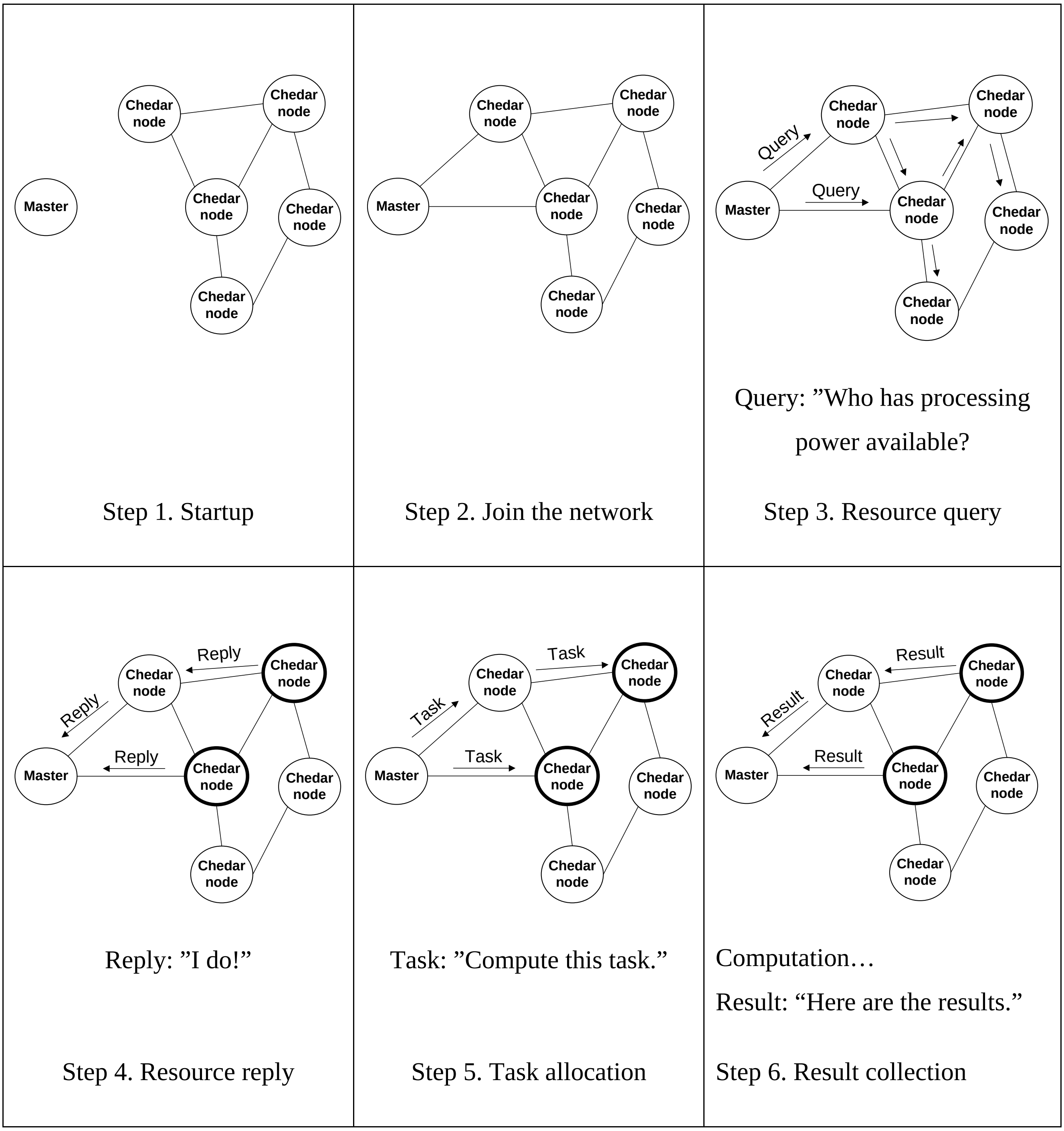


Figure 6. Chedar's layer-wise architecture.

The computation is distributed using the following process. First the node that wants to distribute its computation joins the Chedar network (steps 1 and 2). Next it sends a query to locate free computation resources (step 3) and gets responses from idling nodes (step 4). Then by selecting the most suitable nodes for processing the task it divides the computation tasks and distributes them to the selected nodes (step 5). Finally when the computation is over the nodes send their results back to the initiating node that started the computation (step 6). Calculation ends and everyone is happy.



Step 1. Startup

Step 2. Join the network

Query: ”Who has processing power available?

Step 3. Resource query

Reply: ”I do!”

Step 4. Resource reply

Task: ”Compute this task.”

Step 5. Task allocation

Computation…

Result: “Here are the results.”

Step 6. Result collection

What happens inside a Chedar node is that when Chedar node starts the distributed program its file operations are hijacked. Therefore any Java program that uses files to read input and store output can be distributed without any extra work. The communication between Chedar and distributed program is shown in Figure 7.

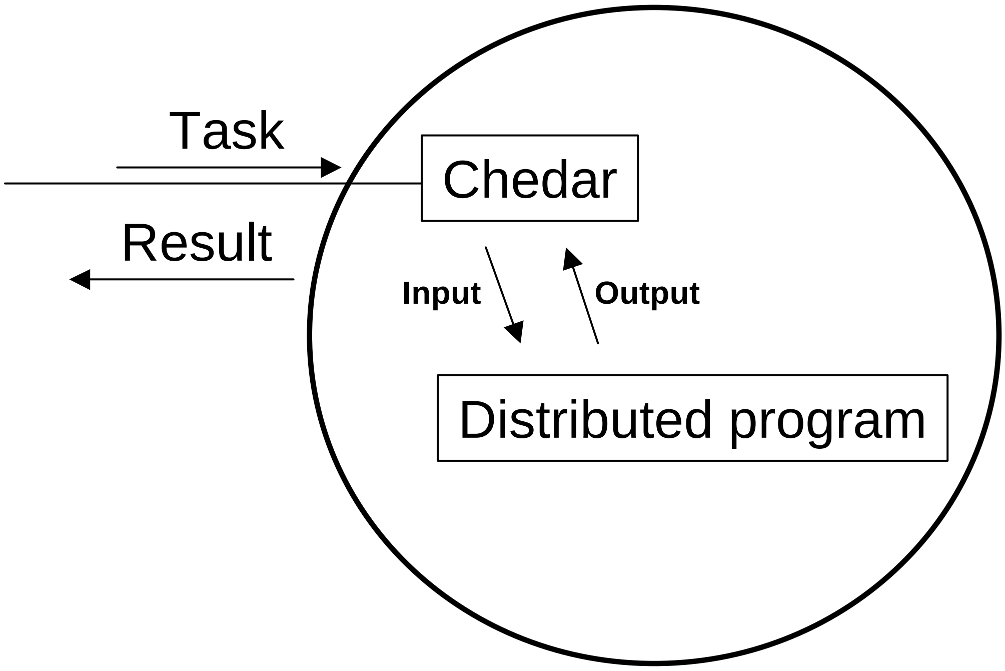


Figure 7. Input and output communication using file streams.

There are clear advantages of this kind of a peer-to-peer design over ordinary clusters:

- The costs are minimal costs because there is no need for new hardware and only the processing power that is idling is being used

- The approach is can handle dynamic scenarios and achieves better fault-tolerance because computers can join or leave the network at any time

- The solution is scalable because peer-to-peer network size can be huge

6 Test Scenario and Results

To test the solution we computed twice the running times of a typical NeuroSearch optimization case where 8 tasks were distributed to 1, 2, 4 and 8 machines in the computer class on Agora's 5th floor. All the 8 machines were identical and running AMD Athlon XP 2400+ processors.

The minimum time of the two runs was taken into account when calculating speedup, efficiency and serial fraction values. These three measures are the most commonly used when measuring the performance of parallel programs [6].

Speedup is defined as $s = \frac{T(1)}{T(p)}$, where *T(1)* is the time used for processing the tasks with one processor and *T(p)* is the time used for processing the tasks with *p* processors.

Efficiency is defined as $e = \frac{s}{p}$, where *s* is the speedup with *p* processors and *p* is the number of processors.

Serial fraction is defined as $f = \frac{1/s - 1/p}{1 - 1/p}$, where *s* is the speedup with *p* processors and *p* is the number of processors.

The results of the tests are shown in Table 1.

Processors	Time (sec)	Speedup	Efficiency	Serial fraction
1	53744	-	-	-
2	26430	2.0334468	1.016723	-0.016448348
4	11330	4.7435128	1.185878	-0.052247693
8	6622	8.115977	1.014497	-0.002041424

Table 1. Results of the test scenario.

The results indicate that because the speedup is greater than number of processors the computation is useful to distribute at least for 8 machines. Serial fraction is below zero meaning that super-linear speedup occurs. Efficiency is highest and serial fraction is lowest with 4 processors thus suggesting that the price to performance is best when the tasks are distributed to 4 processors.

7 Conclusions and Future Work

As a conclusion it seems that the distribution over Chedar peer-to-peer platform was successful. Still there are many things that still require future work to make the system complete.

First, we are planning to make a larger deployment of the system to more machines and see how the system performs.

Second, the system would benefit of some improvements that were not done yet. For example it should be possible that master can leave the network and gather results afterwards when it rejoins the network. The tasks should be load-balanced between peer nodes if one machine stops idling when users start to use the machine and the intermediate results should be sent to the initiating node thus avoiding the need to restart computation when failures or computer resets occur.

Third, in case that there would be a need for example to parallelize computation using multiple-population evolutionary algorithm the system should provide means for the Chedar nodes to communicate directly with each other rather than just using the initiating node for the communication. Also a monitoring interface to measure how computation is proceeding would be beneficial to provide user more control over the computation process.

References

[1] Vapa M., Kotilainen N., Auvinen A., Töyrylä J., Hyytiälä H., Vuori J., ”NeuroSearch: evolutionary neural network resource discovery algorithm for peer-to-peer networks”, being submitted to Elsevier Science Ad Hoc Networks Journal, December 2003

[2] Cheese Factory -project, tisu.it.jyu.fi/cheesefactory

[3] Innovations in Business, Communication and Technology (InBCT), Agora Center, University of Jyväskylä, www.jyu.fi/agora-center/inbctF.html

[4] Cantú-Paz E., ”A Survey of Parallel Genetic Algorithms”, 1998

[5] Alba E., ”Parallel evolutionary algorithms can achieve super-linear performance”, Elsevier Science Information Processing Letters 82(1), 2002

[6] Karp A. H., Flatt H. P., ”Measuring Parallel Processor Performance”, Communications of the ACM 33(5), 1990