Growing Neural Gas – A Parallel Approach

Lukáš Vojáček, Jiří Dvorský

Growing Neural Gas – A Parallel Approach

Lukáš Vojáček[1] and Jiří Dvorský[2]

[1] IT4Innovations Centre of Excellence
Ostrava, Czech Republic
lukas.vojacek@vsb.cz
[2] Department of Computer Science,
VŠB – Technical University of Ostrava, 17. listopadu 15,
708 33 Ostrava, Czech Republic
jiri.dvorsky@vsb.cz

Abstract. The paper deals with the high dimensional data clustering problem. One possible way to cluster this kind of data is based on Artificial Neural Networks (ANN) such as SOM or Growing Neural Gas (GNG). The learning phase of the ANN, which is time-consuming especially for large high-dimensional datasets, is the main drawback of this approach to data clustering. The parallel modification, Growing Neural Gas, and its implementation on the HPC cluster is presented in the paper. Some experimental results are also presented.

Keywords: growing neural gas, high-dimensional dataset, high performance computing

1 Introduction

Recently, the issue of high-dimensional data clustering has arisen together with the development of information and communication technologies which support growing opportunities to process large data collections. High dimensional data collections are commonly available in areas like medicine, biology, information retrieval, web analysis, social network analysis, image processing, financial transaction analysis and many others.

Two main challenges should be solved to process high-dimensional data collections. One of the problems is the fast growth of computational complexity with respect to growing data dimensionality. The second one is specific similarity mea-surement in a high-dimensional space. As presented in [1], Beyer et al. for any point in a high-dimensional space the expected distance, computed by Euclidean measure to the closest and to the farthest point, shrinks with growing dimensionality. These two reasons reduce the effectiveness of clustering algorithms on the above-mentioned high-dimensional data collections in many real applications.

The authors propose an effective data clustering algorithm which is based on *Growing Neural Gas* (GNG) [7]. The computational complexity is resolved by the parallel implementation of GNG. Some technical problems have to be resolved in

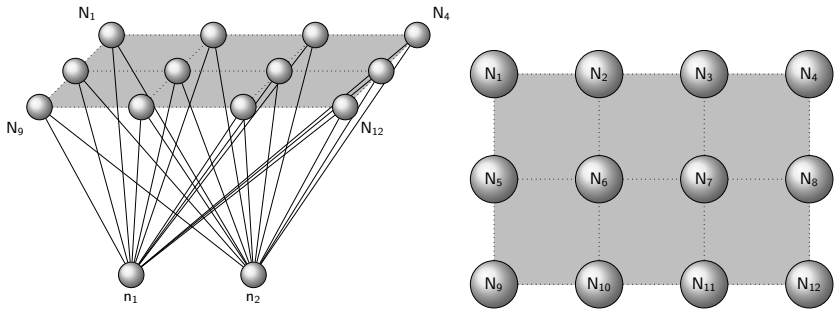
order to effectively train such kind of neural network using an *High Performance Computing* (HPC) cluster with MPI. The traditional serial approach to training GNG is also considered in the paper. The serial learning GNG algorithm is used for benchmarking the parallel version of GNG. In other words, parallel GNG has to produce the same network and should be an order of magnitude faster in the ideal case.

2 Artificial Neural Networks

2.1 Self Organizing Maps

Self Organizing Maps (SOM), also known as Kohonen maps, were proposed by Teuvo Kohonen in 1982 [4]. SOM is a kind of artificial neural network that is trained by unsupervised learning. Using SOM, the input space of training samples can be represented in a lower-dimensional (often two-dimensional) space [5], called *map*. SSuch model is efficient in structure visualization due to its feature of topological preservation using a neighbourhood function.

SOM consists of two layers of neurons (see Fig. 1): an *input layer* that receives and transmits the input information and an *output layer*, the map that represents the output characteristics. The output layer is commonly organized as a two-dimensional rectangular grid of nodes, where each node corresponds to one neuron. Both layers are feed-forward connected. Each neuron in the input layer is connected to each neuron in the output layer. A real number, or weight, is assigned to each of these connections.



(a) Global view on SOM structure (b) SOM output layer

Fig. 1. Basic Schema of SOM

2.2 Growing Neural Gas

The principle of this neural network is an undirected graph which need not be continuous. Generally, there are no restrictions on the topology. The graph

is generated and continuously updated by competitive Hebbian Learning[6, 9]. According to the pre-set conditions, new neurons are automatically added and connections between neurons are subject to time and can be removed. GNG can be used for vector quantization by finding the code-vectors in clusters [3], biologically influenced [10], image compression, disease diagnosis.

GNG works by modifying the graph, where the operations are the addition and removal of neurons and edges between neurons. An example of the operation is shown in Figure 2

To understand the functioning of GNG, it is necessary to define the algorithm. The algorithm described by Algorithm 1 is based on the original algorithm [2] [3], but it is modified for better continuity in the SOM algorithm. The description of the algorithm has been divided for convenience into two parts. In the first part of Algorithm 1 the overall functionality is described. The second part of Algorithm 2 describes one iteration, which means the descriptions of variables without dependence on time.

Remark The notation used in the paper is briefly listed in Table 1.

Algorithm 1 Growing Neural Gas algorithm

1. Initialization of network. Two neurons N_1 and N_2 are created, $\mathsf{E} = \{\mathsf{e}_{12}\}$. Weight vectors $w_1(t)$ and $w_2(t)$ are initialized to random values $w_{kj}(t) \in [0, 1]$.
2. Select arbitrary unused input data vector.
3. Perform the one learning iteration according to the algorithm described in Algorithm 2.
4. Reduce error value e_i for all neurons N_i using factor β.
5. Returns to step 2, until all input data vector have been used.
6. If $t < T$ return to step 2.

3 Parallelization

Parallelization focuses on the equally distribution of neurons, where new neurons are allocated to the process with the lowest number of neurons. The advantage of this distribution is constant workload processes. The disadvantage is increase communication between processes.

After analysing the GNG learning algorithm we identified the one most processor time-consuming area. This part was selected as a candidate for the possible parallelization. The selected area is:

Finding BMU – this part of GNG learning can be significantly accelerated by dividing the GNG output layer into smaller pieces. Each piece is then assigned to an individual computation process. The calculation of the Euclidean distance among the individual input vector and all the weight vectors

Table 1. Notation used in the paper

Symbol	Description
M	Number of input vectors
n	Dimension of input vectors, number of input neurons, dimension of weight vectors in GNG output layer neurons
N	Current number of neurons in GNG output layer
N_{max}	Maximum allowed number of neurons in GNG output layer
n_i	i-th input neuron, $i = 1, 2, \ldots, n$
N_i	i-th output neuron, $i = 1, 2, \ldots, N$
e_{ij}	edge between neurons N_i and N_j for some $i, j = 1, \ldots, N$, where $i \neq j$.
E	set of all edges in GNG
G	undirected graph describing topology of GNG, $G(\{\mathsf{N}_1, \ldots, \mathsf{N}_N\}, \mathsf{E})$
T	Number of epochs
t	Current epoch, $t = 1, 2, \ldots, T$
X	Set of input vectors, $X \subset \mathbb{R}^n$
$\boldsymbol{x}(t)$	Current input vector in epoch t, arbitrarily selected vector from set X $\boldsymbol{x}(t) \in X$, $\boldsymbol{x}(t) = (x_1, x_2, \ldots, x_n)$
$\boldsymbol{w_k}(t)$	Weight vector of neuron N_k, $k = 1, 2, \ldots, N$ $\boldsymbol{w_k}(t) \in \mathbb{R}^n$, $\boldsymbol{w_k}(t) = (w_{1k}, w_{2k}, \ldots, w_{nk})$
N_{c_1}	The first Best Matching Unit (BMU_1), winner of learning competition
N_{c_2}	The second Best Matching Unit (BMU_2), the second best matching neuron in learning competition
$\boldsymbol{w}_{c_1}(t)$	Weight vector of BMU_1
$\boldsymbol{w}_{c_1}(t)$	Weight vector of BMU_2
l_{c_1}	Learning factor of BMU_1
l_{nc_1}	Learning factor of BMU_1 neighbours
e_i	Local error of output neuron N_i, $i = 1, 2, \ldots, N$
α	Error e_i reduction factor
β	Neuron error reduction factor
γ	Interval of input patterns to add a new neuron
a_{max}	Maximum edges age
a_{ij}	Age of edge e_{ij}

Algorithm 2 One iteration of the Growing Neural Gas algorithm

1. Find neurons BMUs neurons N_{c_1} and N_{c_2}.
2. Update the local error e_{c_1} of neuron N_{c_1}

$$e_{c_1} = e_{c_1} + \|\boldsymbol{w}_{c_1} - \boldsymbol{x}\|^2 \tag{1}$$

3. Update the weight vector $\boldsymbol{w}_{c_1}$ of neuron N_{c_1}

$$\boldsymbol{w}_{c_1} = \boldsymbol{w}_{c_1} + \boldsymbol{l}_{c_1}(\boldsymbol{x} - \boldsymbol{w}_{c_1}) \tag{2}$$

4. For all neurons N_k where exists edge $\mathsf{e}_{c_1 k}$ (N_{c_1} neighbourhood)
   (a) Update the weights $\boldsymbol{w}_k$ using l_{nc_1} learning factor

$$\boldsymbol{w}_k = \boldsymbol{w}_k + l_{nc_1}(\boldsymbol{x} - \boldsymbol{w_k}) \tag{3}$$

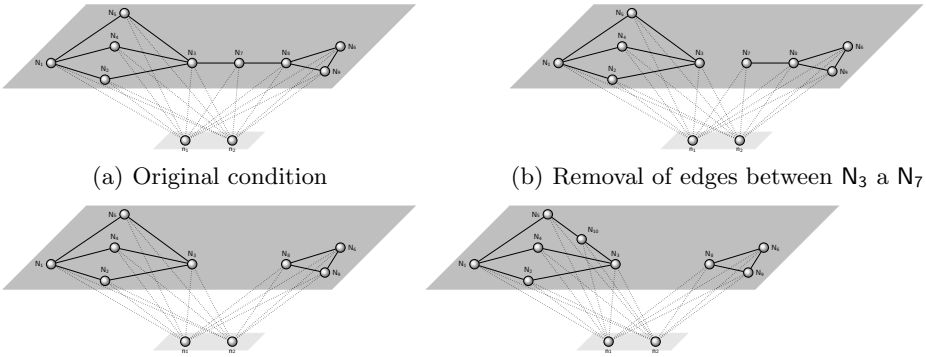
   (b) Increase age a_{kc_1} of edge $\mathsf{e}_{c_1 k}$

$$a_{kc_1} = a_{kc_1} + 1 \tag{4}$$

5. If there is no edge between neurons N_{c_1} and N_{c_2}, then create such edge. If the edge exists, the age is set to 0.
6. If any edge has reached the age of a_{max}, it is removed.
7. If there is a neuron without connection to any edge, the neuron is then removed.
8. If the number of processed input vectors in the current iteration has reached the whole multiple of the value γ and the maximum allowed number of output neurons is not reached, add a new neuron N_{N+1}. The location and error of the new neuron is determined by the following rules:
   (a) Found neuron N_b(NBE) which has the biggest error e_b.
   (b) Found neuron N_c(NSE) among neighbours of neuron N_b and has the biggest error e_c among these neighbours.
   (c) Create a new neuron N_{N+1} and the value of w_n is set as:

$$\boldsymbol{w}_{N+1} = \frac{1}{2}(\boldsymbol{w}_b + \boldsymbol{w}_c) \tag{5}$$

   (d) Creating edges between neurons N_b and N_{N+1}, and also between neurons N_c and N_{N+1}.
   (e) Removed edge between neurons N_b and N_c.
   (f) Reduction of error value in neurons N_b and N_c using the multiplying factor α. Error for neuron N_{N+1} is equal to the new error of neuron N_b.



(a) Original condition

(b) Removal of edges between N_3 a N_7

(c) Removal of edges between N_7 a N_8, remove neuron N_7

(d) Added neuron N_{10} between N_5 a N_3

Fig. 2. GNG examples

to find BMU in a given part of the GNG output layer is the crucial point of this part of GNG learning. Each process finds its own, partial, BMU in its part of the GNG output layer. Each partial BMU is then compared with other BMUs obtained by other processes. Information about the BMU of the whole network is then transmitted to all the processes to perform the updates of the BMU neighbourhood.

A detailed description of our approach to the parallelization process is described in Fig. 3.

The parallelization of GNG learning was performed on an HPC cluster, using *Message Passing Interface* (MPI) technology. MPI technology is based on effective communication between processes. That means that one application can run on many cores. The application uses MPI processes which run on individual cores. The processes are able to send and receive messages and data, communicate etc. Detailed information about HPC and MPI technology is provided, for example, in [8][3].

3.1 Description of Proposed Approach

This subsection includes a detailed description of our approach. Initially, each process reads the training input vectors and on the first process are create two neurons. During the calculation, new neurons are equally distributed on the processes. We do not store the complete GNG graph, we only store parts of it in the appropriate computational nodes.

This approach results in the following significant advantage: neurons are equally distributed among the processes corresponding to the used cores, con-

[3] A specification of MPI is available on the web: http://www.mpi-forum.org/ The functions used in the experiment with a link to MPI.Net are available on the web: http://osl.iu.edu/research/mpi.net

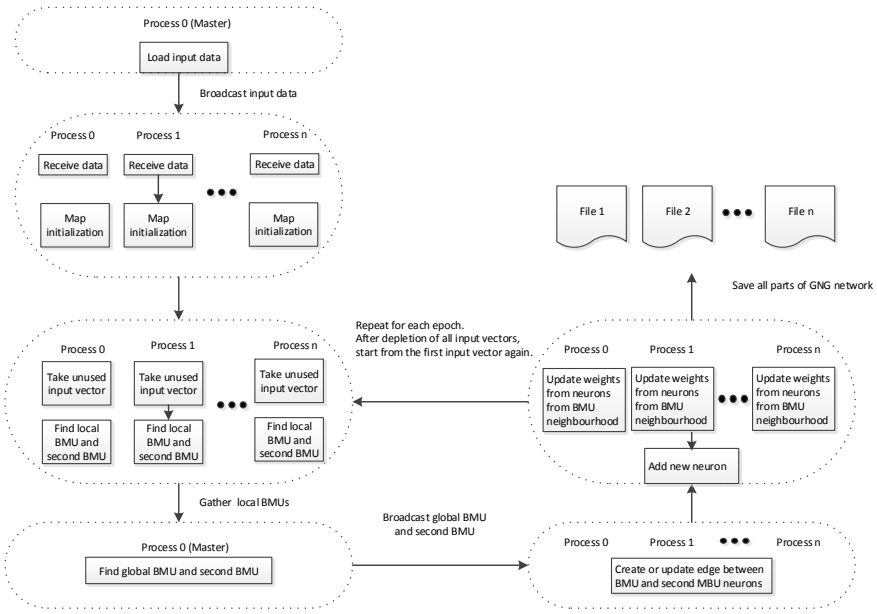


Fig. 3. Parallel Algorithm

trary to the sequential approach, where the whole graph is allocated to only one core (and where there may not be enough memory). For a more precise illustration: consider having three computation servers, each with 16GB of memory. If we use the sequential GNG version, we can only use 16GB of memory on one server. But in the case of the parallel SOM version, we can use all the memory, all 48GB. (The graph can be up to three times larger.)

In the main computational phase, one BMU and a second BMU are founded for each input vector. Thus, each processor needs to compute its local BMU and second BMU within its neurons, after which, each local BMU and second BMU (and their position in the network) are shifted onto one process using the MPI function *GatherFlattened* to determine the global BMU and second BMU. It is possible to use another MPI functions as well, which can provide this selection at one time, but after testing we have found that the experiments took much more time than our presented approach. A global winning BMU and second BMU are then distributed using the MPI function *Broadcast* on all the processes. Now if there is a edge between the first BMU and the second BMU then age is set to zero otherwise creates edge between this two neurons. Next, the neighbourhood of the BMU in each process is known and, consequently, the weights of the neurons matching are actualized. If the condition is met for adding a new neuron, the process with the lowest number of neurons add a new neuron. A detailed description of adding a new neuron can be found in Fig. 4. This

procedure is repeated until all the input training vectors are exhausted (until we have finished one epoch).

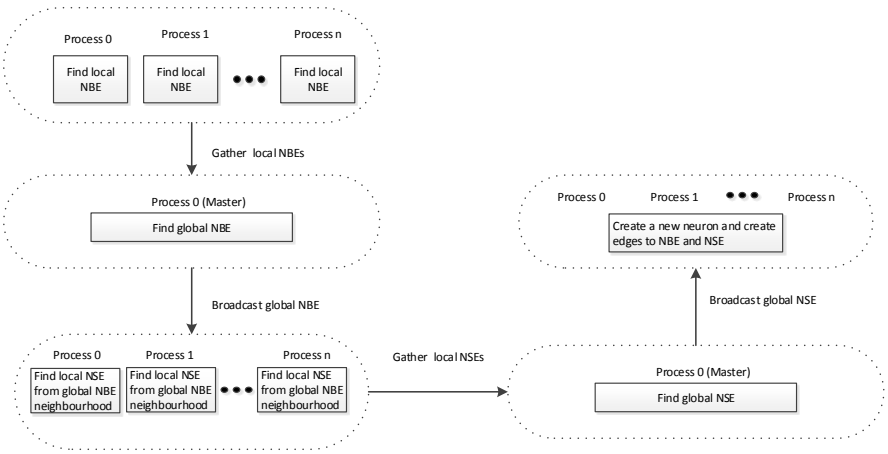


Fig. 4. Parallel Algorithm for adding a neuron

4 Experiments

4.1 Experimental Datasets and Hardware

Two datasets were used in the experiments. The first dataset was commonly used in Information Retrieval – *Medlars*. The second one was the test data for the elementary benchmark for clustering algorithms[11].

Medlars Dataset The Medlars dataset consisted of 1,033 English abstracts from a medical science[4]. The 8,567 distinct terms were extracted from the Medlars dataset. Each term represents a potential dimension in the input vector space. The term's level of significance (weight) in a particular document represents a value of the component of the input vector. Finally, the input vector space has a dimension of 8,707, and 1,033 input vectors were extracted from the dataset.

Clustering dataset Three training data collections called TwoDiamonds, Lsun and Hepta from the Fundamental Clustering Problems Suite (FCPS) are used. A short description of the selected dataset used in our experiments is given in Table 2.

[4] The collection can be downloaded from ftp://ftp.cs.cornell.edu/pub/smart. The total size of the dataset is approximately 1.03 MB.

Table 2. Fundamental Clustering Problems Suite – selected datasets

Name	Cases	#Vars	#Clusters	Main Clustering Problem
Target	770	2	6	outlying clusters
Lsun	400	2	3	different variances in clusters
TwoDiamonds	800	2	2	touching clusters

Experimental Hardware All the experiments were performed on a Windows HPC server 2008 with 6 computing nodes, where each node had 8 processors with 12 GB of memory. The processors in nodes were Intel Xeon 2.27GHz. The topology with the connection between the head node and computing nodes can be found on the web[5] (topology number four). The Enterprise and Private Networks link speed was 1Gbps, the Application link speed was 20Gbps.

4.2 First Part of The Experiment

The first part of the experiment was oriented towards a comparison of the standard GNG algorithm and parallel approach to this GNG learning algorithm. The Medlars dataset was used for the experiment. A parallel version of the learning algorithm was run using 2, 8, 16, 24 and 32 MPI processes. The records with an asterisk (*) represents the results for only one process i.e. this is the original serial learning algorithm and there is no network communication.

GNG parameters are the same for all experiments and are as follows $\gamma = 200$, $e_w = 0.05$, $e_n = 0.006$, $\alpha = 0.5$, $\beta = 0.0005$, $a_{max} = 88$, M = 1000, $\delta = 100$. The achieved computing time is presented in Table 3.

Table 3. Computing Time with Respect to Number of Cores, Standard GNG Algorithm, Dataset Medlars

Cores	Computing Time [hh:mm:ss]
1*	00:35:41
2	00:17:50
8	00:04:47
12	00:03:56
16	00:03:09
24	00:02:45
32	00:02:32

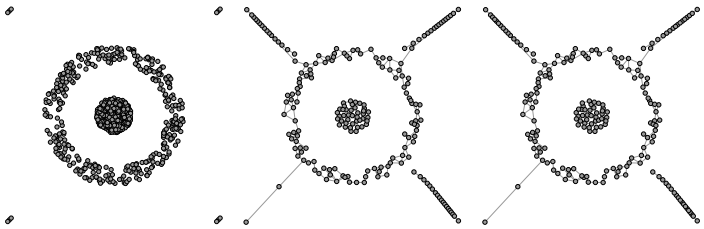
As we can see from Table 3, the computing time depends on the number of used cores as well. With a growing number of processors, the computation effectiveness increases, and the computational time is sufficiently reduced.

[5] http://technet.microsoft.com/en-us/library/cc719008(v=ws.10).aspx

4.3 Second Part of The Experiment

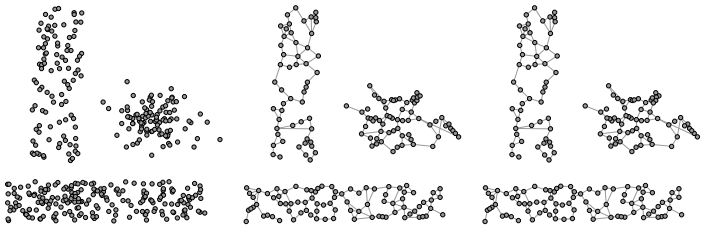
The second part of the experiments was oriented towards comparing the results obtained by the parallel and standard GNG algorithm. The Clustering dataset was used for the experiment. The parallel version of the learning algorithm was run using 16 MPI processes.

GNG parameters are similar to the previous experiment. There are two changes M = 500 and $\delta = 25$.



(a) Input data (b) Standard GNG (c) Parallel GNG using 16 cores

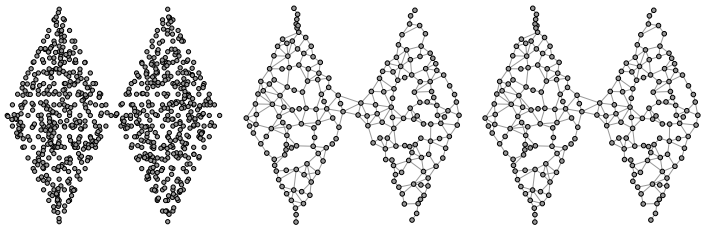
Fig. 5. Results of dataset *Target*



(a) Input data (b) Standard GNG (c) Parallel GNG using 16 cores

Fig. 6. Results of dataset *Lsun*

In Figures 5(a), 6(a) and 7(a) there are a layout view input data, which are used for training GNG. Outputs of standard GNG algorithm are in Figures 5(b),



(a) Input data (b) Standard GNG (c) Parallel GNG using 16 cores

Fig. 7. Results of dataset *TwoDiamonds*

6(b) and 7(b), which are the same as in the parallel version. i.e. both versions produce the same network.

5 Conclusion

In this paper the parallel implementation of the GNG neural network algorithm is presented. The achieved speed-up was very good and the results from the standard and parallel version of GNG are same. So we can say that the operation of the parallel version is correct. However, the effectiveness of a parallel solution is dependent on the division of the output layer. An improper division may cause the communication between processes to be very time consuming.

In future work we intend to focus on the sparse date, use combinations of neural networks for improved result and improved acceleration.

Acknowledgement This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the Development of human resources in research and development of latest soft computing methods and their application in practice project, reg. no. CZ.1.07/2.3.00/20.0072 funded by Operational Programme Education for Competitiveness, co-financed by ESF and state budget of the Czech Republic.

References

1. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is “nearest neighbor” meaningful? In: Database Theory ’99. vol. 1540, pp. 217–235 (1999)
2. Fritzke, B.: A growing neural gas network learns topologies. In: Advances in Neural Information Processing Systems 7. pp. 625–632. MIT Press (1995)

3. Holmström, J.: Growing Neural Gas Experiments with GNG, GNG with Utility and Supervised GNG. Master's thesis, Uppsala University (2002-08-30)
4. Kohonen, T.: Self-Organization and Associative Memory, Springer Series in Information Sciences, vol. 8. Springer, Berlin, Heidelberg (1984), 3rd ed. 1989.
5. Kohonen, T.: Self Organizing Maps. Springer-Verlag, 3rd edn. (2001)
6. Martinetz, T.: Competitive hebbian learning rule forms perfectly topology preserving maps. In: Gielen, S., Kappen, B. (eds.) ICANN '93, pp. 427–434. Springer London (1993), http://dx.doi.org/10.1007/978-1-4471-2063-6_104
7. Martinetz, T., Schulten, K.: A "neural-gas" network learns topologies. Artificial Neural Networks 1, 397–402 (1991), http://web.cs.swarthmore.edu/ meeden/DevelopmentalRobotics/fritzke95.pdf
8. Pacheco, P.: Parallel Programming with MPI. Morgan Kaufmann, 1st edn. (1996)
9. Prudent, Y., Ennaji, A.: An incremental growing neural gas learns topologies. In: Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on. vol. 2, pp. 1211 – 1216 vol. 2 (july-4 aug 2005)
10. Sledge, I., Keller, J.: Growing neural gas for temporal clustering. In: Pattern Recognition, 2008. ICPR 2008. 19th International Conference on. pp. 1–4 (2008)
11. Ultsch, A.: Clustering with SOM: U*C,. Proc. Workshop on Self-Organizing Maps, Paris, France, pp. 75–82 (2005)