Growing Neural Gas Based on Data Density

Lukáš Vojáček[1(✉)], Pavla Dráždilová[2], and Jiří Dvorský[1,2]

[1] IT4Innovations, VŠB - Technical University of Ostrava,
17. listopadu 15/2172, 708 33 Ostrava, Czech Republic
{lukas.vojacek,jiri.dvorsky}@vsb.cz
[2] Department of Computer Science, VŠB – Technical University of Ostrava,
17. listopadu 15/2172, 708 33 Ostrava, Czech Republic
pavla.drazdilova@vsb.cz

Abstract. The size, complexity and dimensionality of data collections are ever increasing from the beginning of the computer era. Clustering methods, such as Growing Neural Gas (GNG) [10] that is based on unsupervised learning, is used to reveal structures and to reduce large amounts of raw data. The growth of computational complexity of such clustering method, caused by growing data dimensionality and the specific similarity measurement in a high-dimensional space, reduces the effectiveness of clustering method in many real applications. The growth of computational complexity can be partially solved using the parallel computation facilities, such as High Performance Computing (HPC) cluster with MPI. An effective parallel implementation of GNG is discussed in this paper, while the main focus is on minimizing of interprocess communication which depends on the number of neurons and edges among neurons in the neural network. A new algorithm of adding neurons depending on data density is proposed in the paper.

Keywords: Growing neural gas · High-dimensional dataset
High performance computing · MPI · Data density

1 Introduction

The size and complexity of data collections are ever increasing from the beginning of the computer era, while the dimensionality of the data sets is rapidly increasing in recent years. Contemporary and especially future technologies allow us to acquire, store and process large high dimensional data collections that are commonly available in areas like medicine, biology, information retrieval, web analysis, social network analysis, image processing, financial transaction analysis and many others.

To have any chance to process such amount of the data we have to reduce amounts of raw data by categorizing them in smaller set of similar items, we have to identify groups that occurs in the data, we have to reveal structures hidden in

K. Saeed and W. Homenda (Eds.): CISIM 2018, LNCS 11127, pp. 314–323, 2018.
https://doi.org/10.1007/978-3-319-99954-8_27

the data. These tasks are precisely the purpose of methods known as *clustering*. There are many clustering methods, we will focus on clustering methods based on unsupervised learning in this paper.

Unfortunately, there are two major issues faced by clustering algorithms based on unsupervised learning, such as *Growing Neural Gas* (GNG) [10], that prevent them to be effective, in many real applications, on vast high dimensional data collection:

1. The fast growth of computational complexity with respect to growing data dimensionality, and
2. The specific similarity measurement in a high-dimensional space, where the expected distance, computed by Euclidean metrics to the closest and to the farthest point of any given point, shrinks with growing dimensionality [1].

The growth of computational complexity can be partially solved using the parallel computation facilities, such as *High Performance Computing* (HPC) cluster with MPI technology. Obviously, it is necessary to resolve technical and implementation issues specific to this computing platform, such as minimizing of interprocess communication, to provide effective parallel implementation of GNG.

The amount of interprocess increases with the growing number of neurons and edges connecting the neurons. The addition of a new neuron and edges to GNG is driven by condition given at the startup – neuron is added after predefined amount of time regardless the data collection properties. Respecting the data collection properties it is easy to see that some addition of a new neuron and edges is not necessary, for example the addition of a new neuron in dense data area caused just by precalculated condition. Similar approach can be used for outlying data. When a new neuron is created to cover this part of the data collection, there is no special need to attach a new neuron to more neurons than the nearest. In this way the future edge disposal is eliminated. A new neuron is attached to two nearest neurons only in the case that a new neuron would cover a part of the data collection located just nearby these two neurons. So, the sum of distances between a new neuron to these two neurons should be proportional to the distance between these two neurons themselves.

The proposed approach is based only on standard GNG learning algorithms, there is no need to apply space partitioning method to improve nearest neuron search. The performed experiments shows that our approach clearly adhere the structure of data collection, while quality of the GNG is preserved.

We will first introduce the terminology, the notation which we use in the article and related works to GNG. In the next section we describe a new approach how to add a new neuron to GNG. Section Experiments contains statistics and visualization of results. In conclusion, we discuss the advantages of our approach.

2 Growing Neural Gas

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

generated and continuously updated by competitive Hebbian Learning [9,13]. 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 [8], clustering data streams [7], biologically influenced [14] and 3D model reconstruction [12]. GNG works by modifying the graph, where the operations are the addition and removal of neurons and edges between neurons.

To understand the functioning of GNG, it is necessary to define the algorithm. The algorithm described in our previous article [16] is based on the original algorithm [6,8], but it is modified for better continuity in the SOM algorithm. Here is the Algorithm 1 which describes one iteration.

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

2.1 Related Works

The methods based on Artificial Neural Networks (ANN) are computationally expensive. There are different approaches on how to improve effectivity of these methods – improve computation of the nearest neurons, reduce number of computation (batch), parallel implementation and other.

The authors of the paper [4] propose two optimization techniques that are aimed at an efficient implementation of the GNG algorithm internal structure. Their optimizations preserve all properties of the GNG algorithm. The technique enhances the nearest neighbor search using a space partitioning by a grid of rectangular cells and the second technique speeds up the handling of node errors using the lazy evaluation approach. The authors in [13] propose a algorithm for a GNG which can learn new input data (plasticity) without degrading the previously trained network and forgetting the old input data (stability). Online Incremental Supervised Growing Neural Gas in [3] is an algorithm whose features are zero nodes initialization, the original batch Supervised Growing Neural Gas node insertion mechanism and network size constraint. In [2] is proposed a batch variant of Neural gas (NG) which allows fast training for a priorly given data set and a transfer to proximity data. Author's algorithm optimizes the same cost function as NG with faster convergence than original algorithm. A paper [11] proposes a Growing Neural Gas based on density, which is useful for clustering. An algorithm creates new units based on the density of data, producing a better representation of the data space with a less computational cost for a comparable accuracy. Authors use access methods to reduce considerably the number of distance calculations during the training process.

In the paper [5] the authors combine the batch variant of the GNG algorithm with the MapReduce paradigm resulting in a GNG variant suitable for processing large data sets in scalable, general cluster environments. The paper [15] is focused on the actualizations of neurons weights in the learning phase of parallel implementation of SOM. Authors study update strategies between Batch SOM – updates are processed at the end of whole epoch – and immediately updating after processing one input vector.

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 output neuron, $i = 1, 2, \ldots, N$
X	Set of input vectors, $X \subset \mathbb{R}^n$
$\boldsymbol{x}_i$	i-th input vector, $i = 1, 2, \ldots, M$ $\boldsymbol{x}(t) \in X$, $\boldsymbol{x}(t) = (x_1, x_2, \ldots, x_n)$
$\boldsymbol{w_k}(t)$	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
p	Number of processes
k	Specify an area around BMU_1 without adding a new neuron; $0 < k \leq 1/2$

3 Optimization of Learning Phase

In our paper [16] we dealt with the parallelization of GNG. The main problem that reduces parallelization is communication between processes and threads. This communication increases with the number of neurons in the neural network. The goal of our optimization is to reduce the number of neurons and to manage the addition of new neurons.

By default, new neurons are added after the condition, which is determined at startup calculation - see Algorithm 1 step 8 (a neuron added after a certain period of time). We identified after analysing the GNG learning algorithm that the algorithm does not take into account the input data. The only limitation is the maximum number of neurons in the neural network. Our proposed solution

Algorithm 1. 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.

consists of two parts. The first part is a change in the condition when inserting a new neuron and the second part is a change in the weight of this new neuron.

3.1 A New Approach How to Add a New Neuron

The goal is to change the current condition that adds a neuron (after a certain time) to a new condition that will take into account the data it is working on. We do not have to specify number of steps for which we add a new neuron. The

basic principle of our approach for adding a new one is to evaluate the distance of the input vector from the neurons N_{c_1} and N_{c_2}. If the input vector is close to N_{c_1} then our algorithm does not add the new neuron. Standard approach allways add the new neuron in this situation.

The following inequalities determined when we add a new neuron. A new neuron is added to the neural network, if one of inequalities (6), (7), (8) is true.

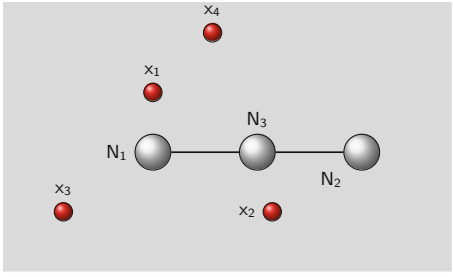
$$||\mathsf{N}_{c_2} - \boldsymbol{x}(t)|| < ||\mathsf{N}_{c_1} - \mathsf{N}_{c_2}|| \quad \wedge \quad ||\mathsf{N}_{c_1} - \boldsymbol{x}(t)|| \geq k||\mathsf{N}_{c_1} - \mathsf{N}_{c_2}|| \tag{6}$$

$$||\mathsf{N}_{c_2} - \boldsymbol{x}(t)|| \geq ||\mathsf{N}_{c_1} - \mathsf{N}_{c_2}|| \quad \wedge \quad ||\mathsf{N}_{c_1} - \boldsymbol{x}(t)|| \geq k||\mathsf{N}_{c_1} - \mathsf{N}_{c_2}|| \tag{7}$$

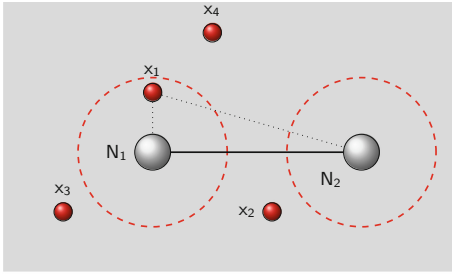
$$||\mathsf{N}_{c_2} - \boldsymbol{x}(t)|| > ||\mathsf{N}_{c_1} - \mathsf{N}_{c_2}||, \tag{8}$$

where $\boldsymbol{x}(t)$ is input vector, parametr k specifies the size of area around N_{c_1} and $0 < k \leq 1/2$.

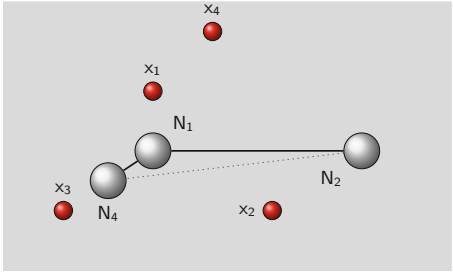
In Fig. 1, two neurons (N_1 and N_2) and four input vectors ($\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$ and $\boldsymbol{x}_4$) can be seen. Neurons N_1 and N_2 represent BMU N_{c_1} and second BMU N_{c_2} for all input vectors in the example. The input vector $\boldsymbol{x}_1$ is too closed to N_1 ($||\mathsf{N}_1 - \boldsymbol{x}_1|| < k||\mathsf{N}_{c_1} - \mathsf{N}_{c_2}||$) and a new neuron is not added (Fig. 1(b)). If the input vectors $\boldsymbol{x}_2$, $\boldsymbol{x}_3$ and $\boldsymbol{x}_4$ are selected then a new neuron is added. The input vector $\boldsymbol{x}_2$ satisfies inequality (6) and it is in the standard situation (Fig. 1(a)). The new added neuron N_3 obtains weight which is calculated: $1/2(w_{c_1}(t) + w_{c_2}(t))$. The edge between N_1 and N_2 is deleted and new edges are created which connect a new neuron N_3 with neuron N_1 and N_2. The age of new edges is set to zerro.



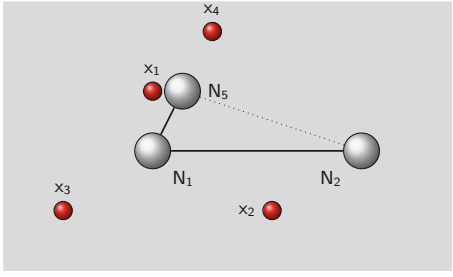
(a) Input vector x_2, standard procedure



(b) Actualization area for $k = 0.3$



(c) Input vector x_3



(d) Input vector x_4

Fig. 1. Addition of a new neuron based on given input vectors $\boldsymbol{x_2}$, $\boldsymbol{x_3}$ and $\boldsymbol{x_4}$

The inequality (8) is true for the input vector $\boldsymbol{x}_3$ (see Fig. 1(c)) and the inequality (7) is true for the input vector $\boldsymbol{x}_4$ (see Fig. 1(d)). Only one edge connect the new added neuron N_4 (N_5) with BMU N_1 in both situations. The weight of new neuron and the age of the new edge is set in the standard way.

4 Experiments

4.1 Experimental Datasets and Hardware

One dataset was used in the experiments. The dataset was commonly used in benchmark – *Clustering dataset*.

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

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

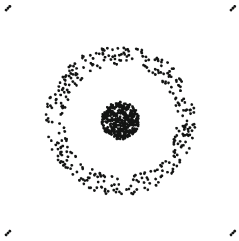
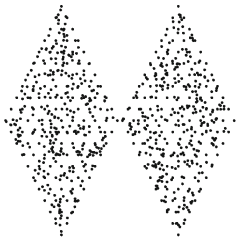
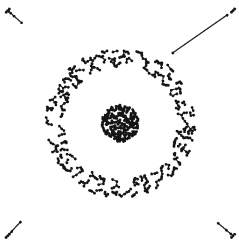
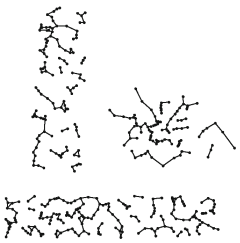
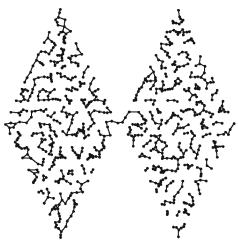
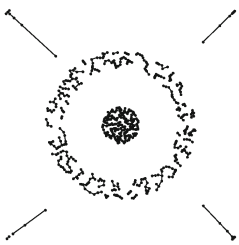
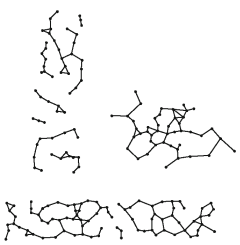
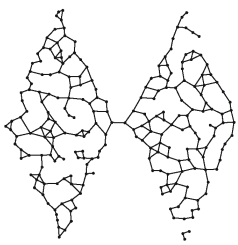
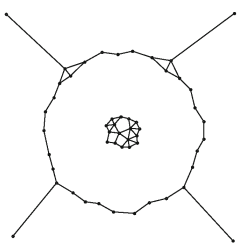
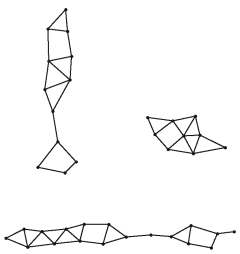
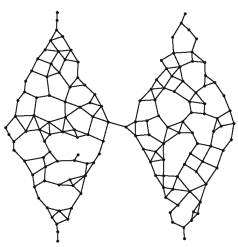
Experimental Hardware. The experiments were performed on a Linux HPC cluster, named Anselm, with 209 computing nodes, where each node had 16 processors with 64 GB of memory. Processors in the nodes were Intel Sandy Bridge E5-2665. Compute network is InfiniBand QDR, fully non-blocking, fat-tree. Detailed information about hardware is possible to find on the web site of Anselm HPC cluster[1].

4.2 The Experiments

The first part of the experiments was oriented towards comparing the results obtained in density versions (k = 0.3 and k = 0.5) 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. The GNG parameters are the same for all experiments and are as follows $e_w = 0.05$, $e_n = 0.006$, $\alpha = 0.5$, $\beta = 0.0005$, $a_{max} = 100$, M = 1000, $\delta = 200$.

[1] https://support.it4i.cz/docs/anselm-cluster-documentation/hardware-overview.

Table 3. Graphical representations of data set layout and corresponding GNGs

	Data Collection		
	Target	Lsun	TwoDiamonds
Input data			
Standard GNG			
Parallel GNG; k=0.3			
Parallel GNG; k=0.5			

The first row in the Table 3 shows a layout view of the input data, which are used for training GNG. The outputs of standard GNG algorithms are in the second row. In third and fourth rows are result of our proposal method, but each for different size areas that will not update.

In the Table 4, we can see number of neurons which have been used.

For data collection TwoDiamonds, the time computation of standard GNG is 6.05 s, parallel GNG with k = 0.3 is 3.9 s and parallel GNG with k = 0.5 is 2.2 s.

Table 4. Number of used neurons

Algorithm	Dataset		
	Target	Lsun	TwoDiamonds
Standard GNG	771	401	801
Parallel GNG; k = 0.3	583	187	231
Parallel GNG; k = 0.5	51	38	148

5 Conclusion

In this paper the parallel implementation of the GNG neural network algorithm based on data density is presented. The achieved speed-up was better than our previous approach. That's because there are only fewer neurons in the network. Therefore, it takes less time to locate the BMU. For parallelization, we generally try to keep communication as small as possible, which reduces this communication due to fewer neurons; this is most evident when adding and removing neurons.

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

Acknowledgments. This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center – LM2015070" and co-financed by SGS, VŠB – Technical University of Ostrava, Czech Republic, under the grant No. SP2018/126 "Parallel processing of Big Data V".

References

1. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "Nearest Neighbor" meaningful? In: Beeri, C., Buneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 217–235. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49257-7_15
2. Cottrell, M., Hammer, B., Hasenfuß, A., Villmann, T.: Batch neural gas. In: 5th Workshop On Self-Organizing Maps, vol. 102, p. 130 (2005)
3. Duque-Belfort, F., Bassani, H.F., Araujo, A.F.: Online incremental supervised growing neural gas. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 1034–1040. IEEE (2017)
4. Fišer, D., Faigl, J., Kulich, M.: Growing neural gas efficiently. Neurocomputing **104**, 72–82 (2013)

5. Fliege, J., Benn, W.: MapReduce-based growing neural gas for scalable cluster environments. Machine Learning and Data Mining in Pattern Recognition. LNCS (LNAI), vol. 9729, pp. 545–559. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41920-6_43
6. Fritzke, B.: A growing neural gas network learns topologies. In: Advances in Neural Information Processing Systems 7, pp. 625–632. MIT Press (1995)
7. Ghesmoune, M., Lebbah, M., Azzag, H.: A new growing neural gas for clustering data streams. Neural Netw. **78**, 36–50 (2016)
8. Holmström, J.: Growing Neural Gas Experiments with GNG, GNG with Utility and Supervised GNG. Master's thesis, Uppsala University (2002–08-30)
9. Martinetz, T.: Competitive hebbian learning rule forms perfectly topology preserving maps. In: Gielen, S., Kappen, B. (eds.) ICANN 1993, pp. 427–434. Springer, London (1993)
10. Martinetz, T., Schulten, K.: A "neural-gas" network learns topologies. Artif. Neural Netw. **1**, 397–402 (1991)
11. Ocsa, A., Bedregal, C., Guadros-Vargas, E.: DB-GNG: a constructive self-organizing map based on density. In: International Joint Conference on Neural Networks, 2007. IJCNN 2007, pp. 1953–1958. IEEE (2007)
12. Orts-Escolano, S., et al.: 3D model reconstruction using neural gas accelerated on GPU. Appl. Soft Comput. **32**, 87–100 (2015)
13. Prudent, Y., Ennaji, A.: An incremental growing neural gas learns topologies. In: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, 2005. IJCNN 2005, vol. 2, pp. 1211–1216 (2005)
14. Sledge, I., Keller, J.: Growing neural gas for temporal clustering. In: 19th International Conference on Pattern Recognition, 2008. ICPR 2008, pp. 1–4 (2008)
15. Vojáček, L., Dráždilová, P., Dvorský, J.: Self organizing maps with delay actualization. In: Saeed, K., Homenda, W. (eds.) CISIM 2015. LNCS, vol. 9339, pp. 154–165. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24369-6_13
16. Vojáček, L., Dvorský, J.: Growing neural gas – a parallel approach. In: Saeed, K., Chaki, R., Cortesi, A., Wierzchoń, S. (eds.) CISIM 2013. LNCS, vol. 8104, pp. 408–419. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40925-7_38