Tumble Tree – Reducing Complexity of the Growing Cells Approach

Hendrik Annuth and Christian-A. Bohn

Wedel University of Applied Sciences
Wedel, FR Germany

Abstract. We propose a data structure that decreases complexity of unsupervised competitive learning algorithms which are based on the *growing cells structures* approach. The idea is based on a novel way of ordering the cells in a tree like data structure in a way that random access during training is replaced by tree traversals.
Overall time complexity is reduced from $O(n^2)$ to $O(n \log n)$ which opens new application fields to the growing cells structures approach.

Key words: neural networks, unsupervised learning, reinforcement learning, growing cells structures, growing neural gas.

1 Introduction

The importance of automatic analysis and interpretation of complex, huge and arbitrary data sets has notably increased in the field of computer science. This is driven by the fact that these data sets nowadays can easily be created, retrieved and stored.

The analysis and interpretation of such data using unsupervised competitive learning algorithms has shown to be a great success. One important approach from the field of unsupervised learning is the growing cells structures (GCS) concept [1]. It is deduced from Kohonen's *self organizing map* (SOM) [2] and shares its advantages like autonomy, robustness, scalability and the ability of retrieving information from very complex data. In contrast the GCS approach is able to locally and globally adapt to a given problem under consideration by adding and removing cells in the network. This makes the method very flexible and adaptive. The GCS approach achieves extraordinary results in the area of classification, clustering, dimensionality reduction and data mining, such that there exist a huge amount of applications.

Despite the success of the GCS approach, due to its quadratic complexity, the restricted network size hinders many applications where huge amounts of data arise. This comes from the fact that at every iteration cycle each network cell has to be visited once and a local counter at each cell has to be decreased or increased. Up to now, approaches which try to circumvent this problem, like [3], mostly fail if input data grows significantly and complexity issues surface again.

Previous Work. A very important preparatory work for the GCS approach was [2]. They propose the self organizing map which iteratively adapts a 2D mesh to a distribution of samples. While a SOM has a fixed number of elements, the growing cells structures approach [1, 4] allows the network for dynamically adapting to the complexity of the sample data.

Optimizations of the basic GCS approach have been presented in [5] where an advanced clustering algorithm enables analysis of huge data sets. The ability of handling high-dimensional and very noisy data sets has been proven in the field of medicine [6, 7]. Further useful example applications of the GCS approach are document categorization [8]. And [9] describes an interesting approach supporting classification of measurements from a technical gas sensor.

Further development has been driven by the application of surface reconstruction from scanned 3D point sets. This comes from the fact that complexity issues are very significant in this area since the amount of training samples easily exceeds hundreds of thousands of samples per application case. Surface reconstruction with GCS have first been tackled in [10] and [11]. Based on this, [12] proposes a mesh optimization algorithm, and to achieve a better surface quality [13] introduces an edge swap operation. In [14] surface reconstruction is further optimized by the *smart growing cells* (SGC) network which proposes a framework of extending general unsupervised learning by application related rules.

In the following, we first explain the vital steps of the GCS algorithm, which are important to analyze runtime issues. Then we introduce the *tumble tree* data structure which reduces the complexity of the standard GCS approach. Finally, we show that this is a major breakthrough for the GCS method through its application to surface reconstruction where execution time is reduced from several weeks to just hours.

2 The Tumble Tree Idea

2.1 Standard GCS Approach

Basic Loop. The basic training process of the growing cells approach is exposed in Fig. 1. Growing and shrinking of the network within the GCS process depends on the signal counters at each cell. Every time a cell is moved to match a sample its signal counter is affected. In [1] α is constant, but for huge data sets adaption of α during network training is required as shown in [3].
See [1] for a detailed description of the basic GCS approach.

Handling Signal Counters. If a sample is presented to the network, finding the relevant BMU (step 1 of Fig. 1) has $O(\log n)$ time complexity with n the number of cells and if the cells are structured in an octree, for example. Nevertheless, adapting the signal counters τ_k and finding the cell with the highest and those cells with a low counter for adaption of the network size (step 3 and 4 in Fig. 1) has time complexity of $O(n)$.

1. Select a random sample from the data set and find the network cell lying closest to it — the *winning* or *best matching unit* (BMU). Move the BMU and its neighbors according to a certain amount in the direction of the sample.
2. Increment the *signal counter* τ_{BMU} of the BMU and decrease signal counters of all other cells by multiplying them with a coefficient $\alpha < 1$.
3. After a certain number of iterations (steps 1 and 2) determine the cell with the greatest signal counter and add a new cell next to it, since a high signal counter indicates an underrepresentation of cells compared to the amount of samples at that cell.
4. After a certain number of iterations of step 3 determine those network cells which contain a signal counter value below a certain threshold $\overline{\tau}$ and delete them, since they are not represented through an adequate number of samples at that location.

Fig. 1: Algorithm of the classical GCS approach.

Since during learning the GCS grows to about the size of the sample set, the above steps have to be executed n times leading to complexities of $O(n \log n)$ and $O(n^2)$, respectively, leading to overall complexity of $O(n^2)$. See Fig. 2 for a visualization of what happens with the signal counters if cells are moved.

This quadratic complexity is the reason for the lack of applications with sample set sizes above few hundreds of thousands of samples. From our experiences, we learned that calculation time rises from a few minutes to several weeks if network sizes rise from about 10^5 to just about 10^6 cells.

Thus, for these two tasks of updating signal counters and searching cells (steps 2, 3, and 4 from Fig. 1) we propose a new algorithm as follows.

2.2 Signal Counters in the Tumble Tree

We add a concept which we term the *tumble tree*. The tumble tree is a tree-like data structure concurrent to the GCS network with n nodes each of them linked to a single network cell — each cell in the GCS network has its counterpart in the tumble tree. Essentially, the tumble tree is a binary tree — each node k has two children. The child subtree on the left contains only nodes with a signal counter smaller than τ_k, those on the right have signal counters which are greater than or equal to that of node k.

All nodes in the tree contain a cell signal counter and a local value α_k. To determine a signal counter at a node, all α_k above the node have to be multiplied by it. The reason for that will be explained on the following pages.

Considering the algorithm from Fig. 1, our method starts again by finding the BMU (step 1), but for updating the signal counter and for searching the

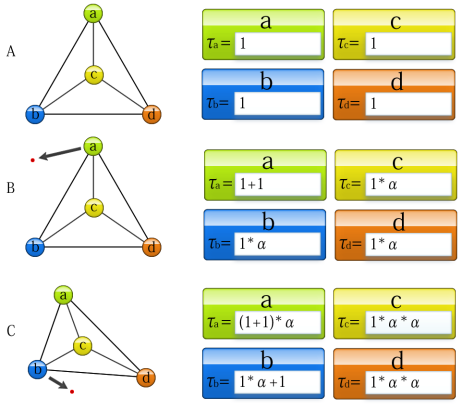

Fig. 2: On the left, a simple GCS network is exposed. In the middle, cell (a) is moved, at the bottom cell (b) is moved. The right side shows the change of the signal counters τ_k of each node. In case (A) all τ_k equal one, in (B) node (a) is moved while incrementing its signal counter, the remaining nodes' τ_k are decreased by a multiplication with α. In case (C) the similar happens with node (b).

highest and the low signal counters (steps 2, 3, and 4) it switches to the tumble tree and accomplishes the operation from there.

Searching Cells regarding its Signal Counters. The first of the two tasks to be realized in the tumble tree is determining the cells with the highest and the lowest signal counters. In the standard GCS method this is achieved by a linear walk through a simple list of the cells. In our proposal, we assume at the moment that the tumble tree nodes contain the correct signal counter values. Then, finding the highest τ_k can be simplified by choosing the outer right node at the bottom of the tree.

Determining all nodes with τ_k below a certain threshold $\overline{\tau}$ is accomplished by descending the tree from the root node and comparing the signal counter τ_k of the visited node k with the threshold $\overline{\tau}$. As long as $\tau_k < \overline{\tau}$ hold we take a branch to the right. If $\tau_k < \overline{\tau}$ does not hold, then the left child of the actual node is the root node of a subtree which only contains nodes of cells for which $\tau_k < \overline{\tau}$ hold.
Obviously, the complexity of the traversal — the search task — is $O(\log n)$.

Topology Maintanance. If a BMU of the GCS has been determined, its signal counter has to be incremented and those of all other cells have to be decreased by a certain factor α. This could change the validity of the tumble tree since its topology is determined from the size of each τ_k.

To keep the tumble tree valid, nodes where signal counters change must be repositioned in the tree. This means the BMU's node must be taken off and be

reinserted regarding its new τ_k. The remaining nodes can be kept untouched since their signal counters change to the same amount each.

Reinserting the BMU's node with a certain new τ_{BMU} is done by traversing the tree from the root node and comparing its τ_{BMU} with the signal counters at the visited nodes. Depending on the result a branch to the left or to the right child is taken until the leaves of the tree are reached. At this position the BMU's node is inserted as a new leave node.

The complexity of this step is also $O(\log n)$.
Up to now we just assumed that signal counter values are known. Finally, we explain their management in the following.

Handling Signal Counters. Up to now, the only task left with complexity $O(n)$ is the update of the signal counters in all cells after a new BMU has been determined. We solve this issue as follows.

The signal counters of all cells but the BMU must be multiplied by an actual value α. This multiplication is postponed, and instead the relevant cells are marked by multiplying this α with a node-local α_k contained in the root node of the subtree of those cells. This subtree is just the complete tumble tree without the BMU node.

Postponing this multiplication through one root node prevents from visiting all cells at the moment, but if one τ_k in a node is needed, it first has to be generated from the postponed α-values spread over the whole tumble tree. It is required for each of the following three cases.

1. If a BMU node is to be taken out of the tree, then for later reinsertion τ_{BMU} must be known. This is accomplished by determining the path back to the root and multiplying all visited (postponed) α_k with the actual τ_{BMU}. Now the BMU node with a valid signal counter can be taken out of the tree.
   What remains is the α-value which is contained in and which vanishes with the node. It virtually serves as postponed α-value for the subtrees and must be propagated to the left and the right child of the BMU node in order keep the signal counters underneath the missing node being valid.
2. When reinserting a BMU's node into the tumble tree again, then for the comparison, all signal counters on the way down to the right place of the node are required. This is simply accomplished by accumulating the values α_k of all nodes which are visited on the way down.
   Additionally, the α-values are propagated through the nodes by multiplying them with each τ_k, and finally they are set to 1, such that nodes on this path contain the true τ_k and a neutral α_k. The reason for that is, that the node which is to be reinserted even contains a valid signal counter (from step 1 from above). Keeping the old α_k in the nodes above would virtually change the signal counter of the new node.
   Additionally, since all α_k on the way down to the inserted node are combined with the relevant τ_k — and thus set to a neutral value — the former postponed α-values are now missing at the child subtrees of each of the nodes.

To solve this, the propagated α_k are also propagated to the left and the right child nodes on the way down to the new node.[1]

3. The highest and the low signal counters are also determined on the fly by accumulating the α_k of the visited nodes while traversing the tumble tree like described above.

All three tasks from above can be accomplished in $O(\log n)$ time complexity and without touching every node in the GCS. For an example of a simple network see Fig. 3.

Thus, the linear complexity for accessing the signal counters is now driven down to $O(\log n)$, and with complexity of step 1 in Fig. 1 this leads to an overall complexity of our GCS approach of $O(n \log n)$.

3 Conclusion

One could think that there is nothing to add at "the algorithm is driven down from $O(n^2)$ to $O(n \log n)$" — nevertheless, in many cases n is not big enough to have a favorable effect on an application. Due to that, we justify our algorithm with the task of surface reconstruction which is a vital application for the GCS approach like mentioned in section 1. To assure logarithmic complexity we used a *red-black-tree* [15] as basis for the tumble tree.

Table 1 exposes the results. The acceleration of our approach compared to using the classical GCS algorithm with squared complexity is impressive. It can be seen that computation times rise dramatically if the tumble tree is not utilized. Our approach enables creating networks with several millions of cells in an acceptable time. Besides the numbers, Fig. 4 shows some visual results.

	Network size [number of cells]				
	20k	50k	200k	1,500k	3,000k
Classical GCS	5 min	24 min	7 h	2 weeks	8 weeks
GCS with Tumble Tree	1 min	3 min	14 min	3 h	7 h

Table 1: Computing time of the classical GCS method rises quadratically with network size. Using the proposed tumble tree reduces complexity to $O(n \log n)$ and calculation times are significantly diminished in a way that an application depending on a huge number of cells can now be realized in an acceptable amount of time of few hours instead of weeks.

[1] The way down the tree looks like "tumbling" from the left to the right which delivers the name of our approach.

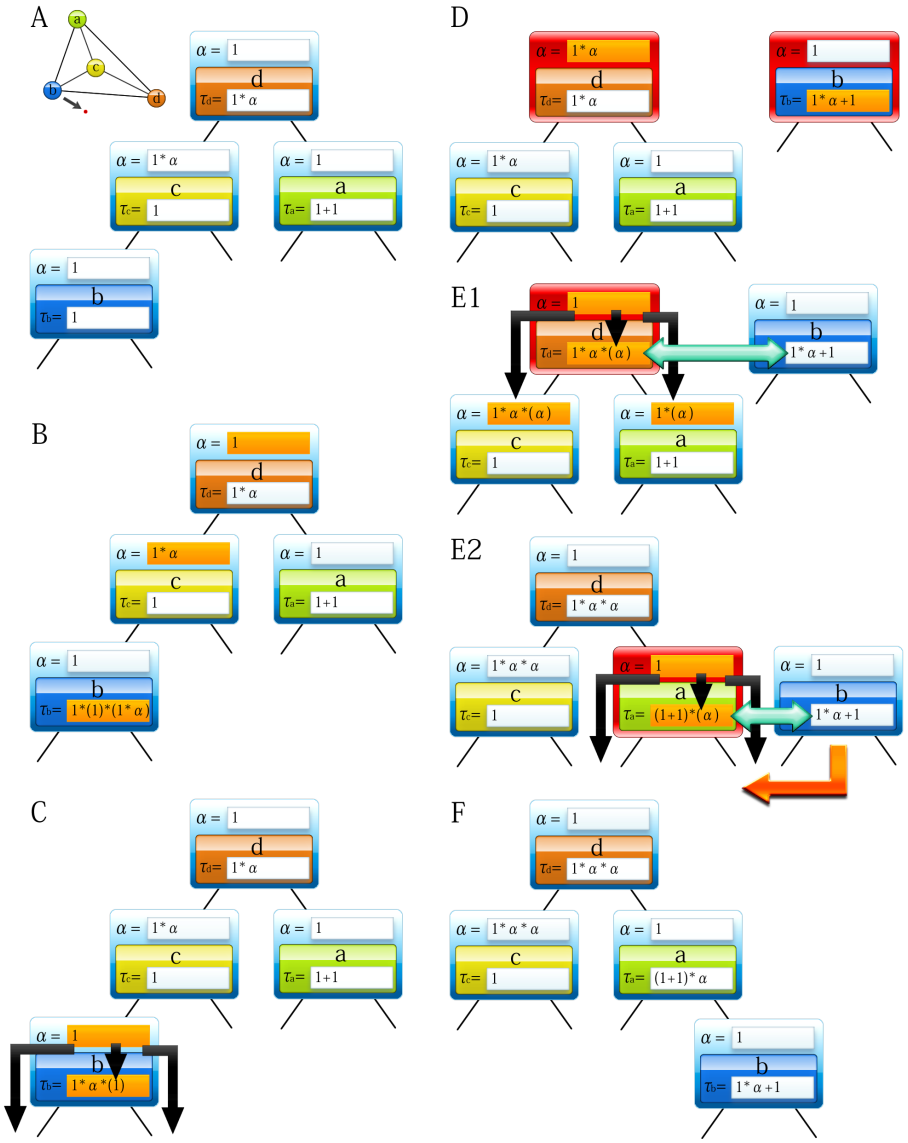

Fig. 3: A: The cell (a) has recently been selected as winning cell. Now (b) is the winner cell and the signal counter update is imminent. B: All alpha values from nodes above (b) are collected and multiplied with the signal counter of (b). C: Since (b) will be detached soon, the children of (b) must receive an update of their α-values. Therefore the update information is propagated to the neighboring nodes and α_k of (b) is set to 1. D: The signal counter of (b) is now valid and does not contain any postponed α-values. It can be detached from the network and its signal counter be incremented since it has been selected as BMU. The remaining cells are multiplied with α by postponing the multiplication in the α-value of node (d). E: The node (b) is reinserted. To determine its position its signal counter is compared with the counters of the visited cells, but before, each of the cell counters have to be determined by propagating the α_k down the tree. Then, the node will be remounted in the tree when it reaches an empty node. F: (b) has been reinserted and the signal counter update process is finished.

Summarizing, we showed how to reduce complexity of the common GCS approach from $O(n^2)$ to $O(n \log n)$. Essentially this is achieved by organizing the network cells in a tree-like data structure, and instead of a linear loop through the set of cells for certain operations a tree-traversal can be accomplished which requires logarithmic time complexity.

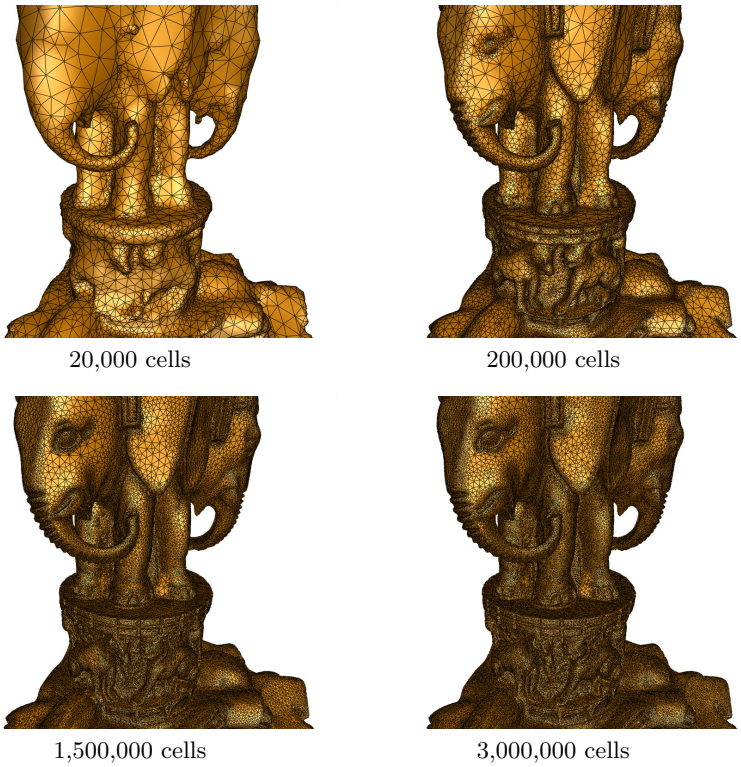

Fig. 4: A vital example application which emphasizes the importance of allowing the GCS network for growing to sizes of several millions of cells is surface reconstruction from arbitrary 3D point samples. Here, the quality of the reconstructed surface strongly depends on the number of triangles. When using a GCS for this tasks the number of cells equals the number of triangle vertices and thus the application is simply not possible being realized if few hundreds of thousands of samples (which is a typical amount in this type of tasks) require several days of computation time.

As an example, we presented an application case which was not possible being realized up to now. Since the GCS algorithm is utilized in a huge variety of applications, we are confident, that this approach might be more than simply an optimization of a classical method.

For future work, we are planning a parallelized version of the algorithm. Since even the general GCS algorithm is suitable for parallelization, we think we could

achieve interactive or real-time training rates — at least for specific simplified tasks.

References

1. Fritzke, B.: Growing cell structures - a self-organizing network for unsupervised and supervised learning. Neural Networks **7** (1993) 1441–1460
2. Kohonen, T.: Self-Organized Formation of Topologically Correct Feature Maps. Biological Cybernetics **43** (1982) 59–69
3. Ivrissimtzis, I., Jeong, W.K., Lee, S., Lee, Y., Seidel, H.P.: Neural meshes: surface reconstruction with a learning algorithm. Research Report MPI-I-2004-4-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany (October 2004)
4. Fritzke, B.: A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D.S., Leen, T.K., eds.: Advances in Neural Information Processing Systems 7. MIT Press, Cambridge MA (1995) 625–632
5. Hodge, V.J., Austin, J.: Hierarchical growing cell structures: Treegcs. IEEE Transactions on Knowledge and Data Engineering **13** (2001) 207–218
6. Wong, J.W.H., Cartwright, H.M.: Deterministic projection by growing cell structure networks for visualization of high-dimensionality datasets. J. of Biomedical Informatics **38**(4) (2005) 322–330
7. Mahony, S., Benos, P.V., Smith, T.J., Golden, A.: Self-organizing neural networks to support the discovery of dna-binding motifs. Neural Netw. **19**(6) (2006) 950–962
8. Deng, W., Wu, W.: Document categorization and retrieval using semantic microfeatures and growing cell structures. In: DEXA '01: Proceedings of the 12th International Workshop on Database and Expert Systems Applications, Washington, DC, USA, IEEE Computer Society (2001) 270
9. Cheng, G., Zell, A.: Externally growing cell structures for data evaluation of chemical gas sensors. Neural Computing and Applications **10**(1) (2001) 89–97
10. Hoffmann, M., Vrady, L.: Free-form surfaces for scattered data by neural networks. Journal for Geometry and Graphics **2** (1998) 1–6
11. Yu, Y.: Surface reconstruction from unorganized points using self-organizing neural networks yizhou yu. In: In IEEE Visualization 99, Conference Proceedings. (1999) 61–64
12. Álvarez, R., Noguera, J.V., Tortosa, L., Zamora, A.: A mesh optimization algorithm based on neural networks. Inf. Sci. **177**(23) (2007) 5347–5364
13. Mari, Jo a.F., Saito, J.H., Poli, G., Zorzan, M.R., Levada, A.L.M.: Improving the neural meshes algorithm for 3d surface reconstruction with edge swap operations. In: SAC '08: Proceedings of the 2008 ACM symposium on Applied computing, New York, NY, USA, ACM (2008) 1236–1240
14. Annuth, H., Bohn, C.A.: Growing cells meshing. In: In Proceedings of 3IA — The 13th International Conference on Computer Graphics and Artificial Intelligence. (2010)
15. Bayer, R.: Symmetric binary b-trees: Data structure and maintenance algorithms. Acta Inf. **1** (1972) 290–306