TreeGNG - Hierarchical Topological Clustering

K.A.J.Doherty, R.G.Adams, N.Davey

Department of Computer Science, University of Hertfordshire,
Hatfield, Hertfordshire, AL10 9AB, United Kingdom
{K.A.J.Doherty, R.G.Adams, N.Davey}@herts.ac.uk

Abstract. This paper presents TreeGNG, a top-down unsupervised learning method that produces hierarchical classification schemes. TreeGNG extends the Growing Neural Gas algorithm by maintaining a time history of the learned topological mapping. TreeGNG is able to recover from poor decisions made during the construction of the tree, and provides the novel ability to influence the general shape of the hierarchy.

1 Background

In this paper we present TreeGNG, a new hierarchical clustering algorithm based on a time audit trail of the graph connectivity of the Growing Neural Gas (GNG) [4] algorithm. GNG grows and prunes the network components in response to incremental learning. The dynamic nature of the network can produce disjoint graph structures, and these disconnected graphs "can be used to identify clusters in the input data" [5]. By maintaining a time audit trail of the graph connectivity, we can uncover hierarchical structure within the data.

The use of unsupervised learning techniques to discover the hierarchy of concepts from unlabelled data, primarily focus on extensions to Kohonen's Self-organising Feature Map (SOM) [8]. Recently, two hierarchical neural clusterers, HiGCS [1] and TreeGCS [6], have been proposed, both based on Bernd Fritzke's Growing Cell Structures (GCS) [3]. However, there are some limitations of GCS, which are detailed in [7] and [1], and the results of our own evaluation of GCS concur with these findings.

The models produced by the GCS and SOM algorithms are a topological mapping only if the topology of the graph matches the topological structure of the manifold from which the data are drawn. In [9], it was shown that an algorithm for the homogeneous distribution of the network nodes combined with the competitive Hebb rule produces the induced Delaunay triangulation of the manifold. GNG is an incremental implementation of this node distribution and competitive Hebbian learning concept, and the resultant graph structures are a topology preserving map of the manifold. It was reported in [2], that GNG converges rapidly and the quality of the clustering has little dependence

on the network parameters. The results of our own evaluation of GNG agree with these findings.

2 TreeGNG

Inspired by the claimed capabilities of TreeGCS, we investigated its performance. Whilst we believe the TreeGCS event auditing approach has potential, the resultant trees are binary, and we remain unconvinced as to the suitability of GCS as the underlying algorithm. In TreeGNG (Fig.1), we use GNG as the partitioning algorithm, and whilst the tree growth and pruning mechanisms follow the general outline of TreeGCS, we believe the inclusion of a growth window can improve the quality of the tree representation of data, as the resultant TreeGNG trees are not necessarily binary.

```
Until stopping criterion is satisfied
 For each input
  Run GNG and generate graph structure
  If (number of graphs has increased)
    Identify the tree node X that now points to multiple graphs
    For tree node Y in all ancestors of X
     If (Y Growth-Window is open)
       Create a tree node for each new graph and grow as children of Y
       Remove X
     Else
       Create a tree node for each new graph and grow as children of X
       Open X Growth-Window
     End If
    End For
  Else If (number of graphs has decreased)
    Insert a new tree node as a child of ancestor of the deleted graphs
    Remove the tree nodes associated with the deleted graphs
    Remove singleton tree nodes
  End If
  For tree node X in all tree nodes
    If (X Growth-Window is open) Reduce X Growth-Window
    If (X Growth-Window < 0) Close X Growth-Window
  End For
 End For
End program
```

Figure 1: The TreeGNG Algorithm

The standard GNG algorithm begins with a graph of two nodes joined by a single edge, and the TreeGNG tree is initialised with the root node pointing to this graph. As the GNG node and edge insertion routines grow the graph structure, the root node maintains the pointer to the revised graph. As a part of the GNG dynamics, edges are aged, and old inactive edges are periodically deleted. A graph edge deletion event can result in a graph splitting into two sub-graphs, and it is this splitting that initiates tree growth. As a graph splits into two sub-graphs, a growth window is activated for the tree node that pointed to the original graph. Any subsequent splitting of the sub-graphs that occurs whilst this window is open, results in tree nodes being inserted into the tree as

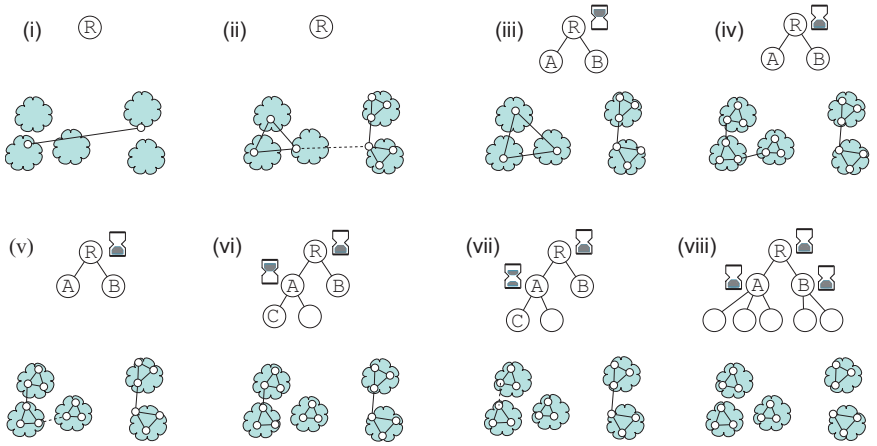

Figure 2: TreeGNG Growth Mechanism. In (i), GNG is initialised with two nodes joined by a single edge. The tree consists of a single root node R. In (ii), following a period of standard GNG dynamics, the dashed edge is marked for deletion in the GNG graph. As the edge is deleted in (iii), the graph splits and node R grows two children A and B to represent the increase in the number of graphs. A growth window is opened for node R. After a further period of GNG dynamics, the growth window for node R closes (iv) and any subsequent graph splitting will result in tree growth under nodes A or B. In (v), the dashed edge is marked for deletion, and in (vi), the edge is deleted producing child growth beneath node A. The growth window for node A is opened. In (vii), the dashed graph edge is marked for deletion, and the natural tree growth should occur as children of node C, but because the growth window for node A is still open, the new node is inserted as a child of node A, i.e. a sibling of node C. Following a period of GNG dynamics and the growth of two children of B, the final tree structure is shown in (viii).

siblings of the sub-graph tree node rather than as its children (Fig. 2). Without this window, the tree growth is event driven and is always binary.

Tree pruning occurs as a result of an edge insertion event occurring in the GNG graph, merging two graphs into a single super-graph. The merging graphs may not be siblings in the tree, and so a new tree node pointing to the new super-graph is created as a child of the common ancestor of the two merged graphs. The tree nodes pointing to the original two merged graphs are removed from the tree. Finally, singleton tree nodes are removed (Fig. 3). In the resultant tree, only leaf nodes of the tree maintain pointers to the GNG graph structures.

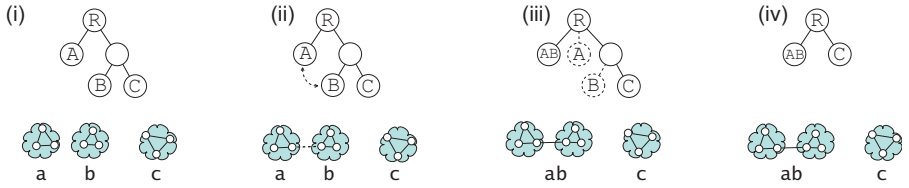

Figure 3: The TreeGNG Pruning Mechanism, also demonstrating how the algorithm can recover from poor decisions made in the construction of the tree. In (i), TreeGNG has formed a poor tree structure in which tree nodes B and C are siblings. With the standard GNG dynamics, it is likely that an input would cause graphs a and b to merge as the dashed edge is inserted. To maintain the correct tree structure, this requires tree nodes A and B to merge (ii). In (iii), as the graphs a and b are merged, then a new tree node AB (pointing to the merged ab graph) is inserted into the tree as a child of the common ancestor of A and B. The dashed tree nodes (A and B) are removed from the tree. Finally, in (iv), the intermediate singleton tree node between R and C is removed.

3 Results

We chose to demonstrate the utility of our approach on synthetic data. In addition to the GNG network parameters, TreeGNG requires a growth window parameter for child growth. With a small growth window, for example every iteration, then the expectation was for the resultant tree to be binary. With a very large growth window, such that the tree is generated only when the user-defined stopping criterion is satisfied, then the expectation was for the tree structure to be either a single root node if the GNG graph structure contains no sub-graphs, or the tree to have a branching factor corresponding to the number of sub-graphs, and a ply of 1. The aim of our investigation was to determine if between these two extremes, the tree structure that was produced corresponded to our expectations of an appropriate tree.

We report the results for a data set consisting of 9 Gaussian sources in $\mathbb{R}^2$ arranged in, what we consider to be, 3 large clusters, each consisting of 3 smaller clusters. The expectation with these data was for the ideal tree to have a branching factor of 3, and a maximum ply of 2. We assessed the quality of a tree by an examination of the branching factors at each ply. Its accuracy was determined by how well it conformed to our expectations. Fig. 4 shows typical resultant trees for a range of growth windows, and the accuracy of the branching factor at differing ply. For each set of parameters, we repeated the test 10 times, and we report the average accuracy achieved over these runs. We were looking for 3 children beneath the root node, and we achieved 100% accuracy with the growth window in the range 1 to 3 epochs. The wide range for the growth window, and the level of accuracy, was encouraging. Examining the nodes at ply one, we found the peak accuracy at a growth window of 2.5

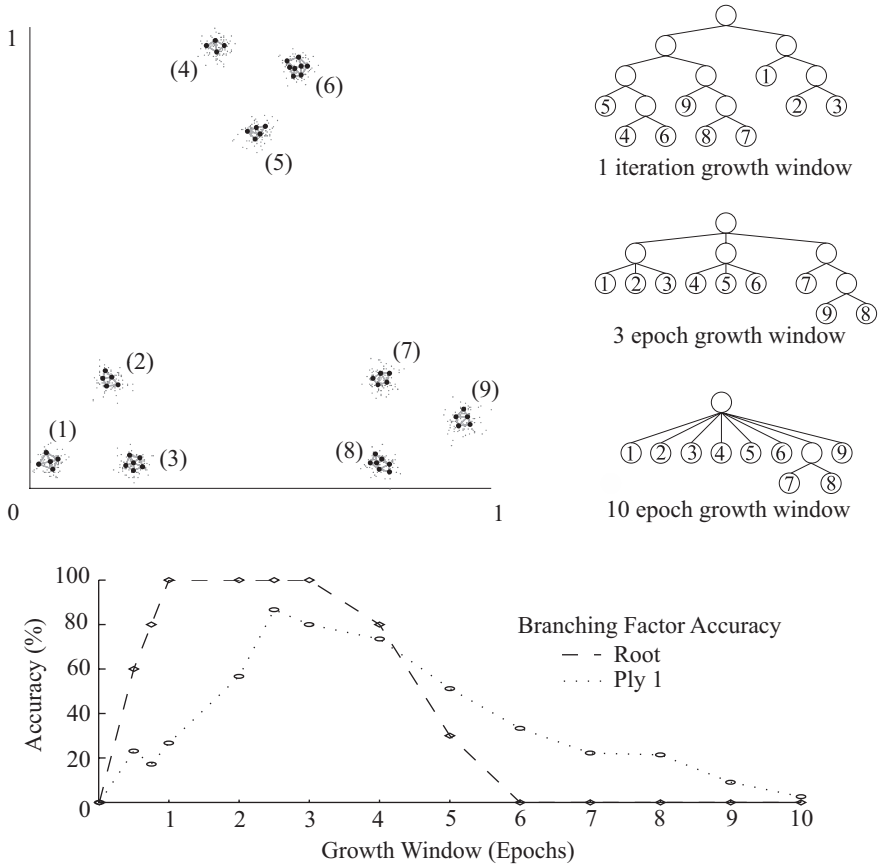

Figure 4: TreeGNG results for a range of growth windows. The scatter plot on the upper left shows the data and the GNG graphs after 100 epochs, with a GNG node insertion rate of two nodes per epoch (maximum of 50 nodes), and a GNG edge deletion age of 1/10 of an epoch. With a growth window of one iteration the tree was binary, and with a growth window of 10 epochs, the tree structure was essentially flat. Of more interest, is that a growth window of approx. 2.5 epochs consistently produced a tree broadly in-line with our expectations of an appropriate hierarchical representation. The accuracy of the tree branching factor at the root and ply one nodes, are shown in the lower graph.

epochs, where 87% of the nodes had the required branching factor of three.

We have repeated these tests on a range of synthetic data sets, and our preliminary results suggest that TreeGNG is capable of generating tree structures broadly in-line with our expectations.

4 Conclusion

TreeGNG incrementally learns and adjusts the tree structure in response to the input data and produces a stable hierarchical representation for a broad range of network parameters. The periodic edge removal and competitive Hebbian learning of the GNG algorithm allows the network to recover from poor decisions in the generation of the hierarchy, and thus overcomes one of the problems with TreeGCS and the divisive statistical methods. The algorithm provides the novel ability to influence the general shape of the hierarchy, and for a range of growth windows, produces trees that we consider to be representative of the data structure.

Work is continuing to clarify the relationships between the GNG node insertion rate, GNG edge deletion age and growth window duration; and in future work, we shall assess the performance of TreeGNG on real-world data.

References

[1] V. Burzevski and C. K. Mohan. Hierarchical growing cell structures. In *Proc. IEEE Int. Conf. Neural Networks*, volume 3, pages 1658–1663, Washington D.C., 1996.

[2] X. Cao and P.N. Suganthan. Hierarchical overlapped growing neural gas networks with applications to video shot detection and motion characteristics. In *Proc. Int. Joint Conf. Neural Networks*, volume 2, pages 1069–1074, Hawaii, USA, 2002. IEEE.

[3] B. Fritzke. Growing cell structures - a self-organising network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[4] B. Fritzke. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, pages 625–632, 1995.

[5] B. Frizke. Unsupervised ontogenic networks. In *Handbook of Neural Computation*, pages C2.4:1–C2.4:16. IOP Publishing Ltd and Oxford University Press, 1997.

[6] V. J. Hodge and J. Austin. Hierarchical Growing Cell Structures: TreeGCS. *IEEE Trans. Knowledge and Data Engineering*, 13(2):207–218, 2001.

[7] M. Köhle and D. Merkl. Visualising similarities in high dimensional input spaces with a growing and splitting neural network. *Lecture Notes in Computer Science*, 1112:581–586, 1996.

[8] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 2nd edition, 1997.

[9] T. M. Martinetz and K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.