Article:
Hodge, V.J. and Austin, J. (2001) Hierarchical growing cell structures: TreeGCS. IEEE Transactions on Knowledge and Data Engineering. pp. 207-218.

https://doi.org/10.1109/69.917561

Hierarchical Growing Cell Structures: TreeGCS

Victoria J. Hodge and Jim Austin

Abstract—We propose a hierarchical clustering algorithm (TreeGCS) based upon the Growing Cell Structure (GCS) neural network of Fritzke. Our algorithm refines and builds upon the GCS base, overcoming an inconsistency in the original GCS algorithm, where the network topology is susceptible to the ordering of the input vectors. Our algorithm is unsupervised, flexible, and dynamic and we have imposed no additional parameters on the underlying GCS algorithm. Our ultimate aim is a hierarchical clustering neural network that is both consistent and stable and identifies the innate hierarchical structure present in vector-based data. We demonstrate improved stability of the GCS foundation and evaluate our algorithm against the hierarchy generated by an ascendant hierarchical clustering dendogram. Our approach emulates the hierarchical clustering of the dendogram. It demonstrates the importance of the parameter settings for GCS and how they affect the stability of the clustering.

Index Terms—Unsupervised, growing, neural, network, hierarchical, cluster, topology.

1 Introduction

The ability to introduce hierarchical structured knowledge into a system using autonomous learning of examples has received much attention, see, for example, [15], [8], [11], or [2], where the papers' authors describe the implementation of a variety of hierarchical systems. Proponents argue that humans intuitively cluster information and, in real-world situations, concepts are related and frequently organized into a generalization hierarchy by a human subject [10]. However, it is very difficult for a human to construct a hierarchy by hand that is both consistent and does not introduce redundancy. By implementing an autonomous clustering process, we can rapidly organize the data space for the system and its users while overcoming the inconsistency and innate redundancy of human-generated abstractions. The hierarchical structure generated may then be used to discover trends and generalization information, to allow knowledge to be stored at the requisite level of granularity, and for further processing in the system. The system using the hierarchy can utilize the distances between nodes in the hierarchy to calculate the similarity of concepts. The hierarchy can be exploited to restrict data searches to regions of the hierarchy, thus minimizing the search space and progressively refining the search from general concepts to specific concepts. Searching is computationally expensive and, by restricting the search to subregions of the hierarchy, the computational expense can be minimized. Hierarchical clustering is frequently used in Information Retrieval to cluster words based on the frequency of occurrence of the words or to cluster documents using the words in the documents as indexing values, see, for example [16].

• *The authors are with the Department of Computer Science, University of York, UK YO10 5DD. E-mail: {vicky, austin}@cs.york.ac.uk.*

Manuscript received 28 July 1999; revised 17 Mar. 2000; accepted 16 June 2000.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 112683.

Clustering may be defined as a process of partitioning a multidimensional input data space into groups of similar objects. The similarity of the objects is determined by a cardinal similarity measurement over the object attributes. A frequently used similarity measure is Euclidean distance. The clustering arranges the objects so that objects within a cluster are more similar to each other than to objects in another cluster; the intracluster similarity is higher than the intercluster similarity.

Clustering algorithms have been investigated previously and cover a vast range of methodologies (see, for example, Jain and Dubes [7] for a discussion of the statistical approaches or Song and Lee [15] for a discussion of neural network techniques). The clustering algorithms described in [7] and [15] are used in a wide variety of applications, including pattern recognition, information retrieval, image processing, and natural language processing. However, nearly all clustering techniques suffer from at least one of the following:

1. They assume specific forms (for example, a normal form) for the probability distribution of the input space being modeled,
2. They require unique global minima of the input probability distribution,
3. They cannot handle identical cluster similarities, i.e., where two pairs of clusters have equal similarity,
4. They do not scale well as the training time is often $O(n^2)$ with respect to the number of input vectors, or
5. They require prior knowledge of the data to allow the cluster topology to be specified and more knowledge to allow the parameters to be set for the clustering method.

Hierarchical clustering superimposes a structure onto the clusters below. In the hierarchy produced, each leaf node represents one object from the entire set of objects and the root node represents the whole set of objects. Thus, the generality of the clusters in the hierarchy increases

monotonically from leaf nodes to the root. The hierarchy may be formed agglomeratively (bottom-up) by progressively merging the most similar clusters. Once two clusters are merged, their union becomes the new cluster and this union is used for later comparisons and merging. Alternatively, the hierarchy may be obtained divisively (top-down) by iteratively dividing a supercluster into finer-grained subclusters, according to some metric. The subclusters are then treated as the objects for future subdivisions.

Probably the most commonly used statistical hierarchical clustering approach is the dendogram [3]. For dendograms, there is initially one data point per cluster. The algorithm iteratively determines the distance between each pair of clusters and the two clusters with the smallest distance are merged, producing a branch in the cluster tree. The merging of the most similar clusters is repeated until only one cluster is left, which represents the entire set of input data points. There are many flavors of dendograms; the difference results from the various similarity criteria employed [3]. We use Ward's method [3] as a comparison for our new method (TreeGCS) in this paper.

One of the problems with dendograms is that they cannot be visualized for large data sets as the diagram is just too complex with too many leaf nodes and branches to allow comprehension. Dendograms also have problems when more than one pair of clusters have equal similarities, as only one new data item may be assimilated at each iteration. If there are two pairs of clusters with identical values, then one pair must be merged on the current iteration and the other pair on the next iteration. The order of the two merges is arbitrary. Therefore, we feel the dendogram is ideal for structure and cluster comparison, but would not be suitable for the learning system outlined in the first paragraph, as it fails on points 3 and 4 above.

Connectionist methods model the brain's fundamental systems through a parallel, distributed processing system comprising units linked by weighted connections. A pattern is input to the network and every unit is used to calculate the network output using the input values and connection weights. Probably the most common connectionist clustering approach is Self-Organizing Maps (SOMs) [9]. SOMs induce a topological map of relationships among vectors in the input data space, mapping each input vector to a single neuron in a lattice topology. However, SOMs only form a flat cluster topology; they are not hierarchical.

Therefore, SOMs have been extended to Hierarchical SOMs (HSOMs) [12], which are multilayered SOMs and thus allow a cluster hierarchy to be formed. Each neuron in the lattice on a metalayer points to an entire SOM on the layer below. The sublayer is a finer grained representation of the knowledge in the higher layer. Mangiameli et al. [11] demonstrated SOMs superiority to seven bottom-up hierarchical clustering methods with respect to accuracy, robustness, and ease of use. However, HSOMs generally require prior knowledge of the input distribution to predict a suitable topology. For example, in [13], the author describes the DISCERN system and states that one of the main liabilities of the DISCERN system is that "*much of the internal knowledge structures are fixed and specified in advance*." Therefore, Song and Lee [15] have produced the Structurally Adaptive Intelligent Neural Tree (SAINT) system that automatically determines a cluster hierarchy composed of SOM networks, removing the need to prespecify the structure. Where the mapping error from the input vector to the best matching node is high, new nodes are added. Any superfluous nodes are deleted and nodes that are very similar are merged. This dynamically creates a tree-structured SOM. However, the lattice topology within the SOM subnetworks fixes the number of neighbors attached to each node and loses flexibility. SOMs also cannot form discrete (disconnected) clusters, thus inhibiting the data representation. The clusters have to be determined by hand after the algorithm terminates and this introduces the innate subjectivity of human judgements. We need a hierarchical methodology that maintains the superiority of SOMs over statistical techniques, but automatically and dynamically determines the cluster hierarchy, thus minimizing human intervention, does not confine nodes to a strict lattice structure, and allows the network to split into separate clusters.

In this paper, we propose and develop such a hierarchical clustering algorithm, called TreeGCS, which is based on Fritzke's GCS neural network [5]. TreeGCS is an unsupervised, growing, self-organizing hierarchy of nodes able to form discrete clusters. In TreeGCS, high dimensional inputs are mapped onto a two-dimensional hierarchy reflecting the topological ordering of the input space. We selected Fritzke's GCS as our foundation rather than Fritzke's Growing Neural Gas (GNG) [6] or Bruske and Sommer's Dynamic Cell Structures (DCS) [1], as GNG and DCS maintain the input data dimensionality in the network representation. Thus, visualization of the hierarchy would not be possible with GNG and DCS unless the input data was limited to an impractical three-dimensions or less to permit visualization. It is also far simpler to superimpose a cluster hierarchy on top of a two-dimensional cluster representation. We feel that these two benefits of dimensionality reduction outweigh the possible drawback of dimensionality reduction where mapping on to a lower-dimensional topology can distort the data and lose similarity information. Fritzke [4] has demonstrated the superiority of this mapping for GCS compared to SOMs. We demonstrate how our hierarchical structure emulates the similarity structure produced by a dendogram in Section 4. TreeGCS is preserving the similarities in the input data space identified by the dendogram.

Our algorithm is similar to HiGS [2]. HiGS (see Fig. 1) is a top-down, self-organizing hierarchy that aims to map the input vector distribution onto a two-dimensional hierarchical structure. The HiGS algorithm inserts a node every second iteration in the subnetwork below the root that has the greatest error[1] and iteratively in each subnetwork below this also with the maximum error to spread and reduce the error. Prior to insertion, the HiGS algorithm collates all signal counters and, if all counters are approximately equivalent, a new subcluster is generated for each node to

1. The maximal mean Euclidean distance between the winning node and all other nodes for each input where the winner lies in the respective subnetwork.

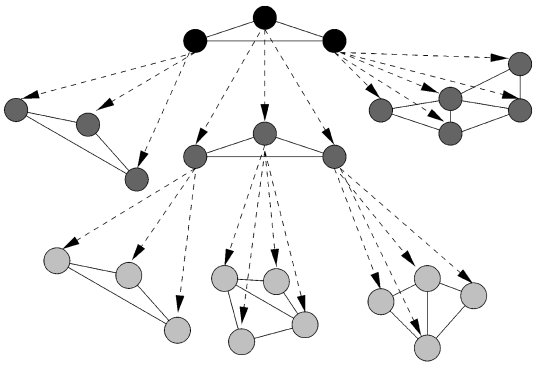

Fig. 1. Illustration of HiGS topology.

allow the hierarchy to grow down. Superfluous nodes are deleted from leaf networks if few inputs map onto them. However, the structure generated from HiGS does not match our requirements. The topology induced for HiGS is not a tree configuration as the parent must be a member of a cluster of cardinality at least three (see Fig. 1). The HiGS algorithm generates child clusters and periodically deletes superfluous children so, at any particular time, the tree representation may be incorrect (i.e., a superfluous child cluster may be due for deletion). Our proposal maintains the correct cluster topology at each epoch.

In the remainder of this paper, we describe and evaluate Fritzke's GCS in Section 2 and identify two weaknesses with the stability and the criticality of the parameter settings. In Section 3, we detail our hierarchical extension and how we improve these two GCS limitations. We qualitatively evaluate TreeGCS against a dendogram to demonstrate our improved stability in Section 4. In Section 4, we also demonstrate the importance of the parameter settings and posit recommendations for selecting suitable values. We further analyze the results in Section 5 and summarize and provide future recommendations in Section 6.

2 GCS

Here, we outline the GCS algorithm that forms the foundation for our hierarchical methodology. The initial topology of GCS is a two-dimensional structure (triangle) of cells (neurons) linked by vertices. Each cell has a *neighborhood* defined as those cells directly linked by a vertex to the cell. The input vector distribution is mapped onto the cell structure by mapping each input vector to the best matching cell. Each cell has a vector attached denoting the cell's position in the input vector space; topologically close cells have similar attached vectors. On each iteration, the attached vectors are adapted toward the input vector. The adaptation strength is constant over time and only the *best matching unit (bmu)* and its direct topological neighbors are adapted, unlike SOMs, where the adaptation occurs in a progressively reducing radius of neurons around the bmu. Cells are inserted where the cell structure under-represents the input vector distribution and superfluous cells which are furthest from their neighbors are deleted. Each cell has a "*winner counter*" variable denoting the number of times that cell has been the bmu. The winner counter of each cell is reduced by a predetermined factor on every iteration. The aim is to evenly distribute the winner counter values so that the probability of any cell being a bmu for a random input is equivalent, i.e., the cells accurately represent the input space.

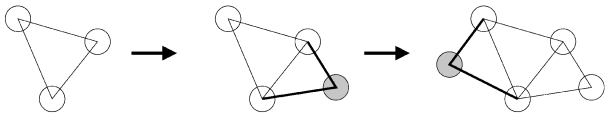

Fig. 2. Illustration of cell insertion: A new cell and associated connections are inserted at each step.

The GCS learning algorithm is described below; the network is initialized in point 1 and points 2 to 7 represent *one iteration*. An *epoch* comprises an iteration (points 2 to 7) for each input vector in the data set.

1. A random triangular structure of connected cells with attached vectors ($w_{c_i} \in \Re^n$) and E representing the winner counter (the number of times the cell has been the winner) is initiated.
2. The next random input vector ξ is selected from the input vector density distribution. The input vector space is represented as real-valued vectors of identical length.
3. The best matching unit (bmu) is determined for ξ and the bmu's winning counter is incremented.

$$bmu = \| \xi - w_c \|_{min_{c \in network}} \text{ where } \| \| \text{ is the Euclidean distance}$$
$$\Delta E_{bmu} = 1.$$

4. The best matching unit and its *neighbors* are adapted toward ξ by adaptation increments set by the user:

$$\Delta w_{bmu} = \epsilon_{bmu}(\xi - w_{bmu})$$
$$\Delta w_n = \epsilon_i(\xi - w_n) \quad (\forall n \in neighborhood).$$

5. If the number of input signals exceeds a threshold set by the user, a new cell (w_{new}) is inserted between the cell with the highest winning counter (w_{bmu}) and its farthest *neighbor* (w_f)—see Fig. 2. The weight of the new unit is set according to:

$$w_{new} = (w_{bmu} + w_f)/2.$$

Connections are inserted to maintain the triangular network configuration. The winner counter of all *neighbors* of w_{new} is redistributed to donate fractions of the neighboring cells' winning counters to the new cell and spread the winning counter more evenly,

$$\Delta E_n = -\frac{1}{|\ n\ |} E_n \quad \forall n \in neighborhood\ of\ w_{new}.$$

The winner counter for the new cell is set to the total decremented:

$$E_{new} = \sum \left(\frac{1}{|\ n\ |} E_n \quad \forall n \in neighborhood\ of\ w_{new} \right).$$

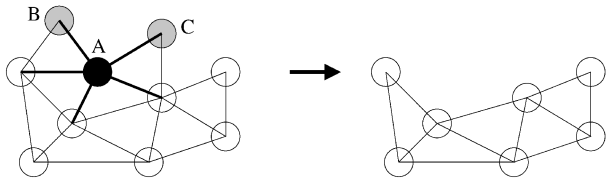

Fig. 3. Illustration of cell deletion: Cell A is deleted. Cells B and C are within the neighborhood of A and would be left dangling by removal of the five connections surrounding A, so B and C are also deleted.

6. After a user-specified number of iterations, the cell with the greatest mean Euclidean distance between itself and its *neighbors* is deleted and any cells within the neighborhood that would be left "dangling" are also deleted (see Fig. 3). Any trailing edges are deleted to maintain the triangular configuration:

$$Del = max_{c \in network}\left(\frac{\sum \parallel w_c - w_n \parallel}{card(n)} \forall n \in neighborhood\right).$$

7. The winning counter variable of all cells is decreased by a user-specified factor to implement temporal decay:

$$\Delta E_c = -\beta E_c \ \ \forall c \in network.$$

The user-specified parameters are: the dimensionality of GCS, which is fixed at 2 here; the maximum number of neighbor connections per cell; the maximum cells in the structure, ϵ_{bmu}; the adaptation step for the winning cell, ϵ_i; the adaptation step of the neighborhood, β; the temporal decay factor; the number of iterations for insertion; and the number of iterations for deletion.

The algorithm iterates until a specified performance criterion (e.g., network size) is met. If the maximum number of epochs along with the maximum number of cells is specified as the termination criterion, then new cells are inserted until the maximum number of cells is reached. Once the maximum has been reached, adaptation continues on every iteration and cells may be deleted. The cell deletion reduces the number of cells to below the maximum allowing one or more new cells to be inserted until the maximum number of cells is reached again. Deletion removes superfluous cells while creating space for new additions in underrepresented regions of the cell structure, so the input distribution mapping is improved while the maximum number of cells is maintained.

Fritzke has demonstrated superior performance for the GCS over SOMs [4]. Superiority with respect to:

- Topology preservation, with similar input vectors being mapped onto identical or closely neighboring neurons ensuring robustness against distortions.
- Neighboring cells having similar attached vectors, ensuring robustness. If the dimensionality of the input vectors is greater than the network dimensionality, as in our evaluation in Section 4, then the mapping usually preserves the similarities among the input vectors.
- Lower distribution-modeling error (which is the standard deviation of all counters divided by the mean value of the counters). This increases fault tolerance as each cell is only related to a small fraction of the input vector space, so any error will only affect a small region of the cell structure.

2.1 GCS Evaluation

The run-time complexity for GCS is: Determine the bmu by calculating the distance between the input vector and each cell's attached vector for every dimension which is $(numberCells * dimension)$. This calculation is performed for every input vector on each epoch. The run-time for the GCS is therefore

$$(numberCells * dimension * numberInputs * epochs).$$

During evaluation of the GCS algorithm, we discovered that it was susceptible to the order of the input data. We rearranged the order of the input vectors, e.g., the first vector was switched to become the last, the second vector was switched to become the second last, etc., until a completely new ordering of the input data space was generated. Each input ordering was presented to GCS. For each input ordering, the cluster topology altered. The algorithm is *consistent*—an identical cluster topology is generated for an identical input space but is *unstable*—a different configuration is produced if the input space is reordered. We define *consistency* as generating the same cell structure when the algorithm is run repeatedly with the same input vector order. We define *stability* as generating the same cell structure if the algorithm is run repeatedly with the same input vectors, but the vectors are input in different orders. We investigated an adaptation of the GCS algorithm in which the quantization error[2] is used to determine insertion and deletion, thus minimizing the overall quantization error (as recommended by Fritzke [5]). However, this did not improve cluster stability and generated inappropriate cluster cardinalities, for example, 76 cells for a cluster that represented six input vectors and six cells representing 36 input vectors as concurred by Fritzke. We concluded that the original winning counter implementation was most appropriate, but that it appeared to be committing to clusters too early and could not recover. We amended the deletion step so that cells may only be deleted once 90 percent of the total required cells have been added to the cell structure. We empirically evaluated various values and found 90 percent to be most consistent; lower values commit to clusters too early and 100 percent can inhibit cluster splitting. While not eliminating the instability problem completely, it was significantly ameliorated during 1,680 epochs by delaying deletion until 90 percent of the maximum number of cells are present.

However, over a greater number of epochs (e.g., 30,000) the stability still remains problematic. We further improved

2. The quantization error is the accumulated distance squared between the cell reference vector and all input signals for which the cell is the bmu.

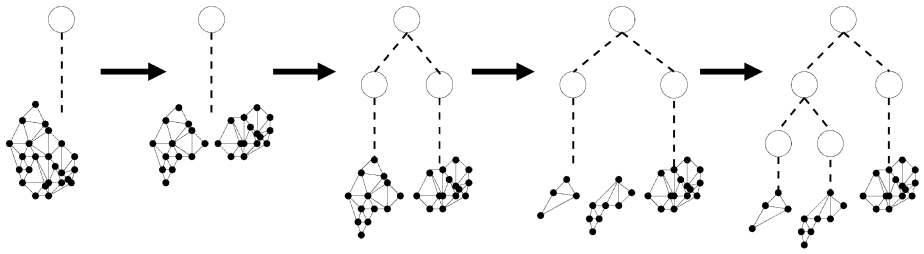

Fig. 4. Illustration of cluster subdivision.

the stability over many epochs by cycling through different data orders (i.e., the order of the vectors in the input space was altered). In this paper, we utilize three data orderings to illustrate the initial susceptibility of the algorithm to the input data order and how cycling improves the stability. The GCS reads the first data order for one epoch, the second for the next, the third and back to the first, etc., until the requisite number of epochs has been completed. This enables the different data orders to counteract each other and, thus, introduces more stability, as demonstrated in Section 4.

There is also a criticality of parameter settings. Köhle and Merkl [12] observed the problem and we illustrate the phenomenon and from the results suggest possible parameter settings.

3 TreeGCS

The tree is constructed from, and superimposed onto, the standard GCS algorithm exposited above. A tree root node points to the initial cell structure and incorporates a list of all cells from the GCS.

When the cluster subdivides, new nodes are added to the tree to reflect the additional clusters (see Fig. 4) and the lists are updated with each leaf node holding a list of all cells in its associated cluster. Only leaf nodes maintain a cluster list. A parent's cluster list is implicitly a union of the children's cluster lists and is not stored for efficiency—minimizing memory usage. No constraints are imposed on the tree, hence, it is dynamic and requires no prior data knowledge—the tree progressively adapts to the underlying cell structure. The hierarchy generation is run once after each GCS epoch. The running time is $O(cells)$ as we essentially breadth-first search through the entire cell structure.

3.1 Algorithm (in pseudocode)

For each epoch,
- **Execute** the GCS epoch, forming an unconnected graph representing the disjoint clusters.
- **Breadth first search** from the final winning cell for the epoch to determine which cells are present in the cluster.
- **While** some cells remain unprocessed,
  - **Breadth first search** from the next unprocessed cell to determine which cells are present in the cluster.
- **If** the number of clusters has increased from the previous epoch, **then** any tree nodes that point to multiple clusters are identified and child nodes are added for each new cluster formed (see Fig. 4). The cluster list of the parent is deleted and cluster lists are updated for the child nodes. If a cluster is formed from new cells (cells inserted during the current epoch), then a new tree node is added as a child of the root and the new cluster cells added to the new node's list.
- **Else if** the number of clusters has decreased, a cluster has been deleted and the associated tree node is deleted. The tree is tidied to remove any redundancy (see Fig. 5).
- **For** each unprocessed cluster, the tree node that points to that cluster is determined, the cluster list emptied, and the new cells are added.

Once the requisite number of GCS epochs have been completed, the GCS cells are labeled. Each input vector is input to the GCS and the cell identifier of the bmu is returned. The bmu can then be labeled with the input vector (or other label as appropriate). All tree nodes except leaf nodes have only an identifier and pointers to their children. The leaf nodes have an identifier, but also maintain a list of the identifiers of the GCS cells in their cluster. When the GCS bmu is identified, the associated leaf node can be identified and the tree can be traversed to find all ancestor nodes.

The superimposed tree will be symmetrically equivalent; it is right/left independent. Trees that appear superficially dissimilar can, in fact, represent identical cluster topologies.

The tree may be degenerate if the clustering subdivides degenerately. This obviously induces an inefficient representation, but is necessary for soundness and completeness of the clustering breakdown.

In the next section, we demonstrate our stability improvement for the underlying GCS algorithm. We restrict when cell deletion may commence in all test runs. We demonstrate how cycling through different data orders produces more stable clusters than just inputting a single data order repeatedly. We also demonstrate the criticality of the parameter settings for GCS and outline how we aim to

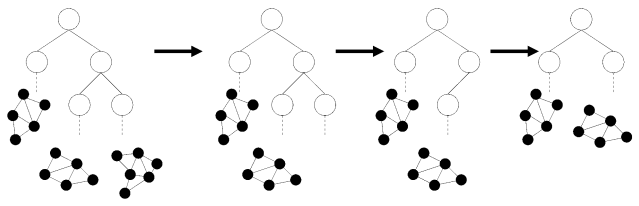

Fig. 5. Illustration of cluster deletion. The rightmost cell cluster is deleted during an epoch (Step 2)—this leaves a dangling pointer. The node with the dangling pointer is removed (Step 3), leaving redundancy in the hierarchy. The redundancy is removed in the final step.

overcome the problem. Selecting suitable parameter settings is a combinatorial problem and affects many statistical and neural clustering methods [3], [15]. We make recommendations for a simple method of parameter selection for GCS.

We evaluate the algorithm on a small data set to permit a simple and concise illustration of the cluster structure produced. Fritzke has previously demonstrated GCS's ability to accurately map benchmark data sets. Fritzke demonstrates GCS's superior performance with respect to the number of training epochs compared to three common neural networks for the spiral benchmark [5]. He also identified GCS's superiority with respect to correctly classified test patterns over six common neural network approaches and the nearest-neighbor approach for mapping the vowel recognition benchmark data set [5]. We do not aim to recreate Fritzke's cluster accuracy evaluations, but focus our evaluation on improving cluster stability and selecting parameter settings.

4 Evaluation

4.1 Evaluation

The data set for the stability investigation is comprised of 41 47-dimensional real-valued vectors with no missing values, representing data for 41 countries in Europe:

$$(x_1, x_2, \ldots, x_{47}) \in \Re^{47}, EuroData = \{x^1, x^2, \ldots, x^{41}\}.$$

The attributes were geographical, population, economic, communication, and transportation factors obtained from the World Factbook [17]. Each vector in the data set has a clear label (the country name), so the position of each country's vector in the hierarchical clustering is easily identified. This allows us to qualitatively compare the various hierarchies. A very similar data set is used in [8]. SOMs have been criticized for their inability to map input distributions with a higher dimensionality than the number of input vectors as stated previously in Section 2. Here, we can use such a data set to demonstrate the stability improvements we have made and also to demonstrate our technique's ability to accurately map such a data set onto a two-dimensional hierarchy emulating a dendogram clustering. We compare stability by assessing the similarity of the hierarchical cluster topology produced for the respective input data orders and comparing the cluster contents. Is the topology a similar shape? Do the same countries fall into each half of the hierarchy? Are the same countries consistently clustered together across the input data orders?

We use three different orderings of the data to evaluate stability.

1. Alphabetical order of the country names (i.e., the 47-dimensional vectors that represent each country are arranged in the input vector space according to the alphabetical order of the respective countries).
2. Middle to front—the second half of the alphabetical order file was moved to the front.
3. Numerical order—sorted on the first attribute of each vector.

4.1.1 Dendogram

We qualitatively evaluate the hierarchies produced by TreeGCS against the hierarchy generated by a bottom-up hierarchical dendogram utilizing Ward's algorithm. The dendogram hierarchically illustrates similarities. In evaluations by Mangiameli et al. [11], Ward's method proved superior to six other hierarchical bottom-up methods with respect to classification accuracy, so Ward's method should be best for structure comparison. In Ward's algorithm, the two clusters to merge at each step are selected to maximize an objective function (error sum of squares), thus, information loss is minimized (see Fig. 9 for an example of a dendogram hierarchy).

We presented the three different data orders described above to the dendogram to produce three hierarchies. There were some minor variations in the dendogram hierarchies produced from the three data orders, but the contents of the clusters were identical for all hierarchies. See Fig. 9 for the hierarchy produced from the alphabetical vector order. If we take the dendogram as showing three clusters, the clusters produced are:

- {Den, Fra, Ger, It, UK}
- {Lux}
- {Alb, And, Aus, Bel, Bos, Bul, Cro, Cyp, Cze, Eir, Est, Far, Fin, Gib, Gre, Hun, Ice, Lat, Lie, Lit, Mac, Mal, Mon, NL, Nor, Pol, Rom, SM, Ser, Slk, Sln, Spa, Swe, Swi, Ukr}.

4.1.2 TreeGCS

The parameter settings for TreeGCS throughout the evaluation were: $\epsilon_{bmu} = 0.02,\ \epsilon_i = 0.002,\ \beta = 0.0002$, and a maximum of 10 connections per cell. These values produced the maximal consistency for the input data during a brief empirical assessment and using our parameter selection approach recommended in Section 4.2 where we reduce ϵ_{bmu} by scale factor 10 to give ϵ_i, which we further reduce by scale factor 10 to give β. The maximum number of cells in the structure was set to 123; there are three times as many cells as countries to ensure maximal spread with minimal redundancy. The number of iterations before a cell insertion was set to one; the insertion value is set to ensure that all input vectors are used for cell insertion in turn. The number of iterations between each delete was set to 5,000; this value ensures maximal adaptation before any cells are deleted. The algorithm was set to run for 30,000 epochs for a thorough evaluation of stability.

Each of the nine data orders (three single-pass and six cyclic) was presented to TreeGCS to allow us to compare the stability of a single data order input to TreeGCS repeatedly against cycling through three different data orders. For the single-pass, the single data order was presented repeatedly for 30,000 epochs. For the cyclic approach, each data order was presented in turn for one epoch, allowing us to cycle through the three data orders in three epochs. There are six permutations of the three data orders (1,2,3)(1,3,2)(2,3,1)(2,1,3)(3,1,2)(3,2,1) with respect to the data order numbers in Section 4, giving six hierarchies.

As stated in Section 2.1, the running time is

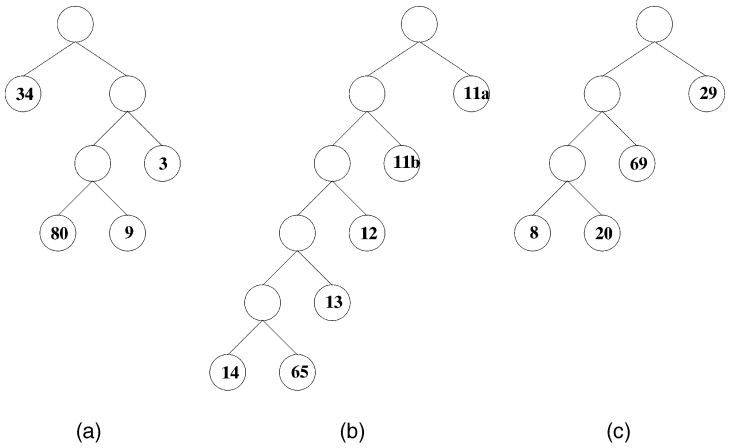

Fig. 6. Illustration of the cluster topology for the three single pass TreeGCS runs. (a) = alphabetical order, (b) = middle-to-front, and (c) = sorted numerically by the first attribute. The numbers indicate the number of GCS cells in the cluster.

$$(numberCells * dimension * numberInputs * epochs)$$

for the GCS foundation and $O(cells)$, run once per epoch for the hierarchy generation. The algorithm took, on average, **440 seconds** to run 30,000 epochs on a 180 MHz MIPS R10000 Processor with the European data.

4.1.3 Single-Pass TreeGCS

We obtained the following cluster configurations from the **three single-pass TreeGCS** (the numbers in the listings correlate to the numbers in the nodes in the figures):

1. **Alphabetical order of countries** (see Fig. 6).
   34 {Lux, Ukr}
   9 {Den, Fra, Ger, It, Spa, UK}
   80 {Alb, And, Aus, Bel, Bos, Bul, Cro, Cyp, Cze, Eir, Est, Far, Fin, Gib, Gre, Hun, Ice, Lat, Lie, Lit, Mac, Mal, Mon, NL, Nor, Pol, Rom, SM, Ser, Slk, Sln, Swi}
   3 {Swe}
2. **Middle to front order of countries** (see Fig. 6).
   11a {Den, Fra, Ger, It, Spa, UK}
   11b {Aus, Bel, NL, Swe, Swi, Ukr}
   12 {Cze, Fin, Gre, Nor, Rom}
   13 {Bul, Eir, Hun, Pol, Slk}
   14 {Lux, Ice}
   65 {Alb, And, Bos, Bul, Cro, Cyp, Est, Far, Gib, Lat, Lie, Lit, Mac, Mal, Mon, SM, Ser, Sln}
3. **Numerical order of first attributes** (see Fig. 6).
   29 {Aus, Bel, Den, Fra, Ger, It, NL, Spa, Swe, Swi, UK}
   69 {Alb, And, Bos, Bul, Cro, Cyp, Est, Far, Gib, Ice, Lat, Lie, Lit, Mac, Mal, Mon, Pol, Rom, SM, Slk, Sln}
   8 {Hun, Lux, Ser}
   20 {Cze, Eir, Fin, Gre, Nor, Ukr}

4.1.4 Cyclic TreeGCS

For the six permutations of the **cyclic** approach, the parameter settings for TreeGCS were the same as previously detailed. The following six hierarchies were obtained for the six possible permutations of the three data orders (in the following: D = alphabetical data order, M = middle to front, and S = sorted numerically by the first attribute). Again, the numbers in the listings correlate to the numbers in the nodes in the figures:

1. **DMS** (see Fig. 7).
   18 {Den, Fra, Ger, It, NL, Spa, UK}
   108 {Alb, And, Aus, Bel, Bos, Bul, Cro, Cyp, Cze, Eir, Est, Far, Fin, Gib, Gre, Hun, Ice, Lat, Lie, Lit, Lux, Mac, Mal, Mon, Nor, Pol, Rom, SM, Ser, Slk, Sln, Swe, Swi, Ukr}
2. **DSM** (see Fig. 7).
   30 {Bel, Den, Fra, Ger, It, NL, Spa, Swe, UK}
   8 {Aus, Lux, Ser, Swi, Ukr}
   88 {Alb, And, Bos, Bul, Cro, Cyp, Cze, Eir, Est, Far, Fin, Gib, Gre, Hun, Ice, Lat, Lie, Lit, Mac, Mal, Mon, Nor, Pol, Rom, SM, Slk, Sln}
3. **MSD** (see Fig. 7).
   10 {Den, Fra, Ger, It, Spa, UK}
   116 {Alb, And, Aus, Bel, Bos, Bul, Cro, Cyp, Cze, Eir, Est, Far, Fin, Gib, Gre, Hun, Ice, Lat, Lie, Lit, Lux, Mac, Mal, Mon, NL, Nor, Pol, Rom, SM, Ser, Slk, Sln, Swe, Swi, Ukr}
4. **MDS** (see Fig. 7).
   17 {Den, Fra, Ger, It, NL, Spa, UK}
   32 {Aus, Bel, Cze, Fin, Gre, Nor, Rom, Swe, Swi, Ukr}
   11 {Bul, Eir, Hun, Lux, Ser, Slk}
   66 {Alb, And, Bos, Cro, Cyp, Est, Far, Gib, Ice, Lat, Lie, Lit, Mac, Mal, Mon, Pol, SM, Sln}
5. **SDM** (see Fig. 7).
   18 {Den, Fra, Ger, It, NL, Spa, UK}
   5 {Cze, Gre, Lux, Ser}
   15 {Aus, Bel, Rom, Swe, Swi}
   12 {Eir, Fin, Hun, Nor, Ukr}
   76 {Alb, And, Bos, Bul, Cro, Cyp, Est, Far, Gib, Ice, Lat, Lie, Lit, Mac, Mal, Mon, Pol, SM, Slk, Sln}
6. **SMD** (see Fig. 7).
   23 {Bel, Den, Fra, Ger, It, NL, Spa, UK}
   90 {Alb, And, Bos, Bul, Cro, Cyp, Cze, Eir, Est, Far, Fin, Gib, Gre, Hun, Ice, Lat, Lie, Lit, Lux, Mac, Mal, Mon, Pol, SM, Ser, Slk, Sln}
   13 {Aus, Nor, Rom, Swe, Swi, Ukr}

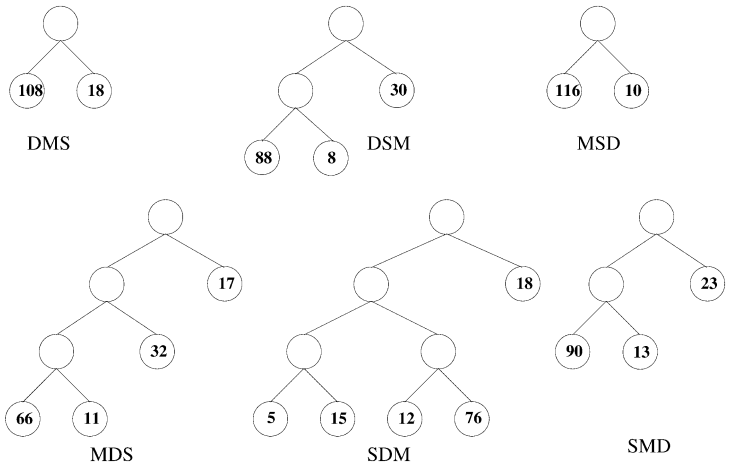

Fig. 7. Illustration of the cluster toplology for the six cyclic TreeGCS runs.

4.2 Parameter Settings

Next, we demonstrate the importance of the parameter settings for GCS that underlies TreeGCS. Köhle and Merkl [12] intimated at the problem for GCS. Parameter selection is a combinatorial problem and affects many statistical and neural network techniques [3], [15]. We make recommendations for effective parameter combinations for TreeGCS that are easily derived. We take the three parameters ϵ_{bmu}, ϵ_i, and β and fix two while altering the third for each parameter in turn (see Table 1) to measure the criticality of each parameter in turn. We then cycle through seven combinations of parameters where each parameter is reduced by a factor of 10 in the order ϵ_{bmu}, ϵ_i to β (see Table 2). For all evaluations, we list the final epoch for deletion from the TreeGCS hierarchy and the final epoch of insertion into the TreeGCS hierarchy for a run of 90,000 epochs. We assume the hierarchy is *static* when there are no more insertions or deletions of nodes. The data is comprised of 52 630-dimensional real-valued vectors. The hierarchy becomes static for the 47-dimensional European data with most parameter settings, so we use the higher dimensionality data for our evaluation and also demonstrate our system's ability to accommodate such high-dimensional data.

We can see differences regarding when the hierarchy becomes static for each parameter variation. From the last three lines of Table 1, varying ϵ_i between 0.002, 0.02, and 0.05 when the other two parameters are fixed creates the largest variation in final deletion/insertion from never static to static in approximately 4,000 epochs to never static. N.B., a static hierarchy, does not necessarily imply cluster quality. Some settings where the hierarchy becomes static early produce poor quality clusters as there are too few clusters and the clusters produced are too general.

Again looking at Table 2, we can see variations in when the hierarchy becomes static, but there is a general trend of "U"-shape with respect to the final deletion/insertion.

5 Analysis

We can see that the cyclic approach we have introduced is more stable than the original single-pass as the different data orders counteract each other and reduce the instability. The shapes of the hierarchies for the cyclic runs are more similar than those from the single-pass. For the cyclic, the DMS and MSD hierarchies have identical shapes as do SMD and DSM and also SDM and MDS. All cyclic hierarchies are very similar, differing only with subdivisions of larger clusters. Although the **single pass** finds a {*Den, Fra, Ger, It, Spa, UK*} cluster, it is not consistently in one-half of the hierarchy. There are two further notable inconsistencies for the single-pass: For the alphabetical order, {*Den, Fra, Ger, It, Spa, UK*} is a subcluster of the main branch with {*Lux, Ukr*} forming the separate half. For the numerical order, {*Aus, Bel, NL, Swe, Swi*} are included with the {*Den, Fra, Ger, It, Spa, UK*} cluster. The single-pass approach does not consistently group the same countries across the three hierarchies. For the **cyclic approach**, the different orderings seem to counteract each other and cancel out the hierarchical variations. Cyclic TreeGCS consistently finds the {*Den, Fra, Ger, It, NL, Spa, UK*} cluster and has the remaining countries as the other half of the cluster.

Comparing the cyclic hierarchies with the dendogram, the dendogram similarities are emulated except that cyclic TreeGCS includes *NL* and *Spa* with the {*Den, Fra, Ger, It, UK*} cluster, unlike the dendogram, omitting NL from the cluster in one instance. Looking at the dendogram, two of the most similar countries to {*Den, Fra, Ger, It, UK*} are *NL* and *Spa*, so this is neither unexpected nor undesirable. The two possible anomalies in cyclic TreeGCS are the inclusion of *Bel* with the {*Den, Fra, Ger, It, NL, Spa, UK*} cluster for the SMD cyclic and *Bel* and *Swe* for the DSM cyclic, but these are minor in comparison with the anomalies of the single-pass approach. We feel our cyclic approach improves cluster stability.

From Figs. 6 and 7, we can see that the number of cells per cluster is proportional to the number of countries in the cluster for the cyclic approach, but not for the single pass. In fact, in only seven instances do two countries have the same cell as their bmu in all six permutations of the cyclic approach: {*Aus and Swi*: DSM}, {*Nor and Ukr*: DMS}, {*Slk and Bul, Eir and Hun*: MDS}, {*Cze and Gre*: MSD}, {*Cze and Gre*: SDM} and {*Eir and Fin*: SMD}. In all cases, these countries are similar according to the dendogram. An unknown input

TABLE 1
Whether the Hierarchy Became Static Over 90,00 Epochs when the First and Second, Second and Third, and First and Third Parameters Are Fixed while the Other Is Varied

ϵ_{bmu}	ϵ_i	β	Epoch	Last Delete	Last Insert	Stability
0.09	0.002	0.0002	90000	89928	89943	F
0.09	0.002	0.002	90000	89928	89943	F
0.09	0.002	0.005	90000	38865	38973	T
0.09	0.05	0.0002	90000	18437	18988	T
0.09	0.05	0.002	90000	1361	1988	T
0.09	0.05	0.005	90000	-	1085	T
0.09	0.02	0.0002	90000	24681	22401	T
0.09	0.02	0.002	90000	2049	2325	T
0.09	0.02	0.005	90000	-	1085	T
0.02	0.002	0.0002	90000	89958	89836	F
0.2	0.002	0.0002	90000	11781	11873	T
0.4	0.002	0.0002	90000	89805	89851	F
0.02	0.04	0.005	90000	67251	67250	T
0.2	0.04	0.005	90000	49148	49378	T
0.4	0.04	0.005	90000	33219	33234	T
0.05	0.002	0.0005	90000	24681	25981	T
0.05	0.02	0.0005	90000	4253	2998	T
0.05	0.05	0.0005	90000	-	10419	T
0.3	0.002	0.003	90000	89943	89897	F
0.3	0.02	0.003	90000	4023	3977	T
0.3	0.05	0.003	90000	89025	88719	F

For the final column, a "T" indicates a static hierarchy and "F" indicates that the hierarchy never became static.

TABLE 2
The Effect of Progressively Reducing the Parameters by Scale Factor of 10

ϵ_{bmu}	ϵ_i	β	Epoch	Last Delete	Last Insert	Static
0.02	0.002	0.0002	90000	89958	89836	F
0.05	0.005	0.0005	90000	64680	64511	T
0.09	0.009	0.0009	90000	7282	5951	T
0.2	0.02	0.002	90000	31260	31444	T
0.3	0.03	0.003	90000	1529	2845	T
0.4	0.04	0.004	90000	36157	36172	T
0.5	0.05	0.005	90000	89852	89607	F

For the final column, a "T" indicates a static hierarchy and "F" indicates that the hierarchy never became static.

can be uniquely identified in all but these cases by finding the bmu for the unknown vector and identifying the appropriate country for which that cell is a bmu. Our cyclic approach has spread the probability of a cell being a bmu for any given input data vector more evenly than the probability spread for the single-pass approach.

For the parameter investigations, we can observe the importance of the three settings. As mentioned previously, there is a trade-off between a static hierarchy and cluster quality. Parameter settings, where the hierarchy is static after relatively few epochs, produce poor quality clusters that are too general. Also, we should note that the number

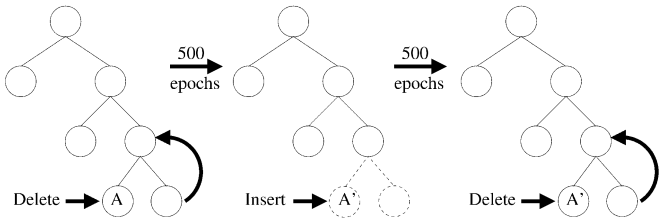

Fig. 8. Illustration of the recognition problem. Node A is deleted and the hierarchy tidied, then A' is inserted. If A' is then deleted, the cluster composition and entire node hierarchy will have changed from A, so indentifying A' as equivalent to A is difficult.

of deletions/insertions varies for parameter settings that become static at the same time, i.e., the epoch when the hierarchy becomes static is not proportional to the number of insertions/deletions. For the parameter settings, $\epsilon_{bmu} = 0.2$, $\epsilon_i = 0.02$, and $\beta = 0.002$, there are no deletions/insertions between epoch 2,493 and epoch 31,000, then two deletions and three insertions occur up to epoch 31,444 and then no more. However, for many parameter settings, the deletions/insertions are occurring constantly throughout the run.

In many cases, the movement in given layers of the hierarchy is due to the innate dynamism of the network below and is caused by repeated deletion and reinstatement of the same clusters. This is difficult to detect at run-time if the contemporaneity of the hierarchy is maintained, i.e., the splitting/deleting of the network clusters is mirrored exactly by the hierarchy (see Fig. 8). During the 1,000 epochs between the first cluster deletion and the second cluster deletion in Fig. 8, the composition of the clusters will have changed, with new cells being added and existing cells deleted, so matching the clusters through their cell contents is not possible. One solution would be to maintain a list of the hierarchy nodes removed with details of parents and siblings. If a deleted cluster was reinstated, the parent will still be the same, but this would slow the algorithm and introduce a storage overhead of a shadow hierarchy. Another solution would be a posteriori manual inspection of the run-time output. Again, the parentage of deleted/reinstated nodes could be compared and any oscillating detected.

If the users felt that the cluster quality was higher for an oscillating parameter combination, then we would recommend using those. However, we would generally recommend using a parameter combination where the hierarchy becomes static, but not too early, e.g., 24,000 to 40,000 epochs for the 630-dimensional data. As parameter setting is a combinatorial problem, we recommend using the scale factor 10 reduction technique shown in Table 2 and selecting the optimal combination from there if possible. This is how we selected the GCS parameters we used for our stability investigation runs in Section 4.1. This minimizes the search space for parameter settings. Only if a suitable cluster quality is not found do we recommend a further combinatorial investigation.

6 Conclusion and Future Development

We have introduced a hierarchical, dynamically formed clustering neural network extending and refining the GCS of Fritzke and partially overcoming an instability problem inherent in the GCS approach. The algorithm adaptively determines the depth of the cluster hierarchy; there is no requirement to prespecify network dimensions as with most SOM-based algorithms. Our superimposed tree will adapt to any variations in the cell structure below and there are no user-specified parameters for the hierarchy. The clustering produced by the cyclic variant is similar to the dendogram, while being able to handle identical similarities and maintaining the superiority of the SOM mapping approach over dendograms, as posited by Mangiameli et al. [11]. A further advantage of our approach over dendograms is that leaf nodes in our hierarchy represent groups of input vectors. In dendograms, each leaf node represents a single data point which precludes visualization of dendograms for large data sets as there are too many leaf nodes and branches to visualize. Due to our use of the GCS that maps high-dimensional input distributions to a two-dimensional network topology, we can visualize the superimposed hierarchical clustering as and when required. We feel TreeGCS is appropriate for any system that requires a hierarchical data representation where the structure needs to be flexible and cannot be prespecified, for example, the number of clusters or the hierarchical depth. We intend using TreeGCS to generate a hierarchical thesaurus for an Information Retrieval system.

With an unsupervised dynamic algorithm, there is no guarantee of the cluster configuration—the ideal algorithm should be totally consistent and stable to the order of data presentation. We have improved the stability of the GCS base by introducing a cardinality threshold for deletion and cycling through different data orders, so the topological variances may counter each other. One possible extension that may improve the stability further would be to present the various data orders and combine the resulting cluster configurations to produce a single average cluster configuration. There are three possible methods of combination: *Bagging*, *Boosting*, or a simple ensemble [14]. All three methods generate an ensemble of classifiers and combine them to give a single classifier. For Bagging and Boosting, the input data set is varied and, for the simple ensemble, the standard training set is used for each classifier, but the parameter settings of the classifier are varied. Bagging uses a random redistribution of the input data set to train each classifier, while the input data space for each classifier in Boosting is selected based on the performance of the earlier classifiers. However, Boosting "*can create ensembles that are less accurate than a single classifier—especially with neural networks.*" Bagging, however, could be used to combine the cluster hierarchies and produce a single combined cluster topology, but the training sets would need to be altered and it may be more difficult to provide cyclic data sets. We feel the most appropriate for TreeGCS would be a simple ensemble. Opitz and Maclin [14] noted that a simple ensemble of neural networks where each network differs only in the parameter settings is "*surprisingly*

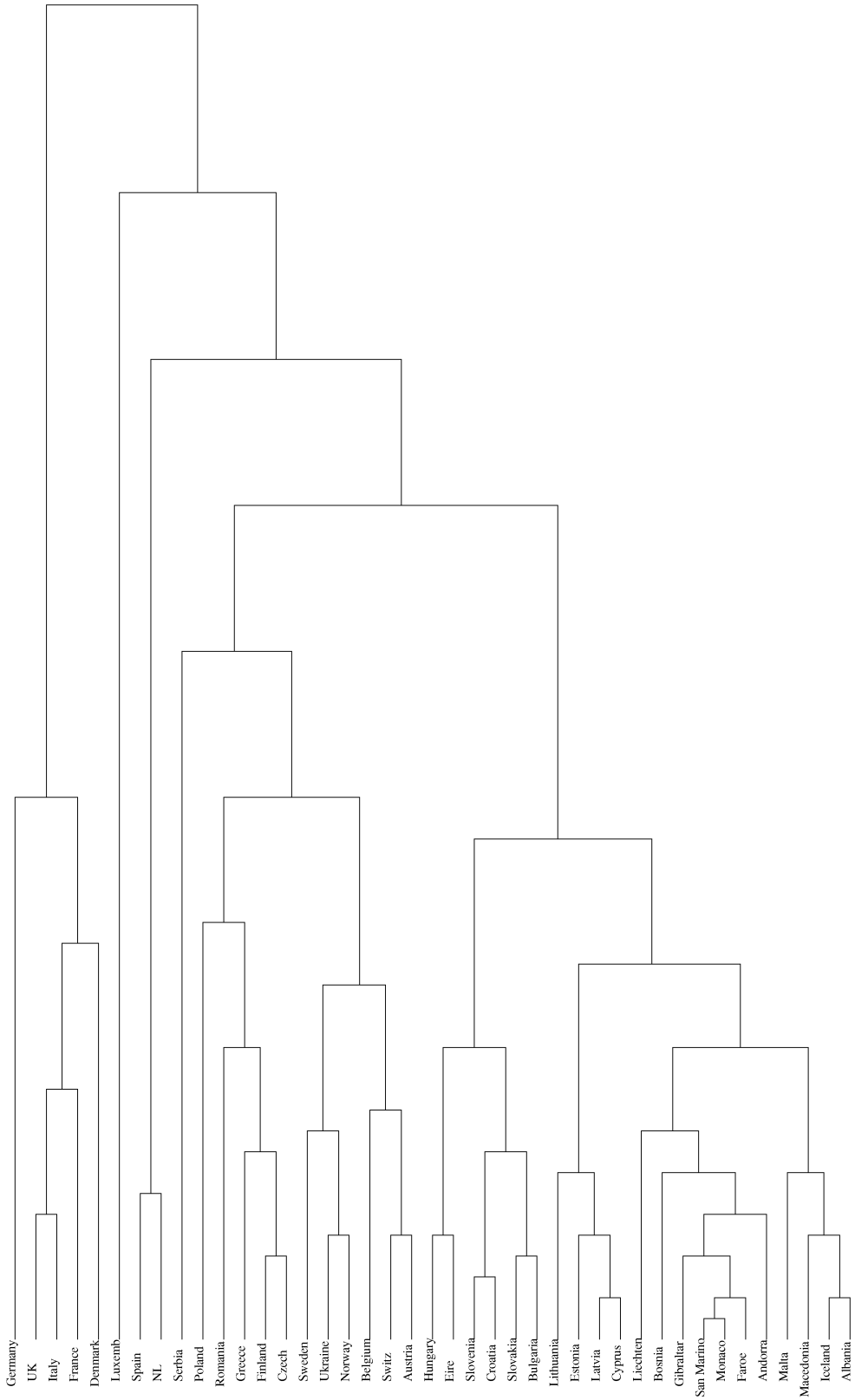

Fig. 9. Illustration of the ascendant hierarchical clustering for the alphabetical data order. The links are logarithmic and rounded.

effective." The hierarchies produced by the single pass TreeGCS approach were very dissimilar and there is little commonality. Opitz and Maclin state that the classifiers in the ensemble need to be accurate. Our improved cyclic approach produces more consistent clusters that could be run with different parameter settings and readily combined to identify an average combined cluster. If the user preferred any individual cluster produced by the cyclic approach, then they could use that cluster. However, if the user wanted a consensus, then they could produce a

number of cyclic cluster hierarchies using different parameters, knowing that our stability improvement has minimized cluster instability and determine the average combined cluster topology.

We have ensured that unknown inputs can be identified as each country maps to a different cell in nearly all instances (discussed in Section 5) using only three times as many cells as countries. We can input an unknown vector, determine the bmu, and identify the country for which that cell is a bmu. We have highlighted the criticality of the parameter setting and recommended selecting parameters that stabilize between 24,000 to 40,000 epochs to ensure an equitable trade-off between quality and stability.

In future work, we will demonstrate TreeGCS against SOMs with higher dimensionality data; SOMs are criticized for skewing the representation for high dimensionalities when mapping on to the lower-dimensional network topology [4]. Also, we will demonstrate TreeGCS with a larger data set (higher number of input vectors) to evaluate the scalability. We will demonstrate the use of the hierarchy and its subsequent traversal, allowing the system to handle the specificity and generality of concepts. We can zoom in to the hierarchy at the required level of specificity and traverse the hierarchy from there. We will demonstrate the utility of the hierarchy for a hierarchical thesaurus and how it may be linked to documents for document similarity retrieval and estimation.

ACKNOWLEDGMENTS

This research was supported by an EPSRC studentship.

REFERENCES

[1] J. Bruske and G. Sommer, "Dynamic Cell Structure Learns Perfectly Topology Preserving Map," *Neural Computation,* vol. 7, no. 4, 1995.
[2] V. Burzevski and C.K. Mohan, "Hierarchical Growing Cell Structures," technical report, Syracuse Univ., Syracuse, N.Y., 1996, an abbreviated version appears in *ICNN '96: Proc. Int'l Conf. Neural Networks,* June 1996.
[3] B.S. Everitt., *Cluster Analysis.* Edward Arnold, 1993.
[4] B. Fritzke, "Kohonen Feature Maps and Growing Cell Structures—A Performance Comparison," *Advances in Neural Information Processing Systems–5–(NIPS '92),* C.L. Giles, S.J. Hanson, and J.D. Cowan, eds., 1993.
[5] B. Fritzke, "Growing Cell Structures—A Self-Organizing Network for Unsupervised and Supervised Learning," Technical Report, TR-93-026, Int'l Computer Science Inst., Berkeley, Calif., 1993.
[6] B. Fritzke, "A Growing Neural Gas Network Learns Topologies," *Advances in Neural Information Processing Systems—7,* G. Tesauro, D.S. Touretzky, and T.K. Leen, eds., 1995.
[7] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data.* Englewood Cliffs, N.J.: Prentice Hall, 1988.
[8] K. Kaufman and R. Michalski, "A Method for Reasoning with Structured and Continuous Attributes in the INLEN-2 Knowledge Discovery System," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD-96),* Aug. 1996.
[9] T. Kohonen, *Self-Organizing Maps,* vol. 2. 1997.
[10] M. Kubat, I. Bratko, and R. Michalski, "A Review of Machine Learning Methods," *Machine Learning and Data Mining: Methods and Applications,* R.S. Michalski, I. Bratko, and M. Kubat, eds., pp. 3–69, 1998.
[11] P. Mangiameli, S.K. Chen, and D. West, "A Comparison of SOM Neural Network and Hierarchical Clustering Methods," *European J. Operational Research,* vol. 93, pp. 402–417, 1996.
[12] M. Köhle and D. Merkl, "Visualizing Similarities in High Dimensional Input Spaces with a Growing and Splitting Neural Network," *Proc. Sixth Int'l Conf. Artificial Neural Networks (ICANN '96),* 1996.
[13] R. Miikkulainen, "Script-Based Inference and Memory Retrieval in Subsymbolic Story Processing," *Applied Intelligence,* vol. 5, pp. 137–163, 1995.
[14] D. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *J. Artificial Intelligence Research,* vol. 11, pp. 169–198, 1999.
[15] H.-H. Song and S.-W. Lee, "A Self-Organizing Neural Tree for Large Set Pattern Classification," *IEEE Trans. Neural Networks,* vol. 9, no. 3, May 1998.
[16] R. Weiss, B. Vélez, M.A. Sheldon, C. Namprempre, P. Szilagyi, A. Duda, and D.A. Gifford, "HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering," *Proc. Seventh ACM Conf. Hypertext,* Mar. 1996.
[17] World FactBook, 1997, http://www.odci.gov/cia/publications/factbook/country-frame.html.

Victoria J. Hodge is a postgraduate research student in the Department of Computer Science, University of York. She is a member of the Advanced Computer Architecture Group investigating the integration of neural networks and information retrieval.

Jim Austin has the chair of neural computation in the Department of Computer Science, University of York, where he is the leader of the Advanced Computer Architecture Group. He has extensive expertise in neural networks as well as computer architecture and vision. Dr. Austin has published extensively in this field, including a book on RAM-based neural networks.