Exploiting Homogeneity of Density in Incremental Hierarchical Clustering

Dwi H. Widyantoro

School of Electrical Engineering and Informatics, Institute of Technology Bandung
Bandung, 40132 INDONESIA
dwi@if.itb.ac.id

Abstract. Hierarchical clustering is an important tool in many applications. As it involves a large data set that proliferates over time, reclustering the data set periodically is not an efficient process. Therefore, the ability to incorporate a new data set incrementally into an existing hierarchy becomes increasingly demanding. This article describes HOMOGEN, a system that employs a new algorithm for generating a hierarchy of concepts and clusters incrementally from a stream of observations. The system aims to construct a hierarchy that satisfies the homogeneity and the monotonicity properties. Working in a bottom-up fashion, a new observation is placed in the hierarchy and a sequence of hierarchy restructuring processes is performed only in regions that have been affected by the presence of the new observation. Additionally, it combines multiple restructuring techniques that address different restructuring objectives to get a synergistic effect. The system has been tested on a variety of domains including structured and unstructured data sets. The experimental results reveal that the system is able to construct a concept hierarchy that is consistent regardless of the input data order and whose quality is comparable to the quality of those produced by non incremental clustering algorithms.

Keywords: *Clustering; Conceptual Clustering; Incremental Hierarchical Clustering.*

1 Introduction

Generating a hierarchy of clusters incrementally in a dynamic environment is a crucial process especially when (1) a complete set of data may not be available on the onset, (2) the data set grows over time and (3) the need for incorporating the new arrived data may be critical. A system working under these conditions has to be able to put a new observation properly into the existing hierarchy, update the concept descriptions and then restructure the hierarchy.

Due to the "information overload" phenomenon in which data proliferation is inevitable, periodically reclustering the whole data set in order to incorporate the new incoming data is fundamentally not an efficient process. The ability to perform incremental clustering becomes increasingly appealing because it offers a viable option to the problem faced by a batch process. An incremental

Received July 28th, 2006, Revised September 11th, 2006, Accepted for publication September 11th, 2006.

algorithm for hierarchical clustering should be capable of capturing intrinsic cluster structures. More importantly, its hierarchy quality should be comparable to the quality of those generated by non incremental methods. While no consensus yet exists on what constitutes intrinsic structures, it is likely that such structures cannot be assumed to have certain shapes or distributions.

The sensitivity to input orderings is a long-standing problem in incremental conceptual clustering [1]. Two major issues that can affect the sensitivity problem are (1) nodes misplacement and (2) early commitment on cluster membership. The former is mainly due to the changes of hierarchy structures while processing new observations so that nodes that are previously well placed become misplaced. The latter refers to the use of a fixed threshold value for deciding an observation's cluster membership, for example, those applied in INC [2] and UNIMEM [3], which despite its practicality has its limitation in that it cannot adapt a cluster membership test to local properties of the cluster. Hence, early commitment on a cluster membership decision could prevent capturing an intrinsic hierarchical structure in the data set.

This paper presents a new incremental conceptual clustering algorithm, HOMOGEN, that addresses the quality issue. The conceptual clustering approach works on a metric space model that views an object (e.g., observation, cluster or node) as a point in a high-dimensional space. The density of points is used to define the characteristic of a good cluster and as guidance to hierarchically organize a set of clusters. Informally, the density describes the spatial distribution of points, measured in terms of the average distance from a point to its nearest neighbor (this will be formally defined in Section 3.1). A hierarchy is represented as a tree structure in which a node in the tree denotes a cluster in the hierarchy. HOMOGEN's approach to concept formation aims to construct a tree structure with two properties:

Property 1 (Homogeneity) A tree structure satisfies a homogeneity property if every node in the tree consists of child nodes with similar density locally, w.r.t. the distances to nearest sibling among the child nodes.

Property 2 (Monotonicity) A tree structure satisfies a monotonicity property if the density of a node is always at least as high as the density of its parent. That is, the density of nodes monotonically increases along any path in the tree structure from the root to a leaf node.

These two properties serve as guiding principles for minimizing the occurrence of misplaced nodes during the hierarchy construction. The homogeneity requirement is needed in order to form clusters with local density properties, that is, the densities of objects vary in intrinsic cluster structures. This property

also does not bias toward the shape and the class distribution of clusters that makes it suitable for tracking evolving clusters in an online situation. In fact, the homogeneity property also relaxes the commitment in the cluster membership function by flexibly defining it based on the cluster density. Accordingly, a new object can be a member of a cluster if the inclusion of the new object in the cluster will not violate the homogeneity property of the cluster. Additionally, the monotonicity property requirement is based on the observation that higher-level hierarchies in most hierarchical systems are generally used to represent entities with broader contexts. This characteristic can be captured with the notion of monotonicity, also in terms of cluster density. Thus, the monotonicity property helps properly organize the hierarchical structures of clusters. The structure needs to be changed whenever the property is violated, and construction of the new structure aims to satisfy this property. Taken together, both properties are expected to construct a natural hierarchical structure such that nearby (resp. distant) clusters share a lower (resp. an upper)-level ancestor.

2 Related Work

Previous work has mitigated the effect of input ordering by applying restructuring operators such as cluster merging, splitting, and promotion [4]. The strategies for applying these operators can be broadly divided into local and global approaches with their advantages and shortcomings. The local approaches apply restructuring operators on the neighborhood of a hosting node (i.e., a node that serves as the parent of a new observation) [2]-[4]. Although relatively efficient to recover nodes misplaced at neighboring nodes, the local approaches in general suffer from their inability to deal with major structural changes. The global approaches address the sensitivity issue by iteratively reinserting nodes into the entire hierarchy [5], which is clearly expensive.

The restructuring strategy in HOMOGEN represents a tradeoff between the local and the global approaches. The system pinpoints nodes whose structures are potentially affected by the presence of new observations and then applies restructuring operators only to nodes that actually experience structural change. The structural change problems are detected through checking the nodes' conformity with the homogeneity and monotonicity properties. Intuitively, this strategy improves the ability of HOMOGEN to recover from major structural changes while preserving the incremental nature of the algorithm.

HOMOGEN's approach that uses a set of conceptual constraints (e.g., the homogeneity and monotonicity properties) as the guiding principles during the hierarchy restructuring can be related to the ARACHNE [6] and the HIERARCH [7] systems. Unlike these systems that rely exclusively on their constraints as

the only guiding principles, HOMOGEN also explicitly detects and rectifies structural problems that cannot be recovered by satisfying the imposed constraints. The premise is that no single approach covers all cases, and a complementary approach that addresses a different restructuring objective can be implanted to handle the uncovered cases. Although differing greatly in detail, this idea is similar in spirit to COP-COBWEB [8] and COP-KMEANS [9], a version of COBWEB (KMEANS) that enforces instance-level hard constraints irrespective to the clustering decision of the main approaches. The instance-level constraints in these systems are heavily dependent on the input domains so that a different set of hard constraints needs to be defined on a different data set. In contrast, HOMOGEN's approach is more general because it deals only with a structural property, allowing it to work across data sets without additional efforts.

The clustering process of HOMOGEN can be viewed as the incremental version of hierarchical agglomerative clustering (HAC) methods [10]-[12] with two respects. First, it works in a bottom-up fashion, which is the same as to the manner HAC algorithms form cluster hierarchies in batch modes. The second similarity is that HAC also produces cluster hierarchies that tend to be monotonic. Unlike HAC that biases toward generating tree structures with the fewest branching factors, HOMOGEN relaxes this restriction that allows it to construct a more comprehensible hierarchical structure.

3 Cluster Hierarchy Construction

3.1 Formal Foundations

A hierarchy $H = \{N_1, N_2, \ldots, N_n\}$ is a tree consisting of n nodes. Each node in the tree maintains two types of information: *concept* and *density*. The *concept* summarizes the descriptions of all observations covered by a node. The *density* describes the spatial distribution of the child nodes. An *internal node* has at least two child nodes. A node in the tree represents a cluster whose members are the set of child nodes. A leaf node is a singleton cluster covering a single observation whose concept description is the description of the observation itself.

Concept Representation. Let an observation $o_i = \{o_{i1}, o_{i2}, \ldots, o_{id}\}$ be a d-dimensional point where o_{ij}, represents the value of the j^{th} dimension of the i^{th} observation. A concept $C = \{c_1, c_2, \ldots, c_d\}$ also has the same dimension as that of the observation. Let $\mathcal{E}(N)$, the extension of N, denote the set of observations (leaf nodes) that are descendants of N.

Definition 1 (*Concept Description***)**. The concept description C of a node N is

the center of m observations (leaf nodes) that are descendants of N, that is, $C=\{c_1, c_2, \ldots, c_d\}$ where $c_j = \frac{1}{m}\sum_{i=1}^{m} o_{ij}$ and $o_{ij} \in \mathcal{E}(N)$.

Density Representation. The density of a node is defined as the average distance to the closest neighbor among the child nodes. A natural way of obtaining the distances to the nearest neighbors is from the path given by the minimum spanning tree (MST) of the child nodes. The density representation of a node N is a triple $D=\langle NDP,\mu,\sigma\rangle$ where $NDP=\{d_i | d_i \in \Re\}$ is a population of nearest distance d_i, μ and σ are the average and the standard deviation of NDP. Each d_i in NDP is the length of an edge, measured by the distance from a child node to its nearest sibling, in the MST structure connecting the child nodes of N. Thus, the μ and σ values are locally defined over the distances among the child nodes. The distance between two nodes, with respect to the concept descriptions of the two nodes, in general can be measured by using L_n distance functions as defined by

$$L_n(N_i, N_j) = \left(\sum_{k=1}^{d} \left(c_{ik} - c_{jk}\right)^n \right)^{\frac{1}{n}} \quad (1)$$

where C_i and C_j are the concept descriptions (i.e., clusters centers) of nodes N_i and N_j, respectively. The average value of NDP, μ, characterizes the density of a node (cluster) in which the density is higher with lower μ value. The average distance of a leaf node is defined to be zero (i.e., the distance between the leaf node and itself). Hence, a leaf node represents a cluster with infinitely large density.

Definition 2 (Monotonic Node). Let μ_N and μ_P be the average nearest distances with respect to the density representations of nodes N and its parent P respectively. N is a monotonic node if only if $\mu_N \leq \mu_P$, that is, the density of N is higher than or equal to the density of its parent.

Definition 3 (Homogeneous Node). Let $D_N = \langle NDP,\mu,\sigma\rangle$ be a density representation of a node N. Given a lower limit $L_L = \mu - k\sigma$ and an upper limit $U_L = \mu + k\sigma$ where k is a positive constant, the node N is homogeneous, with respect to k, if and only if $L_L \leq d_i \leq U_L$ for $\forall d_i \in NDP$. The functions L_L and U_L define the lower and upper bounds based on the mean and the variance of the population.

Thus, a node is homogeneous if its distribution of the distances to the closest

neighbors among the child nodes is within a bounded range around the mean. The variance factor k in L_L and U_L functions controls the tightness of the bounds. Definition 4 interprets the effects of observing a new point that is not within the bounds of a node.

Definition 4 (Low and High Density Regions Formation). Let N be a homogenous node with L_L and U_L as the node's lower and upper limits, respectively. Given a new point A, let B be an N's child node that is the nearest neighbor to A. Let d be the distance from A to B. If $d < L_L$, the region covering A and B is said to ***form a high-density region*** on N. If $d > U_L$, then A (and B) is said to ***form a low-density region*** on N.

3.2 A Preliminary Analysis of Problem Complexity

HAC algorithms produce *binary tree* structures that, *w.r.t.* Definition 3, always meet the homogeneity property due to the fact that a node with two child nodes is homogeneous. Many variants of HAC algorithm, except the *Centroid*-based HAC, always satisfy the monotonicity property [11] because a new higher-level cluster is formed in the order of increasing distance between two clusters. The time complexity of these algorithms is at least $O(N^2)$ [11]. In strictly on-line setting, these two properties can be preserved by rebuilding the tree each time encountering a new observation and its time complexity is therefore at least $O(N^3)$, which is clearly not interesting. However, it is also not obvious whether there exists an algorithm with time complexity of less than $O(N^2)$ that can incrementally incorporate a new point into an existing tree while still preserving the tree properties. Rather than pursuing both properties, the incremental algorithm of HOMOGEN takes a strategy that guarantees producing only a tree satisfying the homogeneity property. The algorithm relies only on heuristic rules for building a tree that tends to be monotonic.

3.3 The Algorithm Development

The approaches for generating a concept hierarchy incrementally can be divided into two stages. During the first stage, the algorithm locates a node in the hierarchy that can accept a new observation in a bottom up fashion, and then inserts the new observation into the hosting node. The second stage performs hierarchy restructuring. This two-stage algorithm is applied on observing the third and subsequent data points. The initial hierarchy is created by merging the first two points (the merging process will be described later).

3.3.1 First Stage: Locating the Initial Placement in Concept Hierarchy

Locating the initial placement of a new observation is performed in the following sequence:

1. Find the best match concept over leaf nodes based on the closest distance to the new observation. To avoid exhaustive search by scanning the entire leaf nodes, the system performs a beam search, which maintains k best search paths, through the hierarchy in order to approximate the best match leaf node.
2. Starting from the parent of the closest leaf node, perform upward search to locate a cluster (or create a new cluster hierarchy) that can host the new observation. Heuristic rules are employed during this search in order to minimize disturbance of the hierarchy monotonicity.

Let's first define two operators needed to place a new observation in the hierarchy: node insertion operator (Figure 1a) and hierarchy insertion operator (Figure 1b). For both operators, let N_j be the new observation.

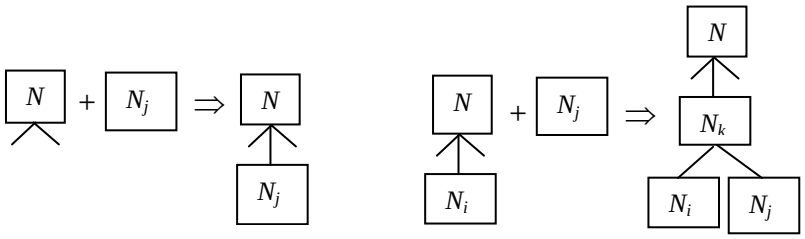
Definition 5 (Node Insertion Operator) The node insertion operator, denoted by *INSERT_NODE*(N, N_j), inserts N_j as a new child of a node N.

Definition 6 (Hierarchy Insertion Operator) Let N_i be one of N's child nodes. The hierarchy insertion operator, denoted by *INSERT_HIERARCHY*(N_i,N_j), inserts a new node N_k in the hierarchy so that N_k becomes a parent of N_i and N_j, and is a child node of N.

The *upward search* employs two heuristic rules to determine which insertion operator to apply. By utilizing the monotonicity property, the general idea of upward search is similar to the strategy of inserting a new element into a sorted list of bins.

Heuristic 1 (Node Insertion). Perform *INSERT_NODE*(N, N_j) if $L_L \leq d \leq U_L$ where d is the distance from a new observation N_j to the nearest child node of N, and L_L & U_L be the lower and upper bounds of N, respectively, as in Definition 3. For N with two child nodes, these bounds are defined to be $L_L = k_L.d_N$ and $U_L=k_U.d_N$ where $0 < k_L < 1$ is a lower limit constant, $k_U > 1$ is an upper limit constant, and d_N is the distance between the two N's child nodes.

In a node with two child nodes, the zero variance in the node's density representation would hardly allow the heuristic to insert a third child node. The heuristic addresses this problem by providing bounds derived only from the mean value. These special case bounds also play the role of determining the



(a) *INSERT_NODE*(N, N_j) (b)

Figure 1 Node and hierarchy insertion operators.

Hierarchy Restructuring Algorithm

1. Let *crntNode* be the hosting node.
2. **While** (*crntNode* ≠ *null*)
3. Let *parentNode* ← Parent(*crntNode*).
4. Detect and recover the siblings of *crntNode* that are misplaced.
5. Perform homogeneity maintenance process on *crntNode*.
6. Let crntNode ← parentNode.
7. **End of While**

Figure 2 Hierarchy restructuring algorithm.

allowable variation in the distances to nearest neighbors. The bound constants are $k_L = 2/3$ and $k_U = 3/2$ (see Section 4.2 for the details).

Heuristic 2 (Hierarchy Insertion). Let N_i be the child node of N closest to a new observation N_j. Perform *INSERT_HIERARCHY*(N_i, N_j), if and only if N_j forms *a high-density region* on N, and if and only if N_j *forms a low-density region* on at least one of N's child nodes.

The applicability conditions of Heuristic Rule 2 are an indication that no cluster in the hierarchy can host the new observation without causing a significant density disturbance. Therefore, a new cluster hierarchy needs to be inserted in order to accommodate the new observation while minimizing the perturbation of the hierarchy monotonicity.

On each level in the hierarchy, the algorithm during the *upward search* examines the applicability conditions of each heuristic rule, applies the corresponding insertion operator whenever the conditions are satisfied and then stops. If none of the rules can be applied, the search proceeds to the next higher-level cluster (i.e., the parent of current cluster). If the search process reaches the

top-level cluster, a new cluster hierarchy will be inserted at the top level using the hierarchy insertion operator, replacing the root node with the new cluster.

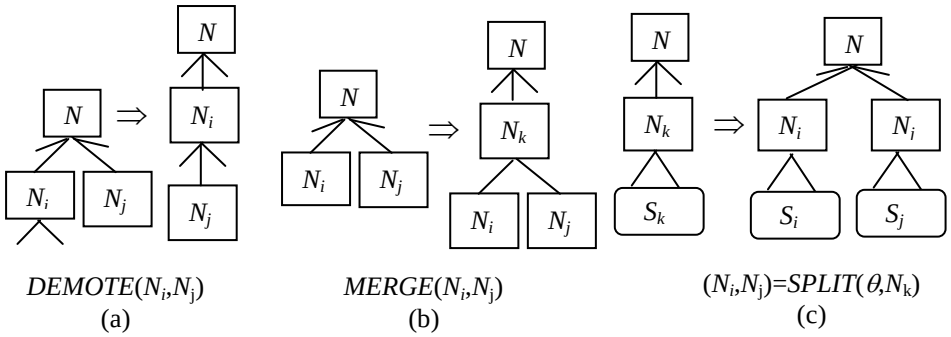


Figure 3 Demotion, merging and splitting restructuring operators.

3.3.2 Second Stage: Hierarchy Restructuring

Changes in the hierarchy structures always occur after incorporating new observations during their initial placement. The restructuring process is performed to adapt the hierarchy to new structures by (1) recovering any misplaced nodes and (2) repairing the homogeneity property that has been violated. To do this effectively, the algorithm pinpoints nodes in the tree affected by the change of a node's structure once a new observation is incorporated in the hierarchy. Then, local operators are applied systematically on these affected nodes.

A node is affected if its concept description changes. The notion of concept descriptions in Definition 1 implies that the affected nodes are the hosting node and its ancestors. Figure 2 summarizes the hierarchy-restructuring algorithm that performs the restructuring process on these affected nodes. The following two sections discuss steps 4 and 5 described in the figure.

Detection and Recovery of Misplaced Nodes

There is a case during the first stage of algorithm in which the hierarchy insertion operator causes a node misplaced eventhough the homogeneity and the monotonicity properties are still maintened. Definition 7 formally defines this misplaced problem and a demotion operator is suggested to eliminate it.

Definition 7 (Misplaced Sibling) Let N_i and N_j be siblings to one another. N_j is said to be misplaced as the sibling of N_i, denoted by **Misplaced_Sibling**(N_i,N_j), if N_j does not form a *low-density region* on N_i.

Detection and Recovery of Misplaced Nodes (N_i) Algorithm
1. Let the input N_i be the recipient of demoted nodes.
2. Let *Siblings* ← the set of N_i's siblings.
3. Let *No_Misplaced_Sibling* ← **false**.
4. **Repeat**
5. Let $N_j \in$ *Siblings* be the closest node to a child node of N_i.
6. **If Misplaced_Sibling** (N_i, N_j) (i.e., see Definition 7)
7. **Then** *DEMOTE*(N_i, N_j),
8. Remove N_j from *Siblings*.
9. **Else** Let *No_Misplaced_Sibling* ← **true**.
10. **Until** (*No_Misplaced_Sibling* = **true**) or (*Siblings* = null).

Figure 4 Misplaced node detection and recovery algorithm.

A demotion operator, denoted by *DEMOTE*(N_i, N_j), is a process of retracting N_j from its parent and inserting it as a child node of N_i (see Figure 3a).

Since applying a single demotion operator could also lead to further problems to the N_i's remaining siblings, the algorithm checks the rest of them and reapplies the demotion operator, repeatedly, until no misplaced sibling is found.

Figure 4 describes the detail process. The restriction on the next sibling chosen in Line 5 guarantees that once the selected node is found to be not a misplaced sibling, then neither do the remaining siblings. If the algorithm terminates by the second condition (i.e., *Siblings* = *null*), additional minor restructuring is performed (not shown in the algorithm) in order to satisfy the requirement that an internal node must have at least two child nodes.

Homogeneity Property Maintenance

This section describes the process of repairing a cluster whose homogeneity property is violated. In such a case, some areas in the cluster *form high and/or low-density regions*. A high-density region is eliminated by merging two nearest nodes using a *merging operator*.

Definition 9 (Merging Operator) *MERGE*(N_i,N_j) is *INSERT_HIERARCHY*(N_i,N_j), where N_j is a sibling of N_i (see Figure 3b).

The merging operator replaces two nodes in a cluster with a single node that is

the center of the two nodes. If the merged nodes are restricted to those with the smallest nearest distance and the distance is below the cluster's lower limit, the merging operator will remove a high-density region from the cluster. Repeating the merging process on these nodes will eventually eliminate all high-density regions.

Homogeneity Maintenance (N_k) Algorithm

1. Let an input N_k be the node that is being examined.
2. **Repeat**
3. Let N_i and N_j be the pair of neighbors among N_k's child nodes with the closest distance.
4. **If** N_i and N_j *form a high-density region* with respect to N_k,
5. **Then** *MERGE*(N_i,N_j),
6. **Until** there is no high-density region found in N_k during the last iteration.
7. Let M_i be the child of N_k with the largest d_i and M_j be M_i's nearest neighbor where d_i is the distance from node *i* to its nearest neighbor.
8. **If** M_i and M_j *form a low-density region* in N_k,
9. **Then** Let $S_k \leftarrow$ the set of N_k's child nodes.
10. Let (N_i, N_j) = *SPLIT*(θ, N_k)
11. **If** $N_i \notin S_k$ **Then** *Call* **Homogeneity Maintenance** (N_i).
12. **If** $N_j \notin S_k$ **Then** *Call* **Homogeneity Maintenance** (N_j).

Figure 5 Homogeneity maintenance algorithm.

A low-density region can be removed by splitting the cluster into two or more smaller ones using sparser regions as the cutting points.

Definition 10 (Splitting Operator) Let N_k be a child node of N, and S_k be a set of child nodes of N_k (see Figure 3c). Let θ be a splitting function that divides S_k into two disjoint subsets S_i and S_j, that is, (S_i, S_j) = $\theta(S_k)$ satisfying S_i and S_j, S_k=$S_j \cup S_j$, and $S_j \cap S_j = \varnothing$. Let ($N_i$,$N_j$)=*SPLIT*($\theta$,$N_k$) where *SPLIT* is a splitting operator. The *SPLIT* operator retracts N_k from N and makes N_i and N_j, as N's child nodes where S_i and S_j are the sets of child nodes of N_i and N_j, respectively. If S_i or S_j contains a single child node, then that node becomes N_i or N_j, that is, effectively promoting the child node one level higher in the tree.

Using the MST graph of the cluster being split, the algorithm employs a splitting function θ that cuts a path connecting an object with the farthest distance to its nearest neighbor. If the splitting operation is performed only when the farthest distance to the nearest neighbor exceeds the cluster's upper bound, recursively applying this operator on each new split will eventually

obtain a cluster that is free from low-density regions, in which case the splitting process stops. Figure 5 provides the detail of the homogeneity maintenance process. Working in a *divide and conquer* fashion, the algorithm receives an input cluster N_k and replaces N_k by a non empty set of homogeneous nodes S_v.

4 Evaluation

4.1 Quantifying the Hierarchy Quality

4.1.1 Measuring the Quality of Hierarchy Structures

Given a hierarchy produced by a system H_L, the quality of H_L is quantified by measuring the degree of its match with a known target hierarchy H_T. Generally speaking, the degree of match between H_T and H_L is calculated by the number of nodes in H_T, except the root node, that match with their corresponding nodes in H_L. Furthermore, a node in H_L is said to be the corresponding node in H_T if both nodes match *conceptually* and *structurally*.

Let $N_T \in H_T$ and $N_L \in H_L$ be nodes in the target hierarchy H_T and in the hierarchy produced by a system H_L, respectively, where both hierarchies are derived from the same set of observations. Let $\mathcal{E}(N)$ denote the set of observations (singleton nodes) that are descendants of node N. The degree of conceptual match between N_T and N_L, denoted by $CMatch(N_T,N_L)$ is as follows.

$$CMatch(N_T, N_L) = \frac{|\mathcal{E}(N_T) \cap \mathcal{E}(N_L)|}{|\mathcal{E}(N_T) \cup \mathcal{E}(N_L)|} \tag{2}$$

For each node N_T in H_T, let N^*_L be the corresponding node in H_L such that:

$$N_L^* = \underset{(N_L \in H_L) \wedge (N_L \neq Root)}{\arg\max} \{CMatch(N_T, N_L)\} \tag{3}$$

Then, the degree of structural match between N_T and N_L, denoted by $SMatch(N_T,N^*_L)$, is defined by

$$SMatch(N_T, N_L^*) = CMatch(Parent(N_T), Parent(N_L^*)) \tag{4}$$

Finally, the degree of match between H_T and H_L, denoted by $HMatch(H_T,H_L)$, is computed as follows:

$$HMatch(H_T, H_L) = \sum_{(N_T \in H_T) \wedge (N_T \neq Root)} CMatch(N_T, N_L^*) \cdot SMatch(N_T, N_L^*) \tag{5}$$

The maximum score is determined by the number of target nodes in the target hierarchy.

4.1.2 Measuring the Quality of Distinct Clusters

In this measure, the hierarchy generated by a clustering algorithm is examined whether a distinct target cluster can be rediscovered. Let *DATA* be the set of all observations, and $TC_i \in TC$ be the *i*th target cluster in a set of target clusters *TC*. Let $\mathcal{E}(TC_i)$ denote the set of observations belonging to the target cluster TC_i such that $DATA = \bigcup_i \mathcal{E}(TC_i)$ for $\forall TC_i \in TC$ and $\mathcal{E}(TC_i) \cap \mathcal{E}(TC_j) = \phi$. Moreover, let H_L be a hierarchy produced by a system using all observations in *DATA*. For each $TC_i \in TC$, let N^*_L be the corresponding node in H_L and be determined similarly as in Equation 2. The quality of H_L is then calculated as an *accuracy* measure denoting the percentage of match between the target clusters and their corresponding clusters in H_L, as defined by Equation 6 below.

$$Accuracy(T_C, H_L) = \frac{\sum_{TC_i \in TC} |\mathcal{E}(TC_i)| \times CMatch(TC_i, N_L^*)}{|DATA|} \times 100\% \qquad (6)$$

4.2 Parameter Determination of HOMOGEN

To determine the appropriate tightness of the bound functions, during preliminary experiments HOMOGEN was run using a synthetic data set and the variance factor *k* (see Definition 3) was varied from 0.3 to 2 in all nodes with three or more child nodes. The lower bound constant k_L was also varied from 0.1 to 0.9 and from 1.1 to 2 for the upper bound constant k_U particularly for nodes with two child nodes in the Heuristic Rule 1.

From these experiments, k=1, k_L=2/3, and k_U=3/2 were found to be among those that gave good measures of hierarchy quality. These settings were then fixed for other data sets in the rest of experiments.

4.3 Performance Comparison with Other Incremental Systems

In this section the performances of HOMOGEN are compared with those of COBWEB [4] and two versions of ARACHNE systems. The first version of ARACHNE, denoted by ARACHNE-L(ocal), implements the original ARACHNE's control strategy as described by McKusick and Langley [6]. This version applies restructuring operators on neighboring nodes that violate the nodes' constraints. The second version, ARACHNE-G(lobal), extends ARACHNE-L by pushing the power of tree constraints employed by the system further into its

limit.

Table 1 Summary of data sets.

	Synthetic Data Sets			Natural Data Sets [a]		
	Gird	Triangle	Symbol	Soybean Small	Soybean Large	Voting
#Observations	288	108	27	47	307	435
#Target Clusters [b]	38	12	12	5	19	2
#Distinct Clusters	24	9	9	4	19	2
#Target Hierarchy Levels	4	2	2	2	–	–
#Attributes [c]	2	2	3	35	35	15
Dimension Size	2	2	39	76	132	48
Attribute Value Types	Cont	Cont.	Nom.	Nom.	Nom.	Nom.
Distance Functions	L_2	L_2	L_1	L_1	L_1	L_1

[a]From UCI repository of machine learning database [13]
[b]#TargetClusters = #Distinct Clusters + #Internal Nodes, except the root node, that groups the distinct concepts and their larger groups.
[c]The number of attributes does not include the target (class) attribute.

The experiment uses six data sets as summarized in Table 1. The data sets Grid, Triangle, Symbol, and Soybean Small have known, clear target hierarchy structures as shown in Figure 6 while the hierarchy structure of the Soybean Large is unknown. Since the Voting data set contains only two target classes, it has the simplest hierarchy structure. The first four data sets are used to evaluate the performance of HOMOGEN in discovering both the distinct clusters and their organizational structures inherent in the data sets. The experiments were performed in two ordering scenarios: *random* and *bad* orderings. The observation in random ordering was selected randomly from one of the unseen observations regardless of the observations' classes. In *bad ordering*, the stream was ordered by observations' classes [1] where observation of a different class will not be given until all observations of the same class had been processed. In each case the experiment results were averaged over 25 trials.

Table 2 provides the performance comparison of HOMOGEN with other incremental systems with respect to the systems' abilities to rediscover distinct clusters inherent in the data and to properly organize the discovered clusters into higher-level clusters. Table 3 summarizes the systems' performances on rediscovering distinct clusters. The table shows that HOMOGEN performs comparably well to or better than the other systems. The clustering of HOMOGEN is relatively not affected by the cluster shapes. For example, the cluster boundaries on *Voting* and *Soybean Large* are not clear-cut, indicating the irregularity of cluster shapes and/or the overlap between clusters. Yet HOMOGEN performs better on these data sets. To some extent, this confirms the expectation that the homogeneity property can guide the incremental process of HOMOGEN

to reconstruct clusters of fairly arbitrary shapes.

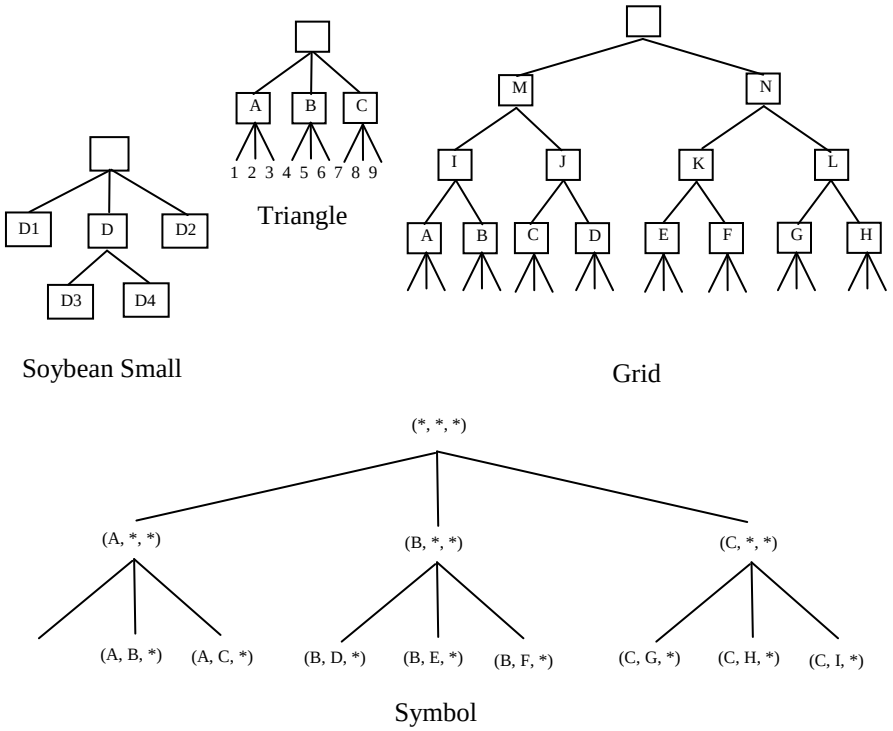


Figure 6 Target hierarchy structures of Triangle, Soybean Small, Grid and Symbol data sets.

Table 2 The quality of hierarchy structures. The quality is measured according to Equation 5 averaged over 25 trials.

	HOMOGEN	COBWEB	ARACHNE-L	ARACHNE-G	Max Scores
	Random Ordering				
Grid	38.00	–	24.20	24.25	38
Triangle	12.00	–	11.99	12.00	12
Symbol	12.00*	9.62	9.75	11.31	12
Soybean Small	4.67*	4.23	3.81	4.29	5
	Bad Ordering				
Grid	37.97	–	26.65	26.65	38
Triangle	12.00*	–	11.81	11.85	12
Symbol	12.00*	7.12	10.18	11.19	12
Soybean Small	4.61	2.91	2.78	3.30	5

* The differences are statistically significant at most at 0.026 levels.

4.4 Performance Comparison with HAC Algorithms

A subset of the Reuters-21578 1.0 test collection [13] was used for this evaluation. Only six topics were used from the training set part of the *ModApte* split [14] with moderate topic sizes. The number of selected documents was 951 consisting of target topics *Coffee* (90), Crude (253), *Gold* (70), *Interest* (190), *Sugar* (97) and *Trade* (251).

Table 3 The quality of distinct clusters. (measured using Equation 6) The differences of bold numbers are statistically significant from non-bold numbers on the same row at 0.001 levels.

	HOMOGEN	COBWEB	ARACHNE-L	ARACHNE-G
	Accuracy (%) on Random Ordering			
Soybean Small	**96.00**	**94.03**	83.38	**96.83**
Soybean Large	**59.18**	55.91	47.61	53.66
Voting	**79.07**	75.22	74.10	76.42
	Accuracy (%) on Bad Ordering			
Soybean Small	**97.28**	72.32	67.96	85.92
Soybean Large	**61.61**	50.31	49.74	53.26
Voting	**79.60**	68.40	63.79	75.22

Each document was represented by a feature vector containing a set of unique terms and their term frequencies. All stop words such as “a”, “the”, "although”, etc were removed. The feature selection process was applied to remove irrelevant terms using two alternatives of *heuristics*:

MDF-FS: minimum document frequency-based feature selection that selects terms occurring in at least *n* documents.

MTF-FS: minimum term frequency-based feature selection that selects a term *t* if there exists at least one document in which *t* occurs at least *m* times. It assumes that term frequency is an indicator for topical words.

HOMOGEN performs feature selection and weighting on the fly as it receives a new document to learn. Therefore, MDF-FS feature selection and TF-IDF weighting method are not applicable for this system. In batch systems, the feature selection and weighting processes are performed over all documents well before the clustering process begins.

Seven variants of HAC (i.e., non-incremental algorithms for hierarchical clustering) were considered for performance comparison. These variants were *Single-link, Complete-link, Group-average, Weighted-average, Centroid,*

Median-method and *Ward*'s method [10], [11]. They differed from each other in their methods in calculating the distances of a cluster to a non-singleton cluster. The distance between two documents or clusters in HOMOGEN was measured by the Euclidean distance function.

Table 4 Peak accuracies achieved by HOMOGEN and HAC methods. The accuracy of HOMOGEN is averaged over 25 runs. The italic numbers following the accuracies are the parameter values of their respective feature selection methods (i.e., *mtf* or *mdf* values) that produce the results.

	Accuracy (%) (*parameter value*)			
Feature Selection	MTF-FS		MDF-FS	
Term Weighting	TF	TF-IDF	TF	TF-IDF
HOMOGEN	89.32 (*5*)	–	–	–
Single-link	70.20 (*9*)	61.71 (*12*)	59.80 (*48*)	62.45 (*64*)
Complete-link	72.81 (*2*)	68.04 (*1*)	72.81 (*4*)	68.01 (*1*)
Group-average	89.16 (*4*)	86.86 (*4*)	81.12 (*2*)	80.24 (*8*)
Weighted-average	88.62 (*5*)	83.94 (*4*)	76.05 (*8*)	80.42 (*32*)
Centroid	83.43 (*14*)	79.05 (*14*)	58.31 (*64*)	50.18 (*80*)
Median Method	74.37 (*8*)	67.80 (*12*)	62.09 (*80*)	61.34 (*64*)
Ward's Method	84.36 (*5*)	83.37 (*6*)	77.19 (*8*)	80.11 (*32*)

Two feature weighting methods are considered: *term frequency* (TF) and *term frequency inverse document frequency* (TF-IDF) [15]. All features are then normalized using Eucledian Normalization.

The experiments exploited the peak accuracies that could be achieved by HOMOGEN and the seven HAC variants on the full data set (951 documents consisting of six topics). More specifically, the best result was taken by varying the minimum term frequency values from 1 to 15 for the MTF-FS feature selection, or by varying the minimum document frequency (*mdf*) values to 2, 4, 8, 12, 16, 32, 48, 64, 80 and 96 for the MDF-FS feature selection.

Table 4 presents the best results for each variation of feature selection and weighting methods as well as their corresponding parameter settings. The best accuracy from HAC algorithms is achieved by the *Group-average* method (89.16%) and the peak performance attained by HOMOGEN is slightly higher (89.32%). A higher parameter value, shown next to the accuracy in the table, is an indication that the corresponding clustering algorithm is more sensitive to noise since it needs to be more aggressive in removing irrelevant features in order to maximize its performance.

Table 5 The confusion matrices of clusters generated by HOMOGEN and *Group-average* HAC methods.

HOMOGEN

	Document Topics						Total
	Coffee	*Crude*	*Gold*	*Interest*	*Sugar*	*Trade*	Docs.
Cluster-1	81.28	0.20	–	0.16	–	0.72	82.36
Cluster-2	0.08	226.72	1.60	1.24	0.16	3.44	233.34
Cluster-3	–	0.08	64.88	–	–	0.08	65.04
Cluster-4	1.04	2.96	0.24	175.32	0.88	6.64	187.08
Cluster-5	0.60	0.32	0.68	–	90.28	0.12	92.00
Cluster-6	0.24	0.96	0.04	5.24	0.04	227.92	234.44
#Excluded docs.	6.76	21.76	2.56	8.04	5.64	12.08	56.84

Group-average Hierarchical Agglomerative Clustering

	Document Topics						Total
	Coffee	*Crude*	*Gold*	*Interest*	*Sugar*	*Trade*	Docs.
Cluster-1	83	–	–	–	–	1	84
Cluster-2	1	226	–	–	1	1	229
Cluster-3	–	1	66	–	–	–	67
Cluster-4	3	12	–	186	–	11	212
Cluster-5	–	–	–	–	94	–	94
Cluster-6	–	10	3	2	–	233	248
#Excluded docs.	3	4	1	2	2	5	17

The detail results of HOMOGEN and *Group-average* algorithms are provided by Table 5. The fractional numbers in HOMOGEN are due to the averaging of the experiment results over 25 trials. Let *precision* be the percentage of correct assignment of documents in all found clusters and *recall* be the percentage of correct assignment over all 951 documents. The precision and recall of HOMOGEN are 96.9% and 91.1%, respectively. The group-average algorithm, on the other hand, produces clusters with slightly lower precision (95.1%) but higher recall (93.4%).

5 Concluding Remarks

This paper highlights the problem inherently faced by batch hierarchical clustering methods in an online situation. It has also described a new concept formation system called HOMOGEN that addresses the problem by incrementally creating a concept hierarchy from a sequence of instances. Experiments conducted on a variety of domains involving structured and unstructured data sets indicate the effectiveness of the system. It is relatively insensitive to input

ordering and can produce a quality structure inherent within the input data. Its performance in the given unstructured data set is also comparable to the best performance achieved by HAC methods.

The main contribution of this paper is the exploitation of homogeneity property coupled with the monotonicity property for incremental hierachical clustering. Both properties are essential for discovering intrinsic hierarchical structures in which one cannot assume about the shape and the class distribution of clusters. Although systems such as DBSCAN [16], CURE [17] and CHAMELEON [18] can handle clusters with complex shapes and/or different sizes, these systems employ non incremental methods. In incremental systems, COBWEB and its family [4], [5], [8] prefer clusters with similar sizes. ARACHNE tends to build compact clusters. Similar cluster shapes are also formed by the INC system [2]. HIERARCH's constraints exhibit bias toward certain cluster shapes [7].

References

[1] Fisher, D. H., Xu, L., & Zard, N., *Ordering effects in clustering*, In Proceedings of the 9th International Conference on Machine Learning, pp. 163–168 (1992).

[2] Hadzikadic, M., & Yun, D., *Concept formation by incremental conceptual clustering*, In Proceedings of the 11th International Joint Conference on Artificial Intelligence (1989), pp. 831–836.

[3] Lebowitz, M., *Experiments with incremental concept formation: unimem*, Machine Learning, **1**, 103–138 (1987).

[4] Fisher, D. H., *Knowledge acquisition via incremental conceptual clustering*, Machine Learning, **2**, 139–172 (1987).

[5] Biswas, G., Weinberg, J., & Fisher, D., *ITERATE: A conceptual clustering algorithm for data mining*, IEEE Transactions on Systems, Man, and Cybernetics, **28** (2), 100–111 (1998).

[6] McKusick, K. B., & Langley, P., *Constraints on tree structure in concept formation*, In Proceedings of the 12th International Conference on Artificial Intelligence (1991), pp. 810– 816.

[7] Nevins, J., *A Branch and Bound Incremental Conceptual Clusterer*, Machine Learning, **18**, 3–22 (1995).

[8] Wagstaff, K., & Cardie, C., *Clustering with Instance-Level Constraints*, In Proceedings of the 17th International Conference on Machine Learning (2000), pp. 1103–1110.

[9] Wagstaff, K., Cardie, C., Rogers, S., & Schroedl, S., *Constrained K-Means Clustering with Background Knowledge*, In Proceedings of the 18th International Conference on Machine Learning (2001), pp. 577–584.

[10] Everitt, B. S., Landau, S., & Leese, M., *Cluster Analysis*, New York, NY: Oxford University Press Inc (2001).

[11] Jain, A., & Dubes, R. C., *Algorithms for Clustering Data*, Englewood Cliffs, NJ: Prentice Hall (1988).
[12] Miyamoto, S., *Fuzzy Sets in Information Retrieval and Cluster Analysis*, Boston: Kluwer Academic Publishers (1990).
[13] Blake, C., & Merz, C. (1998), UCI Repository of Machine Learning Databases,http://www.ics.uci.edu/mlearn/MLRepository.html, University of California, Irvine, Department of Information and Computer Sciences.
[14] Apt´e, C., Damerau, F., & Weiss, S. M., Automatic Learning of Decision Rules For Text Categorization, ACM Transactions on Information Systems, **12** (3), 233–251 (1994).
[15] Salton, G., & McGill, M. J., *Introduction to Modern Information Retrieval*, New York, NY: McGraw-Hill (1983).
[16] Ester, M., Kriegel, H.-P., Sander, J., & Xu, X., *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (1996), pp. 226–231.
[17] Guha, S., Rastogi, R., & Shim, K., *CURE: An Efficient Clustering Algorithm For Large Databases*, In Proceedings of the 1998 ACM SIGMOD international conference on Management of data (1998), pp. 73–84.
[18] Karypis, G., Han, E. H., & Kumar, V., *Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling*. COMPUTER, **32**, 68–75 (1999).