An Incremental Approach to Building a Cluster Hierarchy

Dwi H. Widyantoro & Thomas R. Ioerger
Texas A&M University
Department of Computer Sciences
College Station, TX 77843-3112 USA
dhw7942,ioerger@cs.tamu.edu

John Yen
The Pennsylvania State University
School of Information Sciences and Technology
University Park, PA 16802-2117 USA
jyen@ist.psu.edu

Abstract

In this paper we present a novel Incremental Hierarchical Clustering (IHC) algorithm. Our approach aims to construct a hierarchy that satisfies the homogeneity and the monotonicity properties. Working in a bottom-up fashion, a new instance is placed in the hierarchy and a sequence of hierarchy restructuring process is performed only in regions that have been affected by the presence of the new instance. The experimental results on a variety of domains demonstrate that our algorithm is not sensitive to input ordering, can produce a quality cluster hierarchy, and is efficient in terms of its computational time.

1. Introduction

Hierarchical clustering is an important tool in data *warehouse* analysis [8], ontology construction of a dynamic text collection (e.g., similar to the one in Yahoo!) and in interactive information retrieval [9]. As it involves a large data set that grows rapidly over time, re-clustering the data set periodically is not an efficient process. Due to the "information overload" phenomenon in recent years, the ability to perform a clustering process incrementally is increasingly appealing because it offers a viable option to a problem faced by a non incremental clustering process.

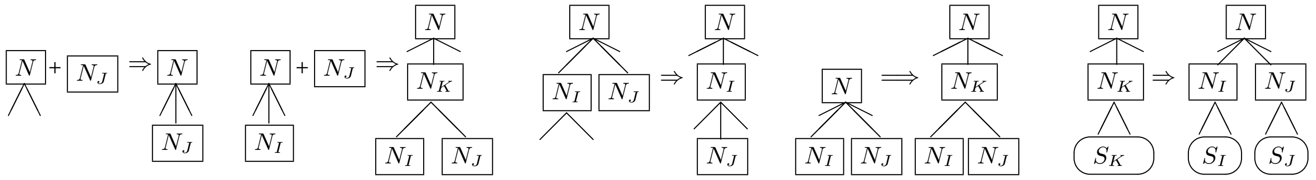
The sensitivity to input ordering is one of the major issues in incremental hierarchical clustering [6]. Previous works mitigate the effect of input ordering by applying restructuring operators, which can be broadly categorized into *local* and *global* approaches. Although relatively efficient to recover nodes misplaced at neighboring nodes, the local approaches [5, 11] in general suffer from their inability to deal with major structural changes. The global approaches [1, 4] alleviate these problems but these approaches are very expensive and make the algorithm non-incremental. By contrast, our method described in this paper represents a trade-off between the local and the global approaches while preserving the incremental nature of the algorithm.

Incremental clustering algorithm has also been developed by Ester et al. from *Data Mining* perspective [2]. Specifically, they develop an incremental version of DBSCAN, a density-based clustering algorithm. However, DBSCAN and its incremental version are *partitional* clustering algorithms. Our approach is more related to the agglomerative hierarchical clustering techniques [3, 10], and therefore, can be viewed as the incremental version of the more traditional bottom-up hierarchical clustering methods.

2. Our Incremental Algorithm

Our approach aims to construct a concept hierarchy with two properties: *homogeneity* and *monotonicity*. Informally, a homogeneous cluster is a set of objects with similar density. A hierarchy of clusters satisfies the monotonicity property if the density of a cluster is always higher than the density of its parent. That is, the density of clusters monotonically increases along any path in the concept hierarchy from the root to a leaf node.

A cluster hierarchy is basically a tree structure with leaf nodes represent singleton clusters covering single data points. Each node in the tree maintains two types of information: *cluster center* and *cluster density*. The cluster density describes the spatial distribution of child nodes of a node. We define a cluster's density as the average distance to the closest neighbor among the cluster's members. A natural way of obtaining the distances to the nearest neighbors is by creating the minimum spanning tree (MST) of objects in the cluster. Specifically, the density representation of a node is a triple $D = \langle NDP, \mu, \sigma \rangle$ where $NDP = \{d_i \mid d_i \epsilon \Re\}$ is a population of nearest distance d_is, 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 between two nodes, of the MST structure connecting a node's child nodes. In general, the distance between two nodes, *w.r.t.* the nodes' cluster center, can be measured by using L^n distance function family.



INS_NODE(N, N_J) (a) INS_HIERARCHY(N_I, N_J) (b) DEMOTE(N_I, N_J) (c) MERGE(N_I, N_J) (d) (N_I, N_J) = SPLIT(Θ, N_K) (e)

Figure 1. Restructuring operators: (a) node insertion operator, (b) hierarchy insertion operator, (c) demotion operator, (d) merging operator, and (e) splitting operator. S_K, S_I & S_J are the sets of child nodes of N_K, N_I & N_J, respectively; $S_K = \{S_I \cup S_J\}$ and $(S_I, S_J) = \Theta(S_K)$. Θ is a splitting function that separates S_K by disconnecting an edge in the cluster's MST structure into two disjoint sets S_I and S_J. (N_I, N_J) = SPLIT(Θ, N_K) splits N_K into N_I and N_J *w.r.t.* the splitting function Θ.

Definition 1 *Let $D_C = \langle NDP, \mu, \sigma \rangle$ be a density representation of a cluster C. Given a lower limit $L_L = f(\mu, \sigma) \leq \mu$ and an upper limit $U_L = g(\mu, \sigma) \geq \mu$, the cluster C is homogeneous with respect to f and g if and only if $L_L \leq d_i \leq U_L$ for $\forall d_i \epsilon NDP$.*

Definition 2 *Let C be a homogenous cluster. Given a new point A, let B be a C's cluster member that is the nearest neighbor to A. Let d be the distance from A to B. A (and B) is said to **form a higher (lower) dense region** in C if $d < L_L$ ($d > U_L$, respectively).*

Our approaches to incorporating a new data point into a cluster hierarchy incrementally can be divided into two stages. During the first stage, the algorithm locates a node in the hierarchy that can host the new data point. 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.

Locating the initial placement of a new data point during the first stage is carried out in a bottom-up fashion:

1. Find the closest point over leaf nodes.
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 point with minimal density changes and minimal disruption of the hierarchy monotonicity. Let N be the node being examined at current level. The placement of a new point N_J in the hierarchy is performed according to the following rules:
    - if $L_L \leq d \leq U_L$ then perform INS_NODE (N, N_J) (see Figure 1a) where d is the distance from the new point N_J to the nearest N's child node.
    - if N_J *forms a higher dense region* on N, and N_J *forms a lower dense region* on at least one of N's child nodes then perform INS_HIERARCHY (N_I, N_J) (see Figure 1b) where N_I is the child node of N closest to the new point N_J.

    If none of the rules applies, the search proceeds to the next higher-level cluster. If the search process reaches the top-level cluster, a new cluster will be inserted at the top level using the hierarchy insertion operator.

The second stage aims to recover any structural changes that occur after incorporating a new data point. The following algorithm describes the hierarchy restructuring process.

Algorithm **Hierarchy Restructuring**

```
1. Let crntNode be the node that accepts the new point.
2. While ($crntNode \neq \emptyset$)
3.     Let parentNode ← Parent(crntNode)
4.     Recover the siblings of $crntNode$ that are misplaced.
5.     Maintain the homogeneity of crntNode.
6.     Let crntNode ← parentNode
```

One of the most common problems is that a node is stranded at an upper level cluster. In such a case, a node N_J, which is supposed to be a child node of N_I, is misplaced as N_I's sibling. Line 4 addresses this issue by utilizing Definition 2 to detect the problem. Specifically, a node N_J, which is the sibling of N_I, is said to be misplaced as N_I's sibling if and only if N_J *does not form a lower dense region* in N_I. If such a problem is detected, we iteratively apply DEMOTE(N_I, N_J) (see Figure 1c).

Line 5 in the Hierarchy Restructuring algorithm repairs a cluster whose homogeneity property has been violated. Intuitively, the recovery process involves the elimination of both the lower and the higher dense regions, repeatedly, until all nearest distances in the cluster are within the cluster's bounds. The following algorithm describes the homogeneity maintenance process of a cluster. Working in a *divide and conquer* fashion, it receives a cluster N and replaces N by one or more homogeneous clusters.

```
Algorithm Homogeneity Maintenance($N$)
Let an input $N$ be the node that is being examined.
Repeat
    Let $N_I$ and $N_J$ be the pair of neighbors among $N$'s child nodes with the smallest nearest distance.
    If $N_I$ and $N_J$ form a higher dense region,
    Then MERGE $(N_I, N_J)$ (see Figure 1d)
Until there is no higher dense region found in $N$ during the last iteration.
Let $M_I$ and $M_J$ be the pair of neighbors among $N$'s child nodes with the largest nearest distance.
If $M_I$ and $M_J$ form a lower dense region in $N$,
Then Let $(N_I, N_J)$ = SPLIT $(\Theta, N)$. (see Figure 1e)
    Call Homogeneity Maintenance($N_I$).
    Call Homogeneity Maintenance($N_J$).
```

3. Evaluation

Let $\mathcal{DATA}$ be the set of all data points, and $TC_i \,\epsilon\, \mathcal{TC}$ be the i^{th} target cluster in a set of target clusters $\mathcal{TC}$. Let $\mathcal{E}(TC_i)$ denote the set of data points belong to a target cluster TC_i such that $\mathcal{DATA} = \bigcup_i \mathcal{E}(TC_i)$ for $\forall\ TC_i \,\epsilon\, \mathcal{TC}$, and $\mathcal{E}(TC_i) \cap \mathcal{E}(TC_j) = \emptyset$ for $i \neq j$. Moreover, let $N \epsilon \mathcal{H}$ be a node in a cluster hierarchy $\mathcal{H}$ that is produced by a clustering algorithm using all data points in $\mathcal{DATA}$. Let $\mathcal{E}(N)$ denote the set of data points (i.e. leaf nodes) that are descendants of a node N. For each $TC_i \,\epsilon\, \mathcal{TC}$, let N_i^* be the corresponding node in $\mathcal{H}$ such that:

$$N_i^* = \arg\max_{N \epsilon \mathcal{H}} \left\{ \frac{\|\mathcal{E}(TC_i) \cap \mathcal{E}(N)\|}{\|\mathcal{E}(TC_i) \cup \mathcal{E}(N)\|} \right\}$$

Thus, N_i^* is a node in $\mathcal{H}$ that represents a target cluster TC_i. The quality of cluster hierarchy $\mathcal{H}$ is then calculated as an accuracy measure, which is defined as follows.

$$Acc_{\mathcal{H}} = \frac{\sum_{TC_i \epsilon \mathcal{TC}} \|\mathcal{E}(TC_i) \cap \mathcal{E}(N_i^*)\|}{\|\mathcal{DATA}\|} \times 100\% \qquad (1)$$

In all experiments, we use $U_L = \mu + \sigma$ as the upper bounds and $L_L = \mu - \sigma$ as the lower bounds of clusters with three or more cluster members. We set the upper bounds $1.5d$ and the lower bounds (2/3)d for two-member clusters where d is the distance between the two clusters' members.

To test the sensitivity of our algorithm to input ordering, we use three natural data sets taken from the UCI repository: *Soybean Small*, *Soybean Large*, and *Voting*. The distance between two instances or clusters on these domains is calculated using the L^1 distance function. The experiments are performed in two settings: *random* and *bad* orderings [6]. We also run COBWEB [5] and ARACHNE [12] for performance comparison with other incremental systems.

Table 1. The performance of various incremental hierarchical clustering algorithms.

	Our IHC	COBWEB	ARACHNE
	Accuracy (%) on Random Ordering		
Soybean Small	96.00	97.27	85.10
Soybean Large	69.49	64.27	60.80
Voting	85.55	84.37	79.40
	Accuracy (%) on Bad Ordering		
Soybean Small	97.36	81.02	78.38
Soybean Large	71.97	60.13	64.26
Voting	85.58	77.81	66.75

Table 1 summarizes the experiment results, averaged over 25 trials. As indicated in the table, the performances of our algorithm are better than the other incremental systems in most cases. Our algorithm is also relatively not sensitive to the presentation of input ordering.

The next experiment is to test whether the performance of our IHC algorithm is still competitive with those of non incremental algorithms particularly the agglomerative hierarchical clustering methods [3, 10]. In this experiment we use a subset of the Reuters-21578 1.0 test collection obtained from the UCI KDD archive. We select only a subset of training stories that are assigned a single topic category. Each document is preprocessed (i.e. by removing stop words, performing features selection and weighting) and is represented by a feature vector. We measure the distance between two documents or clusters using the L^2 distance function. Because a document topic should be independent of the length of document, the feature vectors and cluster centers are normalized by L^2 normalization.

Table 2 depicts the best hierarchy quality produced by each clustering algorithm and the time needed to perform the clustering process. Because our IHC algorithm could produce different result on different input ordering, we average the result over 25 trials. As shown in table, the best result from the agglomerative methods is achieved by the *group-average* with the accuracy of 93.38%. This is only about 2.1% higher than the accuracy achieved by our IHC algorithm, which is 91.28%.

The last column of Table 2 reveals that the best result of our IHC algorithm is obtained by taking a much shorter time than those of the agglomerative approaches. While the best result of our algorithm requires only 56.1 seconds to perform the clustering process, the agglomerative methods need at least 5600 seconds, about 100 times longer than ours. Not only does our IHC algorithm offer an incremental process that can avoid data re-clustering, it also efficiently performs the clustering process.

Table 2. The performance on Reuter data set.

Clust. Method	Accuracy(%)	Execution Time (seconds)
IHC Algorithm	**91.28**	**56.1**
Single Linkage	81.60	5749.2
Complete Linkage	81.18	5692.3
Group-Average	**93.38**	5749.3
Centroid	80.55	5897.7
Ward	**91.27**	5820.7

4. Discussion

The worst case analysis of our IHC algorithm reveals that it needs $B^3 D \log_B n$ time to incorporate a single object, where B is the average branching factor of the tree, D is the dimension (e.g., #features) of objects, and n is the number of objects that have been previously incorporated in the tree. Tests on a variety of domains indicate that the tree branching factor is relatively small, ranging from 2 to 6. D can be associated with the cost for calculating the distance between two objects. The most expensive operation is recalculating the distances among pairs of child nodes during the MST update[1], which is $B^2 D$ time. Given the incremental update time as above, the time complexity of our IHC algorithm is $\mathcal{O}(N \log N)$. This is better than the time complexity of an agglomerative method, which is $\mathcal{O}(N^2)$.

Our notion of density, which provides a basis for defining the homogeneity property, is based on a graph theoretic approach [10]. Despite the advantage of recognizing clusters of any shapes, using MST or a similar structure in a clustering process also has a drawback in that it easily chains several clusters together particularly in batch clustering such as the *single linkage* agglomerative clustering [3]. If there exists a formation of data points that could link clusters, the *single linkage* method tends to find those links because the underlying MST-like structure is formed over all data points. However, this problem is not necessarily the case in our algorithm. Because the calculation of distance between two nodes in our algorithm is based on the nodes' cluster center and the MST structure of a node is built over only the child nodes, the underlying MST structures are local to the nodes' levels. As a result, the MST structures in our IHC algorithm are scattered into several hierarchy levels and are dependent on the input ordering. The chance is therefore very small for encountering an ordering of several objects that can chain two clusters at the same hierarchy levels and clusters‘ parents. Hence, the clusters chaining problem that is deterministic in batch clustering is a random variable in incremental setting with very low probability of occurrence.

5. Conclusion

This paper highlights the inefficiency problem faced by non incremental hierarchical clustering methods in a dynamic environment. In response to this problem, we present a new incremental hierarchical clustering algorithm. Experiments conducted on a variety of domains indicate the effectiveness of our algorithm that illuminates its potential as a valuable tool for Data Mining task.

Acknowledgments

This research is partially supported by Contract DAAD17-00-P-0649 from the U.S. Army Research Laboratory (ARL) and by a grant from Dell Foundation.

References

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

[2] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th VLDB Conference*, 1998.

[3] B. S. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. New York, NY: Oxford University Press Inc, 2001.

[4] D. Fisher. Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research*, 4:147–180, 1996.

[5] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Journal of Machine Learning*, 2:139–172, 1987.

[6] D. H. Fisher, L. Xu, and N. Zard. Ordering effects in clustering. In *Proceedings. of the 9th International Conference on Machine Learning*, pages 163–168, 1992.

[7] G. Frederickson. Data structures for on-line updating of mst, with applications. *Siam J. on Comput.,* 14(4):781–798, 1985.

[8] J. Han and M. Kamber. *Data mining : concepts and techniques*. San Francisco : Morgan Kaufmann Publishers, 2001.

[9] M. Hearst and J. Pederson. Reexamining the cluster hipothesis: Scatter/gather on retrieval results. In *Proceedings of ACM SIGIR*, pages 76–84, 1996.

[10] A. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[11] M. Lebowitz. Experiments with incremental concept formation: Unimem. *Journal of Machine Learning*, 1:103–138, 1987.

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

---

[1] Currently we use *Prim*‘s algorithm to rebuild the MST structure, which has an every-case time complexity of $\Theta(B^2)$. Fortunately, there exists an incremental MST algorithm [7] with $\Theta(\sqrt{(B-1)})$ update time that could be used to improve the efficiency of the MST update.