Instance Driven Hierarchical Clustering of Document Collections

Hassan H. Malik, and John R. Kender

Department of Computer Science, Columbia University,
New York, NY 10027, USA
{hhm2104, jrk}@cs.columbia.edu

Abstract. The global pattern mining step in existing pattern-based hierarchical clustering algorithms may result in an unpredictable number of patterns. In this paper, we propose IDHC, a pattern-based hierarchical clustering algorithm that builds a cluster hierarchy without mining for globally significant patterns. IDHC allows each instance to "vote" for its representative size-2 patterns in a way that ensures an effective balance between local and global pattern significance. The number of patterns selected for each instance is dynamically determined using a local standard deviation based scheme, and the rest of the cluster hierarchy is obtained by following a unique iterative cluster refinement process. By effectively utilizing instance-to-cluster relationships, this process directly identifies clusters for each level in the hierarchy, and efficiently prunes duplicate clusters. Furthermore, IDHC produces cluster labels that are more descriptive (patterns are not artificially restricted), and adapts a soft clustering scheme that allows instances to exist in suitable nodes at various levels in the cluster hierarchy. We present results of experiments performed on 16 standard text datasets, and show that IDHC almost always outperforms state-of-the-art hierarchical clustering algorithms in terms of entropy, and achieves better FScores in most cases, without requiring tuning of parameter values.

Keywords: Pattern based hierarchical clustering, interestingness measures.

1 Introduction and Motivation

Clustering is the partitioning of a dataset into subsets (clusters), so that the data in each subset (ideally) share some common trait. The quality of clustering achieved by traditional flat clustering algorithms (e.g., *k*-means clustering) heavily relies on the desired number of clusters (i.e., the value of *k*), which must be known in advance. Unfortunately, finding the right number of clusters is a non-trivial problem and no successful methods exist to automatically determine this value for a new, previously unseen dataset. Therefore, these algorithms require the user to provide the appropriate number of clusters. This approach, however, may be problematic because users with different backgrounds and varying levels of domain expertise may provide different values for *k*. Consequently, a clustering solution obtained by one user may not satisfy the needs of other users.

Additionally, large clusters in a flat clustering solution may not provide further insights about intra-cluster relationships (e.g., a large cluster containing instances, formally defined in Section 1.1, about animals may not provide additional information to distinguish land animals from marine animals). Similarly, small clusters may not provide further information about inter-cluster relationships.

In an attempt to avoid these problems, hierarchical clustering is widely used as a practical alternative to flat clustering. Nodes in a hierarchical clustering solution are organized in a general to specific fashion, and users have the option to analyze data at various levels of abstraction by expanding and collapsing these nodes. Most importantly, hierarchical clustering algorithms do not require the number of clusters to be known in advance.

The most successful hierarchical clustering algorithms include agglomerative algorithms such as UPGMA [18] and partitioning based algorithms such as bisecting *k*-means [18]. Additionally, a number of pattern-based hierarchical clustering algorithms have achieved success on a variety of datasets [1, 6, 10, 16, 17]. These algorithms come with an added advantage of automatically identifying cluster labels (i.e., the set of binary attributes defining each cluster), and many of them easily support soft clustering (i.e., a technique that assigns instances to one or more clusters). These features are not readily available in agglomerative and partitioning based algorithms. We identified four major problems with existing pattern-based hierarchical clustering algorithms. These problems are discussed in sections 1.2-1.5.

1.1 Notations and Definitions

Let D be a dataset, $I = \{i_1, i_2, i_3, \ldots, i_n\}$ be the complete set of distinct items (i.e., binary attributes) in D. An instance X is denoted as a pair $<id, Y>$ such that id is an identifier that uniquely identifies X and $Y \subseteq I$ represents the set of items in X. A pattern $P = \{p_1, p_2, p_3, \ldots, p_n\}$ is a subset of I. The set of data that contains P is denoted as $D_P = \{(id, Y) \in D \mid P \subseteq Y\}$. The *support* of a pattern P is defined as:

$$Support(P) = \frac{|D_P|}{|D|}$$

P is called frequent if $Support(P) \geq \text{min\_sup}$, where min_sup is the minimum support threshold. A frequent pattern F is called closed if there exists no proper superset $T \supset F$ with *Support* (T) = *Support* (F).

The function *global* (P, D, m) uses measure m to determine the global significance of pattern P in dataset D. Similarly, the function *local* (P, X, m) uses measure m to determine the local significance of pattern P in instance X.

1.2 Problem 1: Sensitivity of Globally Significant Patterns to Threshold Values

Most of the existing pattern-based hierarchical clustering algorithms [1, 6, 10, 16, 17] follow a similar framework. These algorithms first mine a set of globally significant patterns, and then use these patterns to build a cluster hierarchy. The set of globally significant patterns is obtained by first calculating global significance values of a set

of candidate patterns, and then selecting patterns with significance values that satisfy a user defined threshold. Common measures used to calculate global significance values include support [1, 6], support with closed itemset pruning [17], h-confidence [16], and a variety of interestingness measures [10]. Each selected pattern defines a cluster and instances are assigned to clusters if they contain the pattern. Various heuristics are applied to prune clusters and reduce or avoid overlap among clusters.

Because of their inherent dependence on the user defined global significance threshold, existing pattern-based hierarchical clustering algorithms face two potential problems. First, the set of selected globally significant patterns might not cover all instances (i.e., some instances might not contain any globally significant pattern), especially on datasets with a high degree of imbalance in cluster sizes. Second, the number of globally significant patterns found heavily depends on the threshold value used. On high dimensional, highly correlated datasets with many shared patterns, the number of these patterns can be thousands of times higher than the number of instances in the dataset. As we show in Section 6.2, this is not merely a matter of elegance; the excessive number of patterns can even cause global pattern-based algorithms to fail. In our previous work [10], we replaced minimum support with an interestingness threshold, which reduced the number of globally significant patterns. Still, there was no way to set an upper bound on the number of patterns, and the final set of global patterns sometimes did not cover all instances.

1.3 Problem 2: Insensitivity to Local Feature Frequencies

Instances in real-life text and web datasets may contain a feature (i.e., an item) more than once, and these locally frequent features may better represent the main topic of the instance as compared to other, locally infrequent features. As an example, we consider a recent news article on *cnn.com* about certain types of dinosaurs that are believed to be good swimmers. The word "dinosaurs" occurs 19 times in the article whereas the word "marine" occurs only once. Clearly, considering both of these words with equal importance can be problematic. Unfortunately, existing pattern-based hierarchical clustering algorithms do not fully utilize these local feature frequencies. Some approaches [6, 10] use these values in scoring functions to select suitable hierarchy nodes for instances, or to select node parents. However, none of the existing pattern-based hierarchical clustering algorithms utilize a local pattern significance measure in the process of mining the initial set of patterns used for clustering. Note that the idea of having a quantitative basis instead of a binary one for pattern mining in general is not new. For example, the share measure [2] captures the percentage of a numerical total that is contributed by the items in an itemset.

1.4 Problem 3: Unnecessary Coupling between Pattern Size and Node Height

Many existing pattern-based clustering algorithms [6, 10, 17] tightly couple the sizes of cluster labels with the node heights in the initial cluster hierarchy. In these algorithms, the first level in the cluster hierarchy contains all size-1 patterns, the

second level contains all size-2 patterns, and so on. This tight coupling is merely a consequence of the way global patterns are discovered (i.e., by first discovering size-1 patterns, which are used to form size-2 candidates etc.), and does not necessarily reflect a real-life setting. Users would surely appreciate more descriptive cluster labels (i.e., labels that reflect the cluster structure of the dataset with all appropriate patterns, regardless of their corresponding node heights).

Some of these approaches [6] later merge some child nodes with their parents if certain conditions are met, which does increase the label sizes. Still, a large percentage of nodes may remain with labels that have this property.

1.5 Problem 4: Artificial Constraints on Soft Clustering

Instances in real datasets may contain multiple patterns in the corresponding cluster hierarchy. As a consequence, pattern-based hierarchical clustering algorithms more easily support soft clustering when compared with traditional hierarchical clustering algorithms. However, existing algorithms require the user to provide "maximum instance duplication" [10, 17] as an input parameter, and always select the maximum number of clusters whenever possible for each instance. This approach may be problematic for applications to document clustering, where different documents can belong to a varying number of topics, and the same maximum number of topics may not be appropriate for all documents.

Additionally, instead of allowing instances to exist in the most suitable clusters at any level in the hierarchy, some of these approaches first force all instances to their most specific levels (i.e., called "inner termset removal" [10, 17]), and then select the top-*n* (with user defined *n*) most suitable clusters at that level. This restriction appears to be a matter of convenience (i.e., a quick way of constraining instance duplication), and may not be useful for real-life hierarchies. For example, we consider four retirement related nodes in the open directory [5]. Two of these nodes (i.e., Business->Investing->Retirement_Planning, and Home->Personal_Finance->Retirement) are found at level 3, one of these nodes (i.e., Business->Human_Resources->Compensation_and_Benefits->401k_and_Retirement) is found at level 4 and the last node (i.e., Society->People->Generations_and_Age_Groups->Seniors->Retirement) is found at level 5. The Open Directory is currently maintained by a large group of human editors. But if one were to automate the hierarchy generation process, the "inner termset removal" step in [6, 17] would assign a general retirement related instance only to the most specific node at level 5, and eliminate this instance from nodes at levels 3 and 4, which might contrast to the user's expectations.

1.6 IDHC: A More Flexible Instance-driven Hierarchical Clustering Algorithm

Led by these observations, we propose IDHC (i.e., **I**nstance **D**riven **H**ierarchical **C**lustering), a novel pattern-based, hierarchical clustering algorithm which is briefly summarized here. We do not follow the usual framework of first mining globally significant patterns and then using these patterns to build a cluster hierarchy. Instead, IDHC first allows each instance to "vote" for a variable number of representative size-

2 patterns in a way that ensures an effective balance between local and global pattern significance. At this step, the number of votes permitted is dynamically determined using a standard deviation based scheme, upper bounded by a small fixed constant *maxK*. Since there is no global pattern mining step, we do not need to use a global threshold (i.e., minimum support). Furthermore, the number of initial size-2 patterns is guaranteed to be linear in the total number of instances in the dataset, and all instances that contain at-least two unique items are guaranteed to be covered.

Next, these initial clusters are refined to obtain the rest of the cluster hierarchy by following an iterative, instance-driven process that inherently avoids combinatorial explosion. This process directly finds clusters for the next level, and simultaneously prunes duplicate clusters. In addition, this process produces more descriptive cluster labels than previous approaches, without tightly coupling node label sizes with node heights in the hierarchy. This also allows us to avoid forcing instances to their longest pattern clusters and enables instances to exist at multiple levels in the hierarchy.

We present results of experiments performed on 16 standard text datasets, and show in Section 6.1 that IDHC outperforms state-of-the-art hierarchical clustering algorithms both in terms of FScore and entropy measures [18]. Furthermore, we show that our parameters are robust across datasets and that the same untuned parameter values achieved high clustering quality on all datasets used in our experiments.

2 Related Work

The history of pattern-based clustering goes back to the early years of data mining. Han et al. [8] proposed a pattern-based flat clustering framework that uses association rules to generate a hypergraph of patterns (i.e., with items used as vertices and rules used to form hyperedges). An efficient hypergraph partitioning algorithm is then applied to obtain pattern clusters, and instances are clustered by assigning each instance to its best pattern cluster. This framework was later used in many applications, such as topic identification [3]. In another approach, Wang and Karypis [15] applied efficient search space pruning techniques to obtain a global summary set that contains one of the longest frequent patterns for each transaction. This set is later used to form clusters**.**

Based on globally frequent itemsets, Beil et al. [1] proposed an early pattern-based hierarchical clustering framework. This framework was later enhanced by Fung et al. [6] and Yu et al. [17], who improved various stages of the clustering process. In a different approach, Xiong et al. [16] first mine globally significant maximum hyperclique (i.e., with high h-confidence) patterns, and then associate instances to all applicable pattern clusters. These clusters are later merged by applying hierarchical agglomerative clustering (i.e., UPGMA), which was also used by [6] and [17] to merge top level nodes. Results in [16] show that this approach results in clustering quality that is similar to UPGMA, with an added advantage of automatically identifying cluster labels.

In our previous work [10], we improved the framework in [1, 6, 17] by using closed interesting itemsets as globally significant patterns used for clustering, and by using an interestingness measure to efficiently select hierarchical relationships. We

showed that this approach outperformed both existing pattern-based hierarchical clustering algorithms, and the best known agglomerative (i.e., UPGMA [18]) and partitioning based (i.e., bisecting *k*-means with I_2 criterion function [18]) algorithms on 9 commonly used datasets. All previous global pattern-based approaches, including ours, suffer from many of the limitations discussed in Section 1.

Our work also relates to subspace clustering, "an extension of traditional clustering that seeks to find clusters in different subspaces within a dataset. Subspace clustering algorithms localize the search for relevant dimensions allowing them to find clusters that exist in multiple, possibly overlapping subspaces" [9]. These algorithms either follow a top-down, or a bottom-up search strategy. Top-down algorithms find an initial clustering in the full set of dimensions and evaluate the subspaces of each cluster, iteratively improving the results. On the other hand, bottom-up algorithms find dense regions in low dimensional spaces and combine them to form clusters.

3 Dimensionality Reduction

Large real life document collections often suffer from high dimensionality, and this dimensionality varies greatly. For example, the commonly used Reuters 21578 dataset, which uses a restricted vocabulary, contains over 19K unique features in less than 11K news documents. More realistic news data from high frequency news streams may contain more than 150K features in less than 20K documents [12]. Therefore, reducing the dimensionality of the feature space can significantly improve the performance of pattern-based clustering algorithms, as the number of discovered patterns directly depends on the number of initial items. The availability of a labeled training set in supervised problems like classification allows for applying more sophisticated dimensionality reduction (i.e., feature selection) techniques, such as *Information Gain*. In contrast, there is limited information (i.e., global and local feature frequencies) available in unsupervised problems, such as clustering. Therefore, existing pattern-based clustering algorithms use a global threshold like minimum support as the primary dimensionality reduction technique. As discussed in Section 1.2, these approaches may not guarantee coverage.

To address the need to reduce dimensionality while attempting to ensure coverage, we adapt a two-phased heuristic approach in this paper:

***Step 1 (select initial features)*:** Heuristically select the globally most useful features by applying Zipf's law to select features that are neither too frequent nor too infrequent. All experiments reported in this paper selected features that exist in less than or equal to 95% of the instances, and at least two of them.

***Step 2 (ensure local coverage)*:** For each instance *i* in the dataset, first sort all features in *i* in the decreasing order of their local frequencies. Next, select the top-*k* highest frequency features and add them to the set of selected features. All experiments reported in this paper used a fixed value of $k = 10$, ensuring that each document is represented by at least ten locally most frequent features (Section 1.3).

4 Instance-driven Hierarchical Clustering

As discussed in Section 1.2, the threshold-based global pattern mining step in existing pattern-based hierarchical clustering algorithms may result in an unpredictable number of patterns, with no coverage guarantees. The IDHC algorithm in Figure 1 addresses these issues by using a novel approach. Subsections 4.1-4.3 explain the three major stages in the IDHC algorithm.

4.1 Stage 1: Select Significant Patterns with Respect to Each Instance

After reducing the dimensionality of the feature space and initializing the necessary data structures, instances in the dataset are processed in a purely local way (i.e., on an instance by instance basis). Each size-2 pattern in an instance is processed (lines 7-12 in Figure 1) to compute its "overall" significance with respect to the current instance, considering the pattern significance at both local and global levels.

First, we use the "local" method in Figure 2 to determine the local pattern significance. In this paper, we follow the simple approach of calculating local pattern significance by averaging the local frequencies of both of the items (i.e., $pattern_1$ and $pattern_2$) in the size-2 pattern (i.e., *pattern*), and leave exploring more sophisticated techniques for future work. Next, we call the "global" method on Figure 2, which uses a common interestingness measure to determine the global pattern significance. In our previous work [10], we evaluated 22 interestingness measures [7, 14], in the context of global pattern-based hierarchical clustering, and found that only a small number of measures were stable across the 9 datasets used in [10]. We have found that the same measures are useful to determine the global significance values in this context. See [7, 14] for computational details of these measures. Following the intuition that most of the interestingness measures are based on probability, we then multiply local and global significance values to determine the overall pattern significance with respect to the current instance (line 10).

All size-2 patterns are then sorted in decreasing order of their overall within-instance significance values (line 13), and these significance values are also used to calculate the mean and standard deviation of local significance (line 14). Considering the problem in Section 1.5, we adopt a dynamic standard deviation based scheme that selects a variable number of the most significant patterns (i.e., initial clusters) for each instance. This scheme selects up to *maxK* patterns with significance values that are greater than or equal to "*min_std_dev*" standard deviations from the mean, where *maxK* and *min_std_dev* are user definable parameters. Furthermore, we ensure coverage and account for boundary conditions (i.e., instances with a very small number of patterns) by also always selecting the most-significant pattern (line 16).

Once size-2 patterns are selected for all instances, each unique size-2 pattern forms an initial cluster, and instances are associated with the pattern clusters they selected (i.e., lines 18-22 in Figure 1 and method "add-to-cluster"). We maintain a list of pointers for each instance to track instance-to-cluster relationships.

Example: Figure 3(a) provides an example transaction dataset. Using *Added Value* [7, 14] as the interestingness measure, and *min_std_dev* = 1.0, we obtain the most significant patterns with respect to each instance, as shown in Figure 3(c). These

patterns are used to form the initial clusters in Figure 3(d), which also shows instance-to-cluster pointers. For demonstration purposes, this example used a small value for *min_std_dev*, which results in a relatively high number of initial patterns. Experiments in Section 6.1 used a more realistic value for this parameter.

```
01) build-hierarchy(dataset, min_std_dev, maxK, measure)
02)  {reduce dimensionality as explained in Section 3}
03)  top_level_clusters =  Φ
04)  instance_cluster_pointers =  Φ
05)  forall transactions t ∈ dataset do begin
06)      list = Φ
07)     forall size-2 patterns p ∈ t do begin
08)         significance_local = local(p, t, "LOCAL_FREQUENCY")
09)         significance_global = global(p, dataset, measure)
10)         significance(p) = significance_local * significance_global
11)         append (list, p)
12)      end
13)      {sort list in decreasing order of significance values}
14)      {calculate mean and standard_deviation of significance values in list}
15)      add-to-cluster(top_level_clusters, list(1), t)
16)      append(instance_cluster_pointers(t) , list(1))
17)      min_significance = mean + (standard_deviation * min_std_dev)
18)      for (i = 2; i <= maxK; i ++) do begin
19)         if significance(list(i)) < min_significance then break
20)         add-to-cluster(top_level_clusters, list(i), t)
21)         append(instance_cluster_pointers(t), list(i))
22)      end
23)  end
24)  prune-duplicates(instance_cluster_pointers, top_level_clusters)
25)  clusters_to_refine = top_level_clusters
26)  while size(clusters_to_refine) > 0 do begin
27)      refined_clusters = refine-clusters(instance_cluster_ptrs, clusters_to_refine)
28)      {regenerate instance_cluster_pointers using refined_clusters}
29)       prune-duplicates(instance_cluster_pointers, refined_clusters)
30)       clusters_to_refine = refined_clusters
31)  end
32)  {apply bisecting k-means to merge top_level_clusters}
33) end
```

Fig. 1. The IDHC algorithm; Φ represents an empty set

Figure 3(c) also demonstrates how the algorithm "balances" local and global pattern significance. As an example, instance "T4" contains one item (i.e., 'E') with local frequency = 4, three items (i.e., 'B', 'J' and 'K') with frequency = 3, two items (i.e., 'D' and 'L') with frequency = 2, and one item (i.e., 'C') with frequency = 1. In contrast, a pattern selection scheme that only considers local significance would rank size-2 patterns that include two of {'E', 'B', 'J' and 'K'} higher than the other size-2 patterns in this instance. Similarly, considering the global significance values in Figure 3(b), a pattern selection scheme that only considers global significance would rank patterns ('J', 'L') and ('B', 'L') higher than the other patterns. The final set of

```
01) add-to-cluster(cluster_list, pattern, instance)
02)  if pattern not in cluster_list then
03)    {add a new cluster with label = pattern to cluster_list}
04)  end
05)  {append instance to cluster with label = pattern in cluster_list}
06) end

01) prune-duplicates(instance_cluster_ptrs, cluster_list)
02)  forall ptr lists l ∈ instance_cluster_ptrs do begin
03)    m = {clusters in l that also exists in cluster_list}
04)    forall cluster pairs p(cluster1, cluster2) ∈ m do begin
05)      if cluster1 and cluster2 contain same instances then
06)        label(cluster1) = merge-labels(cluster1, cluster2)
07)        remove(cluster_list, cluster2)
08)      end
09)    end
10)  end
11) end

01) refine-clusters(instance_cluster_ptrs, cluster_list)
02)  refined_clusters = Φ
03)  forall ptr lists l ∈ instance_cluster_pointers do begin
04)    m = {clusters in l that also exists in cluster_list}
05)    {for each cluster c in m remove c from m if size(c) < 2}
06)    forall cluster pairs p(cluster1, cluster2) ∈ m do begin
07)      cluster_label = merge-labels(cluster1, cluster2)
08)      if cluster_label not in refined_clusters then
09)        {Add a new cluster to refined_clusters with label = cluster_label}
10)      end
11)      common_instances = cluster1 ∩ cluster2
12)      {add instances in common_instances to cluster with label = cluster_label in
13)       refined_clusters}
14)      {mark instances in common_instances for elimination in both cluster1 and cluster2}
15)      {Add cluster with label = cluster_label as a child node to both cluster1 and cluster2}
16)    end
17)  end
18) {In one pass over clusters in cluster_list, prune all instances that were marked for
19)  elimination}
20)  return refined_clusters
21) end

01) local(pattern, instance, measure)
02)  return average(freq(pattern 1, t), freq(pattern 2, t))
03) end

01) global(pattern, dataset, measure)
02)  {compute the contingency table using pattern 1 and pattern 2 as variables}
03)  {apply measure on the contingency table and return the outcome}
04) end
```

Fig. 2. Supporting methods for the IDHC algorithm; methods "local" and "global" reflect the reference implementation used in this paper, many alternative implementations are possible

(a) A transaction dataset as running example

Instance ID	*Features and their local frequencies*
T1	(A:2), (B:4), (D:1), (H:2), (J:4), (L:1)
T2	(A:3), (C:1), (D:6), (E:1), (G:4)
T3	(B:2), (C:3), (D:1), (I:5), (K:2)
T4	(B:3), (C:1), (D:2), (E:4), (J:3), (K:3), (L:2)
T5	(B:7), (C:2), (D:1), (H:3), (I:2)
T6	(A:1), (B:1), (C:1), (E:1), (J:3), (K:1)
T7	(B:9), (C:3), (F:4), (H:5), (J:1), (L:5)
T8	(C:6), (D:2), (G:1), (I:1), (K:3)
T9	(B:3), (D:2), (J:4), (K:1), (L:8)
T10	(A:4), (B:2), (D:7), (F:3), (I:6)
T11	(C:1), (E:1), (F:1), (G:2), (H:1), (I:4), (J:1)

(b) Global significance values of some size-2 patterns using Added Value (transformed to positive scale)

Pattern	*AV*	*Pattern*	*AV*	*Pattern*	*AV*	*Pattern*	*AV*
(B,D)	0.52	(B, K)	0.57	(E, J)	0.70	(J, K)	0.55
(B, E)	0.38	(B, L)	0.77	(E, K)	0.54	(J, L)	0.95
(B, J)	0.60	(D, E)	0.38	(E, L)	0.38	(K, L)	0.54

(c) Instance pattern selection

Instance ID	*Size-2 patterns*	*Significance range*		*Min sig.*	*Selected patterns*
T1	15	0.52	2.42	1.95	(B, J) (J, L), (H, J)
T2	10	0.77	2.38	2.14	(D, G), (A, D)
T3	10	0.78	2.29	1.76	(C, I)
T4	21	0.57	2.46	1.93	(E, J) (J, L)
T5	10	0.59	2.61	2.05	(B, H) (B, D)
T6	15	0.33	1.40	1.03	(E, J) (B, J) (J, K)
T7	15	0.90	5.40	3.70	(B, L)
T8	10	0.71	2.70	2.16	(C, G) (C, K)
T9	10	0.85	5.72	3.79	(J, L) (B, L)
T10	10	1.19	3.72	2.95	(D, I) (F, I)
T11	21	0.38	2.13	1.33	(G, I) (F, I) (C, I) (H, I)

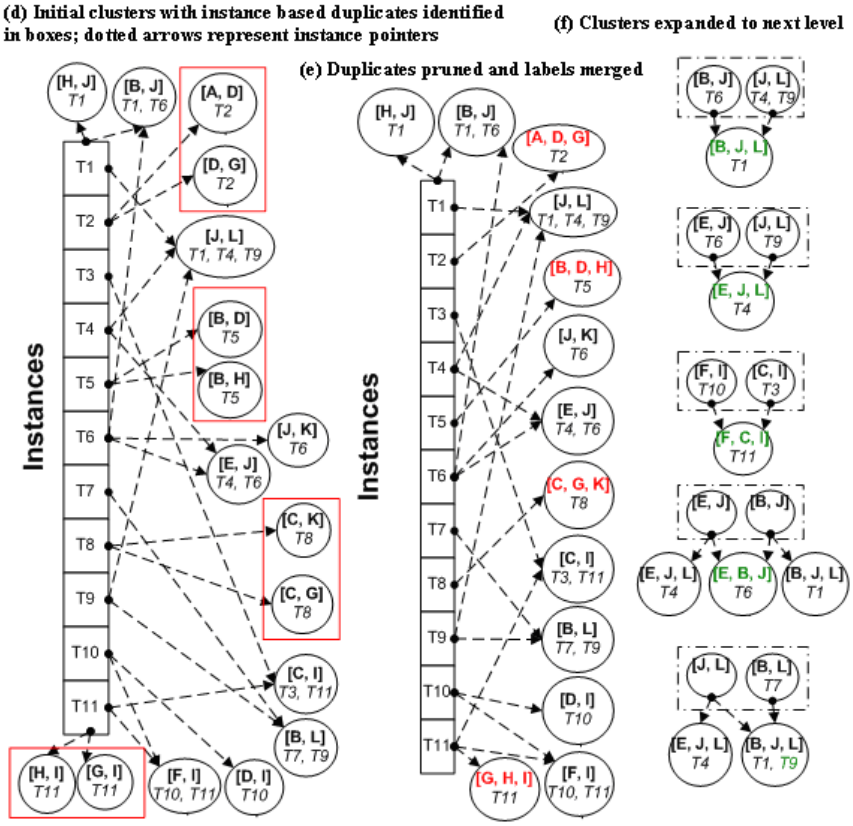

Fig. 3. A running example of various stages in our clustering process; in (d), candidates for deletion are boxed and in (f), newly added clusters and instances are marked in green

patterns selected for this instance (i.e., ('E', 'J') and ('J', 'L')) does include the most frequent local item (i.e., 'E'), but does not include two of the three items with frequency = 3. Instead, the algorithm selects pattern ('J', 'L') that has a higher global *Added Value*, providing a better balance between local and global significance.

Finally, we observe that the number of patterns selected by our standard deviation based scheme is not necessarily proportional to the number of size-2 patterns available in an instance. As an example, both T4 and T11 contain 21 size-2 patterns but our scheme selected twice as many patterns for T11.

4.2 Stage 2: Prune Duplicate Clusters

The set of initial clusters may contain duplicates (i.e., clusters with different labels but the exact same instances). As an example, there are four such duplicates identified in boxes in Figure 3(d). From an instance-based perspective, these duplicate clusters may not be very meaningful to the users, so we prune them in a way that enhances the label of the retained unique cluster. The naïve way of performing this operation requires comparing each cluster with all other clusters (quadratic time). Fortunately, as a positive side effect of our instance-driven approach, we already know instance-to-cluster relationships. It is not difficult to prove that checking for and pruning duplicate clusters locally (i.e., by comparing cluster pairs in each instance's pointer list, and repeating this process for all instances) also prunes all global duplicates.

Method "prune-duplicates" in Figure 2 implements pruning based on this observation, avoiding quadratic time cluster comparisons. As a side effect, in addition to removing cluster duplication, it also expands cluster labels. For this purpose, it merges the label of the retained cluster with the duplicate cluster being pruned. This results in increasingly meaningful (more specific) labels as the duplication level increases, partially addressing the problem described in Section 1.4; Section 4.3 addresses the rest.

Example: Considering the 17 initial clusters in Figure 3(d), the naïve way of identifying duplicate clusters will need up to 136 cluster-pair comparisons. Using instance-to-cluster relationships reduces the number of these comparisons to no more than 18 (i.e., we perform only three cluster pair comparisons for T1; {('H', 'J'), ('B', 'J')}, {('H', 'J'), ('J', 'L')} and {('B', 'J'), ('J', 'L')}). After processing all instances, we easily identify four duplicates (marked in boxes in Figure 3(d)). These duplicates are pruned and their labels are merged to obtain the 13 clusters in Figure 3(e).

4.3 Stage 3: Generate the Cluster Hierarchy

The initial clusters form the top level nodes in the cluster hierarchy, and the rest of the hierarchy is obtained by following an iterative cluster refinement process (lines 26-31 in Figure 1) that makes patterns progressively longer and cluster memberships progressively sparser. We make two intuitive observations that are responsible for high efficiency. First, atomic clusters (i.e., clusters with only one instance) can not be any more specific. Therefore, there is no need to consider these clusters for refinement (i.e., to generate child nodes for the next level). Second, refinement is only

necessary when a cluster *c*1 shares some instances with another cluster c2. These common instances can be removed from both *c*1 and *c*2, and added to a node that is a child to both of these nodes. This refined node still retains the instance memberships of the originating clusters for retrieval purposes (i.e., as child nodes are merely a specialization to, and are considered as a part of, their parents). Furthermore, this determination of overlap exploits instance-to-cluster pointers in a way similar to our duplicate cluster pruning scheme in Section 4.2.

Method "refine-clusters" in Figure 2, based on these observations, finds clusters for the next level. For this purpose, on an instance by instance basis, it first identifies cluster pairs from non-atomic clusters that share some instances (lines 4-6). Next, it appends these shared instances to a child cluster, the label of which is obtained by merging labels of the cluster pair itself (line 7-15).

A child cluster with a "merged" label may already exist, for two possible reasons. First, the same cluster pair may have existed in the pointer list of another instance that has already been processed. Second, merging labels of two different cluster pairs may result in a single label. As an example, merging labels of cluster pairs {('B', 'J'), ('J', 'L')} and {('J', 'L'), ('B', 'L')} in Figure 3(f) results in a single label (i.e., ('B', 'J', 'L')). However, it is easy to prove that in all cases, that first appending shared instances to the cluster with the resulting label, and then adding this cluster as a child to both the originating clusters does not affect instance memberships of the originating clusters. One final note: since clusters may share instances with several other clusters, deleting shared instances from originating clusters immediately after adding them to a new child cluster may result in eliminating some valid child clusters. Therefore, shared instances are marked for elimination as they are found (line 14), but pruned only after all cluster pairs are processed (line 18).

Example of one level of refinement: Figure 3(f) demonstrates refining clusters in Figure 3(e) to the next level. Processing cluster pointers from T1, we find only one pair of non-atomic clusters (i.e., {('B', 'J'), ('J', 'L')}), with T1 itself as the only shared instance. We then merge labels of the cluster pair to obtain ('B', 'J', 'L'), which is used to form the new child node. Cluster pointers from T2 to T11 are processed in a similar fashion to obtain 4 clusters for the next level. This process may result in adding several children to the same cluster (i.e., two child clusters added to ('E', 'J')) or appending several instances to an existing child cluster (i.e., two instances added to cluster ('B', 'J', 'L')).

Hierarchy refinement continues from level to level. Efficiency is maintained by tracking pointers to newly generated clusters (line 9 of method "refined-clusters"). These pointers are later used to regenerate instance-to-cluster pointers (line 28 of Figure 1) in one pass over the newly generated clusters. Since at each step, newly generated clusters can contain duplicates, we apply the duplicate cluster pruning process (Section 4.2) in each iteration. The full process is repeated until all clusters are refined (not shown in Figure 3).

Pattern-based clustering algorithms can result in a large number of top level clusters. Most of the existing algorithms [1, 6, 17] use agglomerative clustering to merge these top level nodes. In [10], considering those high computational costs, we replaced agglomerative clustering with bisecting *k*-means (using I_2 criterion function [18]). Following a similar approach, IDHC first obtains a joint feature vector for each top level cluster in the hierarchy by combining the feature vectors of all documents in

the cluster, and then applies bisecting *k*-means on these vectors. Unlike existing algorithms, first-level clusters in our initial hierarchy are not based on size-1 patterns.

5 Discussion

On the surface, it might seem like IDHC merely replaces some global thresholds (i.e., minimum support [1, 6, 16, 17] or minimum interestingness [10], and maximum instance duplication [10, 17]) with a set of local thresholds (i.e., *maxK* and *min_std_dev*). However, as we have discussed in Section 1, selecting a dataset-independent value for any of the commonly used global thresholds (i.e., minimum support) is non-trivial. Any selected value can result in a very large or a very small number of patterns, with no upper bound on the number of patterns mined. In contrast, our main threshold "*min_std_dev*" is supported by existing statistical principles. As an example, all the experiments in Section 6.1 used a fix value of 1.5 for *min_std_dev* on all 16 datasets. In the case of normally distributed significance values, with support from the central limit theorem, this value would result in selecting patterns with significance values in the 93rd percentile. Even if the distribution is not normal, Chebyshev's inequality provides an alternative, but still fixed, upper bound on the number of patterns selected. Furthermore, our empirical parameter *maxK* ensures that the number of initial patterns selected is always linear in the number of instances in the dataset. Section 6.1 shows that both of these parameters are robust across datasets.

Additionally, we note that the computational complexity of calculating global pattern significance using an interestingness measure (line 9 of Figure 1) depends on the underlying dataset representation used for frequency counting. Experiments in this paper represented datasets as compressed bitmaps [11], with frequency counting time complexity of O (*n*), where *n* is the number of instances in the dataset. However in practice, frequency counting using compressed bitmaps takes close to constant time on document datasets because of their inherent sparseness. Similarly, the sparseness of features limits the number of size-2 patterns considered for each instance even though it can theoretically be quadratic in the number of features in the entire dataset.

Finally, we note that in practice, IDHC scales linearly with the number of instances on real-life text datasets. Informally analyzing major steps in the IDHC algorithm, the first step of selecting size-2 patterns for instances (Section 4.1) scales linearly because the number of size-2 patterns considered for a given instance always remains the same. Further, adding instances to the dataset can only add a linear number of bits to the compressed bitmaps used for pattern frequency counting. Next, the pruning of duplicate clusters (Section 4.2) scales linearly because its processing of instance-to-cluster pointers is entirely local and instance by instance. The iterative hierarchy refinement process in Section 4.3 also scales linearly for the same reason. Note that the number of pointers in the instance-to-cluster list for each instance is upper bounded by *maxK* (or some proportion to *maxK*, depending on the level). Finally, the last step of merging top-level hierarchy nodes (Section 4.3) scales linearly because it uses bisecting *k*-means, which scales essentially linearly with the number of non-zero entries in the dataset [10, 18].

6 Experimental Results

We conduced an extensive experimental study, and evaluated the performance of IDHC on 16 standard text datasets with varying characteristics. With the exception of Reuters [13], we obtained all datasets from Cluto clustering toolkit [4]. We used the two standard hierarchical clustering evaluation metrics, FScore and entropy, as defined in [18], to compare the quality of cluster hierarchies produced by IDHC with two state-of-the-art hierarchical clustering algorithms.

FScore combines the standard precision and recall functions commonly used in Information Retrieval, and evaluates the overall quality of hierarchical tree using a small number of its nodes. For each ground truth class (i.e., each unique class label assigned to any instance in the dataset), FScore identifies the node in the hierarchical tree that best represents it and then measures the overall quality of the tree by evaluating this subset of clusters. Specifically, given a particular class L_r of size n_r and a particular cluster S_i of size n_i, suppose n_r^i documents in the cluster S_i belong to L_r, then the F value of this class and the cluster is defined as:

$$F(L_r, S_i) = \frac{2 * R(L_r, S_i) * P(L_r, S_i)}{R(L_r, S_i) + P(L_r, S_i)}$$

where $R(L_r, S_i) = n_r^i / n_r$ is the recall value and $P(L_r, S_i) = n_r^i / n_i$ is the precision value defined for the class L_r and the cluster S_i. The FScore of class L_r is the maximum F value attained at any node in the hierarchical tree T. That is,

$$FScore(L_r) = \max_{S_i \in T} F(L_r, S_i)$$

The FScore of the entire hierarchical tree is defined to be the sum of the individual class specific FScores weighted according to the class size. That is,

$$FScore = \sum_{r=1}^{c} \frac{n_r}{n} FScore(L_r)$$

where c is the total number of classes. In general, higher FScore values indicate a better clustering solution.

On the other hand, entropy takes into account the distribution of documents in all nodes of the tree. Given a particular node S_r of size n_r, the entropy of this node is defined to be:

$$E(S_r) = -\frac{1}{\log q} \sum_{i=1}^{q} \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r}$$

where q is the total number of classes and n_r^i is the number of documents of the ith class that were assigned to the rth node. Then, the entropy of the entire tree is:

$$E(T) = \sum_{i=1}^{p} \frac{1}{p} E(S_r)$$

where p is the total number of non-leaf nodes of the hierarchical tree T. In general, lower entropy values indicate a better clustering solution.

We selected bisecting k-means with I_2 criterion function [18] as the first state-of-the-art algorithm for comparison. I_2 criterion function is defined as:

$$maximize \sum_{r=1}^{k} \sum_{d_i \in S_r} \cos(d_i, C_r) = \sum_{r=1}^{k} \|D_r\|$$

We also selected our previous global pattern-based hierarchical clustering algorithm, referred to as GPHC here as the second state-of-the-art algorithm for comparison with IDHC. We limited our comparison to these two existing algorithms, since we have shown in [10] that GPHC significantly outperformed FIHC [6] and TDC [17]. Furthermore, [18] reported that bisecting *k*-means with I_2 criterion function outperforms UPGMA. Our findings in [10] were also consistent with this observation. Consequently, we do not compare our algorithm against FIHC, TDC and UPGMA. In addition, we do not compare our algorithm against HICAP as it is reported [16] to have a performance that is comparable to UPGMA.

All three algorithms we tested used bisecting *k*-means as implemented in the Cluto clustering toolkit [4], which uses randomized heuristics. Therefore, we ensured fairness by using the same dedicated machine (i.e., a 64-bit server with two Xeon processors and 8 GB of memory) to execute each clustering algorithm on each dataset 10 times, and reported the averages. In addition, since Cluto produces both flat (with number of clusters as a user defined parameter), and hierarchical clustering solutions for bisecting *k*-means. We ensured an apples-to-apples comparison by always considering the full hierarchical solution and by using the same code to calculate FScore and entropy values for cluster hierarchies produced by all three algorithms.

6.1 Clustering Performance

We first evaluated the effectiveness of various interestingness measures [7, 14], to determine global significance values (i.e., Section 4.1) on 5 datasets (Table 1), and found that the top measures we reported in [10] also consistently performed well in this context. We observed that *Added Value* generally outperformed other measures, while *Chi-Square*, *Certainty Factor*, *Yule's Q* and *Mutual Information* achieved very similar performance. Consequently, results in this section used *Added Value* as the interestingness measure, with *min_std_dev* and *maxK* fixed to 1.5 and 6 respectively. We obtained this value for *min_std_dev* in a principled way by executing IDHC on one dataset (Reuters) and varying *min_std_dev* between 1.0 and 4.0, in intervals of 0.1, selecting the value giving the best entropy score on the selected dataset.

We compared the clustering performance of IDHC against bisecting *k*-means and GPHC on 16 common text datasets, 9 of which were also used in [10]. Since GPHC also uses an interestingness measure, we used MI (i.e., *Mutual Information*), which we found to be the top measure on text datasets in [10]; see Section 6.2 for a note on the MI thresholds used for GPHC.

Table 2 presents the results of this experiment. Among the 16 datasets, our IDHC algorithm with fixed parameter values achieved the best FScores on 7 datasets, and was ranked second on another 7 datasets, resulting in the highest average FScores across all datasets. Most significantly, IDHC outperformed existing algorithms on all datasets in terms of entropy, and achieved average entropy that is about 3 times smaller than the best competitor. As noted above, entropy considers the distribution of instances in all nodes of the tree whereas FScore only considers a small number of nodes (i.e., one node per ground truth class) for evaluation, and ignores the quality of all other nodes. Consequently, entropy may be a more effective measure to evaluate the quality of cluster hierarchies.

Table 1. FScores and entropies using top interestingness measures; MI = Mutual Information

Dataset	*FScores*					*Entropies*				
	Certainty Factor	*MI*	*YulesQ*	*Chi Square*	*Added Value*	*Certainty Factor*	*MI*	*YulesQ*	*Chi Square*	*Added Value*
Reuters	0.825	0.832	0.839	0.837	0.846	0.005	0.012	0.004	0.022	0.005
la12	0.711	0.595	0.710	0.486	0.748	0.066	0.031	0.049	0.045	0.038
re0	0.642	0.619	0.667	0.639	0.615	0.020	0.026	0.018	0.037	0.016
sports	0.847	0.851	0.853	0.819	0.870	0.007	0.003	0.007	0.004	0.005
tr12	0.764	0.689	0.756	0.675	0.769	0.042	0.033	0.032	0.031	0.037
	0.758	0.717	0.765	0.691	**0.769**	0.028	0.021	0.022	0.028	**0.020**

Table 2. Clustering quality on text datasets; higher FScores and lower entropies are better

Dataset				*FScores*			*Entropies*		
Name	*# classes*	*# instances*	*# features*	*bi-k I_2*	*GPHC*	*IDHC*	*Bi-k I_2*	*GPHC*	*IDHC*
reuters	90	10,787	19,127	0.835	**0.851**	0.846	0.075	0.155	**0.005**
classic	4	7,094	41,681	0.782	**0.880**	0.759	0.060	0.025	**0.021**
hitech	6	2,301	22,498	0.528	0.540	**0.544**	0.224	0.172	**0.074**
k1a	20	2340	21,839	0.668	0.654	**0.676**	0.106	0.045	**0.041**
k1b	6	2340	21,839	0.882	**0.903**	0.897	0.042	0.042	**0.021**
la12	6	6,279	30,125	0.741	0.661	**0.748**	0.120	0.062	**0.038**
mm	2	2521	126,373	0.774	**0.943**	0.909	0.073	0.053	**0.014**
ohscal	10	11,162	11,465	**0.601**	0.530	0.554	0.198	0.237	**0.081**
re0	13	1,504	2,886	0.610	**0.672**	0.615	0.115	0.077	**0.016**
reviews	5	4,069	36,746	0.801	0.818	**0.833**	0.073	0.048	**0.013**
sports	7	8,580	27,673	0.882	**0.886**	0.870	0.030	0.016	**0.005**
tr11	9	414	6,429	**0.795**	0.519	0.790	0.107	0.141	**0.038**
tr12	8	313	5,804	0.689	0.604	**0.769**	0.133	0.161	**0.037**
tr23	6	204	5,832	0.667	0.487	**0.679**	0.136	0.042	**0.038**
tr31	7	927	10,128	0.837	0.584	**0.840**	0.041	0.114	**0.013**
wap	20	1,560	8,460	**0.683**	0.663	0.670	0.106	0.047	**0.043**
Average				0.736	0.700	**0.750**	0.102	0.090	**0.031**

Note that in order to demonstrate the robustness of IDHC parameters across datasets, we used the same fixed parameter values on all 16 datasets to obtain results in Table 2. Additional experiments indicate that tuning these parameters to each dataset may increase performance even further. As an example, replacing *Added Value* with *Chi-Square* improved the FScore from 0.897 to 0.916 and improved entropy from 0.021 to 0.012 on the k1b dataset.

6.2 Robustness in the Number of Patterns

In Section 1.2, we noted that the threshold-based global pattern mining step in existing algorithms may result in an unpredictable number of patterns. During our experiments, we found many examples that clearly demonstrate this problem. We provide a few here and also show that IDHC does not suffer from this problem.

Table 3. Number of size-2 patterns

Dataset	*#instances*	*#features*	*Approx. number of size-2 patterns*		
			GPHC, MI	*GPHC, YulesQ*	*IDHC*
mm	2,521	126,373	2.4 million	{fails}	3,651
reviews	4,069	36,746	2.6 million	{fails}	5,952
sports	8,580	27,673	1.4 million	{fails}	12,607
tr11	414	6,429	4.3 million	11.4 million	604
tr12	313	5,804	3.6 million	8.8 million	464
tr23	204	5,832	7.6 million	12.2 million	282
tr31	927	10,128	7.0 million	{fails}	1,360

In [10], we showed that the GPHC algorithm worked well on all 9 datasets with interestingness threshold values of 0.1 and 0.85 for MI and YulesQ respectively. However, when we used the same threshold values on some of the highly correlated datasets, we obtained an extremely large number of size-2 patterns (Table 3), so much so that the mining process could not continue because it exhausted the available system resources. In some cases, GPHC could not even finish mining size-2 patterns. This required us to increase these threshold values to obtain the results reported in Table 2.

The problem of generating too many patterns is not unique to GPHC. All global pattern mining based clustering algorithms suffer from a similar problem. In fact, a pure frequent or closed frequent itemset based algorithm would be expected to find an even higher number of itemsets [10]. In contrast, our local standard deviation based pattern selection scheme selected up to three orders of magnitude fewer size-2 patterns, without loss of performance in general (Table 2). We observe that in practice the upper limit guarantees provided by the central limit theorem and Chebyshev's inequality are sufficient and *maxK* is rarely used to limit the number of size-2 patterns.

7 Conclusions and Future Work

The instance-driven approach uses standard deviation of pattern significance scores as a local threshold to select representative size-2 patterns for each instance in the dataset. This eliminates the need to use highly unstable global thresholds, guarantees that the final set of selected patterns covers all instances, and also utilizes both global and local pattern significance. The selected patterns form initial clusters, and a cluster hierarchy is obtained by following an iterative cluster refinement process that avoids global processing by utilizing instance-to-cluster relationships, produces more meaningful cluster labels, and allows for more flexible soft clustering. Our experiments indicate that this approach outperforms existing hierarchical clustering algorithms both in terms of FScore and entropy. Most importantly, this approach was equally effective on highly correlated datasets.

In the future, we plan to investigate more sophisticated ways of calculating local pattern significance and conduct a more comprehensive study on the effectiveness of

dimensionality reduction, and pattern significance at each level. In addition, we plan to apply IDHC on non-textual dense datasets such as the UCI dataset collection.

8 Acknowledgements

We would like to thank Prof. Howard Hamilton from University of Regina, Fabian Moerchen from Siemens Corporate Research, and the anonymous reviewers for their useful comments that helped a lot in improving this paper.

References

1. Beil, F., Ester, M., Xu, X.: Frequent term-based text clustering. In: International Conference on Knowledge Discovery and Data Mining, pp. 436-442 (2002)
2. Carter, C. L., Hamilton, H. J., Cercone, N.: Share Based Measures for Itemsets. In: First European Conference on the Principles of Data Mining and Knowledge Discovery (1997)
3. Clifton, C., Coolie, R., Rennie. J.: TopCat: Data Mining for Topic Identification in a Text Corpus. IEEE Transactions on Knowledge and Data Engineering, Volume 16, No. 8 (2004)
4. Cluto. http://glaros.dtc.umn.edu/gkhome/views/cluto
5. The Open Directory. http://dmoz.org
6. Fung, B., Wang, K., Ester, M.: Hierarchical document clustering using frequent itemsets. In: SIAM International Conference on Data Mining, pp. 59-70 (2003)
7. Geng, L., Hamilton, H. J.: Interestingness Measures for Data Mining: A Survey, ACM Computing Surveys, Volume 38, No. 3 (2006)
8. Han, E.-H., Karypis, G., Kumar, V., Mobasher, B.: Clustering based on association rule hypergraphs. In: Research Issues on Data Mining and Knowledge Disc (1997)
9. Parsons, L., Haque, E., Liu, H.: Subspace Clustering for High Dimensional Data: A Review. ACM SIGKDD Explorations Newsletter , Volume 6, No. 1 (2004)
10. Malik, H. H., Kender, J. R.: High Quality, Efficient Hierarchical Document Clustering Using Closed Interesting Itemsets. In: Sixth IEEE International Conference on Data Mining, pp. 991-996 (2006)
11. Malik, H. H., Kender, J. R.: Optimizing Frequency Queries for Data Mining Applications. In: Seventh IEEE International Conference on Data Mining, pp. 595-600 (2007)
12. Mörchen, F., Brinker, K., Neubauer, C.: Any-time clustering of high frequency news streams. In: Data Mining Case Studies Workshop, the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2007)
13. Reuters 21578. http://kdd.ics.uci.edu/databases/reuters21578
14. Tan, P., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: 8th international conference on Knowledge discovery and data mining (2002)
15. Wang, J., Karypis, G.: SUMMARY: Efficient Summarizing Transactions for Clustering. In: Fourth IEEE International Conference on Data Mining (2004)
16. Xiong, H., Steinbach, M., Tan, P.-N., Kumar, V.: HICAP: Hierarchical Clustering with Pattern Preservation. In: SIAM International Conference on Data Mining (2004)
17. Yu, H., Searsmith, D., Li, X., Han, J.: Scalable Construction of Topic Directory with Nonparametric Closed Termset Mining. In: Fourth IEEE International Conference on Data Mining (2004)
18. Zhao, Y., Karypis, G.: Hierarchical Clustering Algorithms for Document Datasets. Data Mining and Knowledge Discovery, Volume 10, pp. 141--168, No. 2 (2005)