Efficient Algorithms for Clustering and Classifying High Dimensional Text and Discretized Data using Interesting Patterns

Hassan H. Malik

Submitted in partial fulfillment of the
Requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2008

Abstract

Efficient Algorithms for Clustering and Classifying High Dimensional Text and Discretized Data using Interesting Patterns

Hassan H. Malik

Recent advances in data mining allow for exploiting patterns as the primary means for clustering and classifying large collections of data. In this thesis, we present three advances in pattern-based clustering technology, an advance in semi-supervised pattern-based classification, and a related advance in pattern frequency counting.

In our first contribution, we analyze numerous deficiencies with traditional pattern significance measures such as support and confidence, and propose a web image clustering algorithm that uses an objective interestingness measure to identify significant patterns, yielding measurably better clustering quality.

In our second contribution, we introduce the notion of closed interesting itemsets, and show that these itemsets provide significant dimensionality reduction over frequent and closed frequent itemsets. We propose GPHC, a sub-linearly scalable global pattern-based hierarchical clustering algorithm that uses closed interesting itemsets, and show that this algorithm achieves up to 11% better FScores and up to 5 times better entropies as compared to state-of-the-art agglomerative, partitioning-based, and pattern-based hierarchical clustering algorithms on 9 common datasets.

Our third contribution addresses problems associated with using globally significant patterns for clustering. 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, produces more descriptive cluster labels, and allows a more flexible soft clustering scheme. Results of experiments performed on 40 standard datasets show that IDHC almost always outperforms state-of-the-art hierarchical clustering algorithms and achieves up to 15 times better entropies, without requiring any tuning of parameter values, even on highly correlated datasets.

In our fourth contribution, we propose CPHC, a semi-supervised classification algorithm that uses a pattern-based cluster hierarchy as a direct means for classification. All training and test instances are first clustered together using our instance-driven pattern-based hierarchical clustering algorithm, and the resulting cluster hierarchy is then used directly to classify test instances, eliminating the need to train a classifier on an enhanced training set. For each test instance, we first use the hierarchical structure to identify nodes that contain the test instance, and then use the labels of co-existing training instances, weighing them proportionately to their pattern lengths, to obtain class label(s) for the test instance. Results of experiments performed on 19 standard datasets show that CPHC outperforms a number of existing classification algorithms even with sparse training data.

Our final contribution deals with the problem of finding a dataset representation that offers a good space-time tradeoff for fast support (i.e., frequency) counting and also automatically identifies transactions that contain the query itemset. We compare FP Trees and Compressed Patricia Tries against several novel variants of vertical bit vectors. We compress vertical bit vectors using WAH encoding and show that simple

lexicographic ordering may outperform the Gray code rank-based transaction reordering scheme in terms of RLE compression. These observations lead us to propose HDO, a novel Hamming-distance-based greedy transaction reordering scheme, and aHDO, a linear-time approximation to HDO. We present results of experiments performed on 15 common datasets with varying degrees of sparseness, and show that HDO-reordered, WAH encoded bit vectors may take as little as 5% of the uncompressed space, while aHDO achieves similar compression on sparse datasets. With results from over 10^9 database and data mining style frequency query executions, we show that bitmap-based approaches result in up to 10^2 times faster support counting, and that HDO-WAH encoded bitmaps offer the best space-time tradeoff.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deepest appreciation to my advisor Professor John Kender for his continuous support, dedication, patience, appreciation, and for always believing in me. I was not a typical graduate student for him. Both in terms of my research interests and because I worked full time, all this time. This required conducting research meetings at odd times and handling many special situations, including some that only a great teacher with infinite patience could have handled. I will always consider myself the most fortunate to have had Professor Kender as my advisor.

I would like to thank Dr. Paul Natsev for serving on all three of my doctoral committees, and for providing great feedback on my earlier clustering algorithms. I would like to thank Professor Luis Gravano for serving on my thesis proposal and defense committees, and for providing extremely valuable feedback that helped me focus on core data mining. I would like to thank Professor Howard Hamilton for his work on interestingness measures that has had significant impact on my research, and for taking the time to serve on my thesis defense committee. I would also like to thank Professor Tony Jebara for serving on my thesis defense committee.

I am extremely grateful to my friends and colleagues at Libgo Travel, Inc. for always understanding my educational priorities, and for holding the fort when I was at Columbia taking classes, or when I had both an exam and a work-related priority to balance. In particular, I would like to extend my sincere thanks to Majid Khan, Jawwad Sultan, Parshuram Patki, Rao Manepalli, Armughan Rafat, Bryan Woody, Dory Velten-Lerescu, Rich Meyer and Mitch Talisman.

I would like to thank my parents for always supporting me, and for making endless sacrifices to help me achieve my dreams. I would also like to thank my grandmother for continuously encouraging me to pursue further education.

Last but not the least, I would like to thank my beloved wife Annie, and my daughters Rakiya and Arfa for their unconditional love, countless sacrifices, continuous support, and for bringing in joy and happiness to my everyday life.

This thesis is dedicated to my late grandfather Iftikhar H. Malik; the most honest and selfless person I have ever known. I also dedicate this thesis to Nobel Laureate Dr. Abdus Salam and his life-long efforts to promote scientific research as a shared heritage of all mankind.

1. Introduction and Prior Work

Clustering is the partitioning of a dataset into subsets (clusters), so that the data in each subset (ideally) share some common trait, and it continues to be the focal point in unsupervised learning research. Recent advances in data mining allow for exploiting patterns (e.g., a set of binary attributes) as primary means for clustering large collections of data. Pattern-based clustering algorithms differ from traditional clustering algorithms in that they first find a set of patterns, and then build hard (i.e., each instance is assigned to exactly one cluster) or soft (i.e., each instance is assigned to one or more clusters) clusters around these patterns. The patterns used for clustering are typically selected by a mining process (i.e., itemset mining or association rule mining) that considers patterns based on their global significance.

In this thesis, we first analyze numerous deficiencies with traditional pattern significance measures used in association rule mining (i.e., support and confidence), and evaluate 20 alternative objective interestingness measures. We compare these measures by using the resulting association rules for clustering web images, in a way similar to [34]. Our experiments indicate that objective interestingness measures significantly outperform support and confidence in terms of clustering quality. Additionally, we show that combining textual and linear-time-extractable signal features result in better clustering as compared to signal-only or text-only approaches.

Next, we consider the problem of using patterns to generate cluster hierarchies. A cluster hierarchy is a hierarchical representation of a set of (possibly non-disjoint) clusters such that each node in the hierarchy represents a cluster, and each node may have zero or more child nodes, without allowing for cycles. We introduce the notion of closed interesting itemsets (i.e., closed itemsets with high interestingness), and show that these itemsets provide significant dimensionality reduction over frequent and closed frequent itemsets, meaning that this approach leads to a reduction in the number of attributes under consideration (Appendix A defines "dimension"). We propose GPHC, a global pattern-based hierarchical clustering algorithm that uses closed interesting itemsets as the primary means to generate a cluster hierarchy. In addition, GPHC uses an objective interestingness measure to efficiently select node parents strictly based on their cluster labels (i.e., patterns), without requiring inspection of node contents. This allows us to improve the efficiency of the framework followed by existing pattern-based hierarchical clustering algorithms [25, 5, 88]. These improvements include the use of bitmaps (i.e., spatially mapped arrays of bits), and the replacement of UPGMA with bisecting *k*-means for merging top-level nodes. Results of experiments performed on 9 common datasets show that GPHC achieves up to 11% better FScores and up to 5 times better entropies as compared to state-of-the-art agglomerative (i.e., bottom-up), partitioning-based (top-down), and pattern-based hierarchical clustering algorithms. Furthermore, we show that GPHC scales sub-linearly on the size of document collections, since the number of features (i.e., attributes) in these collections grows slower than their number of documents, as anticipated by Heaps' law [37].

Even though closed interesting itemsets provide substantial improvements over frequent and closed frequent itemsets, we found that using globally significant patterns for clustering can be problematic in general. In order to prune an exponentially large search space, global pattern mining algorithms use a threshold (i.e., minimum support, minimum confidence, or minimum interestingness in the case of closed interesting itemsets). Consequently, these algorithms may result in an unpredictable number of patterns. Even so, the final set of globally significant patterns may not cover all instances, especially on unbalanced datasets. On high dimensional, highly correlated datasets, we found that the number of globally significant patterns of closed interesting itemsets can even be tens to thousands of times higher than the number of instances in the dataset.

Led by these observations, 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 principled local standard deviation-based scheme, and the rest of the cluster hierarchy is obtained by following a unique iterative cluster refinement process that simultaneously grows patterns and clusters. In addition, this algorithm accounts for local feature frequencies, produces more descriptive cluster labels, and adopts a soft clustering scheme that allows instances to exist in suitable nodes at various levels in the cluster hierarchy. Results of experiments performed on 40 standard datasets show that IDHC almost always outperforms state-of-the-art

hierarchical clustering algorithms and achieves up to 15 times better entropies, without requiring any tuning of parameter values. Most importantly, IDHC is stable and is equally effective on highly correlated datasets, where it uses only a few hundred patterns, as compared to millions of patterns used by global pattern-based clustering algorithms.

Next, we use the cluster hierarchy produced by our pattern-based hierarchical clustering algorithm as a direct means for classification. Classification is a procedure in which individual instances (Appendix A) are placed into groups (classes) based on information on one or more characteristics inherent in the instances (referred to as traits, variables, characters, etc.) and based on a training set of previously labeled instances, where each label is a string that uniquely identifies a class in the output space. Traditional inductive classifiers are trained on instances in the training set to produce a classification model (such as decision trees). This model is later used to classify previously unseen test instances. Considering that these classifiers may not fully exploit the distribution of test instances in the context of the whole dataset, a number of recent approaches [68, 89, 45] adopted a semi-supervised model for classification. These approaches first apply an unsupervised, flat clustering algorithm (i.e., *k*-means clustering) to cluster all (i.e., training and test) instances in the dataset, and then use the resulting clustering solution to enhance the training set (i.e., by using clusters to generate additional training instances). A classifier is then trained on the enhanced training set.

Pattern-based hierarchical clustering algorithms have numerous advantages over flat clustering algorithms. They organize data in a general to specific fashion without

requiring the number of clusters to be known in advance, automatically produce cluster labels, and more easily support soft clustering. Our semi-supervised classification algorithm CPHC therefore applies our IDHC algorithm on the whole dataset to produce a cluster hierarchy. We empirically observe that the average node entropies almost always linearly improve (i.e., decrease) with increasing pattern lengths. Motivated by this observation, CPHC uses node-pattern-lengths to determine cluster (i.e., node) weights. To classify a test instance, CPHC first uses the hierarchical structure to identify nodes that contain the test instance, and then uses the labels of co-existing training instances, weighing them by node pattern-lengths to obtain class label(s) for the test instance. Results of experiments performed on 19 standard datasets show that CPHC outperforms a number of existing classification algorithms both on single-label classification problems (measured in terms of classification accuracy) and on multi-label classification problems (measured in terms of micro-averaged precision-recall breakeven point), even with sparse training data.

A common challenge faced by data mining algorithms in general, and pattern-based hierarchical clustering algorithms in particular, is to find an effective dataset representation that supports fast support (i.e., frequency) counting while also automatically identifying transactions that contain the query itemset. We compare the memory requirements and support counting performance of FP Trees and Compressed Patricia Tries (see Appendix A) against several novel variants of vertical bit vectors (i.e., bitmaps). First, borrowing ideas from the very large databases (VLDB) domain, we compress vertical bit vectors using Word-Aligned Hybrid

(WAH) encoding. Second, we evaluate the Gray code rank-based transaction reordering scheme and show that, in practice, simple lexicographic ordering, obtained by applying least-significant-bit first (LSB) radix sort, outperforms this scheme.

Led by these results, we propose HDO, a novel Hamming-distance-based greedy transaction reordering scheme, and aHDO, a linear-time approximation to HDO. We present results of experiments performed on 15 common datasets with varying degrees of sparseness, and show that HDO-reordered, WAH encoded bit vectors can take as little as 5% of the uncompressed space, while aHDO achieves similar compression on sparse datasets. Finally, with results from over 10^9 database and data mining style frequency query executions, we show that bitmap-based approaches result in up to 10^2 times faster support counting, and HDO-WAH encoded bitmaps offer the best space-time tradeoff.

1.1 Prior Work

This subsection provides a brief overview of prior art in the areas of pattern-based clustering, semi-supervised pattern-based classification, and pattern frequency counting. Each of the four following chapters includes a more detailed overview of their corresponding previous work.

1.1.1 Pattern-based Clustering

The history of pattern-based clustering goes back to the early years of data mining. A number of pattern-based clustering algorithms were proposed as the field evolved, and have achieved initial successes on a variety of clustering problems. However, the

progress in pattern-based clustering remained limited because of persistent problems with underlying data mining methods such as association rule mining and itemset mining. One of these problems is due to the use of frequency-based measures for identifying significant patterns, which may result in pruning highly discriminating but less frequent patterns. Another problem is using global thresholds for mining patterns, which does not guarantee coverage and also does not provide an upper bound on the number of initial patterns mined.

Han et al. [34] proposed a pattern-based flat clustering framework that uses association rules to generate a hypergraph of patterns (i.e., with atomic patterns (binary attributes) 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 [16] and our web image clustering algorithm [55]. However, we found that the quality of clustering achieved by hypergraph-partitioning-based clustering methods may significantly vary on each execution, as hypergraph partitioning algorithms use various randomized heuristics to approximately solve this NP-complete problem. In addition, this framework does not provide clear ways of associating features with multiple feature clusters, or of generating cluster hierarchies.

In another approach, Wang and Karypis [80] 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**.** As noted

in Section 8 of [15], this approach has many shortcomings, including its dependence on difficult-to-specify minimum support threshold values.

In contrast to the flat clustering algorithms mentioned above, Beil et al. [5] proposed a pattern-based hierarchical clustering framework that is based on globally frequent itemsets. This framework was later enhanced by Fung et al. [25] and Yu et al. [88], who improved various stages of the clustering process. In a different approach, Xiong et al. [85] first mine globally significant maximum hyperclique patterns (i.e., patterns with high h-confidence), 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 [25] to merge top level nodes. Results in [85] show that this approach results in clustering quality that is similar to that of UPGMA, with an added advantage of automatically identifying cluster labels. Recent research [91], however, suggests that partitioning-based hierarchical clustering algorithms such as bisecting *k*-means outperform agglomerative algorithms both in terms of clustering quality and runtime performance. Consequently, our GPHC algorithm (Chapter 3) [56] replaces UPGMA with bisecting *k*-means (using the I_2 criterion function) to merge top-level nodes in the initial cluster hierarchy.

1.1.2 Semi-supervised Pattern-based Classification

A number of existing algorithms use patterns as primary means for classification. These algorithms mine patterns from instances in the training set to form a classification model, which is later used to classify previously unseen test instances.

Rule-induction-based classifiers like FOIL [66], RIPPER [19], CPAR [87] and C4.5 [65] use heuristics, such as *Gini Index* and *Information Gain* (or *Information Gain* variants), to identify the best literal by which to grow the current rule [81]. Many of them follow the sequential database covering paradigm (i.e., learn one rule, remove the data it covers, then repeat). In contrast, association rule-based classifiers such as CBA [50], CAEP [23], CMAR [48], ARC-BC [4], and DeEPs [46] first mine a large set of association rules that satisfy user-defined support and confidence thresholds, and then extract the final set of classification rules by following a sequential database covering technique. With Harmony [81], Wang and Karypis proposed an instance-centric approach to mine classification rules. Harmony builds the classification model by directly mining some user-defined number of highest-confidence rules for each training instance that satisfy minimum support. Furthermore, Harmony simultaneously mines rules for all classes in the training set.

Considering that these classifiers may not fully exploit the distribution of test instances in the context of the whole dataset, a number of recent approaches [68, 89, 45] adopted a semi-supervised model for classification. In a way similar to transductive learning [78], which allows the structure of the test set to play a role in classification, these semi-supervised approaches use clustering as a way of enhancing the training set. Raskutti et al. [68] used unlabeled data that is not part of the test set to improve the performance of text classification. This is achieved by clustering labeled and unlabeled instances together, and extracting new features from these clusters to enhance the classification model (i.e., features that may only exist in unlabeled instances). In another approach, Zeng et al. [89] first clustered training and

test sets together. The resulting clustering solution is then used to obtain labels for some of the unlabeled test instances, and the newly labeled instances are added to the training set. The extended training set is finally used to train a classifier. In a similar approach [45], Kyriakopoulou and Kalamboukis first clustered training and test sets together. The dataset is then augmented with meta features extracted from the resulting clusters, and a classifier is trained on the expanded training set.

However, existing pattern-based classification algorithms and existing semi-supervised classification algorithms suffer from many related problems. In addition to ignoring the distribution of test instances in the context of the whole dataset, traditional pattern-based inductive classifiers rely excessively on frequency-based measures for identifying significant patterns, and on global thresholds. Similarly, existing semi-supervised classification algorithms excessively depend on the underlying flat clustering algorithm, which requires the number of clusters to be known in advance. Finally, these algorithms ignore the significance of pattern-lengths.

1.1.3 Pattern Frequency Counting

Calculating itemset support (or frequency counting) is a fundamental operation that directly affects space and time requirements of many widely used data mining algorithms. For example, frequent itemset mining and association rule mining are often only concerned with identifying the support and confidence of a given query itemset, while pattern-based clustering algorithms must in addition identify the transactions that contain the query itemset.

First generation data mining algorithms used the trie data structure to improve the itemset support counting performance. In the following years, a number of improvements [9, 3] were proposed to further optimize support counting using tries. These approaches, however, did not address the major drawback of overwhelming space requirements, possibly exponential in depth [86]. With FP Trees, Han et al. [35] eliminated the need to insert each transaction into all paths corresponding to the subsets of the transaction by first preparing a global *F-List* [35] (i.e., a list that contains items in the dataset in their frequency descending order), and then inserting transactions to the trie in the order of this list. In another approach, Yang et al. [86] first generated a binary trie that limits the branching factor to 2 by considering presence or absence of all items in each transaction. All degree-1 nodes in the trie are later merged with their children to obtain a Compressed Patricia Trie.

However, using trie-based structures for frequency counting can be problematic for two reasons. First, the time needed to execute frequency queries may greatly vary on datasets with different characteristics. These queries may execute faster on datasets that have many transactions sharing common prefixes. On the other hand, the frequency queries may take significantly longer to execute on datasets that do not have many transactions sharing common prefixes because there are more upward paths to consider. Second, trie structures do not directly identify the transactions that contain the query itemset. One solution, proposed by Yu et al. in a hierarchical clustering algorithm [88] is to store a list containing the applicable transaction IDs at each node of the trie. This approach may work for small datasets but is clearly impractical for large datasets because of its significant space requirements. In the

worst case (i.e., where each transaction contains each item), IDs of all transactions are replicated at each node in the trie.

As an alternative to trie-based representations, a number of recent approaches [56, 39, 57, 79, 11, 54] represented the dataset as a set of uncompressed bitmaps (i.e., vertical bit vectors). In these approaches, a bitmap is generated for each item in the dataset, where each bit represents presence or absence of the item in a transaction. Itemset frequency is calculated by ANDing bitmaps of all items in the itemset, and counting the number of one-bits in the resulting bitmap. Transactions containing the query itemset are also readily available in the resulting bitmap. However, uncompressed vertical bit vectors may still have high space requirements; for a dataset containing n transactions and m items, the amount of space needed is always m x n bits.

1.2 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 analyzes deficiencies with traditional association rule mining pattern significance measures and presents our hypergraph-partitioning-based web image clustering algorithm, which utilizes both textual and visual features. Chapter 3 introduces closed interesting itemsets, presents the GPHC algorithm, and evaluates its performance. Chapter 4 analyzes issues with global pattern mining, and presents and evaluates our instance-driven hierarchical clustering algorithm. Chapter 5 introduces and evaluates our semi-supervised classification algorithm that uses a pattern-based cluster hierarchy as a direct means for classification. In Chapter 6, we provide an empirical comparative analysis of

various trie and bitmap-based structures for frequency counting. We present HDO, a Hamming-distance-based transaction reordering scheme and aHDO, a linear-time approximation to HDO, and show that these algorithms outperform existing reordering schemes and increase the compressibility of bitmap indices. Finally, we conclude and present our ideas for future work in Chapter 7.

2. Clustering Web Images using Association Rules, Interestingness Measures, and Hypergraph Partitions

In this chapter, we consider the problem of unsupervised clustering of web images (i.e., images augmented with text extracted from their referring web pages). Given *n* images, each with a (possibly empty) text string describing the image, our goal is to find a (flat) clustering of the images that is as close as possible to a manually derived clustering according to the tree-distance evaluation measure (Section 2.7), while minimizing the space and time requirements of the algorithm.

2.1 Visual and Textual Mining

The last two decades have seen significant research in the field of data mining, resulting in a number of successful techniques such as finding associations between data items by mining association rules [1]. These techniques have proven to be useful in symbolic and structured domains such as market basket analysis. Limited research, however, has been conducted to apply these techniques on unstructured, signal-based domains like images [31, 60, 71].

Unlike structured data where features such as keywords and alphanumeric values can be easily identified and extracted, images contain implicit features and patterns that are not straightforward to identify and extract [38]. The fundamental challenge in

image mining is to determine how low-level pixel representations can be efficiently and effectively processed to identify these high-level patterns [38]. Once identified, these patterns could be used in a variety of methods.

Clustering is one such method that uses features to organize data in a number of groups called clusters. Two major approaches exist to cluster images: content-based and text-based. Content-based clustering is normally used by the image analysis and computer vision communities and focuses on exploiting low-level signal features like color, shape, and texture to cluster images, while text-based clustering is normally used by the web mining and information retrieval communities. A common perception exists in web and information retrieval communities [36] that content-based features are computationally expensive to extract and hence infeasible for the web domain. However, some features such as color and orientation can be extracted in time that is linear in the number of pixels in the image. Furthermore, applying simple techniques like image scaling can further reduce computational requirements.

In contrast, the availability of reasonable textual information is not always guaranteed. A large number of images on the web either do not have any textual information associated with them, or the associated textual information does not provide much information about the image (i.e., insufficient to disambiguate from other images that belong to different semantic categories but share some keywords). Text-only clustering techniques are very likely to assign such images to wrong clusters, resulting in low-quality clustering. Similarly, unless sophisticated and computationally intensive techniques are used to capture semantics, signal-only clustering techniques are also likely to produce low quality clusters. We show that

using a combination of textual and simple signal features results in better clustering as compared to clustering solely based on either textual or signal features.

2.2 A Novel Approach

Hypergraphs have proven useful in data mining and high-dimensional document clustering problems [33, 34]. In a typical hypergraph, each vertex represents a dimension and each hyperedge represents an affinity (or relationship) between two or more dimensions represented by the corresponding vertices. Weights assigned to vertices indicate importance of these vertices and weights assigned to hyperedges indicate the strength of the relationship between dimensions represented by the vertices connected by a hyperedge. In this chapter, we first extract signal and text features from images, calculate their frequencies, and apply well-known dimensionality reduction techniques such as stemming, stop word elimination, and Zipf's law to prune non-interesting features. The remaining features are used to generate association rules.

Similar to [34], we use features as hypergraph vertices and all association rules between a set of vertices to generate hyperedges. In the last decade or so, various researchers questioned the usefulness of support and confidence as association rule interestingness measures and have proposed alternatives [10, 51]. Unfortunately, researchers comparing interestingness measures [27, 76] do not agree on any single domain-independent objective measure. Considering this, we compared the effectiveness of twenty-two objective interestingness measures to assign weights to hyperedges in the context of web image clustering, rather than using confidence [34].

Once the association rule hypergraph is available, we apply a widely used hypergraph partitioning algorithm called hMETIS [41] to obtain partitions (or clusters) of features. Images are assigned to these clusters using a simple scoring function. This clustering method eliminates the need of calculating distances from (or similarities to) other images. Finally, we use a tree-distance-based evaluation measure to evaluate the quality of the resulting image clusters with respect to manually generated ground truth.

Most of the steps in this approach, including feature extraction, reduction, rule generation, feature hypergraph generation, and hypergraph partitioning can be performed offline. Assignment of images to clusters is the only real-time step, which is computationally inexpensive.

2.3 Related Work

We now discuss the literature related to web image clustering, association rule mining, interestingness measures for association rules, mining association rules from images, and clustering by graph partitioning.

2.3.1 Web Image Clustering

There have been several web image clustering and categorization approaches proposed in recent years. We discuss only a few representative approaches here. Lienhart and Hartmann [49] use signal-only features to categorize web images. Images are divided into photo-like images, and graphical images. Photo-like images are further divided into photos and artificial photo-like images; graphical images are

further divided into slides, cartoons, and other images. This approach produces coarse categories containing too many images. Although syntactically meaningful, the resulting clusters are likely to contain images that are not semantically related. In contrast, ImageSeer [36] uses the VIPS algorithm [12] to segment web pages into several semantic blocks. These blocks are further used to extract surrounding text of web images. Page-to-block, block-to-image, and block-to-page relationships are obtained using the link structure and page layout analysis, and an image graph is constructed. Techniques from spectral graph theory and Markov chain theory are applied for image ranking, clustering, and embedding. Like any other text-only approach, this approach is likely to assign images with insufficient textual information to incorrect clusters.

2.3.2 Association Rules and their Interestingness Measures

The problem of mining association rules was first introduced in [1]. If $I = \{i_1, i_2, ..., i_m\}$ is a set of literals, called *items*, and D is a set of transactions, an association rule is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = 0$. The rule $X \rightarrow Y$ holds in the transaction set D with *confidence* c if 100c% of transactions in D that contain X also contains Y. The rule $X \rightarrow Y$ has *support* s in the transaction set D if 100s% of transactions in D contain $X \cup Y$. Although a number of algorithms have been proposed improving various aspects of association rule mining [9, 28, 29], Apriori [2] remains the most commonly used algorithm.

One of the most significant problems with association rules mining is that it often results in too many rules [77], especially when attributes (i.e., features) in a data set

are highly correlated [51]. On one of our small datasets containing 295 images, several of which had both signal and textual features associated, we extracted over 1.5 million rules when *minsup* and *minconf* were set to 0.02 (or 2%) each and rules were limited to at most two features on the left hand side and one feature on the right hand side. Increasing the support threshold significantly reduces the number of rules discovered, but risks losing useful associations. In addition, it is non-trivial to set good values for the support and confidence thresholds; they depend on the size of the dataset, the sparseness of data, and the particular problem under study [10]. Considering these issues, a number of researchers have proposed alternative interestingness measures to evaluate and rank discovered associations. These measures are generally divided into subjective and objective interestingness measures. Brijs et al. [10] provide an overview of a number of symmetric objective interestingness measures, five of which are *Lift* (or *Interest*), *Chi-Square*, *Correlation Coefficient*, *Log linear analysis* and *Empirical Bayes correction*. Shekar et al. [73] proposed three measures for capturing relatedness between item pairs. Based on the *Chi-Square* test, Liu et al. [51] introduce the concept of direction-setting and non-direction-setting rules for summarizing association rules. In a follow up paper [52], they propose a subjective approach that assists the user in finding interesting rules. Hilderman and Hamilton [27] survey various objective and subjective interestingness measures for classification rules, association rules, and generalized relations. Tan et al. [76] discuss the properties of twenty-one objective interestingness measures and analyze the impacts of support-based pruning and contingency table standardization.

2.3.3 Mining Association Rules from Images

Utilizing object generation capabilities of UC Berkeley's BlobWorld content-based image retrieval system [6, 13], Ordonez and Omiecinski [60] proposed an algorithm to extract association rules from images. The BlobWorld system represents an image as a collection of Blobs (i.e., regions). In order to generate association rules, objects extracted by BlobWorld are considered analogous to items and images are considered analogous to transactions. Candidate itemsets are generated from the set of objects, and the support is calculated by checking individual images for presence or absence of objects. This information is further used to calculate confidence. This approach works well on a small set of images containing a small number of simple geometric objects, but may not be suitable for images containing a large number of complex objects because of the high computational costs of the object identification step in [60], which compares all identified objects in the system to all blobs in each image. Haddad and Mulhem [31] proposed a more efficient approach that considers both manual textual annotations and signal features like dominant colors, directions, and texture indicators to generate association rules from images. Images are first segmented into regions based on their spatial connectivity and visual similarity. Principal color, secondary color, principal direction, and texture features are computed for regions, and annotations are added manually using a list of predefined terms. Finally, association rules are generated using regions as transactions and region features as items. Because of its inherent dependence on human intervention for term identification and region annotation, this approach may not be suitable for clustering web images.

2.3.4 Clustering Based on Hypergraph Partitioning

Based on the observation that using association rules directly for clustering may result in clusters that are too granular, Han et al. [34] proposed an approach to cluster transactions using association rule hypergraphs. A *hypergraph* is similar to a graph except that each edge, called a *hyperedge*, can connect two or more vertices. In order to generate a hypergraph from a set of association rules, each unique item that exists in the set is assigned to a unique vertex in the graph. All rules containing a set of items would generate a hyperedge, with average confidence of such rules used as the weight. For example, if {A} → {B, C} and {C} → {A, B} are all possible rules between items A, B, and C with confidences 0.6 and 0.4 respectively, there would be a hyperedge between A, B, and C with a weight of 0.5. The hMETIS [41] hypergraph-partitioning algorithm, which is widely used in the VLSI domain, is used to partition this hypergraph. Transactions are assigned to these partitions using a simple scoring function (Section 2.7) resulting in clusters of transactions.

The hMETIS algorithm [41] is based on the multilevel paradigm, where a sequence of successively coarser hypergraphs is constructed. A bisection of the coarsest hypergraph is computed and it is used to obtain a bisection of the original hypergraph by successively projecting and refining the bisection to the next level finer hypergraph. hMETIS achieves this in three phases. During the coarsening phase, the size of the graph is successively decreased; during the initial partitioning phase, a bisection of the smaller graph is computed; and during the uncoarsening and

refinement phase, the bisection is successively refined as it is projected to the larger graphs.

2.4 Mining Association Rules

We now discuss our approach of mining association rules from web images, including preprocessing and feature extraction.

2.4.1 Data Gathering and Preprocessing

Over 3000 images were crawled from the Internet and saved to local disk, along with referring web pages, preserving the links. These images were divided into two separate datasets, called Dataset1 and Dataset2, and the same set of steps was performed on each dataset.

A hash table was generated using references to images as keys and the lists of their referring web pages as values. All the HTML tags and formatting commands (e.g., " ") were stripped out from the web pages, and stop words were eliminated using the standard list of 571 stop words initially designed for the SMART system [70]. The remaining terms were stemmed using the Paice stemmer [61]. Next, the unique terms from all referring web pages were added as textual features for each image, i.e., the image-word vector was binary valued. Images were scaled to an empirically selected size of 168 x 168 maintaining their aspect ratios, enabling faster processing times for the signal feature extraction phase.

2.4.2 Feature Extraction

In addition to the terms extracted from referring web pages, image file names were processed to extract keywords. Terms separated using standard delimiters like space, underscore, and hyphen were isolated and further parsed for potential words, taking case changes and appearance of numbers into account. The resulting keywords were stemmed, checked against the stop words list, and added to the list of textual features associated with the image. The final set of textual features associated with an image then contains both the keywords extracted from the referring web pages and the keywords extracted from its file name.

HSV color histograms were computed and used to identify the two most dominant colors. In order to calculate significant orientations, horizontal and vertical Sobel filters were applied to the image. The resulting values were used to generate a 2D histogram of gradients. Small image gradients were eliminated and the remaining ones were quantized to acquire a coarse representation of the four most significant orientations. The image was then checked for the presence of two major orientations, by comparing the magnitude of the two most significant orientations against the third orientation. If the first two orientations were found to be close to each other but significantly apart from the third orientation, the image were assumed to contain grid like objects; an extra feature indicating this finding was added for such images. The resulting binary color and orientation features were added as image signal features in a textual form (i.e., Color = BLUE) prefixed as "@SIGNAL_" to avoid potential conflicts with textual features. Color names were assigned to HSV ranges in a way similar to [53], except that we have dealt with relatively fewer colors. Figure 1 shows

an image and the set of extracted signal and textual features. Note that the running time of our signal feature extraction phase was linear in the number of pixels in the image.

A web image:

Textual features extracted from the referring web page(s):
tiger panthera tigris tiger panthera tigris there are five subspecies of tigers and three more have become extinct in recent decades tigers are the largest cats and the only ones besides the jaguar that like water they are distributed over east asia tigers usually make a kill every three days and if they don't eat within ten days chances are they will be too weak to hunt and they will die

Textual features extracted from the image file name:
tiger

Signal features:
@SIGNAL_ORIENTATION=1
@SIGNAL_COLOR=Black
@SIGNAL_COLOR=Brown
@SIGNAL_ORIENTATION=3

Final set of features with textual features stemmed, and duplicates and stop words eliminated:
@SIGNAL_ORIENTATION=1 hunt die subspecy as @SIGNAL_COLOR=Black distribut kil tig chant extinct wat weak @SIGNAL_COLOR=Brown eat jagu ten east day mak panther tigr cat rec decad largest @SIGNAL_ORIENTATION=3

Figure 1: Features extracted from one of the 3364 images crawled from the web, found at http://cogsci.indiana.edu/farg/harry/bio/zoo/tiger.htm

2.4.3 Rule Generation

In terms of classical association rule terminology, images were considered as transactions, and textual and signal features were considered as items. An algorithm similar to Apriori-TID [2] was used to generate association rules. Frequent itemsets were computed by checking for the presence or absence of features in images.

Table 1. A few association rules generated from images of cars and animals along with their support. Note the stemming of "Jaguar", "Horse" and "Animal"

Rule	***Support***
{@SIGNAL_COLOR=Brown}→{suv}	0.027118
{@SIGNAL_COLOR=Brown}→{wild}	0.030508
{model}→{car}	0.183050
{jagu}→{turbo}	0.030508
{import}→{hors}	0.027118
{anim}→{@SIGNAL_ORIENTATION=1}	0.054237
{livestock}→ {@SIGNAL_ORIENTATION=1}	0.040677
{@SIGNAL_ORIENTATION=1}→ {@SIGNAL_COLOR=Pink}	0.020338

Zipf's law states that items that occur too frequently or very infrequently are not significant, and this has been proven as a useful feature reduction technique in the context of classifying hidden-web databases [30]. Checking for the support threshold essentially eliminates infrequent items, and stop word elimination eliminates some of the most frequent items. We applied an additional feature reduction step on frequent 1-itemsets (i.e., an itemset containing a single item) and eliminated items with very high support (greater than 0.9). Once generated, rules were written to a file along with their support, confidence, and additional information required to calculate the values of various interestingness measures discussed in the next section. Table 1 shows a few rules extracted from Dataset1.

2.5 Generating Hypergraphs

A unique vertex was generated from each unique item that existed in the final set of extracted association rules. A hyperedge was generated between a set of vertices if there was at least one association rule containing exactly the features that existed in the set. As an example, three hyperedges were generated for the following set of four rules:

$$\{\text{Color = YELLOW}\} \rightarrow \{\text{bart}\} \quad \text{supp} = 0.2, \text{conf} = 0.4$$

$$\{\text{bart}\} \rightarrow \{\text{Color = YELLOW}\} \quad \text{supp} = 0.2, \text{conf} = 0.8$$

$$\{\text{Color = YELLOW}\} \rightarrow \{\text{lisa}\} \quad \text{supp} = 0.25, \text{conf} = 0.3$$

$$\{\text{bart}\} \rightarrow \{\text{lisa}\} \quad \text{supp} = 0.1, \text{conf} = 0.5$$

The first hyperedge was generated between vertices labeled as ‘Color = YELLOW’ and ‘bart’, the second hyperedge was generated between vertices labeled as ‘Color = YELLOW’ and ‘lisa’, and the third hyperedge was generated between vertices labeled as ‘bart’ and ‘lisa’.

In order to assign weights to these hyperedges, we used one of our set of 22 interestingness measures, taking averages if more than one rule participated in the hyperedge. For example, as in [34], using average “confidence” of all rules covered by the first hyperedge results in a weight of 0.6 (i.e., the average of 0.4 and 0.8). Similarly, the second and third hyperedges will have 0.3 and 0.5 assigned as weights, respectively.

2.6 Interestingness Measures

As discussed above, support and confidence are widely criticized as interestingness measures for association rules. For uneven datasets (i.e., datasets with a high degree of class imbalance), a high support threshold results in pruning useful associations between items [76] that are not present in a large number of transactions, and a low support threshold results in too many rules. Figure 2 shows the number of rules we have obtained on our smallest dataset of 295 images for various support levels. Note the non-linearity of the x-axis.

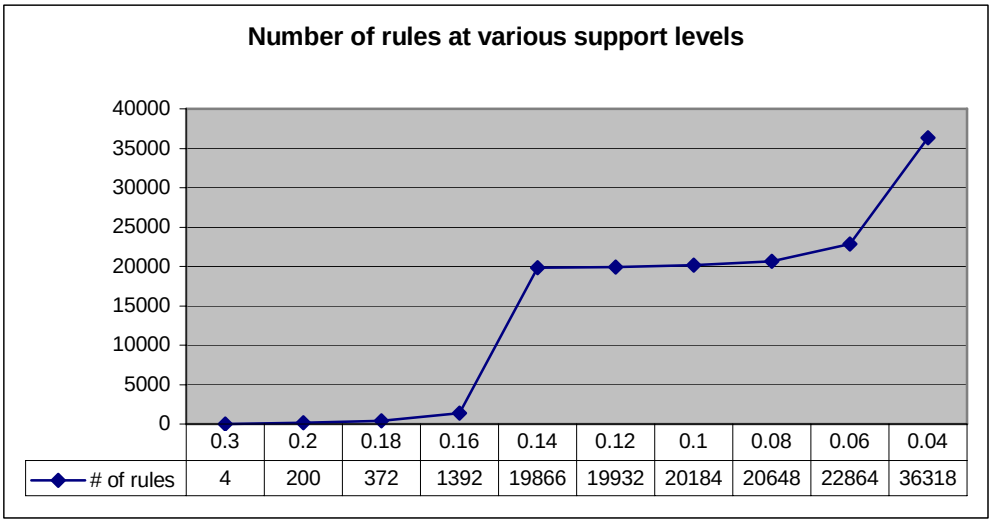


Figure 2. Number of rules generated at various support levels

Nonetheless, [76, 77] show that if the support threshold is set to a very low value, rules that are pruned contain items that are either uncorrelated or negatively correlated, and thus of no value. We use this property to reduce the initial number of rules obtained from our datasets.

On the other hand, confidence is criticized because of its asymmetry and its failure to incorporate the baseline frequency of the consequent [8].

Therefore, we experimented using various statistically inspired interestingness measures as functions to assign weights to hyperedges. Table 2 lists all such measures. See Appendix B for computational details of these measures.

Some of these measures offer properties that can be used to distinguish significant rules from insignificant rules. We used these properties to identify and prune insignificant rules. As an example, Correlation Coefficient and Certainty Factor range between –1 and +1 with a value of 0 indicating independence and negative and positive values indicating negative and positive correlation (or in the case of Certainty Factor, dependence) respectively. We used this property to prune all rules having negative or no correlation and kept only the rules containing positively correlated features. The third column of Table 2 presents the pruning thresholds used in this chapter.

Table 2. List of interestingness measures used

#	*Symbol*	*Interestingness Measure*	*Pruning Threshold*
1	AV	Added Value	<= 0
2	F	Certainty Factor	<= 0
3	χ^2	Chi Square	none
4	S	Collective Strength	none
5	c	Confidence	<= 0.2
6	V	Conviction	none
7	Φ	Correlation Coefficient	<= 0
8	IS	Cosine	none
9	G	Gini Index	<= 0
10	I	Interest	<= 1
11	ζ	Jaccard	none
12	J	J-Measure	none
13	κ	Kappa	<= 0
14	K	Klosgen's	<= 0
15	L	Laplace	none
16	mc	Max Confidence	<= 0.2
17	M	Mutual Information	none
18	α	Odds Ratio	none
19	RI	Piatetsky-Shapiro's Interest	<= 0
20	s	Support	none
21	Q	Yule's Q	<= 0
22	Y	Yule's Y	<= 0

2.7 Clustering via Partitioning

In our preliminary experiments, we used a widely used hypergraph partitioning algorithm, called hMETIS [41], to partition the feature hypergraph. hMETIS produces "balanced k-way" partitions where k, the number of partitions, is specified in advance. For this chapter, we set the number of partitions based on ground truth.

Once features were partitioned, images were clustered by calculating a score of each image against each partition, based on the features in the partition and the features in the image. All images were assigned to partitions with their highest score, with ties broken arbitrarily. A simple function was used to calculate this score:

$$S = \frac{|I \cap P_i|}{|P_i|}$$

where I is the set of image features and P_i is the set of features in cluster i.

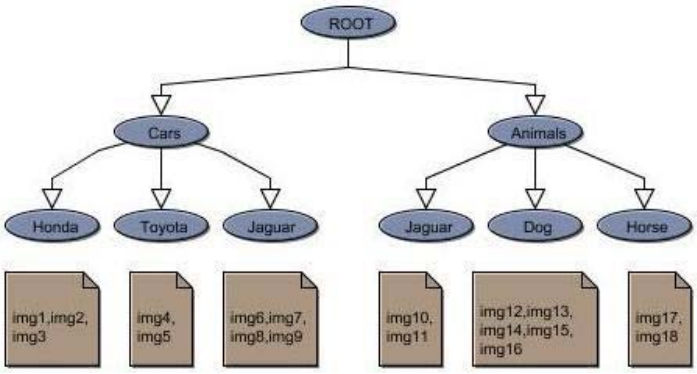


Figure 3. Eighteen images assigned to the ground truth hierarchy

The main goal of clustering is to organize data in clusters so that intra-cluster similarity is maximized and inter-cluster similarity is minimized [34]. We developed a tree-distance-based evaluation measure to evaluate the overall clustering quality, comparing the clustered images with a ground truth hierarchy of image clusters. Each image is evaluated in three ways, and then these individual image scores are summed.

The first way assigns a score of '*p*' to the image for every other image in its cluster that appears in the same ground truth cluster that is the image's ground truth cluster. The second way deducts a score of '*n*' from the image for each image in its cluster that appears in a sibling ground truth cluster of the image's ground truth cluster. The third way deducts a score of '*z*' for any image in its cluster that does not meet the first two conditions (for example, the image appears in a cousin ground truth cluster rather than a sibling).

As an example, Figure 3 presents a hierarchy with 18 images assigned to various root nodes based on ground truth. Suppose a clustering algorithm generates the following six clusters:

Cluster1: img5, img9

Cluster2: img2, img10, img1, img8

Cluster3: img3, img12

Cluster4: img14, img18, img17

Cluster5: img11, img16, img13

Cluster6: img4, img6, img7, img15

We compute the score of cluster 2 as follows:

Using $p = z = 1$ and $n = 0$:

img2: (-1 for img10) + (+1 for img1) + (-0 for img8) = 0

img10: (-1 for img2) + (-1 for img1) + (-1 for img8) = -3

img1: (+1 for img2) + (-1 for img10) + (-0 for img8) = 0

img8: (-0 for img2) + (-1 for img10) + (-0 for img1) = -1

Cluster2: 0 – 3 + 0 – 1 = -4

Scores for other clusters can be calculated in a similar fashion. The overall clustering score is then computed by adding total scores for all clusters. In order to compare clustering quality across datasets of different sizes, max and min bounds for the raw score can be calculated using the ground truth hierarchy, and these extremes can be used to normalize the raw score. We used this technique in this chapter; cluster fit therefore is in a range of [0, 1].

Graph partitioning is an NP-hard problem. Efficient partitioning algorithms such as hMETIS [41] use various randomized heuristics to achieve the desired level of performance. A major drawback of this approach is that multiple executions of the algorithm on the same hypergraph using the same parameters often result in different partitions. As suggested in [42], we executed hMETIS ten times on each feature hypergraph (in part because of the lack of perfect balance in our datasets) and picked the partition with highest overall clustering score.

2.8 Experimental Results

A dataset containing 295 images of cars and animals with a ground truth hierarchy as shown in Figure 3 was used for initial experiments. In the ground truth, the smallest cluster had 7 images and the largest cluster had 80 images. A second dataset containing 3069 images of animals and cartoons was used to validate our results. The smallest cluster in this dataset contained 100 images and the largest cluster contained 1970 images. Both datasets included some categories that could challenge any clustering algorithm because of inherent ambiguity, for example, Jaguar cars and

Jaguar animals in the first dataset, and images of ducks and Donald Duck in the second dataset.

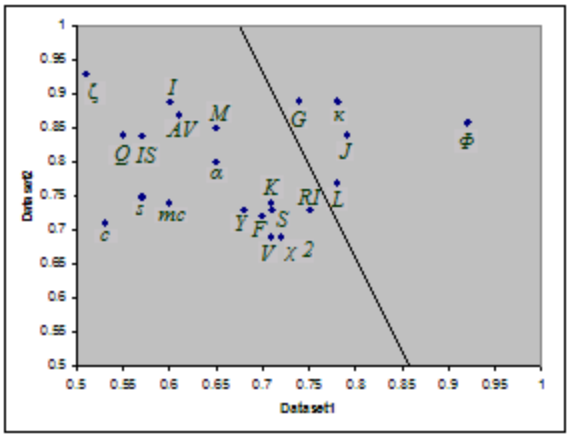


Figure 4. Comparison of clustering quality of various measures across both datasets (the hand-drawn line separates measures succeeding on both datasets from other measures)

Additionally, for computational efficiency purposes, these first experiments were performed using rules that contain one item on the left and one item on the right (see Section 2.8.2 for experiments with more than 2 features). Figure 4 graphs the overall clustering quality of various interestingness measures on each of these datasets. Clearly, support and confidence are among the worst 10 performers on both datasets. Max confidence, a symmetric version of confidence, outperformed confidence on both datasets, which adds credence to the claim that the asymmetric property of confidence is not as useful in the web image domain.

Although individual measures do not appear to be strongly correlated across the two datasets, Correlation Coefficient, Kappa, J-Measure, and Gini Index perform consistently well. However Jaccard poses a surprising problem. We suspect that this is due to the imbalance of cluster sizes in the second dataset.

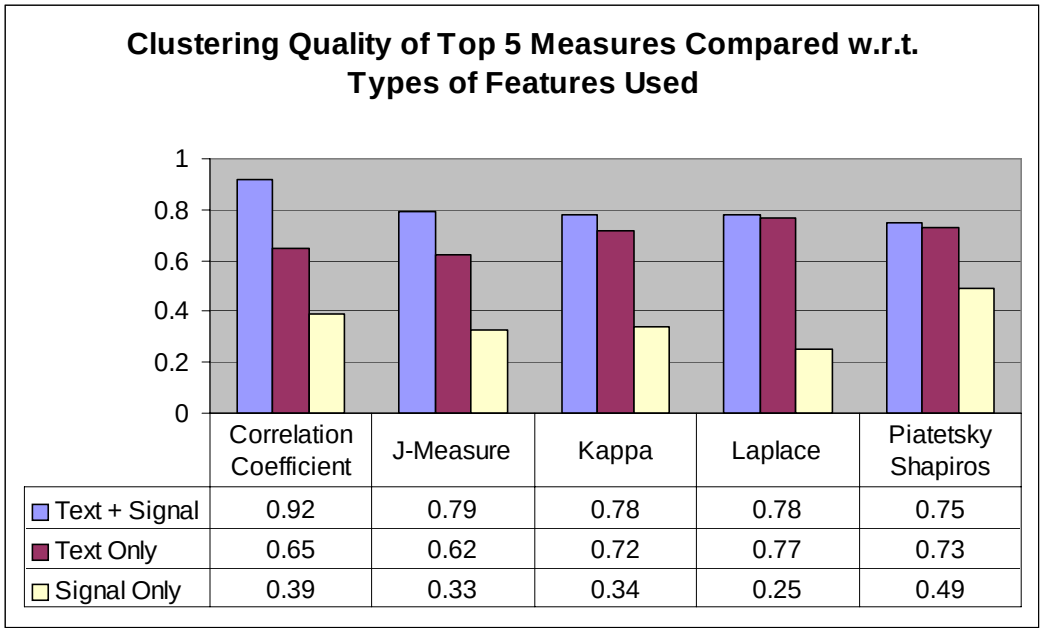


Figure 5. Clustering quality comparison on Dataset1 using text-only, signal-only, and both features

Figure 5 compares the clustering quality of the top 5 measures on the first dataset when signal-only, text-only, or both kinds of features are used. Signal-only techniques performed worst in terms of clustering quality. For example, Figure 6 presents a small signal-only cluster generated using Correlation Coefficient on Dataset1. Although all seven images in this cluster look visually similar, they belong to four different semantic categories.

Figure 6. A small cluster generated from the first dataset using signal-only features and Correlation Coefficient as interestingness measure

Figure 5 also shows that combining textual and signal features provide improvement over clustering using text-only features. Figure 7 presents portion of a cluster generated using text-only features and Correlation Coefficient on Dataset1. While most of the images may have the keyword ‘Jaguar’ associated with them, they lacked further information that could have helped separate animals from cars. When signal features were added, the same clustering technique using the same interestingness measure was able to isolate animals and cars in two separate clusters, achieving a much higher level of clustering quality, as shown in Figure 8.

Figure 7. Portion of a cluster from the first dataset using text-only features and Correlation Coefficient

2.8.1 Cross Validation

To validate our findings, leave-n-out cross validation was applied on the first dataset, using two of the top 5 measures as shown in Figure 5 and both signal and textual features used to generate rules. ‘n’ was set to 10, which resulted in 29 unique sets of randomly selected images. 29 experiments were performed for each of the two measures and one of the image sets was left out in each experiment. The remaining

images were used to generate rules, and all images from the original dataset (images used to generate rules, as well as images that were left out) were clustered using the hypergraph partitions obtained. Experiments performed using Kappa resulted in an average clustering quality of 0.75, with max = 0.85, min = 0.70 and standard deviation = 0.72 whereas experiments performed using J-Measure resulted in an average clustering quality of 0.73 with max = 0.79, min = 0.69 and standard deviation = 0.70, validating our initial results.

Figure 8. Portions of two clusters from the first dataset using combined textual and signal features and Correlation Coefficient

2.8.2 Rules with More Than Two Features

We performed preliminary experiments to find if hypergraph partitions generated using higher order rules would result in better clustering, as compared to rules that contain only one item on the left and one item on the right. Using a relatively higher support threshold on the first dataset, we generated two sets of rules. The first set contained rules with one item on the left and one item on the right and the second set contained rules with two items on the left and one item on the right. All of the top 5 measures were used to generate hypergraphs that were further used to cluster all images in the dataset. We observed that hypergraphs based on rules containing two items on the left results in an average clustering quality improvement of 19% across all measures, as compared to clustering obtained using hypergraphs containing one item on the left and one item on the right. Specifically, Laplace gained the most and Piatetsky-Shapiro's Rule Interest gained the least improvement.

2.9 Conclusions

We conclude that using statistically inspired, objective interestingness measures to assign weights to hyperedges may result in better clustering as compared to using traditional measures such as support and confidence. We also conclude that combining textual and signal based features result in better clustering as compared to signal-only or text-only approaches.

3. High Quality, Efficient Hierarchical Document Clustering using Closed Interesting Itemsets

In this chapter, we consider the problem of unsupervised hierarchical clustering of text documents. Given *n* documents, the goal is to produce a cluster hierarchy of the documents that maximizes the FScore and minimizes the entropy, allowing each document to exist in multiple clusters.

Organizing data into a tree-like hierarchy has many applications. A hierarchy provides a view of the data at different levels of abstraction, helping users deal with the common problem of information overload. With an interactive browser for the hierarchy, the user expands nodes at different levels in the hierarchy, revealing the structure within the broad topic where parent and child nodes are organized in a general to specific fashion. These benefits make hierarchies a logical choice to organize large collections of documents and during last few decades, various approaches were proposed to produce cluster hierarchies from document collections.

Agglomerative and partitioning-based approaches represent the two most popular categories of hierarchical document clustering techniques [91]. Agglomerative approaches start with a singleton cluster for each document and build the hierarchy bottom-up by applying various pair-wise similarity measures on clusters, and merging the cluster pair with highest similarity at each step, until only one cluster remains.

Agglomerative methods generally suffer from their inability to perform adjustments once a merge is performed, resulting in lower clustering accuracy. They also have a high computational cost [91], making them infeasible for large document datasets. On the other hand, partitioning approaches obtain hierarchical clustering solutions via a sequence of repeated bisections [91] and are generally scalable and efficient. Steinbach et al. [75] showed that for document clustering, Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [43] and bisecting *k*-means, a variant of standard *k*-means are the most accurate agglomerative and partitioning methods, respectively [90]. Furthermore, Zhao and Karypis [91] recently showed that the I_2 criterion function outperforms other criterion functions when used with bisecting *k*-means.

A recent trend in hierarchical document clustering is to use frequent itemsets to produce cluster hierarchies. HFTC [5] was the first algorithm in this class and achieves accuracy comparable to 9-secting *k*-means, and worst than bisecting *k*-means. Fung et al. [25] showed that HFTC is not scalable for large document collections and proposed FIHC; a frequent itemset based clustering approach, which they claim outperforms HFTC and the best-known agglomerative and partitioning-based methods (i.e., UPGMA and bisecting *k*-means) both in terms of accuracy and scalability. More recently, Yu et al. [88] proposed TDC, an algorithm that uses only closed frequent itemsets and further reduces dimensionality, while improving document clustering quality and scalability over FIHC. To our surprise, our comparison (Section 3.4.6) revealed that both FIHC and TDC actually perform worse than UPGMA and bisecting *k*-means for clustering documents in our datasets.

Based on the observation that higher frequency does not necessarily mean higher quality, and combining ideas from research in selecting the most interesting association rules, and closed frequent itemset mining, we introduce the notion of closed interesting itemsets in this chapter. A closed interesting itemset is an itemset that satisfies a minimum interestingness threshold, and has support that is different from any of its subset itemsets. We provide a simple, parallelizable algorithm, and necessary heuristics to efficiently mine these itemsets. We present results from extensive experiments performed on standard datasets of varying characteristics and sizes, and show that using the same support threshold for first level (single word) itemsets results in significantly smaller number of closed interesting itemsets as compared to the number of closed frequent itemsets generated. Even so, when used for hierarchical document clustering, we show that a method based on closed interesting itemsets outperforms state-of-the-art clustering algorithms in terms of both clustering quality and runtime performance.

We present GPHC (i.e., Global Pattern-based Hierarchical Clustering), a hierarchical clustering algorithm that uses globally significant closed interesting itemsets to generate a cluster hierarchy. GPHC adopts a hierarchy assembling approach that supports soft clustering and prunes unwanted itemsets along the way. In order to make the hierarchy more compact, existing approaches [25, 88] use agglomerative clustering to merge the first-level nodes. Although significantly less expensive than applying agglomerative clustering on the whole dataset, this step is still expensive. We used bisecting *k*-means to reduce the computational complexity of this step. Finally, we propose various implementation-level optimizations throughout

the chapter. Figure 9 provides an overview of the hierarchical document clustering process followed by our GPHC algorithm.

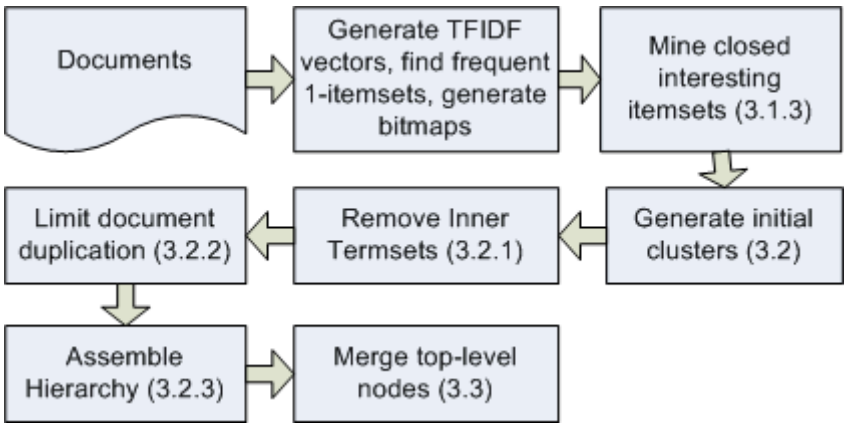


Figure 9. Our hierarchical document clustering process; numbers refer to sections in this thesis

3.1 Mining Closed Interesting Itemsets

We now discuss Closed Interesting Itemsets in detail, and provide a simple algorithm for mining these itemsets.

3.1.1 Motivation

Frequent itemset mining often results in too many itemsets. Using a faster mining algorithm does not always help as it is fundamentally a combinatorial problem and the mining time exponentially increases as support threshold linearly decreases [88, 35], regardless of the mining algorithm used. Researchers found that most of the frequent itemsets found in real life datasets share support with one or more of their parent (subset) itemsets. These itemsets are considered insignificant as they represent specializations of the more general concept represented by the parent itemset. Closed frequent itemset mining utilizes this finding and imposes the additional requirement

of closure for frequent itemset generation. Specifically, in addition of meeting the minimum support threshold, closed frequent itemsets must also have support that is different from (practically less than) any of their subset itemsets. Generally, the same support threshold results in significantly fewer closed frequent itemsets than frequent itemsets on the same dataset. In addition, closed frequent itemsets are more useful than frequent itemsets in a number of applications, such as hierarchical document clustering [88].

Finding the most interesting association rules is another significant thread in data mining research. A number of association rules can be generated from each frequent itemset at each level, which often results in a large association rule base [77], especially when attributes in the data set are highly correlated [51]. A low support threshold results in too many discovered associations. Increasing the support threshold significantly reduces the number of rules discovered, but risks losing useful associations, especially on uneven datasets. On the other hand, confidence is criticized because of its asymmetry and its failure to incorporate the baseline frequency of the consequent [8]. In addition, it is non-trivial to set good values for support and confidence thresholds; it depends on the size of dataset, sparseness of data, and the particular problem under study [10]. Considering these issues, a number of researchers [10, 27, 51, 76] proposed alternative interestingness measures to evaluate and rank discovered associations. Inspired from various statistical and mathematical principles, these measures are considered less sensitive to the properties of specific datasets.

3.1.2 Overview of Closed Interesting Itemsets

We argue that while the closure requirement of closed frequent itemsets is useful and based on solid principles, the other requirement of meeting a minimum support threshold is problematic and difficult to generalize. Combining the stronger aspects of closed frequent itemset mining with research in finding the most interesting association rules, we propose a new kind of itemsets called closed interesting itemsets.

Table 3. List of interestingness measures used with their corresponding threshold values

#	*Symbol*	*Interestingness Measure*	*Threshold*
1	AV	Added Value	0.4
2	c	Symmetric Confidence	0.6
3	F	Certainty Factor	0.4
4	χ^2	Chi-Square	$unit = 50$, $p = 3000$
5	S	Collective Strength	1.45
6	V	Conviction	1.7
7	Φ	Correlation Coefficient	0.35
8	IS	Cosine	0.33
9	G	Gini Index	0.017
10	I	Interest	12
11	Z	Jaccard	0.23
12	J	J-Measure	0.02
13	κ	Kappa	0.35
14	K	Klosgen's	0.068
15	L	Laplace	0.6
16	M	Mutual Information	0.1
17	α	Odds Ratio	25
18	RI	Piatetsky-Shapiro's Interest	0.02
19	Q	Yule's Q	0.85
20	Y	Yule's Y	0.65

These itemsets retain the closure property of closed frequent itemsets, but replace the minimum support requirement with meeting the minimum threshold of a symmetric, statistically inspired objective interestingness measure. Table 3 lists the measures used in our experiments and Section 3.4.4 provides details on the threshold values. See Appendix B for computational details of these measures. Some of these measures are not inherently symmetric and are converted to a symmetric version by

calculating the interestingness values for both directions and selecting the maximum value, as proposed by Tan et al. [33]. In Section 3.4, we compare their relative performance and recommend a small number of measures that we found least sensitive to the properties of specific datasets in our experiments.

Furthermore, most of these measures are meant to calculate correlation or interdependence between two-way contingency tables (i.e., two variables), which makes them unusable for generating closed interesting itemsets with more than two items. While measures like log-linear analysis [10] exist to calculate interdependence between multi-way contingency tables, they are computationally expensive. We define a simple greedy heuristic to deal with this problem:

Super item: If an itemset *p* at level *k* is used to generate a candidate itemset *q* at level *k* + 1 (i.e., itemset *q* contains all *k* items from itemset *p* and exactly one additional item *u*), all items in itemset *p* are used to form a super item *S*, with support (*S*) = support (*p*). Items *S* and *u* are used to form a two-way contingency table and to calculate interestingness values.

***Example*:** Considering a dataset of 200 transactions, support (A) = 98, support (B) = 120, support (C) = 65, support (A, B) = 80 and support (A, B, C) = 45. If itemset {A, B} at level 2 is used to generate a candidate itemset {A, B, C} for level 3, a super item *S* is formed with support (*S*) = support (A, B) = 80. Since C is the additional item in the candidate itemset, a contingency table is formed between *S* and C, as shown in Table 4.

Table 4. A 2 x 2 contingency table between super item *S* and item C

	C	¬C	Total
S	45	35	80
¬*S*	20	100	120
Total	65	135	200

Using the contingency table shown in Table 4 and Correlation Coefficient as the interestingness measure, we get an interestingness value of 0.414, which shows that the super item *S* and item C are positively correlated [33].

Similar to frequent itemset mining, we prune candidate itemsets for level *k* if any of their *k* subsets of size *k* - 1 do not exist in the previous level, with a caveat that frequent itemset mining uses support that has a downward closure property, providing theoretical foundation for this step. We empirically found this step to be useful in increasing the quality and reducing the number of closed interesting itemsets generated. We leave the theoretical analysis for future work.

```
1) result = Φ
2) I1 = U1 = {frequent 1-itemsets}
3) for (k = 2; Ik-1 != 0; k ++) do begin
4)     Ik = find-interesting-itemsets(Ik-1, Uk-1)
5)     append(result, Ik)
6)     Uk = get-unique-items(Ik)
7) end
8) answer = result
```

*(a) Algorithm **CII-MINE***

```
1) find-interesting-itemsets(super_items, unique_items)
2)     interesting_itemsets = Φ
3)     for (i = 0; i < size(super_items); i ++) do begin
4)         for (j = index-of(get-last-item(super_items[i]), unique_items) + 1; j < size(unique_items); j ++) do begin
5)             candidate_itemset = super_items[i] U unique_items[j]
6)             subset_itemsets = find-subset-itemsets(candidate_itemset)
7)             if (contains(Ik-1, subset_itemsets)) then
8)                 if (closed(candidate_itemset, subset_itemsets, Ik-1)) then
9)                     val = apply-measure(super_items[i], unique_items[j])
10)                    if (val >= min_interestingness_threshold) then
11)                        append(interesting_itemsets, candidate_itemset);
12)                    end
13)                end
14)            end
15)        end
16)    end
17)    answer = interesting_itemsets
18) end
```

*(b) Method **find-interesting-itemsets***

Figure 10. A simple closed interesting itemset mining algorithm

3.1.3 Itemset Mining

Figure 10 (a) presents CII-MINE, a simple algorithm to mine closed interesting itemsets. The algorithm starts with mining frequent 1-itemsets (individual words) in a way similar to frequent itemset mining. In our experiments, we found that using a very low support threshold for this step results in best quality itemsets (Section 3.4.7), adding credence to the claim that using a high support threshold results in pruning useful associations [33]. In the k^{th} step (where $k \geq 2$), the method forms candidate itemsets by considering all closed interesting itemsets found in the k - 1^{th} step as super items, and adding the unique individual items that follow the last item in the super item. Each candidate is checked for downward closure and closure, and candidates that satisfy both requirements are checked for meeting the interestingness threshold. Candidates that satisfy all three requirements are added to the set of closed interesting itemsets for step k. Mining stops when all closed interesting itemsets can be formed.

Example: If mining closed interesting itemsets with $k = 2$ and frequent 1-itemsets resulted in the closed interesting 2-itemsets (I_2) in Table 5, U_2 = {a, b, c, d, e, f}. Mining closed interesting itemsets for $k = 3$ would be done as represented in Table 6. Note that super items {b, f}, {d, f} and {e, f} are not considered because there are no items following 'f' in U_2.

Table 5. Interesting 2-itemsets and their support

2-itemset	*Support count*	*2-itemset*	*Support count*	*2-itemset*	*Support count*
{a, b}	150	{b, d}	140	{c, e}	200
{a, d}	280	{b, e}	320	{d, f}	94
{b, c}	120	{b, f}	85	{e, f}	10

Since $k = 3$, size of each super item is $k - 1 = 2$. The algorithm first explores all candidate items for super item {a, b}. Since 'b' is the second item in U_2, four

candidates {a, b, c}, {a, b, d}, {a, b, e} and {a, b, f} are formed, using items that follow 'b' in U_2. Each candidate is checked for the meeting the downward closure requirement, and candidates that do not meet it are pruned (i.e., {a, b, c} is pruned as {a, c} does not exist in I_2). Similarly, candidate itemsets that do not meet the closure requirement (i.e., {b, c, e}) are pruned. Interestingness values of the remaining candidates are calculated by calling "apply-measure", and passing super and unique items, (i.e., {{a, b}, d} and {{b, d}, f}). Candidates that satisfy the minimum interestingness threshold are added to the result. Support for candidate itemsets is only calculated if they meet the downward closure requirement. In addition, we optimized the support calculation performance by using bitmaps that indicate presence / absence of individual, frequent 1-items in all documents (i.e., where each bit represents a document) and ANDing the bitmaps of all items in an itemset.

Table 6. Mining closed interesting 3-itemsets, using 2-itemsets from Table 5, (NC = not calculated)

super_item		***Candidate itemset***	***Supp. Count***	***Comments***
a	b	~~{a, b, c}~~	NC	{a, c} not in I_2
a	b	{a, b, d}	52	Calculate interestingness ({a, b}, d)
a	b	~~{a, b, e}~~	NC	{a, e} not in I_2
a	b	~~{a, b, f}~~	NC	{a, f} not in I_2
a	d	~~{a, d, e}~~	NC	{a, e} not in I_2
a	d	~~{a, d, f}~~	NC	{a, f} not in I_2
b	c	~~{b, c, d}~~	NC	{c, d} not in I_2
b	c	~~**{b, c, e}**~~	120	Not closed: same as (b, c)
b	c	~~{b, c, f}~~	NC	{c, f} not in I_2
b	d	~~{b, d, e}~~	NC	{d, e} not in I_2
b	d	{b, d, f}	72	Calculate interestingness ({b, d}, f)
b	e	~~**{b, e, f}**~~	10	Not closed: same as (e, f)
c	e	~~{c, e, f}~~	NC	{c, f} not in I_2

3.2 Hierarchical Document Clustering and Itemset Pruning

Our hierarchy construction approach is similar to FIHC [25] and TDC [88], with various differences, the most significant of which relates to how parent nodes are

selected. An initial cluster is formed for each closed interesting itemset, containing all documents that contain the itemset, with items in the itemset used as the cluster label. In fact, these clusters are readily available as a byproduct of calculating support using the bitmap-based representation discussed in the previous section. These initial clusters are not disjoint, as a document can contain multiple closed interesting itemsets of varying sizes. Sections 3.2.1 and 3.2.2 discuss our approach to limit document duplication. Section 3.2.3 presents our hierarchy construction algorithm. This step significantly differs from existing approaches as it allows selecting multiple parents, using the difference in interestingness between parent and child nodes without inspecting cluster contents.

3.2.1 Inner Termset Removal

If a document is contained in multiple clusters that are based on itemsets of varying sizes, we reduce document duplication by pruning the document from all but the clusters based on the largest sized itemsets. Later, when these itemsets are used to build the hierarchy, this step results in each document assigned to all applicable nodes at the highest possible (i.e., most specific) level in the hierarchy. Figure 11 presents an algorithm that performs this step in a single pass on discovered closed interesting itemsets, without processing individual documents.

```
1)  {allocate array global_map}
2)  {allocate array lev_maps with size = k}
3)  for (i = k; i >= 1; i--) do begin
4)          forall itemsets t ∈ I_i do begin
5)                  bitmap_t = bitmap_t AND (NOT global_map)
6)                  lev_maps [i] = lev_maps [i] OR bitmap_t
7)          end
8)          global_map = global_map OR lev_maps [i]
9)  end
```

Figure 11. Inner-termset removal algorithm, where *k* = size of the largest discovered itemset

The algorithm starts by allocating a global, and individual coverage maps for each level, where the number of levels is the size of largest discovered itemset. A level coverage map is similar to an itemset bitmap except that an itemset bitmap indicates documents that contain the itemset whereas a level coverage (bit) map indicates documents that contain any itemset at that level. Similarly, the global coverage map indicates documents that contain any discovered itemset. Levels are iterated in largest to smallest order and at each level; bitmaps of all itemsets that exist at that level are ANDed with the inverse of the bits in the global coverage map, which results in eliminating documents that already existed at a higher level. The updated bitmap is used to update the current level's coverage map. Finally, after each level, the current level's documents are added to the global coverage map. This results in pruning documents from all but their largest-sized itemsets.

Example: Considering a dataset of 10 documents, and itemset x at level i, with bitmap_x = {0100100001}, and global map updated with all documents that exist on levels $i + 1$ to k, such as global_map = {0010100101}, we have:

bitmap_x	= {0100100001}
NOT global_map	= {1101011010} AND
bitmap_x	= {0100000000}

Note that two documents were pruned from bitmap_x, as they existed in at least one itemset at a higher level.

3.2.2 Constraining Document Duplication

The inner-termset removal algorithm (Figure 11) also prepares coverage maps for individual levels. These coverage maps are used to limit document duplication at the same (their largest) level, as inner-termset removal eliminates documents from all but their largest applicable itemsets, and documents may still exist in multiple itemsets at their largest level. Using level coverage maps, documents that exist at each level are checked for existence in itemsets (clusters) at that level. If a document exists in more than MAX_DOC_DUP (user defined parameter) itemsets, a score is calculated against each matching itemset and the document is assigned to the MAX_DOC_DUP itemsets with the highest scores. We used a score calculation method similar to TDC [12], which uses the document's TFIDF vector (includes frequent 1-itemsets only) and adds the term frequencies of items that existed in the itemset.

3.2.3 Bottom-up Hierarchy Assembling, Constraining Node Duplication and Pruning of Itemsets

TDC [88] builds a cluster hierarchy by linking each itemset of size k with all of its (up to k) subsets at level k - 1. This approach may result in boosting values for the FScore measure, but would affect the overall clustering quality because of too much node duplication. On the other hand, FIHC [25] applies an expensive similarity calculation method, which first prepares a conceptual document for each node (i.e., by merging the TFIDF vectors of all documents that exist in the node or any of its children) and calculating a score against each of its (up to k) parents. The node is linked to the parent with the highest similarity. This method is expensive because it

requires preparing conceptual documents for nodes at all levels in the hierarchy (conceptual documents for first level are not needed by this step, but at the time of merging first level nodes later), and also because the similarity calculation method uses the notion of "cluster frequent items" which requires an additional step to find these items for each node, using the documents that exist in that node and any of its child nodes. It also adds another parameter to the system (i.e., "minimum cluster support") and as discussed earlier, support thresholds are not easy to generalize. Finally, assigning each node to exactly one parent does not support soft clustering, which is an essential element of real-life hierarchies. As an example, a large number of nodes in the "Yahoo Directory" are cross-linked between various categories.

```
1)  for (i = k; i >= 1; i--) do begin
2)          forall itemsets t ∈ I_i do begin
3)                  if (document-count(bitmap_t) > 0 .OR. contains(parents_i+1, t) then
4)                          if (i = 1) then
5)                                  add(children_root, t);
6)                          else
7)                                  S = get-k-subsets(t)
8)                                  sorted_list = Φ
9)                                  forall itemsets super_item ∈ S do begin
10)                                     if (contains(I_i-1, super_item)) then
11)                                         interestingness_val = apply-measure(super_item, t - super_item)
12)                                         add(sorted_list, super_item, interestingness_val)
13)                                     end
14)                                 end
15)                                 for (j = 0; j < MAX_NODE_DUP .AND. size(sorted_list) >= j; j++) do begin
16)                                     itemset = get-itemset(I_i-1, sorted_list_j)
17)                                     add(children_itemset, t)
18)                                     if (.NOT. contains(parents_i, itemset)) then
19)                                         add(parents_i, itemset)
20)                                     end
21)                                 end
22)                         end
23)                 else
24)                         prune(t)
25)                 end
26)         end
27) end
```

Figure 12. Hierarchy construction

We avoid both extremes (i.e., TDC, which assigns each node to all available parents and FIHC which assigns each node to exactly one parent) and propose a more balanced approach that assigns each node to up to a user-defined number of best matching parents. Our method is also computationally efficient, as it does not prepare conceptual documents for nodes at various levels in the hierarchy and also does not calculate cluster support, and hence, avoids the additional mining step. Instead, we used the same interestingness measure that was used to mine closed interesting itemsets in the previous step, and our super item heuristic to calculate the interestingness between the itemset at level k and its (up to k) parent itemsets at level k - 1 (i.e., by considering the parent itemset as super item). A node is linked to up to MAX_NODE_DUP (user defined parameter) parents with the highest interestingness values. This method does not look into the documents contained in the cluster and selects parents solely using the itemsets (i.e., cluster labels).

Figure 12 presents our bottom-up hierarchy construction algorithm. Because of inner termset removal and constraining maximum document duplication, a number of itemsets may no longer have any documents associated with them (i.e., empty clusters). They are pruned on the way unless they were used as a parent by a node at level $k + 1$.

3.3 Merging First Level Nodes

Generally, itemset mining results in a large number of frequent 1-itemsets (frequent single words), making the first-level nodes very sparse. Removing inner termsets and constraining document duplication results in a number of empty clusters, which are

pruned during the hierarchy construction. Still, there may be a large number of nodes at level 1. Similar to FIHC and TDC, we merge the first level nodes to reduce sparseness of this level.

TDC uses a heuristic to compute pair-wise similarities, and at each step, the pair with highest similarity is merged in a way similar to agglomerative clustering. This heuristic uses the number of common documents between nodes as the primary goodness criteria. We found this heuristic problematic, as it does not support hard clustering (i.e., MAX_DOC_DUP = 1 results in no common docs between nodes), and does not consider the actual similarities between clusters. FIHC, on the other hand, applies agglomerative clustering to the first level nodes and uses a similarity function similar to the one it uses for selecting parents during hierarchy construction. This function uses the notion of "cluster frequent items" and inspects the documents assigned to each node and all of its children to find these items, making it expensive.

We first prepare conceptual documents for first-level nodes by merging term frequencies of frequent 1-itemsets from all applicable documents in the cluster. Unlike FIHC, which prepares conceptual documents for nodes at all levels, we do it only for first-level nodes, which is significantly less expensive. We applied bisecting *k*-means, using the I_2 criterion function on these conceptual document vectors, reducing the computational complexity of this step from $O(n^2 * log(n))$ to $O(e * log(k))$ [91], where n is the number of first-level nodes, and *e* is the number of non-zero entries in the feature vectors of all conceptual documents. Applying bisecting *k*-means to the conceptual document vectors of first-level nodes is significantly less expensive than applying bisecting *k*-means to all document vectors in the data set,

making this approach more scalable than state-of-the-art approaches including bisecting *k*-means (Section 3.4.9).

3.4 Experimental Evaluation

We performed extensive experiments on nine standard datasets of varying characteristics (see Appendix C for more details on the datasets used) and compared GPHC against state-of-the-art agglomerative (UPGMA), partitioning-based (bisecting *k*-means with I_2 criterion function), frequent itemset based (FIHC) and closed frequent itemset based (TDC) approaches using multiple hierarchical clustering evaluation metrics. We used the Cluto clustering toolkit [17] to generate clustering solutions for UPGMA and bisecting *k*-means, and to merge our top-level conceptual document vectors. Furthermore, we did not remove documents assigned to multiple categories from *Reuters* dataset but removed documents without category assignment.

3.4.1 Parallelization

A large percentage of modern computer systems contain multiple processors, processors with multiple cores, or processors that offer hyper-threading capabilities. We utilized SIMD parallelism to take advantage of these features, and to increase run-time performance of various steps used in the clustering process. The closed interesting itemset mining algorithm was extended by creating N threads with IDs 0 to N-1, with each thread using its ID to independently explore a subset of possible candidate itemsets, without requiring any intra-step synchronization. This was achieved by simply replacing line 3 of find-interesting-itemsets method with:

for *(i = thread ID; i < size(super_items); i = i + N)* ***do begin***

A barrier was added between lines 4 and 5 of CSII-MINE to wait for all threads to finish, and append () on the next line was called on all threads.

Similarly, each thread handled a subset of itemsets at each level of inner termset removal, with updates to the current level's bitmap synchronized. To constrain maximum document duplication, each thread independently handled a level as there are no inter or intra-step dependencies in this step. We parallelized the hierarchy generation step by having each thread handle a subset of the itemsets at each level, with updates to parent nodes synchronized. Finally, feature vectors for individual first-level nodes were generated in parallel by a number of threads.

3.4.2 Evaluation Metrics

We used two standard hierarchical clustering evaluation metrics, namely FScore and entropy as defined by Zhao and Karypis [91], to compare the quality of clustering results produced by GPHC with other, state-of-the-art approaches. FIHC and TDC also used FScore in the same way.

FScore combines the standard precision and recall functions commonly used in Information Retrieval [69], and evaluates the overall quality of hierarchical tree using a small number of its nodes. For each ground truth class, 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 defined to be:

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

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.

3.4.3 Setting the Initial Support Threshold for First-level Itemsets

A major issue with any support-based approach, like FIHC [25], is to find the optimal support threshold. Yu et al. [88] proposed to dynamically probe the support threshold by starting with a high value and decreasing the threshold until full coverage of the dataset is achieved. We believe that this approach is problematic, as even a single noisy document would cause this approach to determine 'zero' as support threshold. We addressed this issue by applying a very low support threshold (i.e., 0.2% for the three largest, and 1% for all the other datasets used in this chapter) to generate first level itemsets and generating a "miscellaneous" top-level node containing documents not represented by any itemset. Typically, the number of such documents is very small (i.e., less then 0.1% of the dataset). The support threshold is not used beyond the first level (i.e., to find individual frequent words), and second level and higher itemsets use statistical interestingness measures.

3.4.4 Using Cross Validation to Determine Thresholds for Interestingness Measures

As explained in Sections 3.2 and 3.2.3, GPHC uses an "interestingness threshold" to prune itemsets and to select parent nodes while assembling the hierarchy. This threshold affects both the efficiency and the quality of clustering, which makes it the most important parameter in our system. While it is often possible to tune parameters and achieve good results on individual datasets, such tuning can cause the problem of over-fitting, and has little practical value. One of our most important goals was to find

measures and corresponding threshold values that are robust across datasets with varying characteristics. We attempted to achieve this goal by randomly selecting a dataset (i.e., WAP), and trying a number of threshold values for each interestingness measure. The value that resulted in best results on the randomly selected dataset was blindly used across all datasets. In addition, since the *Chi-Square* test is known to depend on the number of transactions in the dataset, and to overestimate the interestingness of itemsets in large datasets [10], we used a simple heuristic to calculate the *Chi-Square* threshold values for each of the datasets used in our experiments:

$$chiSquareThreshold = unit * (1 + ceil(\frac{dataset\_size}{p}))$$

This heuristic results in a minimum threshold value of (2 * *unit*) which linearly increases in *unit* increments for each *p* documents. In order to maintain consistency, a number of values for *unit* and *p* were applied on our randomly selected dataset and the values that resulted in a threshold that produced best results on the selected dataset were used to produce *Chi-Square* thresholds for all other datasets. Table 3 presents the threshold values obtained using this procedure, for all measures. We used these values throughout our experiments.

3.4.5 Setting Values for MAX_DOC_DUP and MAX_NODE_DUP

MAX_DOC_DUP controls the maximum document duplication at their most specific level, as explained in Section 3.2.2. Recall that documents have already been removed from all but their most specific level because of inner termset removal (Section 3.2.1). Similarly, MAX_NODE_DUP controls the maximum number of

parent nodes allowed. TDC [12] uses a parameter similar to MAX_DOC_DUP, with a value of 10, and does not impose any restrictions on the number of parent nodes. We experimentally found that this approach helps boosting the FScore, but degrades the overall clustering quality (i.e., entropy) because of too much duplication. Therefore, we used a value of 2 for both of these parameters, allowing soft clustering, and avoiding unnecessary duplication.

Table 7. FScore comparison of state-of-the-art hierarchical document clustering algorithms with GPHC, using the top 6 interestingness measures

	UPGMA	bi *k*-means with I_2	FIHC	TDC	Mutual Information	Conviction	Certainty Factor	Added Value	Chi-Square	Yule's Q
Hitech	0.499	0.561	0.458	***0.57***	0.540	0.559	0.541	0.531	0.533	0.498
Re0	0.584	0.590	0.529	0.57	0.672	0.641	***0.701***	0.621	0.593	0.614
Wap	0.640	0.638	0.391	0.47	***0.663***	0.619	0.626	0.628	0.634	0.618
Classic	0.848	0.764	0.623	0.61	***0.880***	0.817	0.786	0.793	0.802	0.781
Reuters	0.729	0.793	0.506	0.46	***0.851***	0.771	0.783	0.815	0.775	0.836
LA12	0.700	***0.741***	0.432	N/A	0.661	0.616	0.626	0.709	0.617	0.669
Ohscal	0.399	0.493	0.325	N/A	0.530	0.515	0.507	0.509	***0.547***	0.485
K1a	0.646	0.634	0.398	N/A	***0.654***	0.610	0.626	0.639	0.638	0.622
K1b	0.892	0.890	0.768	N/A	***0.903***	0.869	0.876	0.879	0.881	0.890

Table 8. Entropy comparison of state-of-the-art hierarchical document clustering algorithms with GPHC, using the top 6 interestingness measures

	UPGMA	bi *k*-means with I_2	FIHC	TDC	Mutual Information	Conviction	Certainty Factor	Added Value	Chi-Square	Yule's Q
Hitech	0.262	0.236	1.258	N/A	0.172	0.210	0.200	0.236	0.153	***0.142***
Re0	0.136	0.136	1.239	N/A	0.077	0.098	0.095	0.117	0.133	***0.064***
Wap	0.131	0.131	1.561	N/A	***0.047***	0.052	0.048	0.067	0.056	0.054
Classic	0.074	0.069	0.886	N/A	0.025	0.069	0.063	0.073	0.029	***0.014***
Reuters	0.101	0.086	1.853	N/A	0.155	0.158	0.149	0.165	0.116	***0.084***
LA12	0.151	0.134	1.076	N/A	***0.062***	0.109	0.102	0.091	0.076	0.072
Ohscal	0.279	0.232	1.775	N/A	0.237	0.300	0.288	0.322	0.230	***0.106***
K1a	0.129	0.126	1.645	N/A	0.045	0.058	0.056	0.077	***0.044***	0.063
K1b	0.043	0.042	0.544	N/A	0.042	***0.033***	0.036	0.056	0.042	0.049

3.4.6 Clustering Quality Comparison

Tables 8 and 9 compare the clustering quality of the GPHC algorithm against state-of-the-art approaches in terms of FScore and entropy. Considering that GPHC uses the Cluto clustering toolkit [17] to merge top-level nodes, which has some

randomness built in to it, we ensured a fair comparison by executing GPHC with each of the interestingness measures exactly once and recorded the results. The same approach was followed to obtain results for UPGMA and bisecting *k*-means.

For FIHC, we executed the software several times on each dataset with a number of support thresholds and recorded the best results. We noticed that support thresholds that worked best on one dataset resulted in low-quality clustering on other datasets. In several cases, applying the same threshold causes itemset mining to take an indefinite amount of time. As an example, a support threshold of 3% resulted in the best FScore on *Classic*. When the same support threshold was applied to *LA12*, it resulted in 100,000+ frequent 1, 2, and 3-itemsets, after which the itemset mining did not return for 10+ minutes and the application had to be manually terminated. Since a TDC implementation was not available, we used results from [88].

Furthermore, Cluto generates both hierarchical and flat clustering solutions for UPGMA and bisecting *k*-means. The hierarchical clustering solution does not change with the number of desired clusters, which only affects the flat clustering solution, since the desired number of flat clusters are obtained from the hierarchical tree using cluster analysis techniques. For existing frequent itemset based approaches [25, 88], comparisons seem to have been made between their hierarchical solutions and flat clustering solutions obtained for UPGMA and bisecting *k*-means, using 3, 15, 30 and 60 as the desired number of clusters. For itemset-based approaches, the number of desired clusters is less significant as it only represents the number of top-level nodes in the hierarchy, and not the total number of clusters in the solution. To obtain values for Tables 8 and 9, we used the hierarchical clustering solutions for UPGMA and

bisecting *k*-means instead, and observed that they perform better than both of the existing frequent itemset based approaches (FIHC and TDC). The FScores we obtained are also similar to the FScores reported by Zhao and Karypis [90] on the same datasets.

We report results of the top six measures here, as determined by averaging the FScores and entropies of each measure on all nine datasets. Note that some measures that are not included in Tables 8 and 9 performed very well on few datasets, but failed to generalize when the same interestingness threshold was applied on other datasets. As an example, j-measure with the threshold given in Table 3 resulted in an FScore of 0.584 on *Hitech*, and entropy of 0.061 on *LA12*. Even though these results are better than all approaches we experimented with, the same threshold did not perform as well on other datasets. Our results (Table 7) indicate that *Mutual Information* results in the best overall FScore, followed by *Added Value* and *Chi-Square*. On the other hand, *Yule's Q* results in best overall entropy (Table 8) followed by *Mutual Information* and *Chi-Square*. We conclude that *Mutual Information* offers the best balance as it outperforms all previous approaches (and interestingness measures using GPHC) in terms of FScore on five out of nine datasets, and performs better than previous approaches on seven out of nine datasets in terms of entropy.

3.4.7 Comparison of Closed Interesting Itemsets with Closed Frequent Itemsets

We compared closed interesting itemsets against closed frequent itemsets, by mining closed frequent itemsets at various support levels on *Reuters* dataset (one of the most

popular research datasets), and applying our clustering process on the mined itemsets. When an interestingness measure is used to mine itemsets, the hierarchy generation process uses the same measure for parent selection, as explained in Section 3.2.3. For closed frequent itemsets, we used support for this purpose in a way that up to MAX_NODE_DUP parents that share the most documents with the child node (i.e., parent nodes with lowest support) were selected. We found that this approach achieves better FScores as compared to TDC [12], which also uses closed frequent itemsets.

Using the support thresholds that were used to generate closed frequent itemsets, we generated frequent 1-itemsets and used them to mine closed interesting itemsets using a few of our top measures. The interestingness thresholds were held constant (i.e., as defined in Table 3). The resulting itemsets were used to cluster the *Reuters* dataset. We report the number of level 2 and higher itemsets generated, along with the corresponding FScores, for closed frequent itemsets and each of the measures used to generate closed interesting itemsets. We omitted the number of 1-itemsets because it remains the same for both closed frequent itemsets and closed interesting itemsets, when the same minimum support threshold is used.

Table 9. The performance of closed interesting itemsets over closed frequent itemsets, at various support levels

Min Supp	*Closed Frequent*		*Mutual Information*		*Yule's Q*		*Added Value*	
	#	*F*	*#*	*F*	*#*	*F*	*#*	*F*
1%	92880	0.71	1613	0.83	2445	0.80	836	0.78
2%	12246	0.67	842	0.80	548	0.78	397	0.78
3%	4015	0.67	435	0.76	209	0.75	235	0.75
4%	1792	0.64	308	0.70	146	0.68	170	0.71
5%	933	0.62	231	0.68	109	0.68	135	0.69

Table 9 presents the results of this experiment. Clearly, the number of closed interesting itemsets found at all support levels is significantly smaller than the number of closed frequent itemsets. Even so, the proposed method achieved better FScores. Also, the quality of clustering decreases for all itemset types, as the minimum support threshold increases, adding credence to the claim that higher support thresholds result in pruning useful associations [76].

3.4.8 Parallel Processing and Hyper-threading

In order to analyze the impact of parallel itemset mining and hierarchy generation, we performed experiments on a system that contains two hyper-threaded 2.8 GHz Intel Xeon based processors. Each hyper-threaded processor is seen as two logical processors by the OS, resulting in a total of four processors available for executing programs. We started with a single thread and executed the clustering process on two largest datasets (i.e., *Reuters* and *Ohscal*) used in our experiments ten times in order to maximize the expected difference, and averaged the execution times. The same process was repeated with number of threads set to 2, 3 and 4. Note that our run-time environment (i.e., 64-bit Java) mapped individual threads to separate processors.

Figure 13 presents results of this experiment. Using four threads resulted in an average total speedup (computed as the ratio of old to new execution times) of 1.67 on *Reuters* and 1.5 on *Ohscal* dataset, when compared with the corresponding single-threaded solutions. Itemset mining enjoyed the most significant performance improvement as threads were added because it does not require intra-step synchronization. On the other hand, hierarchy generation performance improved only

when a new thread could map to a separate physical processor (i.e., from one thread to two threads) and decreased if more threads were added, requiring execution on a logical processor, because of intra-step synchronization on node modifications, and bitmap updates. This suggests that using a different number of threads for each of these steps could result in better overall performance. Finally, comparing the performance of two-threaded solution with four-threaded solution, we can see that hyper-threading resulted in an average itemset mining speedup of 15% and 13% on *Reuters* and *Ohscal* datasets, respectively.

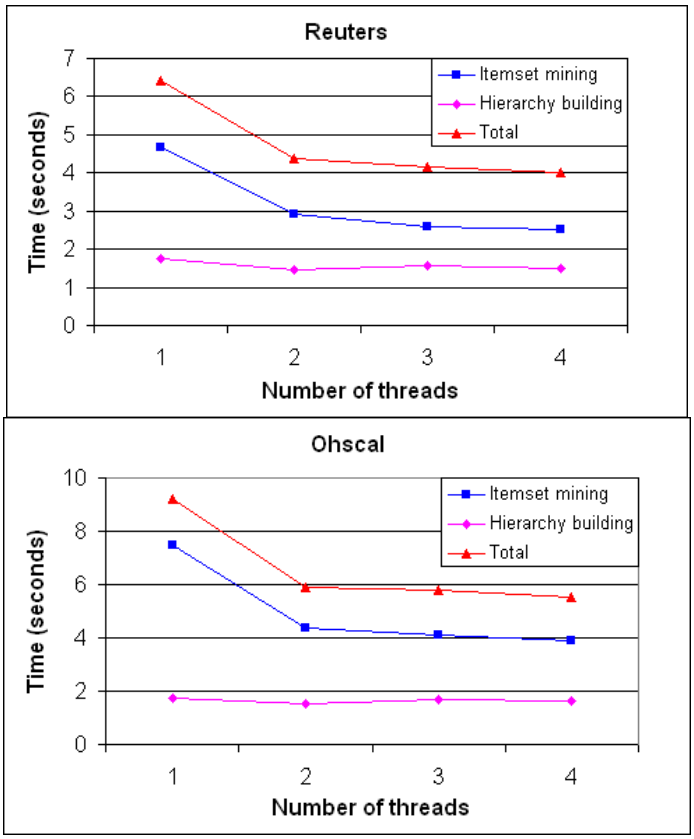


Figure 13. Impact of parallel processing on Reuters and Ohscal datasets with *Mutual Information* as interestingness measure, and threshold as in Table 3

3.4.9 Runtime Performance and Scalability

We used the full *Ohsumed* [59] collection (34,389 unique documents, and 36,250 unique attributes) to evaluate the run-time performance and scalability of GPHC. The *Ohsumed* collection was used to generate ten datasets, containing 20K to 200K documents in 20K increments. Each of these datasets was generated by selecting N documents randomly (where N is the size of desired dataset) from existing documents, and replacing approximately 40% of words with other words from the corpus, retaining the frequencies of replaced words. Using *Mutual Information* as the interestingness measure and the threshold value from Table 3, we executed both the parallel (using 4 threads), and single-threaded versions of GPHC, and also executed bisecting *k*-means, and FIHC on these datasets. For FIHC, we used the support threshold that resulted in best FScore on the full *Ohsumed* collection. In order to ensure a fair comparison, we turned off all cluster analysis, and output options for bisecting *k*-means, and excluded I/O and reporting times. In addition, the reported times of GPHC include execution times of all steps, except offline preprocessing to form document vectors and bitmaps.

Figure 14 presents results of this experiment. We found that bisecting *k*-means scaled up linearly, and FIHC scaled worse than linearly, possibly because of its frequent accesses to document vectors and its agglomerative merging of top-level nodes. The parallel version of GPHC outperformed the single-threaded version, as expected. Both versions of GPHC scaled sub-linearly, because of significant dimensionality reduction achieved by using closed interesting itemsets for clustering, and because GPHC reduces the need to refer to full document vectors. These vectors

are referred to only once: i.e., to generate frequent 1-itemsets. All interesting k (where $k \geq 2$) itemsets are generated using the bitmaps of frequent 1-itemsets, and most documents are clustered without ever referring back to the document vectors. The number of such documents increases with the size of the dataset, as GPHC primarily uses itemsets for forming clusters, and the number of words in the corpus does not linearly increase with new documents. Partial vectors (i.e., applicable frequent 1-itemsets) of a small percentage of documents are referred to remove document duplication from clusters at the same level, and to generate conceptual documents for first level nodes. Finally, we expect GPHC to scale better than TDC, because the number of closed interesting itemsets is significantly smaller than the number of closed frequent itemsets (Section 3.4.7), and other optimizations made throughout the clustering process (i.e., using bisecting k-means to merge the first-level nodes).

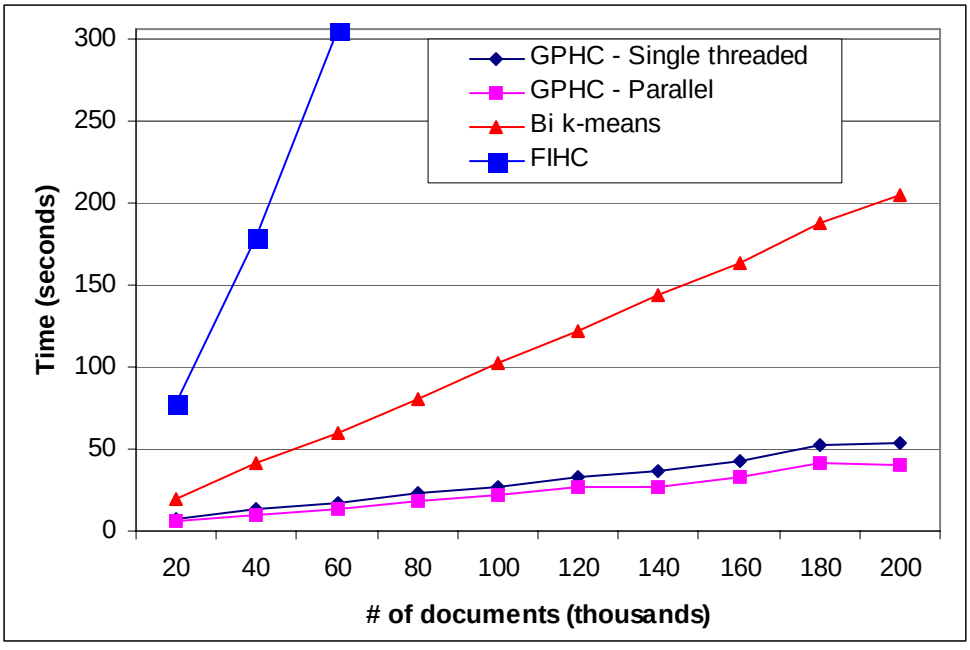


Figure 14. Runtime performance and scalability comparison of GPHC, with bisecting *k*-means and FIHC

3.5 Conclusions

We conclude that using a lower number of closed interesting itemsets may achieve better clustering quality as compared to using a significantly higher number of closed frequent itemsets for the same purpose, and only a small number of interestingness measures were stable across datasets. We also conclude that GPHC may outperform existing hierarchical clustering algorithms in terms of both FScore and entropy.

4. An Instance-Driven Approach to Pattern-Based Hierarchical Clustering

In this chapter, we consider the problem of unsupervised hierarchical clustering of instances in transactional datasets with discrete valued features. Given *n* instances, the goal is to produce a cluster hierarchy of the instances that maximizes the FScore and minimizes the entropy, allowing each instance to exist in multiple clusters at any level in the hierarchy, with each cluster automatically assigned a set of patterns as its label.

4.1 Motivation

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 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.

The most successful hierarchical clustering algorithms include agglomerative algorithms such as UPGMA [91] and partitioning based algorithms such as bisecting *k*-means [91]. Additionally, a number of pattern-based hierarchical clustering algorithms have achieved success on a variety of datasets [25, 5, 88, 56, 85]. These algorithms come with an added advantage of automatically identifying cluster labels (i.e., the set of atomic patterns defining each cluster), and many of them easily support soft clustering. However, these features are not readily available in agglomerative and partitioning based algorithms. There are three major problems with existing pattern-based hierarchical clustering algorithms: sensitivity of globally significant patterns to threshold values, unnecessary coupling between pattern size and node height, and artificial constraints on soft clustering. These problems are discussed in the three subsections of this section.

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

Most of the existing pattern-based hierarchical clustering algorithms [25, 5, 88, 56, 85] follow a similar framework. These algorithms first mine a set of globally significant patterns (e.g., frequent itemsets [25, 5], closed frequent itemsets [88], high h-confidence itemsets [85], or closed interesting itemsets [56]), and then use these patterns to build a cluster hierarchy. Instances are assigned to one or more applicable nodes (i.e., patterns) and various heuristics are applied to eliminate insignificant nodes.

Most of the above mentioned pattern-based hierarchical clustering algorithms use a user defined threshold (e.g., minimum support or minimum h-confidence) to prune an exponentially large search space, and to obtain the final set of globally significant patterns used for clustering. Consequently, these algorithms face two potential problems. First, the final set of globally significant patterns might not cover all instances (i.e., each instance represented by at least one pattern in the final pattern set), 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 as much as thousands of times higher than the number of instances in the dataset. As we show in Section 4.6.2, 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 [56] as described in Chapter 3, 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.

Figure 15. A recent article, found at *www.cnn.com*

Additionally, instances in many text and web datasets may contain a feature (i.e., atomic pattern) more than once. Existing algorithms do not fully utilize these local feature frequencies. Some approaches [25, Chapter 3] 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. For example, we observe that local feature frequencies may provide useful insights about a pattern's significance with respect to an instance. As shown in Figure 15, consider a recent news article 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. 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 [14] captures the percentage of a numerical total that is contributed by the items in an itemset.

4.1.2 Problem 2: Unnecessary Coupling between Pattern Size and Node Height

Many existing pattern-based clustering algorithms [25, 88, Chapter 3] tightly couple the sizes of cluster labels with the node heights in the initial cluster hierarchy. In these approaches, 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, where users would appreciate more descriptive cluster labels (i.e., labels that describe the clusters well, regardless of their corresponding node heights).

Some of these approaches 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.

4.1.3 Problem 3: 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" [88, Chapter 3] 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 instances can belong to a varying numbers of topics, and the same maximum value may not work for all instances.

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" [88, Chapter 3]), and then select the top-*n* (with *n* user defined) 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.

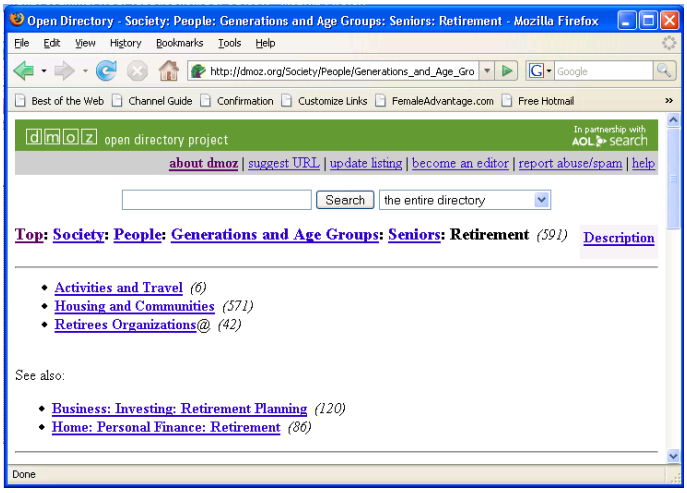


Figure 16. One of the retirement related categories in the open directory, found at http://dmoz.org

Example: Figure 16 presents one of the many retirement related nodes in the Open Directory [22], found at level 5 (i.e., Society->People->

Generations_and_Age_Groups->Seniors->Retirement). Figure 16 also points to two other retirement related nodes (i.e., Business->Investing->Retirement_Planning, and Home->Personal_Finance->Retirement), both of which are found at level 3. 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 [88, Chapter 3] would assign a general retirement related instance only to the most specific node at level 5, and eliminate this instance from both nodes at level 3, which might be against the user's expectations.

4.1.4 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. Instead of following the usual framework (i.e., first mining globally significant patterns and then using these patterns to build a cluster hierarchy), 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 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 to the total number of instances in the dataset, and all instances 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 prunes duplicate clusters in each iteration. In addition, this process produces more descriptive cluster labels than previous approaches, without tightly coupling node label sizes with node heights in the initial cluster 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.

With results of experiments performed on 40 standard datasets, we show in Section 4.6.1 that IDHC outperforms state-of-the-art hierarchical clustering algorithms both in terms of FScore and entropy (Section 3.4.2). 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. We briefly show in Section 4.6.3 that tuning these parameters to each dataset increases performance even further.

4.2 Related Work

The history of pattern-based clustering goes back to the early years of data mining. Han et al. [34] proposed a pattern-based flat clustering framework that uses association rules to generate a hypergraph of patterns (i.e., with atomic patterns 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 [16] and web image clustering [55]. In another approach, Wang and Karypis [80] 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**.** As noted in Section 8 of [15], this approach has many shortcomings including its dependence on the minimum support threshold.

Based on globally frequent itemsets, Beil et al. [5] proposed an early pattern-based hierarchical clustering framework. This framework was later enhanced by Fung et al. [25] and Yu et al. [88], who improved various stages of the clustering process. In a different approach, Xiong et al. [85] first mined 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 [25] to merge top level nodes. Results in [85] show that this approach results in clustering quality that is similar to UPGMA, with an added advantage of automatically identifying cluster labels.

In Chapter 3, we improved the framework in [25, 5, 88] 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 [91]) and partitioning-based (i.e., bisecting *k*-means with I_2 criterion function [91]) algorithms on 9

commonly used datasets. All previous global pattern-based approaches, including ours, suffer from many of the limitations discussed in Section 4.1.

Our work also relates to subspace clustering [62], 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. 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.

4.3 Dimensionality Reduction

Reducing the dimensionality of the feature space can significantly improve the performance of pattern-based clustering approaches, as the number of non-atomic patterns discovered directly depends on the initial atomic patterns (i.e., 1-itemsets). The availability of a labeled training set in supervised problems (i.e., 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 (e.g., minimum support) as the primary dimensionality reduction technique. As discussed in Section 4.1.1, 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 chapter. First, we apply Zipf's law to eliminate too frequent and too rare features. Next considering coverage issues, and the significance of local feature frequencies (Section 4.1.1), we ensure that the final set of selected features contains at least the k locally most frequent features for each instance. Our method consists of the following two steps:

***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 chapter 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. Our empirical evaluation suggests that $k = 10$ works well in practice, and appears insensitive to the dataset. Consequently, we used this value for all experiments in this chapter.

4.4 Instance-Driven Hierarchical Clustering

As discussed in Section 4.1.1, the threshold-based global pattern mining step in existing algorithms may result in an unpredictable number of patterns, with no coverage guarantees. The IDHC algorithm in Figure 17 addresses these issues by using a novel approach. Subsections 4.4.1-4.4.3 explain the three major stages in the algorithm.

```
01) build-hierarchy(dataset, min_std_dev, maxK, measure)
02)  {reduce dimensionality as explained in Section 4.3}
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 = average(freq(p1, t), freq(p2, t))
09)         significance_global = interestingness(p, measure, dataset)
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 - 1; 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
```

Figure 17. The IDHC algorithm

4.4.1 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 17) to compute its "overall" significance with respect to the current instance, considering the pattern significance at both local and global levels.

First, we determine the local pattern significance (line 8) by averaging the local frequencies of both of the atomic patterns (i.e., p_1 and p_2) in the size-2 pattern (i.e., p).

Next, we use a common interestingness measure to determine the global pattern significance (line 9). In Chapter 3, we evaluated 22 interestingness measures [27, 76], 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 Chapter 3. We have found that the same measures are useful to determine the global significance values in this context. See Appendix B 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 11).

All size-2 patterns are then sorted in decreasing order of their overall within-instance significance values (line 14), and these significance values are also used to calculate the mean and standard deviation of local significance (line 15). Considering the problem in Section 4.1.3, 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 defined 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 17).

Once size-2 patterns are selected for all instances, each unique size-2 pattern forms an initial cluster, and instances are associated to the pattern clusters they selected (i.e., lines 18-22 in Figure 17 and method "add-to-cluster"). We maintain a list of pointers for each instance to track instance-to-cluster relationships.

```
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 refined_clusters}
13)        {mark instances in common_instances for elimination in both cluster1 and cluster2}
14)        {Add cluster with label = cluster_label as a child node to both cluster1 and cluster2}
15)     end
16)  end
17)  {In one pass over clusters in cluster_list, prune all instances that were marked for elimination}
18)  return refined_clusters
19) end
```

Figure 18. Supporting methods for the IDHC algorithm

(a) A transaction dataset as running example

Instance ID	***Feature-frequency pairs***
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)

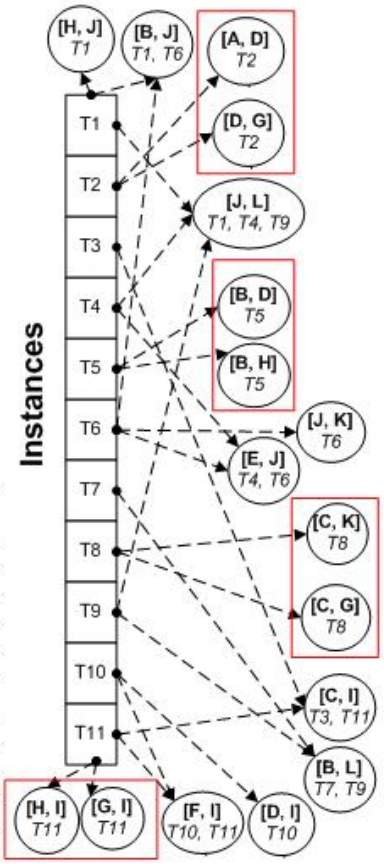
(b) Global significance values of some size-2 patterns using *Added Value* (transformed to positive scale)

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*		*Minimum significance*	*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)

(d) Initial clusters with instance based duplicates identified in boxes; dotted arrows represent instance pointers



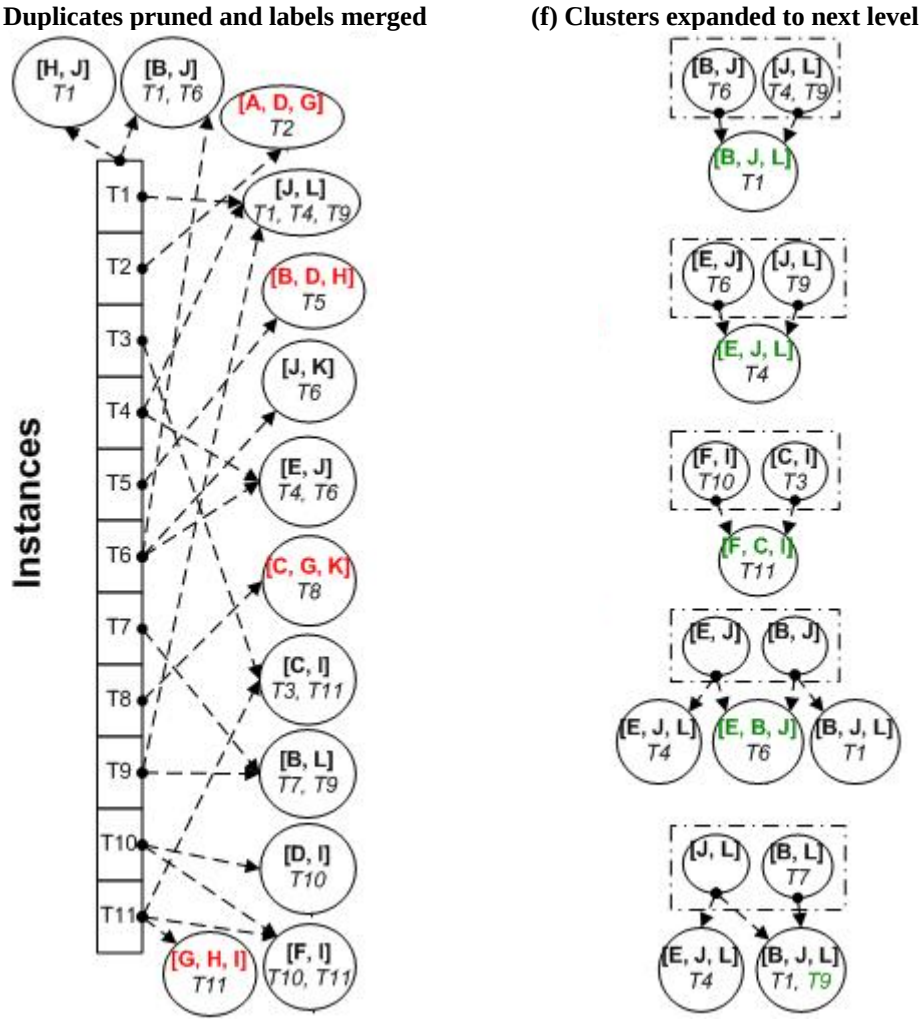


Figure 19. A running example of various stages in our clustering process (see Sections 4.4.1-4.4.3 for details)

Example: Figure 19(a) provides an example transaction dataset. Using *Added Value* [9, 10] 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 19(c). These patterns are used to form the initial clusters in Figure 19(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 4.6.1 used a more realistic value for this parameter.

Figure 19(c) also demonstrates how the algorithm "balances" local and global pattern significance. As an example, instance "T4" contains one atomic pattern (i.e., 'E') with local frequency = 4, three atomic patterns (i.e., 'B', 'J' and 'K') with frequency = 3, two atomic patterns (i.e., 'D' and 'L') with frequency = 2, and one atomic pattern (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 19(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 patterns selected for this instance (i.e., ('E', 'J') and ('J', 'L')) does include the most frequent local atomic pattern (i.e., 'E'), but does not include two of the three atomic patterns with frequency = 3. Instead, the algorithm selects pattern ('J', 'L') that has a higher global *Added Value* as shown in Figure 19(b)), 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 available size-2 patterns. As an example, both T4 and T11 contain 21 size-2 patterns but our scheme selected twice as many patterns for T11.

4.4.2 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 19(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. We can therefore show that checking for and pruning duplicate clusters locally also prunes all global duplicates.

> ***Lemma* 1**: Given a set *S* of instances, an instance *i* in *S*, and a list *L(i)* that contains pointers to all clusters that include *i*, comparing each cluster pair in *L*(*i*) and pruning duplicates results in pruning all duplicate clusters that include *i*. Furthermore, repeating this process for all instances prunes all global duplicates.
>
> ***Proof***: Assume that some cluster *c*1 is a duplicate of another cluster *c*2. For the sake of contradiction, let us assume that *c*1 and *c*2 remain intact after instance-based duplication removal. There must be an instance *i* that exists in both *c*1 and *c*2, and therefore the list *L*(*i*) of cluster pointers for *i* must contain pointers to both *c*1 and *c*2. However, this would mean that the local processing of instance *i* failed to notice these local duplicates, a contradiction.

Method "prune-duplicates" in Figure 18 implements pruning based on this lemma, 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 more meaningful (more specific) labels as the duplication level

increases, partially addressing the problem described in Section 4.1.2; Section 4.4.3 addresses the rest.

Example: Considering the 17 initial clusters in Figure 19(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 up to 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 19(d)). These duplicates are pruned and their labels are merged to obtain the 13 clusters in Figure 19(e).

4.4.3 Stage 3: Generate the Cluster Hierarchy

Once the initial clusters have been created and duplicates from these clusters have been pruned, we follow an iterative cluster refinement process (lines 26-31 in Figure 17) to generate the rest of the cluster hierarchy, by making 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 $c1$ shares some instances with another cluster c2. These common instances can be removed from both $c1$ and $c2$, 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 4.4.2.

Method "refine-clusters" in Figure 18, 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 (lines 7-18).

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 19(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: any cluster can share instances with several other clusters. These shared instances are marked for elimination as they are found (line 15), and are pruned after processing all instances (line 21).

Example of one level of refinement: Figure 19(f) demonstrates refining clusters in Figure 19(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 17) 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 in Section 4.4.2 in each iteration. The full process is repeated until all clusters are refined (not shown in Figure 19).

Pattern-based clustering algorithms can result in a large number of initial clusters, making the first-level nodes in the cluster hierarchy very sparse. Most of the existing algorithms [25, 88, 85] merge first level nodes using agglomerative clustering to reduce the sparseness of this level. In Chapter 3, considering those high computational costs, we replaced agglomerative clustering with bisecting *k*-means (using I_2 criterion function [91]). We follow the same approach in this chapter. Unlike existing algorithms, first-level clusters in our initial hierarchy are not based on size-1 patterns (Section 4.4.1).

4.5 Discussion

On the surface, it might seem like IDHC merely replaces some global thresholds (i.e., minimum support [25, 5, 88, 85] or minimum interestingness [56], and maximum instance duplication [88, 56]) with a set of local thresholds (i.e., *maxK* and *min_std_dev*). However, IDHC offers at least three major advantages over existing global threshold based approaches.

First, 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 4.6.1 used a fix value of 1.5 for *min_std_dev* on all 40 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 to the number of instances in the dataset. Section 4.6.1 shows that both of these parameters are robust across datasets.

Second, most of the existing approaches rely on a global itemset mining algorithm, which only considers boolean presence or absence of items in instances. These approaches therefore inherently ignore local pattern frequencies. On the other hand,

IDHC naturally uses these local values while selecting initial patterns, and does not use a threshold for this purpose.

Third, by always including the most significant pattern for each instance, IDHC guarantees that the resulting hierarchy covers all instances.

Finally, we observe that IDHC is similar to subspace clustering [62] in that it also uses localized information to find clusters that are ignored by traditional clustering algorithms. However, unlike subspace clustering, IDHC does not attempt to find dense regions in low dimensional spaces and combine them to form clusters, or iteratively evaluate subspaces for each cluster. Instead, IDHC first allows each instance to identify its representative size-2 patterns in one pass over the dataset and forms a (possibly non-disjoint) cluster for each of these size-2 patterns. Next, it follows a unique instance-driven refinement process to obtain the rest of the cluster hierarchy.

4.6 Experimental Results

We conduced an extensive experimental study, and evaluated the performance of IDHC on 40 standard datasets with varying characteristics (see Appendix C for more details on the datasets used). We used the two standard hierarchical clustering evaluation metrics, FScore and entropy, as defined in Section 3.4.2, to compare the quality of cluster hierarchies produced by IDHC with two leading state-of-the-art hierarchical clustering algorithms.

We limited our comparison to these two existing algorithms, since we have shown in Chapter 3 that GPHC significantly outperformed FIHC [25] and TDC [88].

Furthermore, [91] reported that bisecting *k*-means with I_2 criterion function outperforms UPGMA. Our findings in [56] 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 [85] 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 [17], which has some randomness built in to it. Therefore, we ensured fairness by using the same dedicated machine to execute each clustering algorithm on each dataset 10 times, and reported the averages. In addition, we used the same code to calculate FScore and entropy values for cluster hierarchies produced by all three algorithms.

We do not analyze the runtime performance of IDHC here, but note that in practice, the runtime of IDHC was linear to the number of instances.

4.6.1 Clustering Performance

We first evaluated the effectiveness of various interestingness measures [27, 76], to determine global significance values (i.e., Section 4.4.1) on a number of datasets, and found that the top measures we reported in Chapter 3 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* 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. Section 4.6.3 discusses further improvements that may be realized by tuning *min_std_dev* and *measure* for individual datasets. See Section 4.6.2 for a note on *maxK*.

4.6.1.1. Standard text datasets. We first evaluated IDHC against bisecting *k*-means and GPHC on 16 common text datasets, 9 of which were also used in Chapter 3. 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 Chapter 3; see Section 4.6.2 for a note on the MI thresholds used for GPHC.

Table 10. Clustering quality on text datasets; higher FScores and lower entropies are better

Dataset	*FScores*			*Entropies*		
	bi-k I_2	*GPHC*	*IDHC*	*bi-k* I_2	*GPHC*	*IDHC*
Reuters	0.835	**0.851**	0.846	0.075	0.155	**0.005**
Classic	0.782	**0.880**	0.759	0.060	0.025	**0.021**
Hitech	0.528	0.540	**0.544**	0.224	0.172	**0.074**
k1a	0.668	0.654	**0.676**	0.106	0.045	**0.041**
k1b	0.882	**0.903**	0.897	0.042	0.042	**0.021**
la12	0.741	0.661	**0.748**	0.120	0.062	**0.038**
Mm	0.774	**0.943**	0.909	0.073	0.053	**0.014**
Ohscal	**0.601**	0.530	0.554	0.198	0.237	**0.081**
re0	0.610	**0.672**	0.615	0.115	0.077	**0.016**
Reviews	0.801	0.818	**0.833**	0.073	0.048	**0.013**
Sports	0.882	**0.886**	0.870	0.030	0.016	**0.005**
tr11	**0.795**	0.519	0.790	0.107	0.141	**0.038**
tr12	0.689	0.604	**0.769**	0.133	0.161	**0.037**
tr23	0.667	0.487	**0.679**	0.136	0.042	**0.038**
tr31	0.837	0.584	**0.840**	0.041	0.114	**0.013**
Wap	**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**

Table 10 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 an average entropy that is about 3 times smaller than the best competitor. As noted in [91], entropy considers the distribution of instances in all nodes of the tree whereas FScore only considers one (best) node for each ground truth class, and ignores the quality of all other nodes. Consequently, entropy may be a more effective measure to evaluate the quality of cluster hierarchies.

Table 11. Clustering quality on UCI datasets; GPHC uses YulesQ outperforming MI on these datasets

Dataset	*FScores*			*Entropies*		
	bi-k I_2	*GPHC*	*IDHC*	*bi-k* I_2	*GPHC*	*IDHC*
adult	0.768	0.811	**0.812**	0.213	0.352	**0.111**
anneal	0.817	0.778	**0.825**	0.049	0.094	**0.012**
auto	0.538	0.555	**0.578**	0.212	0.159	**0.150**
breast	0.853	0.885	**0.922**	0.133	0.362	**0.095**
cylbands	0.674	**0.675**	0.674	0.493	**0.306**	0.414
demotology	0.658	0.531	**0.733**	0.195	0.208	**0.117**
flare	**0.808**	0.807	0.805	**0.133**	0.221	0.256
glass	0.560	0.592	**0.602**	0.246	0.217	**0.135**
heart	0.533	**0.628**	0.623	0.337	0.231	**0.168**
hepatitis	0.834	0.798	**0.858**	0.285	0.136	**0.103**
horseColic	0.726	**0.732**	0.712	0.490	0.162	**0.126**
ionoSphere	**0.829**	0.814	0.790	0.179	0.129	**0.047**
iris	0.771	0.951	**0.953**	**0.106**	0.153	0.334
letRecog	0.218	0.228	**0.279**	0.143	0.180	**0.083**
mushroom	**0.889**	0.777	0.753	**0.004**	0.073	0.006
nursery	0.482	**0.688**	0.506	0.145	0.178	**0.144**
pageBlocks	**0.903**	0.867	0.894	**0.064**	0.206	0.203
penDigits	**0.620**	0.509	0.557	0.055	0.107	**0.043**
pima	**0.733**	0.719	0.694	**0.451**	0.655	0.483
soybean-large	**0.837**	0.532	0.760	**0.047**	0.145	0.059
tictacToe	**0.695**	**0.695**	**0.695**	0.462	0.528	**0.132**
waveform	0.601	**0.643**	0.603	0.361	**0.236**	0.323
wine	0.831	**0.857**	0.836	0.176	0.051	**0.023**
zoo	0.923	0.737	**0.966**	0.066	0.173	**0.037**
average	0.712	0.700	**0.726**	0.210	0.219	**0.150**

4.6.1.2. UCI datasets. We also evaluated IDHC on 24 UCI machine learning datasets obtained from [18]. These datasets are from many different domains and differ from text datasets in that they are low dimensional, and use only local feature presence.

This unfortunately means that our computation of pattern significance (line 10 of Figure 17) reduces to the computation of global significance alone. Still, IDHC ranked first on 11 datasets (Table 11), and second on another 7 datasets in terms of FScores. Furthermore, IDHC achieved the best entropies on 16 datasets, and was ranked second on another 6 datasets. Most significantly, IDHC achieved the highest average FScores and entropies across all 24 datasets.

We conclude that when local feature frequencies are available (i.e., text datasets), IDHC outperforms existing algorithms with a much higher margin in terms of entropy, but is still very effective when these frequencies are not available.

4.6.2 Robustness in the Number of Patterns

In Section 4.1.1, 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 12. 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	126,373	2.6 million	{fails}	5,952
sports	8,580	126,373	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 Chapter 3, 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 12), so much so that the mining process could not continue because it exhausted the available system recourses. 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 10.

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 [56]. 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 10). 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.

4.6.3 Optional Parameter Tuning

All results presented in Section 4.6.1 used the same fixed parameter values. In this section, we demonstrate additional gains that might be realized by tuning our parameters to each dataset. For this purpose, we varied *min_std_dev* between 1.0 and 2.0, in uniform intervals of 0.1, and used six different interestingness measures. Table 13 presents the best FScores and entropies achieved on each dataset used in this experiment, along with the corresponding parameter values. Substituting results in

Table 11, we observe that IDHC almost always outperforms existing approaches with parameter values tuned to each dataset.

Table 13. Parameter tuning on a few UCI datasets; CF = *Certainty Factor*, AV = *Added Value*

Dataset	*FScore*	*Measure*	*min stddev*	*Entropy*	*Measure*	*min stddev*
anneal	0.873	CF	1.1	0.005	yulesQ	1.6
breast	0.934	CF	1.5	0.054	conviction	1.8
cylbands	0.703	conviction	1.5	0.191	MI	1.2
flare	0.811	CF	1.3	0.103	CF	1.8
ionosphere	0.842	chi-square	1.5	0.046	AV	1.4
mushroom	0.862	yulesQ	1.8	0.002	yulesQ	1.7
pageblocks	0.911	CF	1.6	0.087	conviction	1.5
wine	0.935	chi-square	1.5	0.019	conviction	1.4

4.7 Conclusions

We conclude that local thresholds are more stable across datasets as compared to global thresholds that are traditionally used in pattern mining, and also result in guaranteeing instance coverage. We also conclude that the IDHC algorithm results in better clustering quality as compared to existing hierarchical clustering algorithms both in terms of FScore and entropy, even on highly correlated datasets.

5. Classification by Pattern-Based Hierarchical Clustering

In this chapter, we consider the problem of classifying unlabeled instances in transactional datasets with discrete valued features. Given *n* labeled training instances and *m* unlabeled test instances, the goal is to predict labels for the test instances maximizing classification accuracy for single-label classification problems and precision-recall break-even points for multi-label classification problems.

5.1 Introduction and Motivation

Traditional inductive classifiers are trained on instances in the training set to produce a classification model (or knowledge base). This model is later used to classify previously unseen test instances. Considering that these classifiers may not fully exploit the distribution of test instances in the context of the whole dataset (i.e., by building the classification model only from the training instances, while ignoring test instances altogether), a number of recent approaches [68, 89, 45] adopted a semi-supervised model for classification. These approaches first apply an unsupervised, flat clustering algorithm (i.e., *k*-means clustering) to cluster all (i.e., training and test) instances in the dataset, and then use the resulting clustering solution to add

additional instances to the training set. A classifier is then trained on the enhanced training set.

As discussed in Section 4.1, the quality of clustering achieved by traditional flat clustering algorithms (i.e., *k*-means clustering) relies heavily on the desired number of clusters (i.e., the value of *k*), which must be known in advance. Setting a good value for *k* can be non-trivial and an inappropriate value can result in various implications discussed in Section 4.1. In contrast, hierarchical clustering algorithms organize data in a general to specific fashion, and do not require the number of clusters to be known in advance.

Traditional agglomerative and partitioning-based hierarchical clustering algorithms merge exactly two nodes at each step, which may result in a "mechanical looking" hierarchy that may not resemble hierarchies produced by human experts. In addition, these algorithms do not automatically generate cluster labels, and do not support soft clustering. In contrast, pattern-based hierarchical clustering algorithms allow each node in the cluster hierarchy to have a variable number of child nodes, which may in general be closer to a real-life setting. Pattern-based hierarchical clustering algorithms also automatically generate cluster labels (i.e., atomic patterns in the pattern itself), and more easily support soft clustering.

Considering these advantages, we propose CPHC, a novel semi-supervised classification algorithm that uses a pattern-based cluster hierarchy as a direct means for classification. Unlike existing semi-supervised classification algorithms, CPHC directly uses the resulting cluster hierarchy to classify test instances and hence eliminates the extra training step.

The remainder of this section discusses the significance of pattern lengths and also provides a brief overview of the CPHC algorithm. Section 5.2 summarizes existing work that is related to this research. Section 5.3 provides details on various steps in our classification process. Section 5.4 compares the performance of CPHC against state-of-the-art machine-learning and data-mining-based classification algorithms.

5.1.1 The Significance of Pattern Lengths in Pattern-based Cluster Hierarchies

The IDHC algorithm in Figure 17 (Section 4.4) first 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 selected size-2 patterns form initial clusters, and a unique iterative refinement process generates the rest of the cluster hierarchy. In Section 4.6, we showed that this algorithm outperforms existing hierarchical clustering algorithms both in terms of FScore and entropy.

As noted in Chapters 3 and 4, entropy considers the distribution of instances in all nodes of the tree whereas FScore only considers one (best) node for each ground truth class, and ignores the quality of all other nodes. This means that a cluster hierarchy with better (i.e., lower) entropy is expected to have a higher percentage of nodes that contain most instances that belong to the same ground truth class. We performed further experiments to understand the class-label distributions over nodes with varying pattern-lengths.

Intuitively, since IDHC (Figure 17) only assigns instances to nodes (i.e., clusters) that represent their statistically selected patterns, we expected nodes with longer

patterns to have lower entropies. To validate our intuition, we applied the IDHC algorithm to cluster two common machine learning datasets and two common text datasets. We calculated the resulting individual node entropies, and grouped together nodes that represented the same pattern sizes. We report average entropies of each group in semi-log format in Figure 20. We observe that average node entropies linearly decreased (i.e., improved) with increasing pattern-sizes on all four datasets, confirming our intuition.

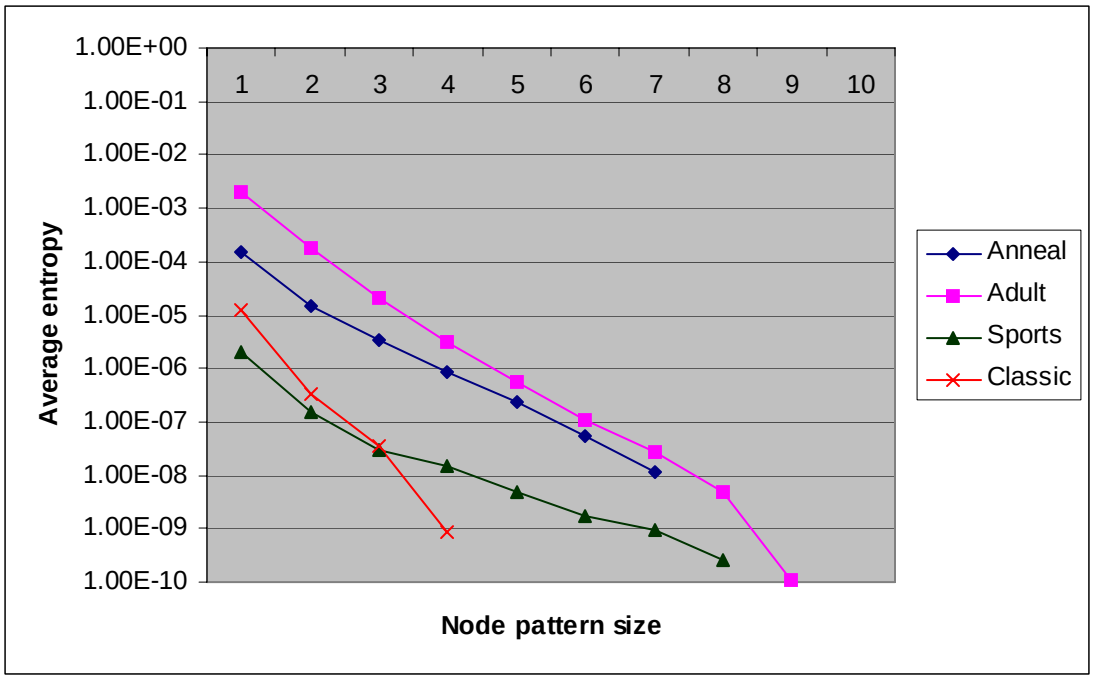


Figure 20. Average entropies of nodes with respect to their pattern sizes on anneal, adult, sports and classic datasets. Note that pattern size = 1 represent "logical" nodes obtained by applying bisecting *k*-means to merge top-level nodes in the initial cluster hierarchy, as discussed in Section 4.4.3. Least-squares regression confirmed that the relationship is essentially linear

5.1.2 CPHC: A Novel Classification Algorithm

Motivated by this observation, we propose CPHC (i.e., **C**lassification by **P**attern-based **H**ierarchical **C**lustering), a novel semi-supervised classification algorithm that uses pattern-lengths as a way of establishing cluster (i.e., node) weights. CPHC first

applies the unsupervised instance-driven pattern-based hierarchical clustering algorithm (i.e., IDHC) in Figure 17 to the whole dataset to produce a cluster hierarchy. Unlike existing semi-supervised classification algorithms [68, 89, 45], CPHC directly uses the resulting cluster hierarchy to classify test instances and hence eliminates the extra training step. To classify a test instance, CPHC first uses the hierarchical structure to identify nodes that contain the test instance, and then uses the labels of co-existing training instances, weighing them by node pattern-lengths to obtain class label(s) for the test instance. This allows CPHC to classify unlabeled test instances without making any assumptions about their distribution in the dataset.

With results of experiments performed on 19 standard datasets, we show in Section 5.4 that CPHC outperforms a number of existing classification algorithms such as FindSim, Naïve Bayes, BayesNets, Trees, ARC-BC, FOIL and CPAR, and achieves classification accuracies that are comparable to (or better than) SVM and Harmony. Most importantly, CPHC was effective even with sparse training data.

5.1.3 Contributions

The main contributions of this chapter include: 1) A novel semi-supervised classification algorithm that uses a unified pattern-based cluster hierarchy as a direct means for classification, 2) Eliminating the need to train any classifier on the enhanced training set and 3) Utilizing pattern-lengths to determine cluster (i.e., node) weights.

Even though a number of existing classification algorithms [65, 66, 19, 87, 50, 23, 48, 4, 81, 46] use patterns (or rules that contain patterns) as a direct means for

classifying data, to the best of our knowledge, none of the existing pattern-based classification algorithms exploit the distribution of test instances in the context of the whole dataset. In addition, we are not aware of any existing classification algorithm (pattern-based or non-pattern-based) that uses a cluster hierarchy as a direct means for classification.

5.2 Related Work

Our work relates to existing rule and pattern-based classification algorithms, with several important differences. Rule-induction-based classifiers like FOIL [66], RIPPER [19], CPAR [87] and C4.5 [65] use heuristics such as Gini Index and *Information Gain* (or *Information Gain* variants), to identify the best literal by which to grow the current rule [81]. Many of them follow the sequential covering paradigm. In contrast, association rule-based classifiers such as CBA [50], CAEP [23], CMAR [48], ARC-BC [4], and DeEPs [46] first mine a large set of association rules that satisfy user-defined support and confidence thresholds, and then extract the final set of classification rules by following a database covering technique. With Harmony [81], Wang and Karypis proposed an instance-centric approach to mine classification rules. Harmony builds the classification model by directly mining some user-defined number of highest-confidence rules for each training instance that satisfy minimum support. Furthermore, Harmony simultaneously mines rules for all classes.

Our work also relates to a number of recently proposed approaches that use clustering as a way of enhancing the training set. We mention only a few of those approaches here. Raskutti et al. [68] used unlabeled data that is not part of the test set

to improve the performance of text classification. This is achieved by clustering labeled and unlabeled instances together, and extracting new features from these clusters to enhance the classification model. In another approach, Zeng et al. [89] first clustered training and test sets together. The resulting clustering solution is then used to obtain labels for some of the unlabeled test instances, and the newly labeled instances are added to the training set. The extended training set is finally used to train a classifier. In a similar approach [45], Kyriakopoulou and Kalamboukis first clustered training and test sets together. The dataset is then augmented with meta features extracted from the resulting clusters, and a classifier is trained on the expanded dataset. In addition, a number of approaches like [63, 74] used clustering as a way of improving the feature selection for classification. These semi-supervised classification algorithms are similar to transductive learning [78] in that transductive learning also allows the structure of the test set to play a role in classification.

Our CPHC algorithm is similar to existing pattern-based classification algorithms in that we also use patterns. But unlike these algorithms, we do not attempt to construct a classification model from the training set. Our approach also differs from existing semi-supervised classification algorithms in that we do not use clustering as a way of enhancing the training set. Instead, we directly utilize a cluster hierarchy to classify test instances and therefore, avoid the extra step of training a classifier after clustering. In addition, existing approaches do not use pattern lengths as a way of establishing cluster weights.

5.3 The CPHC Algorithm

Step 1: Select features (Section 5.3.1)
Input: training instances $trn_1..trn_n$ test instances $tst_1..tst_m$ Select features as explained in Section 5.3.1
Output: $trn'_1..trn'_n$, and $tst'_1..tst'_m$ with reduced features
Step 2: Obtain a cluster hierarchy of training and test instances (Section 5.3.2)
Input: training instances $trn'_1..trn'_n$ test instances $tst'_1..tst'_m$ Apply the IDHC algorithm in Figure 17 on ($trn'_1..trn'_n$ U $tst'_1..tst'_m$)
Output: cluster hierarchy h
Step 3: Classify test instances (Section 5.3.3)
Input: cluster hierarchy h test instances $tst_1..tst_m$ For each test instance tst_i, Traverse h from root to leaves, identify set S of clusters that contain tst_i Use clusters in S, and lengths of associated patterns as their weights to compute class scores Assign the label of top-scoring class (or classes for multi-label problems) to tst_i
Output: predicted labels of test instances $tst_1..tst_m$

Figure 21. The CPHC algorithm

In this section, we explain various steps involved in the CPHC algorithm. Figure 21 summarizes these steps, and subsections 5.3.1-5.3.3 provide details on each step.

5.3.1 Step 1: Noise Elimination and Feature Selection

Studies [26, 72] show that reducing the dimensionality of the feature space may significantly improve the effectiveness and scalability of traditional classification algorithms, especially on high-dimensional datasets. Furthermore, dimensionality reduction tends to reduce overfitting [72]. Pattern-based classification algorithms equally benefit from dimensionality reduction, as both the quality and the number of non-atomic patterns discovered directly depends on the initial, atomic patterns (i.e., 1-itemsets).

Typically, features are selected by first sorting all available features in terms of their significance, and then selecting top-n, or top-n-percent features (with a caveat

that selecting a suitable value for *n* is not straightforward). A recent study [26] evaluated various measures to calculate feature significance and concluded that *Information Gain*, *Chi-Square* and *Bi-normal Separation* worked equally well on a number of datasets, with no statistically significant difference. Considering the comparatively high computational cost of common feature selection methods, a recent hidden-web classification algorithm [30] adapted an efficient, two-phase approach. In its first phase, Zipf's law was applied as an inexpensive heuristic dimensionality reduction technique to eliminate too frequent and too rare features. In its second phase, a more expensive method was applied to select the final set of features.

Unfortunately, none of these approaches guarantee coverage (i.e., that each instance in the corpus is represented by the selected features). Furthermore, the optimal number (or percentage) of features (i.e., the value of *n*) needed to achieve good classification results remains unclear. The literature [72] is inconclusive on *n*: some studies suggest that the number of selected features should be same as the number of training examples, and others suggest that feature selection may make matters worse, especially when the number of available features is small.

Since CPHC first produces a cluster hierarchy of the whole dataset, using a supervised feature selection method (i.e., *Information Gain*) alone may leave some test instances unrepresented in the cluster hierarchy. That is some test instances entirely consist of features that do not exist in any training instance. Traditional classification algorithms may not be able to classify such test instances at all. CPHC however, improves the chances of classifying such test instances by inducing a type

of transitivity: as long as these isolated test instances share some features with more common test instances that overlap the training set, they have a chance of being clustered together in a "logical" node (see Section 5.3.3 for details).

Considering these issues, we adopt a heuristic feature selection method that is efficient, and ensures that the final set of selected features covers all training and test instances. Furthermore, using the number of training instances, and the number of available features, our method automatically estimates the number of features used for classification (i.e., the value of *n*). Our method consists of the following four steps:

***Step 1 (calculate n)*:** $n = i + \left(i \times \log \frac{f}{i} \right)$

where i = number of training instances, and f = total number of available features. This empirically derived formula ensures a reasonable base amount for low dimensional datasets, while moderately growing this number for high dimensional datasets.

***Step 2 (select globally significant features)*:** Heuristically select globally most useful features by first applying Zipf's law to select features that are neither too frequent, nor too infrequent. In other words, select features that exist in less than *min_supp*, and more than *max_supp* instances (where *min_supp* and are *max_supp* are user defined parameters). Further refine these features by first sorting them in decreasing order of their *Information Gain* values (computed using labeled training instances only), and then adding the resulting top-*n* features to set *S* (i.e., the set of "selected" features).

***Step 3 (ensure local coverage of training instances)*:** First find all training instances with no features in *S* (i.e., instances not covered by the selected features), and then process these instances, on an instance-by-instance basis. Sort all features in the current instance in the decreasing order of their (TF * *Information Gain*), where TF = Term Frequency, calculated in the usual way. This "balances" the local significance (i.e., TF) and the global significance (i.e., *Information Gain*). Finally, add the resulting top-*t* features to set *S*. Our empirical evaluation suggests that $t = 10$ works well in practice, and appears insensitive to the dataset.

***Step 4 (ensure local coverage of test instances)*:** First find all test instances with no features in *S* (i.e., instances not covered by the selected features), and then process these instances, on an instance by instance basis. Sort all features in the current instance in the decreasing order of their Term Frequency values. Finally, add the resulting top-*t* features to set *S* (see Step 3 for a note on *t*).

5.3.2 Step 2: Hierarchical Clustering of Training and Test Instances

Once we have the features selected, we apply the IDHC algorithm (Figure 17) on the whole dataset to obtain a cluster hierarchy. The algorithm computes interestingness values for selecting size-2 patterns for instances (i.e., line 17 of Figure 17). However, in the original algorithm these values are not stored, since cluster refinement was done solely using instance-to-cluster pointers. But here, we need to use these values to calculate class scores for test instances (Section 5.3.3), so we modified the algorithm in Figure 17 to track these values. In addition, we obtain interestingness values for patterns longer than size-2 by averaging the interestingness values of

patterns merged during cluster refinement (i.e., line 7 of method "refine-clusters" in Figure 18). We also use the same process in a bottom-up fashion to obtain interestingness values for "logical" nodes (i.e., clusters) generated by merging the top-level nodes (i.e., Section 4.4.3).

5.3.3 Step 3: Classifying Test Instances

We use the following four-step process to classify test instances:

Step 3.1: Given a test instance *t*, and hierarchy *h*, first initialize scores for all classes. Next, traverse *h* from root to leaves, identifying the set *S* of nodes that contain *t*.

Step 3.2: For each node *n* in *S*, compute *w* such that:

$$w = \text{node-pattern-length} * \text{node-interestingness}$$

This weight is based on the relationship presented in Figure 20.

Step 3.3: For each class *c* represented by at least one training instance in *n* (considering all instances in the node as well as instances in all child nodes, as usual), add *x* to the score of *c* such that:

$$x = w \times \frac{|\text{training instances with label} = c \text{ in } n|}{|\text{training instances in } n|}$$

Step 3.4: For single-label classification problems, select the label of the class with the highest score. For multi-label problems, select multiple classes using the "weighted dominant factor-based" scheme in Section V(C-3) of [81], except replacing all uses of confidence with the selected interestingness measure.

Since traditional inductive classifiers only use features in training instances to obtain the classification model, these algorithms may not be able to classify test

instances that entirely consist of features that do not exist in any training instance, even if these isolated test instances share some features with more common test instances that overlap the training set. CPHC improves the chances of classifying such test instances by inducing a type of transitivity: as long as these isolated test instances share some features with more common test instances that overlap the training set, they have a chance of being clustered together in a "logical" node (i.e., node obtained by merging top-level nodes in the initial cluster hierarchy; Section 4.4.3). As a result, the "logical" node may contribute towards score calculation.

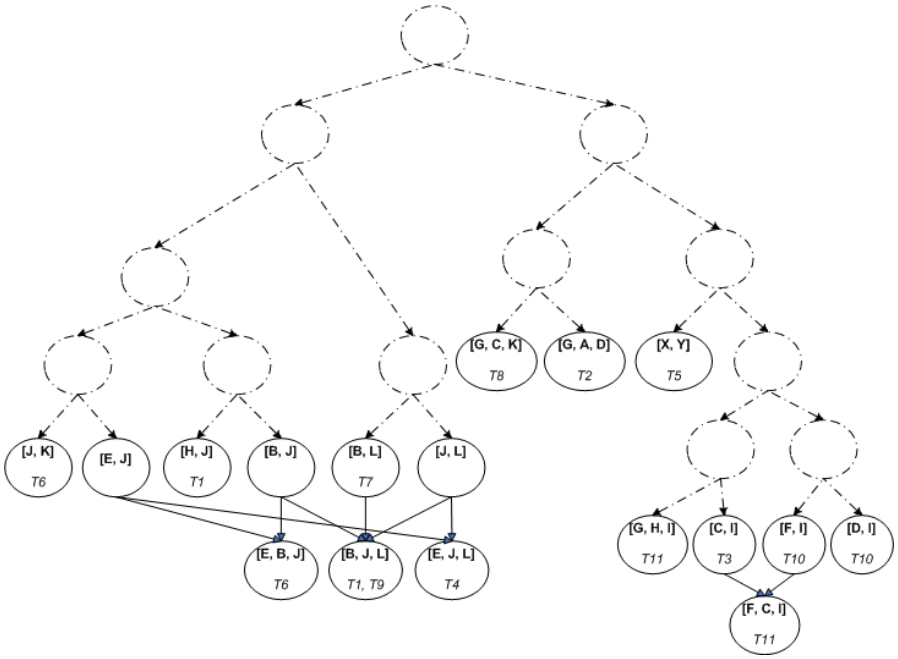


Figure 22. A pattern-based cluster hierarchy obtained by applying the IDHC algorithm in Figure 17. The dotted nodes are "logical" nodes obtained by merging the top-level nodes in the initial cluster hierarchy, as discussed in Section 4.4.3

Example: Figure 22 presents a pattern-based cluster hierarchy obtained by applying the IDHC algorithm (Figure 17). Let us assume that T3 and T5 are test instances that share some features, and the remaining instances are training instances. Let us also assume that T5 is an "isolated" test instance, i.e., T5 does not share any

features with the training set. Since T5 shares some features with T3 (i.e., a test instance that overlaps with the training set), T3 and T5 are clustered together in the logical node formed by merging the node with pattern "X, Y", and a logical node that contains T3. This structure allows the parent of node with pattern "X, Y" to predict class labels for T5.

5.4 Experimental Results

We conduced an extensive experimental study, and evaluated the performance of CPHC on 19 datasets with varying characteristics (see Appendix C for more details on the datasets used). These datasets included both standard text datasets, and numerical datasets from the UCI machine learning dataset collection. For each dataset, we compared the classification results obtained by CPHC against existing classification algorithms. In order to ensure a fair comparison, we obtained data from the same resources and used the same evaluation metrics as used by the best reported classifier.

5.4.1 Classification Performance

Our experiments in Chapters 3 and 4 indicate that *Added Value*, *Chi-Square*, *Yule's Q*, *Mutual Information*, *Certainty Factor* and *Conviction* outperform other interestingness measures [27, 76] in both global and instance-driven pattern-based hierarchical clustering contexts. Since CPHC is also based on pattern-based hierarchical clustering, we limited our experiments to these six measures. See Appendix B for computational details of these measures.

To set values for the parameters for the algorithm in Figure 21 in a principled way, we randomly selected a dataset, and tried a number of values for each parameter. The values that resulted in best results on the randomly selected dataset were blindly used across all datasets. Considering that text and UCI datasets are inherently different, we selected one text and one UCI dataset for the parameter setting purpose. This resulted in selecting *Chi-Square* as the interestingness measure for all text datasets, and *Added Value* as the interestingness measure for all UCI datasets. In addition, we obtained *min_std_dev* = 1.5, and *maxK* = 11. Finally, we fixed *min_supp* to 2 on all small UCI datasets, and to 40 on all other datasets. Section 5.4.3 discusses further improvements that may be realized by tuning *min_std_dev* and *measure* for individual datasets.

Additionally, all results reported here used the 10-fold cross validation scheme (with averages of all 10 experiments reported, as usual), except on Reuters-21578 dataset, where we used the ModApte split [7] to ensure an apples-to-apples comparison with results reported by existing studies.

5.4.1.1. Reuters-21578 (ModApte) text dataset. Reuters-21578 is the most-commonly used benchmark dataset to evaluate the performance of multi-class, multi-label classification algorithms. Existing studies given in [81, 24] used the precision-recall breakeven points on the ten largest categories, as the main performance criteria. We calculated these breakeven points in a way similar to [81], i.e., by changing the dominant factor, and keeping a fixed "score differentia factor" (i.e., 0.6). As mentioned above, we fixed the interestingness measure to *Chi-Square* and *min_std_dev* to 1.5.

Table 14 presents the results of this experiment. The results for Find-Sim, Naïve Bayes, Bayes-Nets, Trees (i.e., Decision-Trees), and linear-SVM are obtained from [24], while the results for ARC-BC are obtained from [4]. Note that [81] also used the same results. Finally, the results for Harmony are obtained from Table VIII of [81]. Among the ten largest categories, CPHC achieved the best break-even performance on 3 categories (i.e., crude, interest and money-fx), and ranked second on another 2 categories (i.e., acq and trade), with ranks 3-5 achieved on the remaining 4 categories. Most significantly, CPHC outperformed all existing classification algorithms in terms of micro-average performance, and also achieved a macro-average that is very close to SVM. Micro-average equally weights all the documents, thus favoring the performance on common classes, while macro-average equally weights all the classes, regardless of how many documents belong to a class.

Table 14. Breakeven performance on Reuters-21578

Category	*Harmony*	*Find Sim*	*Naïve Bayes*	*Bayes Nets*	*Trees*	*SVM (linear)*	*ARC-BC*	*CPHC*
Acq	**95.3**	64.7	87.8	88.3	89.7	93.6	90.9	94.5
Corn	78.2	48.2	65.3	76.4	**91.8**	90.3	69.6	77.2
Crude	85.7	70.1	79.5	79.6	85.0	88.9	77.9	**90.7**
Earn	**98.1**	92.9	95.9	95.8	97.8	98.0	92.8	96.5
Grain	91.8	67.5	78.8	81.4	85.0	**94.6**	68.8	91.1
interest	77.3	63.4	64.9	71.3	67.1	77.7	70.5	**81.0**
Money-fx	80.5	46.7	56.6	58.8	66.2	74.5	70.5	**84.3**
Ship	**86.9**	49.2	85.4	84.4	74.2	85.6	73.6	78.3
Trade	**88.4**	65.1	63.9	69.0	72.5	75.9	68.0	87.9
wheat	62.8	68.9	69.7	82.7	**92.5**	91.8	84.8	83.6
micro-avg	92.0	64.6	81.5	85.0	88.4	92.0	82.1	**92.1**
macro-avg	84.5	63.7	74.8	78.8	82.2	**87.1**	76.7	86.5

5.4.1.2. UCI machine learning datasets. UCI machine learning datasets are also commonly used to evaluate classification algorithms. We compared the performance of CPHC against existing algorithms on 13 small and 2 large UCI datasets, obtained from [18].

Tables 16 and 17 present the results of this experiment. The results for FOIL, CPAR, SVM (i.e., rbf-kernel), and Harmony are obtained from tables XII and XV of [81], which also notes that C4.5, Ripper, and association-based algorithms did not perform as well on these datasets. CPHC outperformed all existing algorithms, with the highest average classification accuracies.

Table 15. Classification accuracies on 13 small UCI datasets

	FOIL	***CPAR***	***SVM***	***Harmony***	***CPHC***
anneal	**96.90**	90.20	83.83	91.51	93.82
auto	46.10	48.00	55.50	61.00	**73.00**
breast	94.40	94.80	**96.80**	92.42	93.33
glass	49.30	48.00	46.00	49.80	**70.00**
heart	57.40	51.10	**60.36**	56.46	58.33
hepatitus	77.50	76.50	81.83	83.16	**83.33**
horsecolic	**83.50**	82.30	83.31	82.53	73.61
ionoSphere	89.50	**92.90**	89.44	92.03	92.57
iris	94.00	**94.70**	94.67	93.32	94.67
pima	73.80	**75.60**	74.18	72.34	73.16
tic-tac-toe	**96.00**	72.20	70.78	92.29	72.74
wine	86.40	92.50	**94.90**	91.94	88.24
zoo	96.00	96.00	86.00	93.00	**97.00**
average	80.06	78.06	78.28	80.91	**81.83**

Table 16. Classification accuracies on 2 large UCI datasets

	FOIL	***CPAR***	***SVM***	***Harmony***	***CPHC***
adult	82.50	76.70	84.16	81.90	**84.95**
mushroom	99.50	98.80	99.67	99.94	**99.98**
average	91.00	87.85	91.92	90.92	**92.46**

5.4.1.3. Sports text dataset. We also evaluated the classification accuracy of CPHC on the Sports text dataset (i.e., TREC, original source: San Jose Mercury News). The results of SVM and Harmony are obtained from [81], which used various parameter values to tune these algorithms. We follow a similar approach and used various values for *min_supp*, which is our noise elimination parameter. The values for all other parameters were kept fixed. From Table 17, we observe that CPHC resulted in better classification accuracies than both of the existing algorithms.

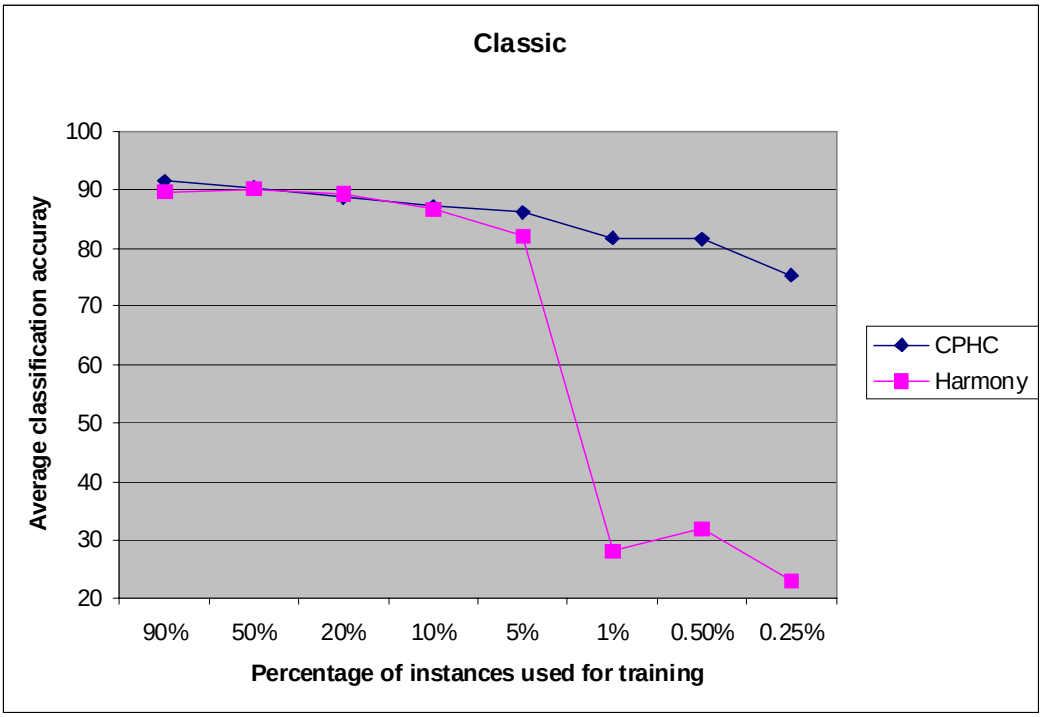
Table 17. Classification accuracy on the Sports dataset. SVM and Harmony used various values for *C* and minimum support. CPHC used various values for *min_supp*

Harmony (Min support)				*SVM (C)*				*CPHC (min_supp)*			
75	100	125	150	2.0	1.0	0.5	0.25	5	10	20	30
94.2	94.9	94.3	94.1	95.79	95.79	95.76	95.72	96.40	96.24	96.12	95.98

5.4.2 Impact of the Percentage of Training Instances on Classification Performance

To evaluate how CPHC reacts to a decreasing ratio of training instances to test instances, we performed a number of experiments on the Classic and Re0 datasets obtained from [17]. In each experiment, we randomly used a varying percentage *p* of the instances as the training set, and the rest as test set. For each value of *p*, we executed the algorithm in Figure 21 ten times and report the average classification accuracies. For comparison, we executed Harmony (i.e., executables obtained from the authors of [81]) in a similar fashion and report the average accuracies in Figure 23. Note that Harmony uses a minimum support threshold which we fixed to 1% of the training instances in each execution.

From Figure 23, we observe that the two algorithms yielded similar accuracies when a large percentage of the dataset was used as the training set. However, CPHC significantly outperformed Harmony as the size of the training set decreased. On Classic and Re0 datasets, the maximum difference in classification accuracy was as great as 53% and 31% respectively! It appears that our algorithm's ability to classify "isolated" test instances, as discussed in Section 5.3.3, is responsible for this difference.



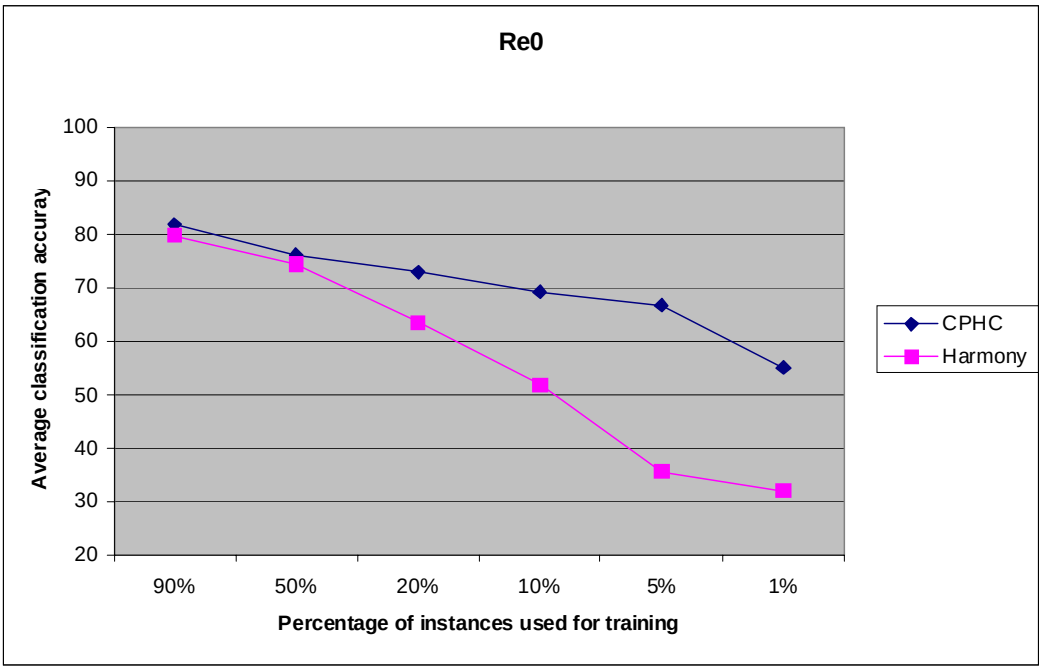


Figure 23. Classification accuracies on Classic and Re0 datasets with increasingly sparser training data. The non-linear scale is approximately logarithmic

5.4.3 Optional Parameter Tuning

Table XVI of [81] presents the classification accuracies achieved on the 13 small UCI datasets by tuning SVM and Harmony using various parameter values. We follow a similar approach to demonstrate additional gains that might be realized by tuning our parameters. For this purpose, we varied *min_std_dev* between 0.9 and 2.0, in uniform intervals of 0.1, and used six different interestingness measures.

Table 18 presents the best classification accuracy achieved on each dataset, along with the corresponding parameter values. For comparison sake, we also include fully tuned Harmony and SVM accuracies as reported in [81]. We observe that CPHC achieved better accuracies on 5/13 datasets, and resulted in the highest average classification accuracy across all 13 datasets.

Table 18. Tuned accuracies on UCI datasets

	Harmony	***SVM***	***CPHC***	***min_std_dev***	***Interestingness measure***
anneal	95.65	**97.26**	95.73	0.9	Certainty Factor
auto	61.50	58.90	**73.00**	1.2	Added Value
breast	**96.14**	95.09	94.06	1.3	YulesQ
glass	49.80	50.53	**75.71**	1.0	YulesQ
heart	58.40	57.46	**62.00**	1.3	Certainty Factor
hepatitus	**85.99**	85.50	84.67	1.8	Added Value
horsecolic	84.64	84.06	76.39	1.4	YulesQ
ionoSphere	93.45	89.43	92.29	1.5	Added Value
iris	**95.99**	93.33	95.33	1.5	Mutual Information
pima	73.79	71.06	**75.92**	1.0	Chi Square
tic-tac-toe	**94.09**	88.52	73.16	1.2	YulesQ
wine	94.90	**97.25**	95.88	1.0	Chi Square
zoo	96.00	97.00	**98.00**	1.1	Added Value
average	83.1	81.95	**84.01**		

5.5 Conclusions

We conclude that the semi-supervised approach of first clustering both the training and test sets together into a single cluster hierarchy, and then using this hierarchy as a direct means for classification eliminates the need to train a classifier on an enhanced

training set. This approach also improves the chances of classifying isolated test instances on sparse training data.

6. Optimizing Frequency Queries for Data Mining Applications

In this chapter, we consider the problem of finding the dataset representation that offers the best space-time tradeoff for counting pattern frequencies.

6.1 Introduction and Prior Work

Calculating itemset support (or frequency counting) is a fundamental operation that directly affects space and time requirements of many widely used data mining algorithms. Some data mining algorithms (i.e., frequent itemset mining [2]) are only concerned with identifying the support of a given query itemset, while others (i.e., pattern-based clustering algorithms [25, 5, 88, 56, 85]) must in addition identify the transactions that contain the query itemset.

6.1.1 Trie-based Representations

First generation data mining algorithms used the trie data structure to improve the itemset support counting performance. In the following years, a number of improvements [9, 3] were proposed to further improve support counting using a trie.

These approaches, however, did not address the major drawback of the overwhelming (possibly exponential in depth [86]) space requirements.

Table 19. A transaction database as running example, assuming minimum support = 2

TID	*Items*	*Frequent items ordered w.r.t. decreasing supports*	*Bitmaps representing each transaction*
T1	{1, 2}	{2, 1}	11000
T2	{1, 3, 4, 5}	{3, 4, 5, 1}	10111
T3	{2, 3, 4}	{3, 2, 4}	01110
T4	{2, 3, 4, 5}	{3, 2, 4, 5}	01111
T5	{2, 3, 4}	{3, 2, 4}	01110
T6	{1, 2, 3, 5}	{3, 2, 5, 1}	11101
T7	{2, 3}	{3, 2}	01100
T8	{3, 4}	{3, 4}	00110
T9	{5}	{5}	00001
T10	{3}	{3}	00100

Han et al. [35] addressed this issue by introducing FP Tree, a trie-inspired data structure that reduces the space requirements of the original trie data structure by eliminating the need to insert each transaction into all paths corresponding to the subsets of the transaction. The FP Tree is generated by identifying frequent 1-items in one pass over the dataset. These items are sorted in descending order of their supports and inserted into the *F-List* (i.e., a list that contains items in the dataset in their frequency descending order). A second pass is made to construct the FP Tree in which items are considered in the order of the *FList*. The first node corresponding to each item is pointed to from a header table and each FP Tree node contains a link to the next node corresponding to the same item.

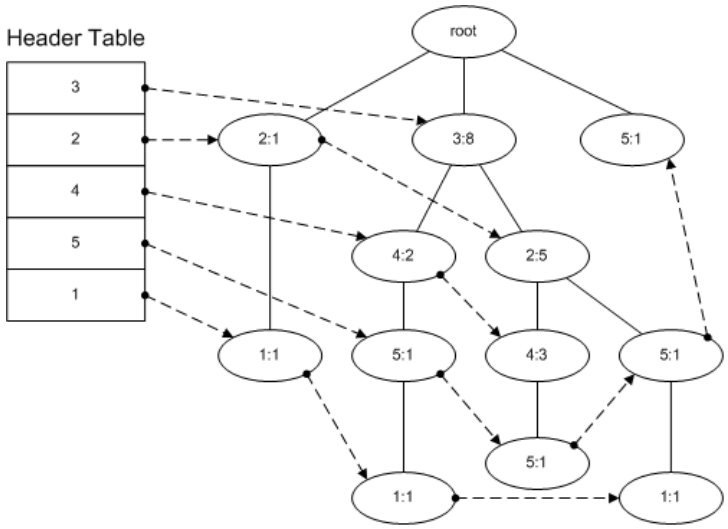


Figure 24. The FP Tree of dataset in Table 19, each node contains an item:frequency pair, and dotted arrows represent node links

***Example 1*:** Considering the transaction database in Table 19, the *FList* contains items in the order (3, 2, 4, 5, 1). Column 3 of Table 19 presents items in each transaction ordered according to the *FList*, and Figure 24 presents the corresponding FP Tree.

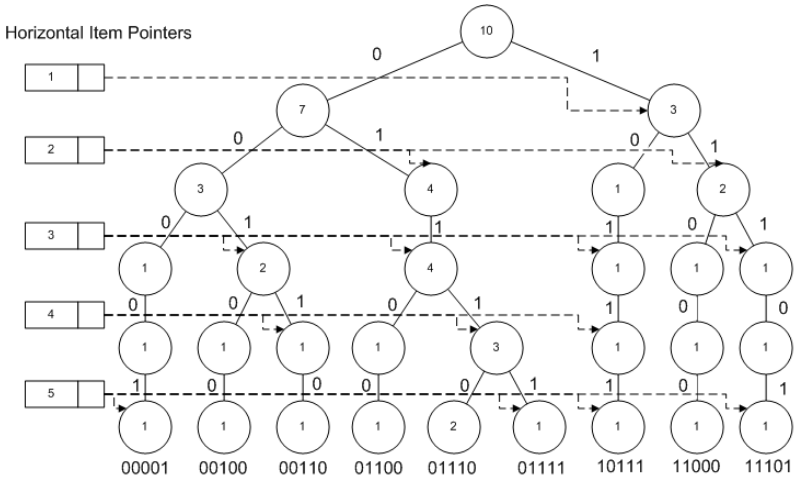


Figure 25. A binary trie, nodes contain the count of transactions with the same prefix, and dotted arrows represent pointers from the horizontal lists

In another approach, Yang et al. [86] reduced the space requirements of the trie data structure by limiting the branching factor to 2. This is achieved by generating a binary trie which considers presence or absence of all items in the transaction, rather than only considering items that exist in the transaction. For each item, a global list of horizontal pointers containing pointers to all nodes that represent the item is maintained. This list enables efficient support counting. The binary trie may contain many single-child nodes, especially on sparse datasets. This observation is used to merge these degree-1 nodes with their children, while maintaining the corresponding horizontal pointer lists. The resulting data structure is called a Compressed Patricia Trie.

Example 2: Column 4 of Table 19 contains a binary representation (i.e., presence or absence of all features) for each transaction. The corresponding binary trie is presented in Figure 25, and the Compressed Patricia Trie obtained by compressing the binary trie is presented in Figure 26. Note that the binary trie presented in [86] contains additional horizontal pointers to represent absence of items. We eliminate these pointers as they are not relevant for support counting purposes.

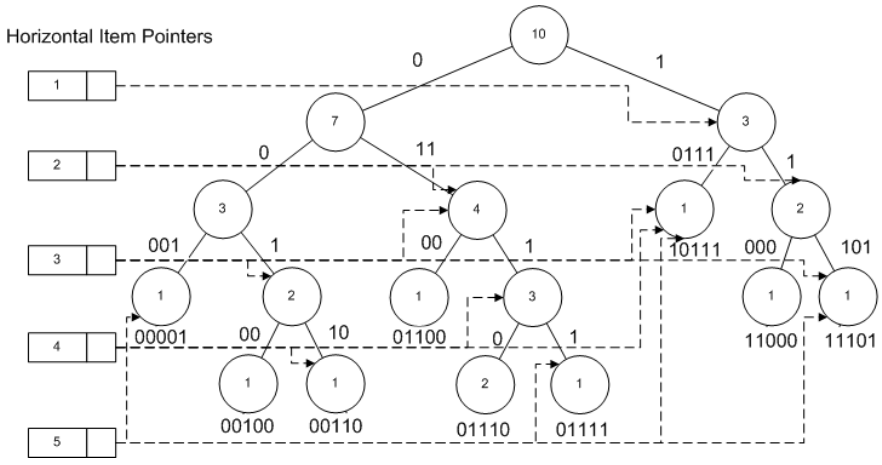


Figure 26. A Compressed Patricia Trie

6.1.2 Bitmap-based Representations

Trie-based representations are suitable for algorithms that are not concerned with the actual transactions that contain the query itemset, but they fell short when these transactions must also be identified. One solution, used by Hu et al. in a hierarchical clustering algorithm [88] is to store a list containing the applicable transaction IDs at each node of the trie. This approach may work for small datasets but is impractical for large datasets because of its significant space requirements. In the worst case (i.e., where each transaction contains each item), IDs of all transactions are replicated at each node. Another possible, but very inefficient solution is to first find the support count using the trie-based data structure and then scan the dataset once to find the applicable transactions.

Considering these issues, a number of recent approaches [56, 39, 57, 79, 11, 54] adapted uncompressed bitmap-based representations (i.e., vertical bit vectors). In these approaches, a bitmap is generated for each item in the dataset, where each bit represents presence or absence of the item in a transaction. Some of these approaches [56] also reduce the number of bitmaps by eliminating infrequent 1-itemsets as a preprocessing step. support is calculated by ANDing (i.e., intersecting) bitmaps of all items in the itemset, and counting the number of one-bits in the resulting bitmap. In level-wise itemset mining algorithms, the number of bitmaps ANDed to find support of an itemset of size k (where $k \geq 2$) is exactly 2, as the previous step would have already generated and preserved bitmaps of all frequent k - 1 sized subsets of the query itemset (i.e., guaranteed by the downward closure property). Unlike trie-based

approaches, no additional processing is needed to find the transactions containing the query itemset, as these transactions are readily available in the resulting bitmap.

Table 20. Vertical bit vectors and corresponding WAH compressed bitmaps for the dataset in Table 19, assuming 4-bit words for WAH encoding

Item	*Vertical bit vector*	*WAH compressed bitmap*
1	1100010000	**0**110 **0**001 **1**001 **0**000
2	1011111000	**0**101 **1**101 **0**100 **0**000
3	0111111101	**0**011 **1**101 **0**110 **0**100
4	0111100100	**0**011 **0**110 **0**010 **0**000
5	0101010010	**0**010 **0**101 **0**001 **0**000

***Example 3*:** Table 20 presents vertical bit vectors for the dataset in Table 19 (Section 6.2.1 provides details on the third column). Given a query itemset {2, 5}, we obtain its support by ANDing 1011111000 with 0101010010. This results in a new vertical bit vector 0001010000. Counting the number of 1-bits, we obtain the itemset support count of 2. The resulting bit vector also identifies the transactions (i.e., 4, 6) that contain the query itemset.

The most significant disadvantage of this approach is that for a dataset containing *n* transactions and *m* frequent 1-items, the amount of space needed for these bitmaps is always *m* x *n* bits, regardless of the characteristics of the underlying dataset. In practice, many datasets are sparse, which results in bitmaps with a lot more zero-bits than one-bits. Moonesinghe et al. [57] attempted to address this problem by first generating a prefix graph that contains a node for each item, and then storing a separate set of variable-sized horizontal bitmaps along with each node. This approach facilitates fast support counting, and frequent itemset mining but does not automatically identify corresponding transactions.

6.1.3 Contributions

In an attempt to find a space and time efficient dataset representation for fast support counting, that also identifies corresponding transactions, we first identified similarities between the support counting problem and the problem of performing logical operations on equality coded index bitmaps in the VLDB domain. We then considered various compressed bitmap representations of database indices, and limit ourselves to schemes that allow efficient logical operations directly on two compressed bitmaps, resulting in a compressed bitmap, without decompressing any of the operand bitmaps. We evaluated the best of these representations (i.e., WAH compressed bitmaps [82, 83, 84], with two different word sizes) against FP Tree, Patricia Trie, and uncompressed vertical bit vectors, both in terms of space requirements, and the query processing performance on more than a billion frequency queries. We then evaluated the effectiveness of recently proposed [64] pre-compression step of applying Gray code sorting to re-order transactions, on bitmaps representing 15 widely used datasets and found that this reordering scheme does not result in an optimal solution on real datasets, because of the large number of empty cells (Section 6.3.2). We observe that in practice, even simple lexicographic ordering, obtained by applying least-significant-bit first (LSB) radix sort on transaction bitmaps, may outperform this scheme.

As a replacement, we propose two novel, Hamming-distance-based transaction reordering schemes (Sections 6.3.3 and 6.3.4) with different space and time characteristics, and show (Section 6.4.1) that these schemes increase the compressibility of bitmaps.

6.2 Compressing Vertical Bit Vectors

We observe that when vertical bit vectors are used, the itemset support counting problem is a specialization (i.e., subset) of the problem of processing bitmap indices to find all rows from a database table that satisfy the given criteria. In database querying, column values in a database table can be both equality and range coded, and the criteria can contain a variety of logical operations (i.e., the 'where' clause in SQL, ignoring joins and other cross-table operations). However, in case of support counting, the values are equality coded (i.e., presence or absence of an item in a transaction) and the problem is to find all rows that contain all items satisfying the given criteria (i.e., matching the query itemset). Considering this observation, existing techniques to optimize the performance of bitwise logical operations on equality coded index bitmaps from the VLDB domain can be directly applied on the vertical bit vectors used for itemset support calculation.

There exists a tradeoff between the degree of compression, and the amount of time needed to perform logical operations on compressed bitmaps. Studies [82, 40] show that well known lossless compression schemes, such as LZ coding, B-W text compression and Huffman coding, are effective in compressing bit sequences, but require decompressing operand bitmaps to perform logical operations. Even though these schemes may achieve a higher compression ratio, the computational cost of performing logical operations makes them impractical for query intensive and real-time applications. Considering these issues, a number of schemes that mix run-length encoding and direct storage were proposed. These schemes allow logical operations

directly on two compressed bitmaps, resulting in a compressed bitmap. Some of these schemes like BBC, PackBits and PBM are byte-based, while other schemes like HRL, WAH, PWC and WBC are word-based. Studies show that word-based schemes, such as WAH, offer the best space-time tradeoff for performing logical operations. See [82, 83, 84] for a detailed comparison of these schemes.

6.2.1 WAH Compressed Bitmaps

Word-Aligned Hybrid code (WAH) [82] is a simple linear-time compression scheme that reads a bit sequence one bit at a time, and produces a word aligned compressed bitmap, where the word size W is configurable. Each word in the resulting compressed bitmap represents either a literal run or a fill run. Literal runs contain uncompressed data while the fill runs contain a word-aligned sequence (i.e., fill) of consecutive zero or one bits. The first bit in each word identifies the run type (i.e., 0 = literal run, and 1 = fill run). In case of a literal run, the rest of the W - 1 bits in the word contain a direct sequence of bits, whereas in the case of a fill run, the second bit in the word identifies the fill bit b, and the remaining W - 2 bits contain a count c which represents a sequence of c * (W - 1), b-bits. For an input bitmap with n bits, the size of the WAH compressed bitmap is upper bounded by $\left\lceil \frac{n}{W-1} \right\rceil \times W$ bits, or O (n). The worst case occurs when there are no fill runs in the resulting WAH compressed bitmap. Furthermore, the absolute value for the worst case (and the overhead) decreases as W increases.

***Example 4*:** Column 3 of Table 20 presents WAH compressed bitmaps for vertical bit vectors in column 2. Bold bits are run identifiers, and the highlighted bits are fill

bits. Many of the resulting WAH compressed bitmaps demonstrate the worst case space scenario, because we used an artificially small value for W, and a very small transaction database for simplicity sake. We show in Section 6.4.1 that WAH encoded bitmaps do not use more space than the corresponding uncompressed bitmaps on real datasets.

6.2.2 Counting Support Using WAH Compressed Bitmaps

Similar to vertical bit vectors, support of a query itemset is obtained by ANDing the corresponding WAH compressed bitmaps, and counting one-bits in the resulting bitmap. Two WAH compressed bitmaps are ANDed by iteratively decoding words from each of the operand bitmaps, and applying the AND operation on the decoded words [82]. The outcome is then added to the output bitmap. If both operand words represent literal runs, the outcome is determined by simply ANDing the two words. If one of the operand words represents a zero-fill, the same number of zeros is added to the output, and an equal number of bits are skipped from the other operand bitmap. Finally, if one of the operand bitmaps represents a one-fill, number of bits equal to the fill size is added from the other bitmap. Since processing fill runs can result in left over bits from either operand word, some bookkeeping is needed to track these leftover bits. Also, when adding a fill run to the output bitmap, the previous word in the output bitmap is checked for the presence of a matching fill, and the existing fill count is incremented by the new fill count, in case of a match. For more details, see [82].

***Example 5*:** Considering WAH compressed bitmaps in the third column of Table 20, and a query itemset {2, 5}, we first decode the first word in the first operand bitmap (i.e., **0**101) and identify it as a literal run. We then decode the first word in the second bitmap (i.e., **0**010), and identify that it is a literal run as well. Therefore, we AND 101 and 010 to obtain 000, which results in adding a zero-fill-run with count = 1 (i.e., **1**001) to the output bitmap. Similarly, processing the next words (i.e., **1**101 and **0**101), we add a literal fill **0**101 to the output bitmap (since the first operand is a 1-fill of size 1). Processing the next words (i.e., **0**100 and **0**001), we add a new zero-fill **1**001 to the output bitmap. Finally, we process the last two words (i.e., **0**000 and **0**000), and add a new zero fill with count = 1. Since the previous word in the output bitmap was a zero-fill, we just increment it and the final output bitmap becomes **1**001 **0**101 **1**010. Next, we count one-bits in the output bitmap to determine itemset support. Decoding the first word (i.e., **1**001), we find a zero-fill and continue to the next word. Decoding the second word (i.e., **0**101), which is a literal fill, we add 2 (i.e., the number of 1-bits) to the support count. Finally, the last word is decoded and ignored as it is a zero-fill and we obtain the itemset support count of 2.

Note that support has an interesting property that the support of an itemset of size k is less than or equal to the support of all of its k - 1 size subset-itemsets. In practice, a large number of itemsets have supports that are less than their subset-itemsets. This results in an important side effect of smaller and smaller WAH compressed bitmaps as the itemset size increases. As an example, to calculate support of the itemset {1, 3, 4}, we AND the compressed bitmap for item {1} with the compressed bitmap for itemset {3, 4} (above), and obtain the output bitmap **0**010 **1**011, with only two words.

Consequently, this side effect makes WAH compressed vertical bit vectors even more feasible (i.e., space efficient) for algorithms that store interim results.

6.3 Increasing Bitmap Compressibility by Reordering Transactions

The amount of compression achieved by run-length-based compression schemes such as WAH encoding depends heavily on the availability of long sequences of 0 or 1 bits in the input bitmap. The best compression is achieved when the transactions are organized in a way that minimizes the total number of bit shifts across all columns. As an example, the first column of Table 21 presents a small transaction dataset. The original order of the rows causes three bit shifts in the first column, three bit shifts in the second column and four bit shifts in the third column, adding to a total of 10 bit shifts. In contrast, the transaction ordering in the second column requires only two bit shifts in each column, adding to a total of six bit shifts for the transaction dataset, which represents a 40% reduction.

Table 21. A transaction dataset in original order, an optimal ordering, and reordered using two schemes

	Original order	***Optimal order***	***Gray code sorted***	***Radix sorted***
	T1: 101 T2: 110 T3: 001 T4: 100	T3: 001 T1: 101 T4: 100 T2: 110	T3: 001 T2: 110 T1: 101 T4: 100	T3: 001 T4: 100 T1: 101 T2: 110
Bit changes in each column	3, 3, 4	2, 2, 2	2, 3, 4	2, 2, 4
Total bit changes	10	**6**	9	8

Unfortunately, reorganizing transactions to achieve such an optimal ordering in general is same as the consecutive block minimization problem (or CBMP) which

was proven NP-complete in 70's by Kou [44]. More recently, even a fairly restricted version of this problem which limits the number of 1's in each row to 2, called 2CBMP [32], was also proven NP-hard.

6.3.1 Reordering Rows Using Gray Code Sorting

Pinar et al. [64] named this problem as the "Tuple Reordering Problem", and proven it NP-Complete by providing a reduction from the Traveling Salesman Problem. They proposed a linear in time and space transaction reordering scheme that is based on Gray code ranks, and showed that the reordered bitmaps achieve better WAH compression. As an example, the Gray code rank-based reordering reduces the total number of bit shifts from 10 to 9, on dataset in Table 21.

6.3.2 Reordering Rows Using LSB Radix Sort

It is important to note that Gray code rank-based transaction reordering results in an optimal solution only if all cells are "full" [64]. This means that for a transaction dataset with c columns, an optimal solution is obtained when there is atleast one transaction covering each of the 2^c possible combinations, which is not realistic. Therefore, even on our toy dataset (Table 21), applying Gray code rank-based reordering resulted in a small improvement.

As an alternative, simple linear-time least-significant-bit first (LSB) radix sort [20], with one bin for zero-bits and one bin for one-bits, can be used which results in a lexicographic ordering of transactions. We show in Section 6.4.1 that lexicographic ordering outperforms the Gray code scheme on many real-life datasets. Column 4 of

Table 21 presents such an example, where lexicographic ordering results in 8 bit shifts, which is better than the Gray code rank-based solution.

6.3.3 HDO, a Greedy, Hamming-distance-based Transaction Reordering Scheme

From Table 21, we observe that both Gray code rank-based, as well as lexicographic reordering may not result in close to optimal solutions on transaction datasets. We propose HDO, a greedy algorithm that reorders transactions in a way that ensures a high degree of similarity between neighboring transactions (i.e., minimizes Hamming-distance), hoping that this greedy choice results in a near-optimal solution. In other words, for each position *i*, HDO finds a transaction *t* that is closest to the transaction at position *i*-1. If there is more than one such candidate, it selects the transaction that results in least impact on the number of existing fill runs.

***Definition 1 (inter-transaction distance)*:** Let t_i be a transaction at position *i* and t_j be a transaction at position *j*, distance between t_i and t_j is defined as

$$tDist(t_i, t_j) = countOneBits(bitmap_{t_i} XOR\, bitmap_{t_j})$$

The function countOneBits(*bitmap*) returns the number of 1-bits in *bitmap*. Furthermore, the smaller is the value of *tDist* between t_i and t_j, the closer are t_i and t_j to each other. If *tDist* = 0, bitmaps for t_i and t_j are exactly the same.

***Example 6*:** Considering transactions T1 and T2 in Table 21, tDist(T1, T2) = countOneBits(101 XOR 110) = countOneBits(011) = 2.

***Definition 2 (set of least-distant transactions)*:** Let *S* be a set of transactions and *t* be a transaction in *S.* Let *S'* is a subset of *S* that does not include *t* and some other

transactions. The set CL_t of transactions that are closest (i.e., least-distant) to t is obtained by:

Step 1: For each transaction x in S', calculate tDist(t, x) and store the outcome in list L. Additionally, track the minimum Distance value *MIN*.

Step 2: For each transaction x in S, add x to CL_t iff tDist(t, x) = *MIN*.

***Definition 3 (HDO)*:** Let S be a set of transactions, assume that transactions S_1 to S_{i-1} are already in HDO. Let $S' = \{S\} - \{S_{1..i-1}\}$, the next transaction S_i is *HDOrdered* by:

Step 1: Using t = S_{i-1}, and S', obtain the set of least-distant transactions CL_t using the method above.

Step 2: If $|CL_t| = 1$, swap the unique transaction with the transaction at S_i. Otherwise, call break-ties(S, i, CL_t) in Figure 28, and swap the resulting transaction with the transaction at S_i. We explain this heuristic peephole (i.e., window) optimization below.

To apply HDO on a transaction dataset with n transactions, we first swap the first transaction in the dataset with a transaction with minimum number of columns, and then iteratively call HDO on transactions 2 to n-1, using the method above. As an example, Figure 27 demonstrates applying HDO on the dataset in Table 21. We can see that the final reordered bitmap achieves a total bit count of 6, which is same as the optimal ordering in this case. Note that our HDO algorithm is an in-place algorithm and works linear in terms of space. However, it has a time complexity of O ($|rows|^2$ x $|cols|$), which is worst than both Gray code rank-based and radix sort-based

reordering schemes, since these schemes has a time complexity linear to the number of bits in the dataset (i.e., O (|*rows*| x |*cols*|)). We address this issue in Section 6.3.4.

Step 1: Find a transaction t with minimum number of 1-bits	*Step 2: Swap row 1 with t (i.e., row 3)*	*Step 3: Calculate difference bitmaps and counts for rows 2 to 4, against row 1*
1: T1: 101 2: T2: 110 **3: T3: 001** 4: T4: 100	**1: T3: 001** 2: T2: 110 **3: T1: 101** 4: T4: 100	1: 001 2: 110 = 111 = 3 **3: 101 = 100 = 1** 4: 100 = 101 = 2
Step 4: Swap rows 2 and 3	*Step 5: Calculate difference bitmaps and counts for rows 3 & 4*	*Step 6: Swap rows 3 and 4*
1: T3: 001 **2: T1: 101** 3: T2: 110 4: T4: 100	1: 001 2: 101 3: 110 = 011 = 2 **4: 100 = 001 = 1**	1: T3: 001 2: T1: 101 **3: T4: 100** 4: T2: 110

Figure 27. Applying HDO

Breaking the ties: If $|CL_t| > 1$ (i.e., there is more than one least-distant transaction to *t*), we break the ties by selecting the candidate that minimizes the bit changes among the three transactions (i.e., the transaction *t*, the transaction prior to *t*, and the candidate itself). In other words, we select the candidate with maximum overlap in difference bits against transaction *t* and its prior transaction, as these bits are part of literal runs started in *t*. Selecting other bits may break existing fill runs and impact the overall compressibility of the transaction dataset.

```
1)  break-ties(S, i, candidates)
2)          d = bitmap of S_{i-1} XOR bitmap of S_{i-2}
3)          L = Φ
4)          for i = 1 to |candidates| do begin
5)                      temp = bitmap of candidates[i] XOR
6)                              bitmap of S_{i-1}
7)                      L[i] = countOneBits(temp XOR d)
8)          end
9)          M = {minimum value in L}
10)         C = {index of first candidate with M in L}
11)         return candidates[C]
12) end
```

Figure 28. The break-ties method

***Example 7*:** Consider t = 1001 and the transaction prior to t = 1101. Let us assume that there are two candidate transactions in set CL_t, i.e., *c1* = 1100 and *c2* = 1010, such that tDist(*t*, *c1*) = tDist(*t*, *c2*) = 2. We first compute the difference bitmap between t and its prior transaction, i.e., d = 1001 XOR 1101 = 0100. Considering *c1*, we calculate the difference bitmap *dc1* between *c1* and t (i.e., *dc1* = 1100 XOR 1001 = 0101), and find the number of different bits *ndc1* between d and *dc1*, i.e., 0100 XOR 0101 = 0001 = 1. Candidate *c2* is processed in a similar fashion, i.e., *dc2* = 1010 XOR 1001 = 0011, and *ndc2* = 0100 XOR 0011 = 0111 = 3. Since *ndc1* < *ndc2*, we select *c1*.

```
1)  aHDO (S, k)
2)          {find a row M with minimum number of columns}
3)          {swap rows 1 and M}
4)          interval = ⌊ |S| / k ⌋
5)          for i = 0 to k - 1 do begin
6)                      L = Φ
7)                      for j = (i * interval) + 2 to |S| do begin
8)                              L[j] =  tDist(S(i*interval)+1, Sj)
9)                      end
10)                     {Using values in L, apply counting sort
11)                     to order transactions S(i * interval)+2 to S|S|}
12)         end
13)         L = Φ
14)         for i = 2 to |S| do begin
15)                     L[i] = tDist(Si, Si-1)
16)         end
17)         for i = 2 to k do begin
18)                     numberOfSwaps = 0
19)                     for j = 2 to |S| - 1
20)                             distj-1Andj+1 = tDist(Sj-1, Sj+1)
21)                             distjAndj+2 = tDist(Sj, Sj+2)
22)                             d1 = L[j] - distj-1Andj+1
23)                             d2 = L[j+2] - distjAndj+2
24)                             if (d1 > 0 OR d2 > 0) AND
25)                                 (d1>=0 AND d2>=0) then
26)                                     numberOfSwaps++
27)                                     {swap rows at j, j+1}
28)                                     L[j] = distj-1Andj+1
29)                                     L[j+1] = distjAndj+2
30)                             end
31)                     end
32)                     if numberOfSwaps = 0 then break
33)         end
34) end
```

Figure 29. The aHDO algorithm

6.3.4 A Linear-time Approximation to HDO

Because of its high worst-case computational cost, HDO might not be suitable for very large, frequently updated transaction datasets. We propose aHDO, an approximation to HDO that has a time complexity linear to the number of bits in the dataset. Even so, it achieves close results, especially on sparse datasets.

Figure 29 presents the aHDO algorithm. The algorithm accepts the transaction dataset S, and a constant k, which is used to select k positions in S at uniform intervals, for the inter-loop processing. Hamming-distances of transactions at positions $i + 1$ to $|S|$ are calculated against each of the selected transaction t_i, and Counting Sort [49] is then applied to reorder these transactions, according to their Hamming-distances against t_i. Note that the linear-time Counting Sort is applicable in this case because the worst case range of Hamming-distances, for a dataset with c columns is already known (i.e., $0..c$). Next, we calculate distances between all consecutive rows (lines 14-16), and make another (up to) k passes over S. In each pass, pairs of consecutive transactions are evaluated, and transactions in the pair are swapped if it reduces the overall number of bit shifts in the solution. Considering four rows at positions $j - 1, j, j + 1$ and $j + 2$, distances between consecutive row pairs $(j - 1, j)$, $(j, j + 1)$ and $(j + 1, j + 2)$ are already available. Rows at positions j and $j + 1$ are swapped only if tDist$(j - 1, j)$ is greater than $(j - 1, j + 1)$ or tDist$(j + 1, j + 2)$ is greater than tDist$(j, j + 2)$, and neither of them results in a difference greater than the current order of the four transactions. This guarantees that swapping a row pair results in reducing the total number of bit changes by atleast 1. Note that reducing the total number of bit changes does not guarantee that the overall size of the compressed

transaction dataset will also reduce (i.e., it may replace a long, existing fill run with two small fill runs), as providing such a guarantee would require checking a number of additional conditions, against all other bits and transactions in worst case, resulting in an exponential-time algorithm. We do not demonstrate applying aHDO here, and note that setting *k* in the range of 50 to 1,000, i.e., a small proportion to the number of transactions, worked well on datasets used in our experiments.

6.4 Experimental Results

We evaluated the data structures and transaction reordering schemes discussed in this chapter in terms of memory requirements, and run-time performance of the support counting operation on fifteen widely used datasets (Table 22, Appendix C), with varying degrees of sparseness.

Table 22. Datasets used in our experiments, #entries correspond to the total number of 1-bits (i.e., columns with non-zero values), Sp = sparseness as the average number of 0's for each 1, rounded to nearest integer

Dataset	*Source*	*#rows*	*#cols*	*#entries*	*Sp*
Flare	UCI ML	1,389	30	13,890	2
Mushroom	UCI ML	8,124	88	176,248	3
Pima	UCI ML	768	36	6,144	4
Anneal	UCI ML	898	66	11,949	4
Adult	UCI ML	48,842	95	677,323	6
FBIS	TREC	2,463	2,000	393,386	12
TR 23	TREC	204	5,832	78,609	14
Hitech	SJMN (TREC)	2,301	22,498	346,881	148
Reviews	SJMN (TREC)	4,069	36,746	781,635	190
LA12	LA Times	6,279	30,125	939,407	200
Sports	SJMN (TREC)	8,580	27,673	1,107,980	213
Reuters	Reuters-21578	10,787	19,127	465,959	442
Ohsumed	Ohsumed-233445	34,389	36,250	2,018,254	617
20NG	20 Newsgroups	9,840	57,675	871,808	650
Classic4	SMART	7,094	41,681	223,839	1320

6.4.1 Space Comparison of Various Structures

Table 23 compares the memory used by the trie-based structures on our datasets. On our test (64-bit) system, each FP Tree node used 24 bytes of memory (i.e., 32-bits for frequency, 32-bits for the item ID, 64-bits for the parent pointer, and 64-bits for the node link), and the header table used 64-bits for each item. On the other hand, each node in the Patricia Trie used 12 bytes (i.e., 32-bits for frequency, and 64-bits for the parent pointer), and each pointer in the horizontal list used 64-bits. Interestingly enough, if the features in each transaction are ordered with respect to FP Tree's *FList*, the corresponding Patricia Trie contains exactly the same number of horizontal pointers as the number of nodes in the FP Tree (trivial proof omitted).

On our test datasets, Compressed Patricia Tries resulted in space savings between 36% and 67%, with greater savings realized on sparser datasets (i.e., higher percentage of degree-one nodes). In order to generate a Compressed Patricia Trie, binary trie generation appears to be a necessary interim step [86], which can be very expensive. Unlike FP Tree, where we only generate nodes for items that are present in a transaction, binary tries consider both the presence and absence of items, resulting in a significantly higher number of nodes. On our test datasets, Patricia Trie generation needed between two and twenty times more computational time as compare to FP Tree, with higher times observed on sparse datasets. Furthermore, it was not always possible to generate the binary trie in memory. As an example, binary trie generation exhausted the available memory on our test system (i.e., about 4GB) on LA12, Sports, Ohsumed, and 20NG datasets when the total number of nodes in the binary trie reached around 175 million.

Table 23. Space comparison of trie-based structures

Dataset	*FP Tree*		*Compressed Patricie Trie*		
	#nodes	*Size (KB)*	*#nodes*	*#ptrs*	*Size (KB)*
Flare	1,361	32.13	599	1,361	17.65
Mushroom	20,799	488.16	10,073	20,799	280.54
Pima	389	9.40	228	389	5.71
Anneal	1,399	33.30	730	1,399	19.48
Adult	21,877	513.48	13,464	21,877	328.70
FBIS	367,553	8,630.15	3,911	367,553	2,917.34
TR 23	75,797	1,822.05	329	75,797	596.02
Hitech	337,474	8,085.31	3,316	337,474	2,675.38
Reviews	760,265	18,105.79	5,949	760,265	6,009.29
LA12	873,862	20,716.49	N/A	N/A	N/A
Sports	1,050,754	24,843.24	N/A	N/A	N/A
Reuters	399,439	9,511.28	15,359	399,439	3,300.61
Ohsumed	1,860,347	43,885.09	N/A	N/A	N/A
20NG	792,123	19,015.97	N/A	N/A	N/A
Classic4	208,414	5,210.34	8,454	208,414	1,727.30

Table 24 compares the space used by uncompressed bit vectors, WAH encoded bitmaps in the original order, and after applying various reordering schemes. Our experiments included both 32 and 64-bit words for WAH encoding, but we only report the 32-bit results here, and note that 64-bit WAH encoded bitmaps used between 4 and 71 percent more space as compare to the corresponding 32-bit bitmaps, because with 64-bit words, uniform bit sequences smaller than 126 bits result in no space savings (i.e., fill count = 2), while 32-bit words realize space savings on shorter (i.e., >=62-bit) uniform bit sequences.

We observe that the uncompressed vertical bit vectors used less space as compare to both trie-based representations on dense datasets (i.e., Mushroom) but used significantly more space on highly sparse datasets (i.e., Classic). WAH encoding resulted in significant space savings, especially on sparse datasets. Also, lexicographic ordering outperformed Gray code rank-based reordering scheme on 12/15 datasets. Furthermore, HDO-WAH encoded bitmaps outperformed all other reordering schemes on 14/15 datasets and resulted in the most significant overall

space savings. HDO even worked well on Hitech, Reviews, and Sports datasets, where both Gray code and lexicographic schemes negatively impacted the compression achieved on the original-ordered bitmap. Finally, aHDO resulted in compression very close to HDO, especially on sparse datasets. The Classic dataset exhibits an interesting behavior, where all reordering schemes negatively affected the WAH compression achieved on the original-ordered bitmap, while HDO still outperformed other reordering schemes.

Table 24. Compression achieved by various reordering schemes. Best results highlighted, WAH compression uses a word size of 32-bit, IF = Improvement Factor as in [64], and all values rounded to 2 decimal places

	Size of the uncomp. bit vectors (Kbytes)	*WAH, original order*		*WAH, Gray code reordered*		*WAH, LSB Radix sorted*		*WAH, HDO*		*WAH, aHDO*	
		Size (KBytes)	*% of original*	*Size (KBytes)*	*IF*	*Size (KBytes)*	*IF*	*Size (KBytes)*	*IF*	*Size (KBytes)*	*IF*
Flare	5.16	5.04	97.73	3	1.68	2.91	1.73	**2.6**	**1.94**	2.67	1.89
Mushroom	87.31	70.26	80.47	22.74	3.09	20.4	3.44	**20.24**	**3.47**	21.55	3.26
Pima	3.38	2.61	77.31	1.22	2.13	1.23	2.13	**0.96**	**2.70**	1.02	2.55
Anneal	7.73	5.02	64.85	3.82	1.31	3.51	1.43	**3.33**	**1.51**	3.58	1.40
Adult	567.03	292.53	51.59	70.24	4.16	**68.3**	**4.28**	71.21	4.11	82.08	3.56
FBIS	609.38	551.57	90.51	456.15	1.21	455.74	1.21	**433.8**	**1.27**	434.64	1.27
TR 23	182.25	173.58	95.24	154.08	1.13	153.67	1.13	**144.11**	**1.20**	144.66	1.20
Hitech	6,327.56	1,222.46	19.32	1,244.64	0.98	1,244.68	0.98	**1,155.45**	**1.06**	1,174.3	1.04
Reviews	18,373	2,689.43	14.64	2,807.85	0.96	2,806.11	0.96	**2,571.97**	**1.05**	2,592.99	1.04
LA12	23,299.80	3,410.82	14.64	3,143.75	1.08	3,144.02	1.08	**2,807.72**	**1.21**	2,875.78	1.19
Sports	29,186.37	3,103.27	10.63	3,445.02	0.90	3,441.88	0.90	**2,949.74**	**1.05**	3,010.04	1.03
Reuters	25,253.62	1,826.09	7.23	1,552.38	1.18	1,549.45	1.18	**1,277.50**	**1.43**	1,359.62	1.34
Ohsumed	152,363.28	7,594.60	4.98	7,119.27	1.07	7,118.09	1.07	**6,502.91**	**1.17**	6,657.52	1.14
20NG	69,390.23	4,121.75	5.94	3,705.17	1.11	3,701.30	1.11	**2,955.54**	**1.39**	3,364.05	1.23
Classic4	36,145.24	**1,280.65**	3.54	1,387.30	0.92	1,386.88	0.92	1,317.56	0.97	1,336.81	0.96

We do not report the times needed to apply various reordering schemes here, and note that Gray code sorting, LSB radix sort and aHDO take comparable amount of time while HDO takes the most amount of time. Counter-intuitively, for small values of k (i.e., 50), we observe that aHDO may take less time than the other linear-time schemes because it calculates the inter-transaction distances by XORing whole words (i.e., 64 bits), while other schemes needs to decode and evaluate each bit, requiring

more operations. Furthermore, we trivially optimized the second most frequent operation in aHDO (i.e., counting 1-bits in a word) by caching bit-patterns.

6.4.2 Performance of Frequency Queries

Database style frequency queries: We first compared the performance of various structures by generating 25 million random frequency queries for each dataset, with 5 million queries for each of the max query sizes 1-5 (i.e., for max query size = 2, there would be about 2.5 million size-1 queries, and an equal number of size-2 queries). This adds to a total of 375 million queries on all datasets, with each query executed on all available structures, adding to many billion query executions. We assumed no prior knowledge about the query itemsets, which means that for a query itemset of size k, all k-bitmaps were used for frequency calculation. This setting is close to real-life database usage where variable-size, random query are common, with a higher percentage of short queries. We report the query execution times on eight datasets in Figure 30.

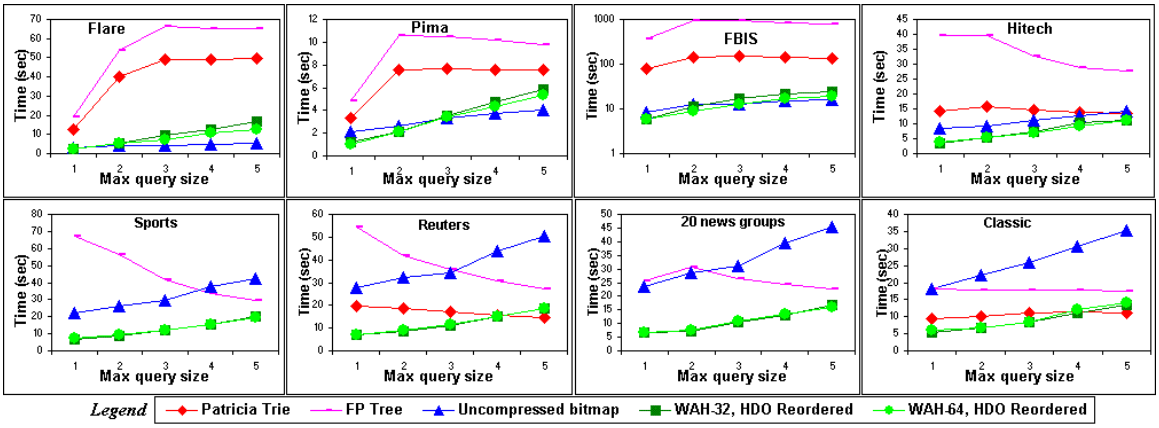


Figure 30. Performance comparison of various structures on 200 million random, variable-sized frequency queries

We observe that bitmap structures resulted in an orders of magnitude faster frequency counting as compare to trie structures on short queries. The performance difference minimized as the query size increased, because the number of bitmaps ANDed linearly increase with the number of items in the query, whereas the number of upward paths considered in a trie remains constant, and more paths can be quickly pruned for longer, randomly generated queries (i.e., decreasing number of co-occurring items). Regardless of (potentially) better frequency counting performance on long queries, trie structures are practically unusable for database style queries because most database style queries (except COUNT) must also identify the corresponding transactions. We also observe that Compressed Patricia Tries outperformed FP Trees, (i.e., a smaller number of nodes traversed). Furthermore, uncompressed vertical bit vectors resulted in shortest query execution times on dense datasets, and HDO-WAH encoded vertical bit vectors outperformed uncompressed vertical bit vectors as the sparseness increased. Finally, we observe that tries performed poorly on datasets that do not have many transactions that share common prefixes (i.e., more upward paths to consider), while the performance of bitmap structures remained un-impacted. For example, on FBIS dataset with 393,386 non-zero entries, the corresponding FP Tree contained 367,553 nodes. Consequently, it took 3,854 seconds to execute 25 million queries using the FP Tree, as compare to only 63 seconds using the HDO-WAH encoded bitmaps, a significant difference!

Data mining style frequency queries: To evaluate the performance of data mining style frequency queries, we applied APRIORI [2] to mine frequent itemsets of sizes 1-5, on datasets in Figure 30. Unlike the previous test (i.e., no prior knowledge), we

stored the bitmaps of frequent itemsets found at each step. Consequently, support calculation was performed by ANDing only two bitmaps (Section 6.1.2). An advanced nanosecond timer was used to record individual query execution times, and the total times are reported in Table 25.

We observe that bitmap structures significantly outperformed both tries. Furthermore, unlike the previous test, the performance gap did not minimize with increasing query sizes for atleast two reasons. First, the number of bitmaps ANDed remained constant (i.e., 2), and second, the percentage of upward paths pruned in tries may actually decrease because unlike the random test, where up to k - 1 items in a query of size k can be non-existent in an upward path, all k - 1 sized subsets of each query are guaranteed to meet minimum support.

Table 25. Itemset mining performance

Dataset	***min supp***	***#itemsets (size 1-5)***	***Time (seconds)***				
			Patricia Trie	***FP Tree***	***Uncompressed***	***WAH 32***	***WAH 64***
Flare	2	21,063	0.36	0.13	0.06	0.07	0.08
Pima	2	3,860	0.05	0.02	0.03	0.03	0.03
FBIS	250	654,525	172.87	361.65	1.07	2.29	1.43
Hitech	50	2,859,310	211.33	478.55	4.00	8.02	5.55
Sports	300	1,297,271	N/A	1543.58	5.04	10.50	7.03
Reuters	100	996,097	269.19	375.08	4.80	6.74	4.72
20NG	200	643,537	N/A	249.44	3.10	5.24	3.37
Classic4	10	5,800,199	212.23	278.21	20.22	15.25	12.77

Finally, we note that the runtime performance of bitmap-based schemes depends on the program structure, and the underlying system architecture, in addition to the total number of operations involved. As an example, in spite of their significantly higher space usage (which translates to more instructions needed to AND bitmaps), uncompressed bitmaps may outperform compressed bitmaps in time. This happens because two uncompressed bitmaps can be ANDed in a simple loop, with no inter-

iteration dependencies. This simple structure allows exploiting maximum instruction level parallelism, and enables compilers to apply techniques like loop unrolling. On the other hand, the decoding logic of compressed bitmaps does not allow exploiting the same level of ILP. Similarly 64-bit WAH compressed bitmaps used more space, but outperformed 32-bit bitmaps on our 64-bit test system, because the system processed twice as much data in each cycle. We conclude that HDO-WAH encoded bitmaps offer the best space-time tradeoff for data mining style queries. For example, performance was comparable to uncompressed bit vectors on Reuters and 20NG, while consuming 20 times less space.

6.5 Conclusions

We conclude that Trie structures are viable for applications that mostly execute long, random queries, as long as we are not concerned with identifying the actual transactions. Furthermore, we conclude that HDO results in better compression, and outperforms other structures on short database style frequency queries. Finally, we conclude that uncompressed bitmaps can be a good choice for data mining applications that are not concerned with high space requirements, while HDO-WAH encoded bitmaps provide the best space-time tradeoff.

7. Contributions, Conclusions and Future Work

7.1 Contributions

The contributions of this thesis include the following:

1. An association-rule-hypergraph-based web image clustering algorithm that replaces commonly used frequency-based rule significance measures with objective interestingness measures, and also combines textual and signal-based features (Chapter 2).

2. Combining the concepts of closure and interestingness into closed interesting itemsets. Our empirical experiments demonstrate that these itemsets provide significant dimensionality reduction over closed frequent itemsets (Chapter 3).

3. A sub-linearly scalable closed-interesting-itemset-based hierarchical clustering algorithm (i.e., GPHC) that outperforms state-of-the-art approaches, in terms of both FScore and entropy, on nine standard datasets (Chapter 3).

4. An instance-driven pattern-based hierarchical clustering algorithm (i.e., IDHC) that builds a cluster hierarchy without mining for globally significant patterns. This algorithm 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, uses instance-to-cluster relationships to refine clusters, produces more descriptive cluster labels, and allows a more flexible soft clustering scheme. Our extensive experiments demonstrate that this algorithm almost always outperforms state-of-the-art

hierarchical clustering algorithms and achieves up to 15 times better entropies, without requiring any tuning of parameter values, even on highly correlated datasets (Chapter 4).

5. A semi-supervised classification algorithm (i.e., CPHC) that uses a pattern-based cluster hierarchy as a direct means for classification, eliminating the need to train a classifier on an enhanced training set. In addition, this algorithm uses pattern-lengths to establish cluster weights. Our experiments indicate that this algorithm outperforms a number of existing classification algorithms even with sparse training data (Chapter 5).

6. A detailed comparative analysis of two trie-based, and several novel bitmap-based structures for itemset-frequency counting. This analysis clearly identified the most suitable structures for a variety of real-life usage scenarios (Chapter 6).

7. Novel use of WAH compressed bitmaps for itemset frequency counting and introducing HDO, a Hamming-distance-based greedy transaction reordering scheme that outperforms existing transaction reordering schemes and provide the best space-time tradeoff on equality coded WAH compressed bitmaps. We also introduce aHDO, a linear time alternative to HDO that yields similar performance on sparse datasets (Chapter 6).

7.2 Conclusions

As a result of our experiments presented in this thesis, we were able to identify many fundamental issues with traditional data mining algorithms and we began to see the field differently. First, we conclude that pure global-pattern-mining-based algorithms

may not be practical for large scale problems. Future research on large scale data mining is likely to adapt more local and instance-driven approaches. Second, we conclude that mining long patterns can be very expensive. Short patterns, as long as they are chosen in a meaningful way, have surprising power and may suffice for a number of applications. Third, we conclude that relying on training data alone may result in suboptimal classification results, especially with sparse training data. Therefore, we expect future research in classification to focus more on semi-supervised approaches. Finally, we conclude that reordered, compressed bitmap-based dataset representations may be more practical for large scale applications. They are likely to replace widely used trie variants in the future.

7.3 Future Work

In the future, we would like to extend our work on instance-driven pattern mining to real-time, incremental scenarios. In particular, we would like to investigate ways of maintaining these pattern sets as the data evolves, and also investigate ways of maintaining cluster hierarchies that use these pattern bases. In addition, we would like to investigate more sophisticated ways of using the pattern-based hierarchical structure to obtain class labels for test instances, possibly by importing ideas from existing co-training and co-clustering research. Finally, we would like to apply our transaction reordering schemes to various real-life server architectures and to identify principled ways of evaluating the feasibility of index compression with respect to target architecture.

A. Glossary of Terms

atomic pattern Same as item.

bitmap A spatially mapped array of bits.

class A group of instances ranked together as possessing common characteristics.

classification **A** procedure in which individual instances are placed into groups (classes) based on information on one or more characteristics inherent in the instances (referred to as traits, variables, characters, etc.) and based on a training set of previously labeled instances.

class label A string that uniquely identifies a class.

cluster A subset (partition) of data, such that the data (instances) in the subset (ideally) share some common trait.

clustering The partitioning of a dataset into subsets (clusters), so that the data (instances) in each subset (ideally) share some common trait.

cluster hierarchy A hierarchical representation of a set of (possibly non-disjoint) clusters such that each node in the hierarchy represents a cluster, and each node may have zero or more child nodes, without allowing for cycles.

cluster label A pattern that uniquely identifies a cluster.

dimension The number of parameters required to describe the position of a point within an abstract space.

dimensionality reduction The process of reducing the number of attributes (random variables) under consideration.

hard clustering A technique that assigns each instance to exactly one cluster.

instance **L**et $I = \{i_1, i_2, i_3, \ldots, i_n\}$ be the complete set of distinct items in a dataset. 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.

item A binary attribute.

itemset Set of items that occur together.

macro-average A measure that weights equally all the classes, regardless of how many documents belong to it.

micro-average A measure that weights equally all the documents, thus favoring the performance on common classes.

multi-label classification A technique that is concerned with learning from a set of examples (instances), each of which is associated with a set of class labels $Y \subseteq L$, where L is the set of all class labels.

pattern Same as itemset.

single-label classification A technique that is concerned with learning from a set of examples (instances) that are associated with a single label l from a set of disjoint labels L, $|L| > 1$.

soft clustering A technique that assigns each instance to one or more clusters.

trie An ordered tree data structure that is used to store an associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. All the descendants of any one node have a common prefix of the string associated with that node, and the root is associated with the empty string.

Values are normally not associated with every node, only with leaves and some inner nodes that happen to correspond to keys of interest.

B. Interestingness Measures

Table 27 presents the formulas of interestingness measures used in this thesis. These measures are functions of a 2 x 2 contingency tables. A contingency table stores the frequency counts that satisfy given conditions [27]. Table 26 is a contingency table for variables A and B, where n(*AB*) denotes the number of instances satisfying both *A* and *B*, and *N* denotes the total number of records.

Table 26. 2 x 2 contingency table for variables *A* and *B*

	B	$\overline{B}$	
A	$n(AB)$	$n(A\overline{B})$	$n(A)$
$\overline{A}$	$n(\overline{A}B)$	$n(\overline{A}\overline{B})$	$n(\overline{A})$
	$n(B)$	$n(\overline{B})$	N

Note that we use symmetric versions of *Added Value*, *Certainty Factor*, *Conviction*, *JMeasure*, and *Laplace* as defined by Tan et al. [76]. Similarly, *Max Confidence* is the symmetric version of *Confidence*. See [27] for a detailed comparison of the properties of these interestingness measures.

Table 27. Formulas of interestingness measures used in this thesis

#	Symbol	Interestingness Measure	Formula
1	AV	Added Value	$\max(P(B\|A) - P(B),\ P(A\|B) - P(A))$
2	F	Certainty Factor	$\max(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)})$
3	χ^2	Chi Square	$\sum_x \sum_y \frac{(O_{xy} - E_{xy})^2}{E_{xy}}$ where O_{xy} is the observed frequency in the contingency table and E_{xy} is the expected frequency [10]
4	S	Collective Strength	$\frac{P(AB)+P(\bar{B}\|\bar{A})}{P(A)P(B)+P(\bar{A})\times P(\bar{B})} \times \frac{1-P(A)P(B)-P(\bar{A})\times P(\bar{B})}{1-P(AB)-P(\bar{B}\|\bar{A})}$
5	c	Confidence	$P(B\|A)$
6	V	Conviction	$\max(\frac{P(A)P(\bar{B})}{P(A\bar{B})}, \frac{P(B)P(\bar{A})}{P(B\bar{A})})$
7	Φ	Correlation Coefficient	$\frac{P(AB)-P(A)P(B)}{\sqrt{P(A)P(B)P(\bar{A})(\bar{B})}}$
8	IS	Cosine	$\frac{P(AB)}{\sqrt{P(A)P(B)}}$
9	G	Gini Index	$\max(P(A)[P(B\|A)^2 + P(\bar{B}\|A)^2] + P(\bar{A})[P(B\|\bar{A})^2 + P(\bar{B}\|\bar{A})^2] - P(B)^2 P(\bar{B})^2,$ $P(B)[P(A\|B)^2 + P(\bar{A}\|B)^2] + P(\bar{B})[P(A\|\bar{B})^2 + P(\bar{A}\|\bar{B})^2] - P(A)^2 P(\bar{A})^2)$
10	I	Interest	$\frac{P(AB)}{P(A)P(B)}$
11	ζ	Jaccard	$\frac{P(AB)}{P(A)+P(B)-P(AB)}$
12	J	J-Measure	$\max(\ P(AB)\log(\frac{P(B\|A)}{P(B)}) + P(A\bar{B})\log(\frac{P(\bar{B}\|A)}{P(B)}),$ $P(AB)\log(\frac{P(A\|B)}{P(A)}) + P(\bar{A}B)\log(\frac{P(\bar{A}\|B)}{P(\bar{A})}))$
13	κ	Kappa	$\frac{P(AB)+P(\bar{A}\bar{B})-P(A)P(B)-P(\bar{A})P(\bar{B})}{1-P(A)P(B)-P(\bar{A})P(\bar{B})}$
14	K	Klosgen's	$\sqrt{P(A,B)}\max(P(B\|A) - P(B), P(A\|B) - P(A))$
15	L	Laplace	$\max(\frac{N\,P(AB)+1}{N\,P(A)+2}, \frac{N\,P(AB)+1}{N\,P(B)+2})$
16	mc	Max Confidence	$\max(P(B\|A),\ P(A\|B))$
17	M	Mutual Information	$\frac{\sum_i \sum_j P(A_i B_j)\times \log_2 \frac{P(A_i B_j)}{P(A_i)P(B_j)}}{\min(-\sum_i P(A_i)\log_2 P(A_i), -\sum_j P(B_j)\log_2 P(B_j))}$
18	α	Odds Ratio	$\frac{P(AB)P(\bar{A}\bar{B})}{P(A\bar{B})-P(\bar{A}B)}$
19	RI	Piatetsky-Shapiro's Interest	$P(AB) - P(A)P(B)$
20	s	Support	$P(AB)$
21	Q	Yule's Q	$\frac{P(AB)P(\bar{A}\bar{B})-P(A\bar{B})P(\bar{A}B)}{P(AB)P(\bar{A}\bar{B})+P(A\bar{B})P(\bar{A}B)}$
22	Y	Yule's Y	$\frac{\sqrt{P(AB)P(\bar{A}\bar{B})}-\sqrt{P(A\bar{B})P(\bar{A}B)}}{\sqrt{P(AB)P(\bar{A}\bar{B})}+\sqrt{P(A\bar{B})P(\bar{A}B)}}$

C. Datasets

Table 27 lists all datasets used in this thesis, along with their corresponding attributes. We obtained *Classic*, *FBIS*, *Hitech*, *K1a*, *K1b*, *LA12*, *MM*, *Ohscal*, *Re0*, *Reviews*, *Sports*, *TR11*, *TR12*, *TR23*, *TR31*, and *WAP* datasets from the Cluto clustering toolkit [17]. The *Reuters* dataset was obtained from [67], and the *Reuters* ModApte training / test set split used in Chapter 5 was obtained from [7]. In addition, we obtained the *Ohsumed* dataset from [59] and *20NG* dataset from [58]. Since the UCI datasets are numerical and the algorithms presented in this thesis require binary valued input data, we used the discretised versions of these datasets available at [18]. Note that many of the existing pattern-based learning algorithms, such as [81], also used the same versions of these datasets.

Table 28. Datasets used in this thesis

Dataset	*Source*	*# classes*	*#rows*	*#cols*	*#entries*
20NG	20 Newsgroups	20	9,840	57,675	871,808
Adult	UCI Machine Learning Repository	2	48,842	95	677,323
Anneal	UCI Machine Learning Repository	6	898	66	11,949
Auto	UCI Machine Learning Repository	7	205	129	5,066
Breast	UCI Machine Learning Repository	2	699	14	6,275
Classic4	SMART Project	4	7,094	41,681	223,839
Cylbands	UCI Machine Learning Repository	2	540	120	17,901
Demotology	UCI Machine Learning Repository	6	366	43	4,384
FBIS	TREC	17	2,463	2,000	393,386
Flare	UCI Machine Learning Repository	9	1,389	30	13,890
Glass	UCI Machine Learning Repository	7	214	40	1,926
Heart	UCI Machine Learning Repository	5	303	45	3,933
Hepatitis	UCI Machine Learning Repository	2	155	50	2,778
Hitech	San Jose Mercury News (TREC)	6	2,301	22,498	346,881
HorseColic	UCI Machine Learning Repository	2	368	81	6,171
IonoSphere	UCI Machine Learning Repository	2	351	155	11,934
Iris	UCI Machine Learning Repository	3	150	16	600
K1a	WebACE	20	2340	21,839	349,792
K1b	WebACE	6	2340	21,839	349,792
LA12	LA Times (TREC)	6	6,279	30,125	939,407
LetRecog	UCI Machine Learning Repository	26	20,000	76	320,000
MM	Movies / Music	2	2521	126,373	490,062
Mushroom	UCI Machine Learning Repository	2	8,124	88	176,248
Nursery	UCI Machine Learning Repository	5	12,960	27	103,680
Ohscal	Ohsumed-233445	10	11,162	11,465	674,365
Ohsumed	Ohsumed-233445	23	34,389	36,250	2,018,254
PageBlocks	UCI Machine Learning Repository	5	5,473	39	54,730
PenDigits	UCI Machine Learning Repository	10	10,992	76	175,872
Pima	UCI Machine Learning Repository	2	768	36	6,144
Re0	Reuters-21578	13	1,504	2,886	77,808
Reuters	Reuters-21578	90	10,787	19,127	465,959
Reviews	San Jose Mercury News (TREC)	5	4,069	36,746	781,635
Sports	San Jose Mercury News (TREC)	7	8,580	27,673	1,107,980
Soybean-Large	UCI Machine Learning Repository	19	683	99	21,568
TicTacToe	UCI Machine Learning Repository	2	958	27	8,622
TR 11	TREC	9	414	6,429	116,613
TR 12	TREC	8	313	5,804	85,640
TR 23	TREC	6	204	5,832	78,609
TR 31	TREC	7	927	10,128	248,903
Wap	WebACE	20	1,560	8,460	220,482
Waveform	UCI Machine Learning Repository	3	5,000	98	105,000
Wine	UCI Machine Learning Repository	3	178	65	2,314
Zoo	UCI Machine Learning Repository	7	101	35	1,616

References

[1] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association rules between set of items in large databases”, In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, DC, May 1993.

[2] R. Agrawal, and R. Srikant, “Fast Algorithms for Mining Association Rules in Large Databases”, In *Proceedings of the 20th International Conference on Very Large Databases,* pp. 487-499, Santiago, Chile, 1994.

[3] A. Amir, R. Feldman and R. Kashi, "A New and Versatile Method for Association Generation", *Information Systems*, Volume 22, No. 6, pp. 333-347, 1997,

[4] M. Antonie and O. Zaiane, "Text Document Categorization by Term Association", In *Proceedings of the 2002 IEEE International Conference on Data Mining,* 2002.

[5] F. Beil, M. Ester, and X. Xu, “Frequent term-based text clustering”, In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 436-442.

[6] S. Belongie, C. Carson, H. Greenspan, and J. Malik, “Recognition of images in large databases using a learning framework”, *Technical Report TR 97-939, U.C. Berkeley,* 1997.

[7] S. Bergsma, "The Reuters-21578 (ModApte) dataset", *Dept. of Computer Science, University of Alberta*, http://www.cs.ualberta.ca/~bergsma/HTML/Courses/650/.

[8] F. Berzal, I. Blanco, D. Sánchez and M.A. Vila, "Measuring the Accuracy and Importance of Association Rules: A New Framework", *Intelligent Data Analysis*, 2002.

[9] F. Bodon, “A fast apriori implementation”, In *Proceedings of the IEEE ICDM Workshop FIM Implementations*, 2003.

[10] T. Brijs**,** K. Vanhoof, and G. Wets, “Defining interestingness for association rules”, *International journal of information theories and applications, 10:4,* 2003.

[11] D. Burdick, M. Calimlim, and J. Gehrke, "MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases", In *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, April 2001.

[12] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, “VIPS: a vision based page segmentation algorithm”, *Technical Report MSR-TR-2003-79, Microsoft*, 2003.

[13] C. Carson, S. Belongie, H. Greenspan, and J. Malik, “Region-based image querying”, In *IEEE Workshop on Content-Based Access of Image and Video Libraries*, 1997.

[14] C. L. Carter, H. J. Hamilton and N. Cercone, "Share Based Measures for Itemsets", In *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery*, pp. 14-24, 1997.

[15] V. Chandola, and V. Kumar, "Summarization - Compressing Data into an Informative Representation", *Knowledge and Information Systems*, Volume 12, No. 3, pp. 355-378, August 2007.

[16] C. Clifton, R. Coolie, and J. Rennie, "TopCat: Data Mining for Topic Identification in a Text Corpus", *IEEE Transactions on Knowledge and Data Engineering*, Volume 16, No. 8, pp. 949-964, August 2004.

[17] Cluto, http://glaros.dtc.umn.edu/gkhome/views/cluto.

[18] F. Coenen, "The LUCS-KDD Implementations of the FOIL, PRM, and CPAR algorithms", http://www.csc.liv.ac.uk/~frans/KDD/Software.

[19] W. Cohen, "Fast effective rule induction", In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms, 2nd Edition", *McGraw Hill / MIT Press*, ISBN: 0-07-013151-1.

[21] W. Dai, G.-R. Xue, Q. Yang, and Y. Yu, "Co-clustering based classification for out-of-domain documents", In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 210-219, 2007.

[22] The Open Directory, http://dmoz.org/.

[23] G. Dong, X. Zhang, L. Wang and J. Li, "CAEP: Classification by aggregating emerging patterns", *Discovery Science*, Volume 1721, 1999.

[24] S. Dumais, J. Platt, D. Heckerman and M. Sahami, "Inductive Learning Algorithms and Representations for Text Categorization", In *Proceedings of the seventh international conference on Information and knowledge management*, pp. 148-155, 1998.

[25] B. Fung, K. Wang, and M. Ester, "Hierarchical document clustering using frequent itemsets", In *Proceedings of the SIAM International Conference on Data Mining,* pp. 59-70, 2003.

[26] E. Gabrilovich and S. Markovitch, "Text Categorization with Many Redundant Features: Using Aggressive Feature Selection to Make SVMs Competitive with C4.5", In *Proceedings of the Twenty-First International Conference on Machine Learning*, pp. 321-328, 2004.

[27] L. Geng and H. J. Hamilton, "Interestingness Measures for Data Mining: A Survey", *ACM Computing Surveys*, Volume 38, No. 3, September 2006.

[28] B. Goethals, "Efficient Frequent Pattern Mining", *PhD thesis*, *Transnational University of Limburg*, Belgium, 2002.

[29] K. Gouda, and M. J. Zaki, "Efficiently mining maximal frequent itemsets", In *Proceedings of the 1st IEEE International Conference on Data Mining*, Nov. 2001.

[30] L. Gravano, P. Ipeirotis, and M. Sahami, "QProber: A System for Automatic Classification of Hidden-Web Databases", *ACM Transactions on Information Systems,* Volume 21, No. 1, Jan. 2003.

[31] H. Haddad, P. Mulhem, **"**Association Rules for Symbolic Indexing of Still Images", In *Proceedings of the 2001 International Conference on Artificial Intelligence*, June 2001.

[32] S. Haddadi, "A note on the NP-hardness of the consecutive block minimization problem", *International Transactions in Operational Research*, Volume 9, No. 6, 2002.

[33] E.-H. Han, G. Karypis, and V. Kumar, “Clustering in a high-dimensional space using hypergraph models”, *Technical Report 97-063, University of Minnesota*, 1998.

[34] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher, “Clustering based on association rule hypergraphs”, *Research Issues on Data Mining and Knowledge Discovery*, 1997.

[35] J. Han, J. Pei, Y. Yin and R. Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach”, *Data Mining & Knowledge Discovery*, 2004.

[36] X. He, D. Cai, J.-R. Wen, W.-Y Ma, and H.-J. Zhang, "ImageSeer: Clustering and Searching WWW Images Using Link and Page Layout Analysis", *Technical Report MSR-TR-2004-38, Microsoft*, 2004.

[37] H. S. Heaps, "Information Retrieval - Computational and Theoretical Aspects", *Academic Press*, 1978.

[38] W. Hsu, M. L. Lee, and Ji. Zhang, “Image Mining: Trends and Developments”, in *Journal of Intelligent Information System (JISS): Special Issue on Multimedia Data Mining*, *Kluwer Academic*, 2002.

[39] X. Hu, T.Y. Lin, and E. Louie, "Bitmap techniques for optimizing decision support queries and association rule algorithms", In *Proceedings of the Seventh International Database Engineering and Applications Symposium*, 2003.

[40] T. Johnson, "Performance Measurements of Compressed Bitmap Indices", In *Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 278-289*,* 1999.

[41] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, “Hypergraph partitioning: Applications in VLSI domain”, *Technical Report TR-96-060, University of Minnesota,* 1996.

[42] G. Karypis, and V. Kumar, “hMETIS user manual”, http://www-users.cs.umn.edu/~karypis/metis/hmetis.

[43] L. Kaufman, and P. J. Rousseeuw, “Finding Groups in Data: An Introduction to Cluster Analysis”, *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics*, New York, March 1990.

[44] L.T. Kou, “Polynomial complete consecutive information retrieval problems”, *SIAM Journal on Computing*, Volume 6, 1977.

[45] A. Kyriakopoulou, and T. Kalamboukis, "Using clustering to enhance text classification", In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007.

[46] J. Li, G. Dong, K. Ramamohanarao and L. Wong, "DeEPs: A New Instance based Discovery and Classification System", *Machine Learning*, Volume 54, No. 2, 2004.

[47] J. Li, and Y. Zhang, "Direct Interesting Rule Generation", In *Proceedings of the Third IEEE International Conference on Data Mining,* 2003.

[48] W. Li, J. Han and J. Pei, "CMAR: Accurate and Efficient Classification based on multiple class-association rules", In *Proceedings of the First IEEE International Conference on Data Mining*, 2001.

[49] R. Lienhart, and A. Hartmann, “Classifying images on the web automatically”, *Journal of Electronic Imaging,* Volume 11, No. 4, pp. 445-454, Oct 2002.

[50] B. Liu, W. Hsu and Y. Ma, "Integrating Classification and Association Rule Mining", In *Proceedings of the Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998.

[51] B. Liu, W. Hsu, and Y. Ma, “Pruning and Summarizing the Discovered Associations”, In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

[52] B. Liu, W. Hsu, S. Chen and W.-Y. Ma, "Analyzing the Subjective Interestingness of Association Rules", *IEEE Intelligent Systems,* Volume 15, No. 5, 2000.

[53] Y. Liu, D. Zhang, G. Lu and W.-Y. Ma, "Region-Based Image Retrieval with High-Level Semantic Color Names", In Proceedings *of the 11th International Multimedia Modeling Conference,* 2005.

[54] C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets", *IEEE Transactions on Knowledge and Data Engineering*, Volume 18, No. 1, Jan. 2006.

[55] H. H. Malik, and J. R. Kender, "Clustering web images using association rules, interestingness measures, and hypergraph partitions", In *Proceedings of the Sixth International Conference on Web Engineering,* 2006.

[56] H. H. Malik, and J. R. Kender, "High Quality, Efficient Hierarchical Document Clustering Using Closed Interesting Itemsets", In *Proceedings of the Sixth IEEE International Conference on Data Mining*, pp. 991-996, 2006.

[57] H. D. K. Moonesinghe, S. Fodeh, and P.-N. Tan, "Frequent Closed Itemset Mining Using Prefix Graphs with an Efficient Flow-Based Pruning Strategy", In *Proceedings of the Sixth IEEE International Conference on Data Mining*, pp. 75-86, 2006.

[58] 20 Newsgroups, http://people.csail.mit.edu/jrennie/20Newsgroups/.

[59] Ohsumed, ftp://medir.ohsu.edu/pub/ohsumed.

[60] C. Ordonez, and E. Omiecinski, “Discovering Association Rules based on Image Content”, In *Proceedings of the IEEE Forum on Research and Technology issues in Digital Libraries*, 1999.

[61] C. D. Paice, "Another Stemmer", *ACM SIGIR Forum,* Volume 24, No. 3*,* pp. 56-61, 1990.

[62] L. Parsons, E. Haque, and H. Liu, “Subspace Clustering for High Dimensional Data: A Review”, *ACM SIGKDD Explorations Newsletter*, Volume 6, No. 1, 2004.

[63] F. Pereira, N. Tishby, and L. Lee, "Distributional clustering of English words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993.

[64] A. Pinar, T. Tao and H. Ferhatosmanoglu, “Compressing Bitmap Indices by Data Reorganization”, In *Proceedings of the 21st International Conference on Data Engineering,* 2005.

[65] J. Quinlan, "C4.5: Programs for Machine Learning", *Morgan Kaufman*, ISBN:1-55860-238-0, 1993.

[66] J. Quinlan and R. Cameron-Jones, "FOIL: A Midterm Report", In *Proceedings of the European Conference on Machine Learning*, 1993.

[67] Reuters. http://kdd.ics.uci.edu/databases/reuters21578.

[68] B. Raskutti, H. Ferr, and A. Kowalczyk, "Using unlabeled data for text classification through addition of cluster parameters", In *Proceedings of the 9th International Conference on Machine Learning*, 2002.

[69] C. J. van Rijsbergen, "Information Retrieval", *Butterworths, London*, 1979.

[70] SMART Project (eds.) Stopword List for English Information Retrieval, http://www.unine.ch/info/clef/englishST.txt.

[71] J. A. Rushing, H. S. Ranganath, and T. H. Hinke, **"**Using Association Rules as Texture Features", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Volume 23, No. 8, 2001.

[72] F. Sebastiani, "Machine learning in automated text categorization", *ACM Computing Surveys*, Volume 34, No. 1, 2002.

[73] B. Shekar and R. Natarajan, "A Transaction-based Neighborhood-driven Approach to Quantifying Interestingness of Association Rules", In *Proceedings of the Fourth IEEE International Conference on Data Mining*, 2004.

[74] N. Slonim, and N. Tishby, "The power of word clustering for text classification", In *Proceedings of the European Colloquium on IR Research*, 2001.

[75] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques", In *Proceedings of the Workshop on Text Mining, 6th ACM SIGKDD International Conference on Data Mining*, 2000.

[76] P. Tan, V. Kumar, and J. Srivastava, "Selecting the right interestingness measure for association patterns", In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 32–41, 2002.

[77] P. Tan, and V. Kumar, "Interestingness measures for association patterns: A perspective", In *Proceedings of the KDD Workshop on Postprocessing in Machine Learning & Data Mining*, 2000.

[78] V. N. Vapnik, "Statistical Learning Theory", *Wiley*, ISBN: 0-47-103003-1, 1998.

[79] F. Verhein, and S. Chawla, "Geometrically Inspired Itemset Mining", In *Proceedings of the Sixth IEEE International Conference on Data Mining*, pp. 655-666, 2006.

[80] J. Wang, and G. Karypis, "SUMMARY: Efficient Summarizing Transactions for Clustering", In *Proceedings of the Fourth IEEE International Conference on Data Mining*, 2004.

[81] J. Wang and G. Karypis, "On Mining Instance-Centric Classification Rules", *IEEE Transactions on Knowledge and Data Engineering*, Volume 18, No. 11, 2006.

[82] K. Wu, E. J. Otoo, A. Shoshani, and H. Nordberg, "Notes on design and implementation of compressed bit vectors", *Technical Report LBNL/PUB-3161*, *Berkeley*, CA.

[83] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression", *ACM Transactions on Database Systems*, Volume 31, No. 1, pp. 1-38, 2006.

[84] K. Wu, E. J. Otoo, and A. Shoshani, "On the Performance of Bitmap Indices for High Cardinality Attributes", In *Proceedings of the Thirtieth international conference on Very large data bases*, pp. 24-35, 2004.

[85] H. Xiong, M. Steinbach, P.-N. Tan, and V. Kumar, "HICAP: Hierarchical Clustering with Pattern Preservation", In *Proceedings of the SIAM International Conference on Data Mining*, 2004.

[86] D.-Y. Yang, A. Johar, A. Grama, and W. Szpankowski, "Summary structures for frequency queries on large transaction sets**",** In *Proceedings of the Data Compression Conference,* pp. 420-429, 2000.

[87] X. Yin and J. Han, "CPAR: Classification based on Predictive Association *Rules*", In *Proceedings of the SIAM International Conference on Data Mining*, 2003.

[88] H. Yu, D. Searsmith, X. Li and J. Han, "Scalable Construction of Topic Directory with Nonparametric Closed Termset Mining", In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pp. 563-566, 2004.

[89] H. J. Zeng, X. H. Wang, Z. Chen, H. Lu, and W. Y. Ma, "CBC: Clustering based text classification requiring minimal labeled data", In *Proceedings of the Third IEEE International Conference on Data Mining*, 2003.

[90] Y. Zhao, and G. Karypis, “Evaluation of hierarchical clustering algorithms for document datasets”, In *Proceedings of the International Conference on Information and Knowledge Management*, pp. 515-524, November 2002.

[91] Y. Zhao, and G. Karypis, “Hierarchical Clustering Algorithms for Document Datasets”, *Data Mining and Knowledge Discovery*, Volume 10, pp. 141-168, No. 2, 2005.