Mining Statistically Important Equivalence Classes and Delta-Discriminative Emerging Patterns

Jinyan Li*
Institute for Infocomm Research (I^{2}R) &
Nanyang Technological University, Singapore
jyli@ntu.edu.sg

Guimei Liu   Limsoon Wong
National University of Singapore
3 Science Drive 2, Singapore
{liugm,wongls}@comp.nus.edu.sg

ABSTRACT

The support-confidence framework is the most common measure used in itemset mining algorithms, for its antimonotonicity that effectively simplifies the search lattice. This computational convenience brings both quality and statistical flaws to the results as observed by many previous studies. In this paper, we introduce a novel algorithm that produces itemsets with ranked statistical merits under sophisticated test statistics such as chi-square, risk ratio, odds ratio, etc. Our algorithm is based on the concept of *equivalence classes*. An equivalence class is a set of frequent itemsets that always occur together in the same set of transactions. Therefore, itemsets within an equivalence class all share the same level of statistical significance regardless of the variety of test statistics. As an equivalence class can be uniquely determined and concisely represented by a closed pattern and a set of generators, we just mine closed patterns and generators, taking a simultaneous depth-first search scheme. This parallel approach has not been exploited by any prior work. We evaluate our algorithm on two aspects. In general, we compare to LCM and FPclose which are the best algorithms tailored for mining only closed patterns. In particular, we compare to epMiner which is the most recent algorithm for mining a type of relative risk patterns, known as minimal emerging patterns. Experimental results show that our algorithm is faster than all of them, sometimes even multiple orders of magnitude faster. These statistically ranked patterns and the efficiency have a high potential for real-life applications, especially in biomedical and financial fields where classical test statistics are of dominant interest.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data mining*

General Terms

Algorithms, Performance

Keywords

Equivalence classes, itemsets with ranked statistical merit

1. INTRODUCTION

Given a database consisting of classes of transactions, it is important to find frequent itemsets that are capable of making statistically fine distinctions between the classes, *e.g.* in the fields of evidence-based medicine, sequence motif detection, financial stocks portfolio construction, risk management, etc. The statistical significance of these frequent itemsets can be measured by various test statistics such as the widely known chi-square, relative risk, odds ratio, and so on [28, 30]. However, there have been a lack of efficient algorithms that can find and rank statistically important patterns from large datasets that meet a certain significance threshold. Many itemsets mining algorithms using support and confidence as thresholds cannot accomplish this—sometimes, they even cause statistical flaws to the results [1, 6, 10]. As an innovative work on statistical data mining, Li et al [15] proposed a so-called "plateau-redundance" theorem which proves that many "plateaus" of itemsets can be located at the same level of significance under relative risk and odds ratio. However, the work [15] did not examine efficient mining and implementation details; also whether the method can handle other test statistics remains unknown. Tan et al [25] conducted an excellent comparative study to select the right interestingness measure for association rules, but their *random-and-disjoint* algorithm is heavily dependent on subjective domain expertise and it is also computationally expensive. As a type of relative risk patterns, minimal emerging patterns [9, 17, 24, 20, 11] have long been proposed to capture statistically important difference between two classes, or trends from temporal databases. However, all known algorithms for mining emerging patterns cannot handle multi-class data efficiently and have not noticed the redundance problem in the minimal patterns. Therefore, in this paper, we investigate (*i*) how to discover a concise representation of statistically important itemsets to get rid of the redundance, (*ii*) how to easily unify the method for a wide range of test statistics, and (*iii*) how to easily extend to handling multi-class data.

The key idea of our method is to mine a concise representation of *equivalence classes* [4] of frequent itemsets [2] from a transactional database $\mathcal{D}$.

[*] Jinyan was full-time working at I^{2}R till June 30, 2007 where this work was done.

KDD'07, August 12–15, 2007, San Jose, California, USA.
Copyright 2007 ACM 978-1-59593-609-7/07/0008 ...$5.00.

An equivalence class EC is a set of itemsets that always occur together in some transactions of $\mathcal{D}$. That is, for all X and $Y \in EC$, $f_{\mathcal{D}}(X) = f_{\mathcal{D}}(Y)$, where $f_{\mathcal{D}}(Z) = \{T \in \mathcal{D} \mid Z \subseteq T\}$. Suppose $\mathcal{D}$ consists of two classes of data $\mathcal{D}_1$ and $\mathcal{D}_2$, let X be an itemset of $\mathcal{D}$, and assume $f_{\mathcal{D}}(X)$ consists of n_1 and n_2 transactions from $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Then n_1 and n_2 are sufficient information to calculate various test statistics of X. For instance, X's *relative risk* in D_i against D_j $(i \neq j)$ is $\frac{n_i/|\mathcal{D}_i|}{n_j/|\mathcal{D}_j|}$. Similarly, X's *odds ratio* between $\mathcal{D}_i$ and $\mathcal{D}_j$ $(j \neq i)$ is

$$\frac{n_i/\left(|\mathcal{D}_i| - n_i\right)}{n_j/\left(|\mathcal{D}_j| - n_j\right)}.$$

By definition of equivalence classes, it is not difficult to see that for any other itemset in the same equivalence class of X, that itemset is at the same level of significance as X under a given test statistics.

Equivalence classes have a nice property that: An equivalence class EC can be uniquely and concisely represented by a *closed pattern* [21] C and a set $\mathcal{G}$ of *generators* [21, 4], in the form of $EC = [\mathcal{G}, C]$, where $[\mathcal{G}, C] = \{X \mid$ there is $G \in \mathcal{G}, G \subseteq X \subseteq C\}$. Therefore, just closed patterns and generators of $\mathcal{D}$ are sufficient to guarantee completeness—all and exactly those frequent itemsets meeting the specified statistical significance threshold can be deterministically identified. One could use the best algorithms for mining closed patterns [26, 12, 27, 32] and generators [7, 16] separately, and then combine their outputs to solve the problem. However, this brute-force approach has to access to the database at least twice, and incurs a large overhead to properly pair up the closed patterns with their corresponding generators. To achieve high efficiency, we mine closed patterns and generators of $\mathcal{D}$ simultaneously under one depth-first search framework. This novel algorithm is even faster than state-of-the-art closed pattern mining algorithms such as LCM [26] (version 3) and FPclose [12] that do not output generators at the same time.

As a more challenging problem in this work, we investigate how to pinpoint a special subtype of equivalence classes called non-redundant *δ-discriminative* equivalence classes. This concept is an extension to *jumping* emerging patterns [9, 17] (recently also known as minimal emerging patterns [20, 11]). Using this concept, we not only can identify patterns with strong relative risk, but also can group the minimal emerging patterns into equivalence classes, reducing the redundance to a minimum degree. As those δ-discriminative equivalence classes are rarely located at the bottom, while very often at the middle, of the search lattice, breadth-first search algorithms would cost huge amount of memory and examine many unnecessary candidates. We refine our main algorithm to skip all low-ranked equivalence classes, so that our algorithm can produce the results with high efficiency. We compare it to the state-of-the-art epMiner algorithms [20, 11]; experimental results show that our algorithm can be multiple orders of magnitude faster for the same mining tasks.

Contribution by this paper: (1) We focus on the mining of a concise representation of statistically important equivalence classes with ranked merit from large datasets. Our work is contrast to the pioneer CBA method by Liu *et al.* [18] or its variants which mine individual classification rules measured by only support and confidence. By our approach, redundant patterns can be greatly reduced, and sophisticated test statistics can be applied. (2) We mine closed patterns and generators simultaneously under one depth-first search scheme. The efficiency is even higher than the best algorithms that are tailored for mining only closed patterns. We are the first to propose such a simultaneous mining. (3) Our algorithm can be easily unified for a wide range of test statistics, and can discover and rank statistically important patterns in one go. This is very helpful to find those top-ranked patterns according to various combinations of statistical merits. (4) Our algorithm and results will be helpful for those machine learning applications containing multiple classes of data. This is because traditional learning methods [3, 14, 31, 22] have to organize the multiple classes of data through exhaustive pairwise coupling approaches (one-vs-one or all-vs-all), leading to very heavy computational redundancy. However, our mining algorithm does not require any pairwise coupling of data.

Paper organization: Section 2 formally defines the research problems. Section 3 presents our computational steps to solve the problems. Section 4 describes details of the mining algorithm. Section 5 reports a performance evaluation of our algorithms in comparison with other methods. Section 6 discusses related works. Section 7 concludes the paper.

2. PROBLEM FORMULATION

We consider n *items* i_1, i_2, ..., i_n. A *dataset* is a set of *transactions*. A transaction is a non-empty set of items. An *itemset* is a set of items. The *support* of an itemset P in a data set $\mathcal{D}$, denoted $sup(P, \mathcal{D})$, is the percentage of transactions in $\mathcal{D}$ that contain P. The *negated* support of P in $\mathcal{D}$ is $\overline{sup}(P, \mathcal{D}) = 1 - sup(P, \mathcal{D})$.

Problem 1: Given k non-empty classes (datasets $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_k$) of transactions, we investigate how to

- Concisely represent *frequent itemsets* (for a support threshold ms) that have the same level of statistical significance under a given test statistics;
- Efficiently discover the concise representations with ranked statistical merit for all of the k classes.

For a pair of datasets $\mathcal{D}_p$ and $\mathcal{D}_n$, we assume that a test statistics on an itemset X, denoted $st(X, \mathcal{D})$ where $\mathcal{D} = \mathcal{D}_p \cup \mathcal{D}_n$, is a function of just the supports of X in $\mathcal{D}_p$, $\mathcal{D}_n$, and/or in $\mathcal{D}$. That is, $st(X, \mathcal{D}) = h(sup(X, \mathcal{D}_p), sup(X, \mathcal{D}_n), sup(X, \mathcal{D}))$, for some function h. This assumption is not very restrictive, as it is sufficiently general to cover a broad range of widely used test statistics. For example, under the odds ratio (OR) test:

$$st^{OR}(X, \mathcal{D}) = \frac{sup(X, \mathcal{D}_p)/\overline{sup}(X, \mathcal{D}_p)}{sup(X, \mathcal{D}_n)/\overline{sup}(X, \mathcal{D}_n)}$$

A chi-square test with 1 degree of freedom is:

$$st^{\chi^2}(X, \mathcal{D}) = \frac{(x * \overline{y} - y * \overline{x})^2 * |\mathcal{D}|}{|\mathcal{D}_p| * |\mathcal{D}_n| * (x + y) * (\overline{x} + \overline{y})}$$

where $x = sup(X, \mathcal{D}_p) * |\mathcal{D}_p|$, $y = sup(X, \mathcal{D}_n) * |\mathcal{D}_n|$, $\overline{x} = (1 - sup(X, \mathcal{D}_p)) * |\mathcal{D}_p|$, and $\overline{y} = (1 - sup(X, \mathcal{D}_n)) * |\mathcal{D}_n|$. A longer list of statistical tests allowable under our assumption includes relative risk test, Student's t, Fisher sign test, likelihood ratio, Yule's Q, and Yule's Y [28, 30, 25].

Only frequent itemsets are in our consideration, because infrequent itemsets are likely to be random noise patterns, and the inclusion of infrequent itemsets would much enlarge the size of mining results which is deemed not unfavorable to the end users. We also note that the ranking of itemsets under different statistical tests can be quite different. Their intersections certainly contain some itemsets that possess many statistical merits. Different statistics may also be better suited for different applications—e.g., OR is applicable to case-control studies, while relative risk is applicable to cohort studies [30]. Thus we are interested in unifying our mining method to cover various test statistics.

Problem 1 can be reduced to the mining of closed patterns [21] and generators [21, 4] of frequent equivalence classes [4] from $\mathcal{D}$.

Definition 1. *Let X be an itemset of a dataset $\mathcal{D}$. The* equivalence class *of X in $\mathcal{D}$, denoted $[X]_{\mathcal{D}}$, is the set $\{A \mid f_{\mathcal{D}}(A) = f_{\mathcal{D}}(X)\}$, where $f_{\mathcal{D}}(Z) = \{T \in \mathcal{D} \mid Z \subseteq T\}$. The maximal itemset and the minimal itemsets of $[X]_{\mathcal{D}}$ are called the closed pattern and the generators of this equivalence class respectively. The closed patterns and generators of $\mathcal{D}$ are all the closed patterns and generators of their equivalence classes.*

Property 1. *Let C be the closed pattern of an equivalence class EC and G be a generator of EC. Then all itemset X satisfying $G \subseteq X \subseteq C$ are also in this equivalence class.*

This *convexity* property says that the entire equivalence class can be concisely bounded as $EC = [\mathcal{G}, C]$, where $\mathcal{G}$ is the set of generators of EC, C is the closed pattern, and $[\mathcal{G}, C] = \{X \mid \text{there is } G \in \mathcal{G}, G \subseteq X \subseteq C\}$.

It is easy to see that equivalence classes do not overlap.

Property 2. *Let $\mathcal{D}$ be a dataset. Let EC_1 and EC_2 be distinct equivalence classes of $\mathcal{D}$. Then $EC_1 \cap EC_2 = \{\}$.*

Corollary 1. *Let $\mathcal{D}$ be a data set and ms be a minimum support threshold. Then the set $\mathcal{F}_{\mathcal{D}}^{ms}$ of frequent itemsets of $\mathcal{D}$ can be exactly partitioned into frequent equivalence classes without overlapping. That is,*

$$\mathcal{F}_{\mathcal{D}}^{ms} = \bigcup_{i=1}^{m} [C_i]_{\mathcal{D}} = \bigcup_{i=1}^{m} [\mathcal{G}_i, C_i]$$

where $C_1, C_2, \ldots, C_m$ are m frequent closed patterns of $\mathcal{D}$, $\mathcal{G}_i$ is the set of generators of $[C_i]_{\mathcal{D}}$.

For $|[C_i]_{\mathcal{D}}| = 1$—i.e., only one itemset in this equivalence class—we simplify $[\{C_i\}, C_i]$ as C_i. Therefore, $\bigcup_{i=1}^{m} [\mathcal{G}_i, C_i]$ is a true concise and lossless representation of $\mathcal{F}_{\mathcal{D}}^{ms}$.

As we assume that the statistical significance of an itemset X is a function h of X's global and local support information, the class label information of the transactions that contain X are important. We refer to this information as the *class label distribution* of X, denoted $n_i(X) = |f_{\mathcal{D}}(X) \cap \mathcal{D}_i|, i = 1, 2, \cdots, k$. As all itemsets within an equivalence class have the same class label distribution, it is easy to see the following property:

Property 3. *All itemsets within an equivalence class share the same level of statistical significance.*

Therefore, Problem 1 can be solved by mining the generators and closed patterns of frequent equivalence classes, and their associated class label distribution information.

Example 1. *Suppose we are given 4 transactions categorized into two classes (class 1 and class 2). Class 1 has two transactions $\{a, b, c\}$ and $\{a, b, c, d\}$; Class 2 has transactions $\{u, v, w\}$ and $\{u, v, w, x\}$. Let the minimum support threshold be 30%. Then there are two frequent closed patterns: abc and uvw. Their corresponding equivalence classes are: $EC_1 = [\{a, b, c\}, abc]$ and $EC_2 = [\{u, v, w\}, uvw]$, which concisely represent the total 14 frequent itemsets (not including the empty set). The class label distribution of abc is: $n_1(abc) = 2$ and $n_2(abc) = 0$; while the class label distribution of uvw is: $n_1(uvw) = 0$ and $n_2(uvw) = 2$. Thus, the statistical importance of EC_1 and EC_2 can be calculated straightforward.*

Problem 2: As a special and most useful case of Problem 1, we also investigate how to directly output a subtype of equivalence classes that are *δ-discriminative* and *non-redundant*.

Definition 2. *Let $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$. Let EC be a frequent equivalence class of $\mathcal{D}$, and C be the closed pattern in EC. Let $n_i(C), i = 1, 2, \cdots, k$, be the class label distribution information of C. Then EC is a δ-discriminative equivalence class if there is $i \in \{1, 2, \cdots, k\}$ such that $\sum_{j \neq i} n_j(C) \leq \delta$, where δ is usually a small integer number 1 or 2.*

In other words, every itemset in a δ-discriminative equivalence class occurs in only one of the k classes with almost no occurrence in any other classes. The following table illustrates an example, where every itemset in EC shows an outstanding occurrence in $\mathcal{D}_1$, while few occurrence in any of the other classes, exhibiting a sharp difference between $\mathcal{D}_1$ versus all other classes.

$\lvert f_{\mathcal{D}}(X) \rvert$	$n_1(X) = \lvert f_{\mathcal{D}}(X) \cap \mathcal{D}_1 \rvert \geq ms * \lvert \mathcal{D} \rvert$
	$\sum_{j \neq 1} \lvert f_{\mathcal{D}}(X) \cap \mathcal{D}_j \rvert = 0$, or 1

The definition of δ-discriminative equivalence classes conceptually differs from jumping emerging patterns (JEPs) [17, 9, 20, 11]. A JEP is an itemset which occurs in one class, but never occurs in the other class. Thus, our notion looks at the itemsets in groups, but the notion of JEPs treats them individually; We highlight the redundancy within an equivalence class, while JEPs did not notice the hidden redundancy problem. Also, itemsets in a δ-discriminative equivalence class are required to be frequent, but JEPs may be infrequent.

Itemsets in a δ-discriminative equivalence class often have a statistical value of infinity under the odds ratio test or the relative risk test. We use $EC_{\mathcal{D}}^{\infty}$ to denote all δ-discriminative equivalence classes of $\mathcal{D}$.

Definition 3. *Let $\mathcal{D}$ be a dataset. Let $EC_1, EC_2 \in EC_{\mathcal{D}}^{\infty}$ be two equivalence classes of $\mathcal{D}$ satisfying $C_1 \subset C_2$, where C_1 and C_2 are the closed patterns of EC_1 and EC_2 respectively. Then EC_2 is said to be a redundant δ-discriminative equivalence class with respect to EC_1.*

This notion of redundancy makes sense, because $C_1 \subset C_2$ and thus $f_{\mathcal{D}}(C_2) \subset f_{\mathcal{D}}(C_1)$ by definition. In other words, C_2's transaction set is entirely subsumed by C_1's transaction set. This is true for every itemset in EC_2. Therefore, EC_2 can be considered as a redundant equivalence class.

It follows that only the most general (minimal) equivalence classes of $EC_{\mathcal{D}}^{\infty}$ are non-redundant. Here equivalence classes are ordered so that an equivalence class EC_1 is said to be more general than another equivalence class EC_2 if EC_1's closed pattern is more general than that of EC_2.

EXAMPLE 2. *Following Example 1, let the minimum support threshold be* 10%. *Then there are 4 frequent equivalence classes:* $EC_1 = [\{a, b, c\}, abc]$, $EC_1' = [\{d\}, abcd]$, $EC_2 = [\{u, v, w\}, uvw]$, *and* $EC_2' = [\{x\}, uvwx]$. *Observe that all the 4 equivalence classes are δ-discriminative, because each of them has a zero occurrence in either Class 1 or Class2. However, only EC_1 and EC_2 are non-redundant: EC_1' is redundant to EC_1' because $abcd \supset abc$, and EC_2' is redundant to EC_2 because $uvwx \supset uvw$.*

Non-redundant δ-discriminative equivalence classes are often located in the middle of the search lattice. Thus, breadth-first search algorithms would cost huge amount of memory and examine many unnecessary candidates. We refine our main algorithm (solving Problem 1) to solve Problem 2.

3. COMPUTATIONAL STEPS

Given k non-empty classes of transactions—$\mathcal{D}_1$, $\mathcal{D}_2$, ..., and $\mathcal{D}_k$—and a support threshold ms, our method to discover and rank statistically important equivalence classes for the k classes of data (Problem 1) consists of the following 5 steps:

1. Let $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$.
2. Mine frequent closed patterns and generators to concisely represent all frequent equivalence classes of $\mathcal{D}$.
3. For every frequent closed pattern X, determine the class label distribution information of $f_{\mathcal{D}}(X)$, namely $\{n_i(X) = |f_{\mathcal{D}}(X) \cap \mathcal{D}_i|, i = 1, 2, \cdots, k\}$.
4. For a test statistics h and a pair of datasets $\mathcal{D}_p$ and $\mathcal{D}_n$, calculate $st^h(X, \mathcal{D}_p \cup \mathcal{D}_n)$ for every closed pattern X, where $\mathcal{D}_p$ or $\mathcal{D}_n$ is the union of some of $\mathcal{D}_i$, and $\mathcal{D}_p \cap \mathcal{D}_n = \emptyset$.
5. Rank all X as per $st^h(X, \mathcal{D}_p \cup \mathcal{D}_n)$, and output significant equivalence classes as $[\mathcal{G}_X, X]$, where $st^h(X, \mathcal{D}_p \cup \mathcal{D}_n)$ meets a pre-specified significance threshold, and $\mathcal{G}_X$ is the corresponding set of generators.

The main computational part is at Step 2 for mining closed patterns and generators from $\mathcal{D}$. Step 3 can be actually integrated into Step 2. Steps 4–5 are repeatable when different test statistics and different class label pairs are specified. This repetition does not require access to $\mathcal{D}$. Therefore, our method can be easily extended to unify a broad range of test statistics without much extra computational cost. The completeness of the algorithm is guaranteed at Step 2 by Corollary 1. All patterns at the same level of statistical significance are concisely represented as shown in Step 5. Thus, our method is indeed a unified style, accessing the data only once, and yet providing sufficient information for determining the concise representation of the statistically important patterns, and for calculating the multiple test statistics in one run. Our method also handles multi-class data.

To solve Problem 2, we refine Step 2. The implementation is easy as we adopt a depth-first search strategy on a set-enumeration tree [23]—we just stop the depth-first search along a branch whenever we reach a δ-discriminative equivalence class. This refinement can save much cost as low-ranked redundant equivalence classes are not touched in the entire mining process. Details are described in Section 4.2.

4. OUR EFFICIENT ALGORITHMS

This section presents our novel ideas for the simultaneous mining of closed patterns and generators on top of a modified data structure of the seminal FP-tree [13]. The FP-tree structure, originally proposed for efficiently mining frequent itemsets [13], has also been used before to mine frequent closed patterns by FPclose [12]. FPclose uses conditional FP-trees to generates candidate closed patterns, then these candidates are inserted into a separate *pattern tree* to check whether they are true closed patterns or not. The critical drawback of FPclose is the separate pattern tree structure which consumes a large amount of memory and which is also a bottleneck for speeding up the efficiency. However, we modify FP-tree in such a way that we can directly output closed patterns one-by-one, without an additional data structure to check whether the patterns are true closed patterns or not. Our novel modifications also include a smaller head table, a smaller projected database, and a new subtree structure storing all *full-support* items. With these subtle and effective modifications, our algorithm is capable of a simultaneous mining of both closed patterns and generators in a high efficiency. The mining speed is even faster than existing best algorithms for mining closed patterns alone. Our algorithm is termed as DPMiner (**d**iscriminative **p**attern **m**iner).

4.1 Mining Equivalence Classes

A brute-force method to mine equivalence classes is to use one of the best closed pattern mining algorithms [32, 27, 19, 12, 26] and the best generator mining algorithms [7, 16], and then associate those generators with their closed patterns. However, this approach explores the search space twice, and has a considerable overhead to match up the closed patterns and generators, as mentioned in Introduction.

DPMiner speeds up the efficiency by integrating the mining of closed patterns and generators into one depth-first framework. It constructs a conditional database for each frequent generator; A tail sub-tree is maintained together with these conditional databases to generate the closure of each generator. The anti-monotonicity [21] of generators (two versions), reviewed below, are frequently exploited in the mining:

PROPERTY 4. *Let $\mathcal{D}$ be a dataset and FG be the set of frequent generators of $\mathcal{D}$. (1) If $X \in FG$, then $Y \in FG$ for all $Y \subseteq X$. (2) If $X \notin FG$, then $Y \notin FG$ for all $Y \supseteq X$.*

PROPERTY 5. *Let X be a frequent itemset of $\mathcal{D}$. X is a generator if and only if $sup(X, \mathcal{D}) < sup(Y, \mathcal{D})$ for every immediate proper subset Y of X.*

4.1.1 Revised FP-Tree for Pruning Non-Generators

In a normal FP-tree [13], all frequent items are sorted into a descending frequency order in the head table, our modification is that frequent items with a *full support*—i.e., those items that appear in every transaction of the original database or conditional projected databases—must be removed from the head table of FP-trees. The reason is that

Table 1: The projected database for $F = \{\mathbf{d}{:}\ 5, \mathbf{b}{:}\ 4, \mathbf{a}{:}\ 4, \mathbf{c}{:}\ 3, \mathbf{h}{:}\ 3, \mathbf{i}{:}\ 2\}$.

TID	Transactions	Projected Transactions
1	a, b, c, d, e, g, h	d, b, a, c, h
2	a, b, d, e, f, h	d, b, a, h
3	b, c, d, e, h, i	d, b, c, h, i
4	a, c, d, e, m	d, a, c
5	a, d, e, n	d, a
6	b, e, i, o	b, i

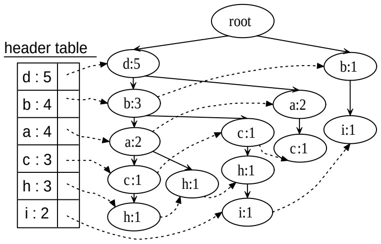


Figure 1: An example of our FP-trees

they and those itemsets containing them are not generators, due to the anti-monotone property of generators. This modification often leads to very small tree structures.

An example of our modified FP-tree is shown in Figure 1, which is constructed from a database in Table 1 containing 6 transactions. The minimum support threshold ms is set to 30%. The frequent item e appears in every transaction, it is thus not allowed to be in the head table. Please refer to [13] for detailed steps for constructing projected transactions, node links, parent links, and conditional FP-trees. After our modified FP-tree is constructed, all subsequent mining are operated on top of the tree.

4.1.2 Deeper Pruning of Non-Generators

We make use of Property 5 for further pruning of non-generators. The idea is that an itemset X is a generator if and only if $sup(X, \mathcal{D}) < sup(Y, \mathcal{D})$ for every immediate proper subset Y of X. Therefore, X can be identified as a generator, or filtered out, by just comparing the support of X with that of X's (immediate) subsets. To be able to do this checking during the mining process, the subsets of an itemset must be discovered prior to that itemset. To ensure this ordering, we built conditional databases according to the descending frequency order of items in the head table.

We present details as follows. When mining a generator l's conditional database D_l, DPMiner first traverses D_l to find frequent items in D_l, denoted as F_l=$\{a_1, a_2, \cdots, a_m\}$, and removes each item a_i from F_l if $l \cup \{a_i\}$ is not a generator, and then constructs a new FP-tree which stores the conditional databases of the remaining items in F_l. More specifically, it checks whether $sup(l \cup \{a_i\}, D)$=$sup(l, D)$ is true for every item a_i in F_l. If $sup(l \cup \{a_i\}, D)$=$sup(l, D)$ is true for some item a_i in F_l, it means that $l \cup \{a_i\}$ is not a generator, and item a_i should be removed from F_l. This checking is performed immediately after all the frequent items in D_l are discovered and it incurs little overhead. DPMiner then checks whether there exists an itemset l' such that $l' \subset (l \cup \{a_i\})$ and $sup(l', D)$= $sup(l \cup \{a_i\}, D)$ for each remaining item a_i in F_l. It is not necessary to compare $l \cup \{a_i\}$ with all of its subsets. It is adequate to compare $l \cup \{a_i\}$ with its immediate subsets based on the anti-monotone property of generators.

DPMiner maintains the set of frequent generators that have been discovered so far in a hash table to facilitate the subset checking. The hash function hashes an itemset to an integer by: $H(l) = \sum_{i \in l} h(i) \bmod L_{table}$, and $h(i) = 2^{i \bmod 32} + i + 2^{order(i) \bmod 32} + order(i) + 1$, where l is a generator, $order(i)$ is the position of item i if the frequent items in the original database are sorted into descending frequency order, and L_{table} is the size of the hash table and it is a prime number. In the above hash function, both the id of an item and the position of an item in descending frequency order are used. It is to reduce the possibility that two different items are mapped into the same value. The reason being that the position of an item in descending frequency order depends on the frequency of the item and is independent of the id of the item. Our experiments show that this hash function is very effective in avoiding conflicts.

The FP-tree structure provides an additional pruning opportunity. If an itemset l's conditional database D_l contains only one branch, then there is no need to construct a new FP-tree from D_l even if there are more than one frequent items in D_l that can be appended to l to form frequent generators. The reason being that if D_l contains only one branch, then for any two items a_i and a_j in D_l, itemset $l \cup \{a_i, a_j\}$ cannot be a generator because $sup(l \cup \{a_i, a_j\}, D)$ = $\min\{sup(l \cup \{a_i\}, D), sup(l \cup \{a_j\}, D)\}$.

4.1.3 Generating Closed Itemsets Simultaneously

Generators in the same equivalence class have the same closure and DPMiner associates generators in the same class together via their common closure. DPMiner maintains the set of closed itemsets in a hash table during the mining process, and the hash function of the hash table is the same as the one used in the previous subsection. When a generator is identified, DPMiner generates its closure, and checks whether the closed itemset is already in the hash table. If it is true, then the generator is added to the equivalence class of the closed itemset. Otherwise, the closed itemset is inserted into the hash table to create a new equivalence class. Next we describe how to generate the closure of each generator.

The closure of a generator l, denoted as C_l, is the intersection of all the transactions containing l. That is, $C_l = \bigcap_{t \in \mathcal{D} \wedge l \subseteq t} t$. Every item i in $(C_l - l)$ satisfies $sup(l \cup \{i\}, \mathcal{D}) = sup(l, \mathcal{D})$, due to convexity of equivalence classes, so we call the items in $(C_l - l)$ *cover items with respect to* l. Let l' be the length-($|l|$-1) prefix of frequent generator l, $D_{l'}$ be the conditional database of l', $T_{l'}$ be the FP-tree constructed from $D_{l'}$, and a_i be the last item of l. Note that $l = l' \cup \{a_i\}$. The FP-tree $T_{l'}$ stores all conditional databases of the frequent items in $D_{l'}$. In FP-tree $T_{l'}$, any path containing a_i represents a set of transactions containing l. Correspondingly, any transaction containing l is stored in some branch of $T_{l'}$ and the branch must contain the item a_i. If an item appears in all the branches containing a_i and has the same support as a_i, then the item should be included in the closure of l. However, not all such items are in D_l as D_l contains only the upper portion of the branches containing a_i, that is, the portion between the nodes containing a_i and the root. To obtain all the cover items with respect to l, we need to traverse the lower portion of these branches as well, that is, the subtrees rooted at the nodes containing item a_i. We call these subtrees *tail subtrees of* l.

We use the example FP-tree shown in Figure 1 to illustrate closure generation. There are three transactions containing itemset c in the example database shown in Table 1. Correspondingly, there are three branches in the FP-tree shown in Figure 1 containing item c: $dbach : 1$, $dbchi : 1$, and $dac : 1$. To obtain all the cover items with respect to c, we need to access all the nodes in these three branches. The FP-tree nodes containing item c split each of the three branches into two parts. The upper portions of the three branches—$dbac$, dbc, and dac—form itemset c's conditional database D_c. Every node in D_c can be visited by following the node-links of item c as in the FP-growth algorithm. The lower portions of the three branches—$h : 1$, $hi : 1$, and ϕ—are the tail subtrees of itemset c. We use the depth-first strategy to traverse each tail subtree, and move to the next tail subtree following the node-links of item c.

If an item appears in l's closure, then the item must appear in l's supersets' closure. If an item a_j does not appear in l's closure but $l \cup \{a_j\}$ is frequent, it is still possible that a_j appears in l's supersets' closure. An item a_j can either appear in D_l or in the tail subtrees of l. Therefore, when constructing the FP-tree storing all the frequent items in D_l, we need to include the frequent items in the tail subtrees of l as well. The frequent items in the tail subtrees of l are not candidate extensions of l as a generator, so they are not put into the header table. In the FP-tree constructed from D_l, these items are always put after the items in the header table so that they can appear in the tail subtrees of l's supersets but they never appear in l's superset's conditional databases. We use an example to illustrate this. In Figure 1, there are two frequent items a and b in c's conditional database D_c. We need to construct an FP-tree from D_c. Item h does not appear in D_c but it is frequent in the tail subtrees of c, so we need to include h in the FP-tree constructed from D_c, and item h is put after item a and b in this FP-tree as shown in Figure 2. When we traverse the tail subtrees of cb, we find that item h is in the closure of cb.

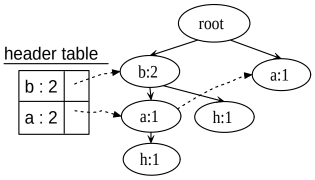


Figure 2: The FP-tree of itemset c with frequent tail items

To generate the closure of a generator l, DPMiner needs to traverse all the tail subtrees of l besides traversing D_l. As an FP-tree node is always in the tail subtrees of all of its ancestors, the number of times that an FP-node is additionally visited is equal to the number of ancestors of the FP-tree node. As we also need to include frequent tail items into FP-trees for closure generation, our FP-trees can be sometimes large.

To solve this problem, DPMiner identifies and prunes tail subtrees that do not contain any potential cover items as soon as possible. Let B be a branch in generator l's conditional database, l_B be the set of items contained in branch B, and T_B be the tail subtree rooted at the last node of branch B. If an item i in T_B is a cover item of l_1, where $l \subseteq l_1 \subseteq l \cup l_B$, then the support of item i in T_B must be the same as that of branch B, because otherwise there exists at least one transaction that contains l_1 but does not contain item i. The support of an FP-tree node is always no larger than that of its ancestors. Therefore, if the support sum of the children node of an FP-node in T_B is less than the support of the FP-node, then there is no need to access the descendants of the FP-node. After T_B is traversed, we check the support of the items in T_B and discard those items whose support in T_B is less than the support of the branch B. After all the tail subtrees are visited, the support of the items are accumulated over all the tail subtrees, and frequent items are included into the FP-tree constructed from D_l. We use an example to illustrate this pruning. In Figure 1, item a's conditional database D_a contains two branches: $abd : 2$ and $ad : 2$. We first visit the first tail subtree of a and get the support of items c and h. The second tail subtree of a is a single branch and the support of the first node of this single branch is 1, which is less than the support of branch $ad : 2$. Therefore, we discard that single branch. After the two tail subtrees are visited, we accumulate the support of items c and h. The support of item c is less than 2, so we discard item c from further consideration. If the above pruning technique is not used, then item c would not be pruned, as its accumulated support in the two tail subtrees is 2. Therefore, the above pruning technique not only avoids traversing unnecessary tail subtrees but also prunes those items that are not cover-items in a very early stage.

4.2 Ranking Equivalence Classes by Statistical Merits

To rank equivalence classes, the class label distribution information of the itemsets are required as shown in Section 3. We accomplish this by replacing the support count of an FP-node with the class label distribution of the FP-node. For example, if transactions 1, 3, and 5 in Table 1 are in class 0 and the other three transactions are in class 1, then the FP-tree is modified as shown in Figure 3. The class distribution of an item in the FP-tree is computed by accumulating the class distribution of the FP-nodes containing that item. The global support of an item is the sum of the support of the item in all the classes. Generators and closed itemsets are identified and generated as described in the previous subsection.

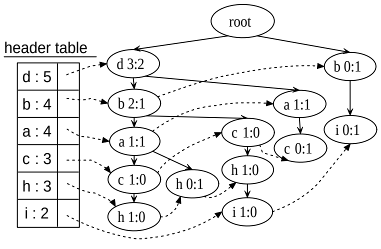


Figure 3: Class distribution information are added into FP-tree nodes for ranking equivalence classes.

Algorithm 1 shows the pseudo-codes of DPMiner to solve our **Problem 1**, which can be also refined to solve our **Problem 2**. As shown, DPMiner maintains the set of

Algorithm 1 The DPMiner Algorithm

```
Input:
    $l$ is a frequent generator;
    $D_l$ is the conditional database of $l$ stored in an FP-tree;
    $C_l$ contains $l$ and the set of cover items on the current path;
    $FG$ is the set of generators and is stored in a hash table;
    $FC$ is the set of closed itemsets and is stored in a hash table;
    $ms$ is the minimum support threshold;
Description:
 1: Scan $D_l$ to count frequent items and sort them into descending
    frequency order, denoted as $F_l=\{a_1, a_2,\cdots, a_m\}$;
 2: $C_{cdb} = \{a_i|sup(l \cup \{a_i\}, D) = sup(l, D), a_i \in F_l\}$;
 3: $C_l = C_l \cup C_{cdb}$; $F_l = F_l - C_{cdb}$;
 4: Scan the tail subtrees of $l$ to obtain $C_{tail}$ and $F_{tail}$, where
    $C_{tail} = \{x|sup(l\cup\{x\}, D) = sup(l, D), x\ is\ in\ tail\ subtrees\ of\ l\}$
    and $F_{tail} = \{x|sup(l \cup \{x\}, D) \neq sup(l, D) \wedge sup(l \cup \{x\}, D) \geq ms, x\ is\ in\ tail\ subtrees\ of\ l\}$;
 5: $C_l = C_l \cup C_{tail}$;
 6: if $C_l$ is not in $FC$ then
 7:     Insert $C_l$ into the hash table storing $FC$;
 8: end if
 9: Insert $l$ into the equivalence class of $C_l$;
10: for all item $a_i \in F_l$ do
11:     if $\exists l' \in FG$ such that $l' \subset l \cup \{a_i\}$ and $sup(l', D)= sup(l \cup \{a_i\}, D)$ then
12:         $F_l = F_l - \{a_i\}$;
13:     else
14:         Insert $l \cup \{a_i\}$ into the hash table storing $FG$;
15:     end if
16: end for
17: if $|F_l| \leq 1$ then
18:     return ;
19: end if
20: if $D_l$ contains only one branch then
21:     return ;
22: end if
23: for all transaction $t \in D_l$ do
24:     $t = t \bigcap (F_l \cup F_{tail})$;
25:     Sort items in $t$ and insert $t$ into the new FP-tree.
26: end for
27: for all item $a_i \in F_l$ do
28:     DPM($l \cup \{a_i\}$, $D_{l\cup\{a_i\}}$, $C_l$, $FG$, $FC$, $ms$);
29: end for
30: $C_l = C_l - C_{cdb} - C_{tail}$;
```

frequent generators FG in a hash table. DPMiner checks whether itemset $l \cup \{a_i\}$ is a generator by searching the immediate subsets of $l \cup \{a_i\}$ in the hash table (Line 10). If $l \cup \{a_i\}$ is not a generator, then it is removed from F_l (Line 11); otherwise it is inserted into the hash table (Line 13). DPMiner also maintains the set of closed itemsets FC in a hash table, and each closed itemset represents an equivalence class. Generators in the same equivalence class are associated together by their common closed itemset. If the closure of a generator is already in the hash table, then the generator is added to the equivalence class of the closed itemset (Line 8). Otherwise, the closed itemset is inserted into the hash table to create a new equivalence class (Lines 6–7). To generate closed itemsets, DPMiner maintains the set of cover items C_l on the current path during the mining process to propagate the cover items with respect to l to l's supersets' closure. Besides the cover items inherited from l's prefix, the other cover items of l can either from l's conditional database (Lines 2–3) or from the tail subtrees of l (Lines 4–5).

To solve **Problem 2** (mining δ-discriminative equivalence classes (ECs) that are also non-redundant), we make use of the following observation. If an EC is δ-discriminative and non-redundant, then none of its subset ECs can be δ-discriminative and none of its superset ECs can be non-redundant. Therefore, mining non-redundant δ-discriminative ECs is just to find a border in the set-enumeration tree [23] such that the patterns above the border (close to the top root node empty set) are not δ-discriminative and the patterns below the border are redundant. So, we refine line 10 of Algorithm 1—We just check whether $l \cup \{a_i\}$ is a δ-discriminative patterns. If it is, then we remove a_i from F_l.

4.3 Optimizations and Discussion

DPMiner maintains two hash tables during the mining process. One hash table stores generators for checking the minimality of generators. The other one stores closed itemsets for associating generators with their closed itemsets. When the number of generators or closed itemsets is large, the space occupied by these two hash tables may exceed the size of the main memory. To solve this problem, we can associate generators with their closed itemsets in a post-processing step to avoid keeping the second hash table in the memory. It is possible that the hash table storing generators is larger than the main memory. In this case, we can use the CFP-tree structure [19] to store frequent generators. The in-core querying time of CFP-tree may be a little longer than the hash table, but CFP-tree requires much less space than the hash table and it also supports efficient retrieval of itemsets on disk.

DPMiner constructs a conditional database for each generator and the closure of each generator is generated during the mining process. An equivalence class may have multiple generators. Therefore, a closed itemset may be generated multiple times in the algorithm. An alternative approach is to construct a conditional database for each closed itemset to ensure every closed itemset is generated only once, and then generate the generators of each closed itemset. The reason for us to adopt the first approach is that items that cannot appear in the closure of a generator are identified and pruned very early in DPMiner. Therefore, generating the closure of generators incurs little overhead in the algorithm. However, it is much more difficult and costly to generate generators from closed itemsets.

The method used by DPMiner for generating the closure of generators can be used alone to mine only frequent closed itemsets. An itemset is a closed itemset if and only if the closure of the itemset is the same as the itemset itself. A trick due to this property is that: If an itemset l is not a closed itemset and the closure of l contains an item i such that item i does not appear in l's conditional database, then there is no need to process l's conditional database. The reason being that any closed itemset containing l must also contain item i (because the closure of l contains item i), and all of the itemsets discovered from l's conditional database contain l, but none of them contain item i. Therefore, none of the itemsets discovered from l's conditional database can be closed, and we can skip mining them. Therefore, if an itemset is not a closed itemset, then there is no need to process the conditional database of the itemset. By using this method, we do not need to maintain the set of frequent closed itemsets in the memory, but most of existing frequent closed itemset mining algorithms such as CLOSET+ [27], and FPclose [12] require this.

5. EXPERIMENT RESULTS AND EVALUATION

We used the following 8 benchmark datasets to evaluate DPMiner:

Datasets	Size	#Trans	#Items	MaxTL	AvgTL	#classes
adult	1.29MB	32,561	149	15	13.87	2
chess	0.34MB	3,196	75	37	37.00	2
connect-4	9.11MB	67,557	129	43	43.00	3
mushroom	0.56M	8,124	119	23	23.00	2
BMS-POS	11.62MB	515,597	1,657	165	6.53	-
pumsb	16.30MB	49,046	2,113	74	74.00	-
ALL-AML	1.1MB	38	1738	865	**865**	2
LungCancer	2.1MB	32	4372	2172	**2172**	2

where 'MaxTL' and 'AvgTL' indicate the maximal and average transaction length. The first four datasets are obtained from the UCI machine learning repository. The BMS-POS and pumsb datasets have been also widely used to evaluate the performance of frequent itemset mining algorithms, and they are available at `http://fimi.cs.helsinki.fi/data/`. These two datasets do not contain class information. The ALL-AML and LungCancer datasets (available at `http://research.i2r.a-star.edu.sg/rp/`) contain gene expression levels of leukemia or lung cancer patients. They are challenging, as discussed in [20], because every transaction contains hundreds (865 in ALL-AML) or even thousands (2172 in LungCancer) of items. The experiments were conducted on a 3.60Ghz Pentium IV with 2GB memory running Fedora core. All codes were compiled using g++.

5.1 On Mining Equivalence Classes

To evaluate the efficiency of DPMiner for mining equivalence classes, we compare DPMiner with LCM [26] and FPclose [12]. We obtained the source codes of LCM and FPclose from their correspondence authors. The LCM algorithm represents the state-of-the-art algorithm for mining frequent closed itemsets. It shows the best performance in a comparative study of frequent closed itemset mining algorithms; see `http://fimi.cs.helsinki.fi/fimi04/`. Here we use its newest version—LCM ver. 3 [26]. The main technique of LCM is a parent-child relationship defined on closed patterns which constructs tree-shaped transversal routes composed of only closed patterns. FPclose is the winning algorithm in FIMI03; see `http://fimi.cs.helsinki.fi/fimi03/`. FPclose uses conditional FP-trees to generate candidate closed patterns, then these candidates are inserted into a separate pattern tree to check whether they are true or not closed patterns. DPMiner differs from all of them as we store the data using a modified version of the FP-tree, and we output both closed patterns and generators in parallel directly from those FP-trees.

As said, the main purpose of DPMiner is to output both closed patterns and generators in parallel. To directly compare with FPclose and LCM, we provide an optional implementation of DPMiner that mines and outputs only closed itemsets by using the closure generation technique described in Section 4.1.3 and Section 4.3. We term this option of DPMiner as "DPMiner-close". We found that our algorithm is not only faster, but also we use less amount of memory as we do not have to store all the closed patterns in the memory.

Figure 4 shows the running time of the algorithms on the first six datasets. DPMiner performs better than LCM3 on four datasets. The DPMiner-close algorithm is significantly faster than LCM3 and FPclose on all the datasets except BMS-POS. This indicates that our closure generation technique is very efficient. The reason why DPMiner-close is slower than LCM3 and FPclose on BMS-POS is that BMS-POS is a very sparse dataset, the cost for building FP-trees from sparse datasets is relatively high.

5.2 On Mining δ-Discriminative Equivalence Classes That Are Non-redundant (Problem 2)

Loekito and Bailey [20] proposed an excellent algorithm, called epMiner, for mining minimal emerging patterns based on zero-suppressed binary decision diagrams. The epMiner algorithm is claimed to significantly outperform another state-of-the-art algorithm proposed by Fan and Kotagiri [11]. Given two classes of data $\mathcal{D}_{pos}$ and $\mathcal{D}_{neg}$, the epMiner algorithm mines those *minimal* patterns occurring frequently ($> \alpha\%$) in the positive class and infrequently ($< \beta\%$) in the negative class. Such patterns are known as minimal emerging patterns [9]—patterns with good relative risk and odds ratio test values.

For a fair comparison with the epMiner algorithm [20], DPMiner takes the option of mining non-redundant δ-discriminative equivalence classes under the setting

$$ms = (|\mathcal{D}_{pos}| \cdot \alpha\%)/(|\mathcal{D}_{pos}| + |\mathcal{D}_{neg}|)$$

and

$$\delta = |\mathcal{D}_{neg}| \cdot \beta\%$$

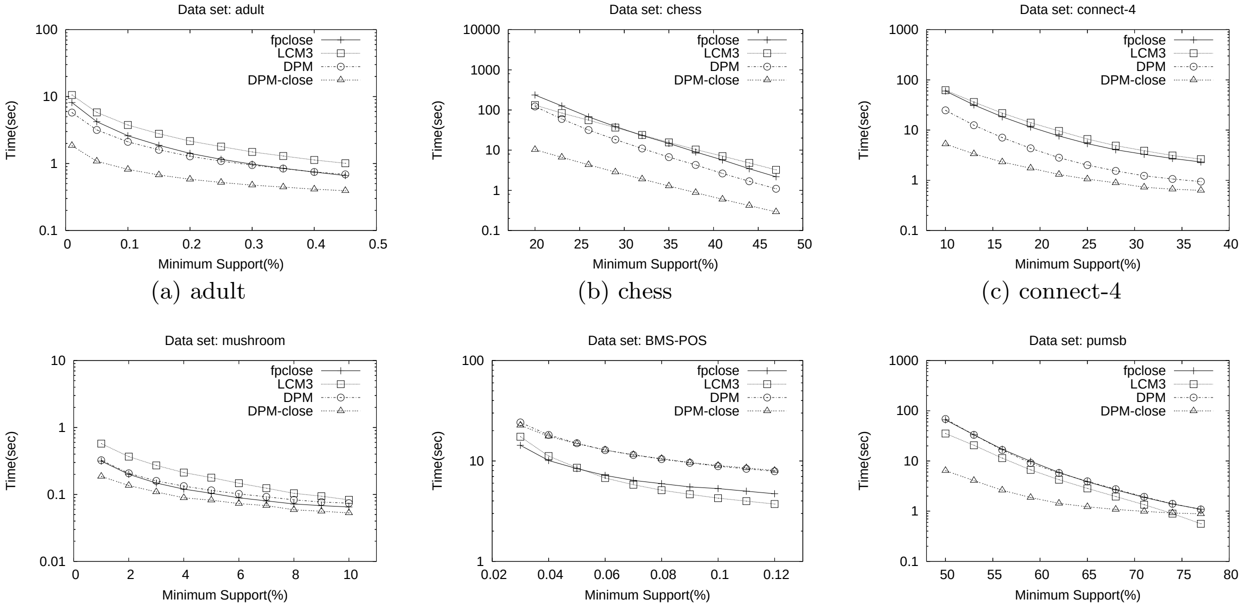
Then, all the generators of these equivalence classes are exactly the minimal $\alpha\%$-$\beta\%$ emerging patterns that the epMiner algorithm outputs. As DPMiner also outputs the closed patterns of the equivalence classes, strictly speaking, we can actually ignore the mining of closed patterns to save more cost for a fairer comparison.

Figure 5 shows the running time of DPMiner and epMiner. From this figure, we can see that DPMiner is constantly faster than epMiner with multiple orders of magnitude on the two high-dimensional gene expression datasets when varying α and β. (Our running time included the time for mining the closed patterns. The experiments on the ALL-AML dataset was terminated when the time reached 50000 seconds; on the LungCancer dataset was terminated when reached 10000 seconds.)

The high efficiency of DPMiner is mainly attributed to the prefix trees (the revised FP-tree structure) to store the data, and the use of closed patterns and generators to concisely represent the equivalence classes. We also note that for handling multiple-class transactions, DPMiner can gain even more efficiency, as the epMiner algorithm [20] has to be run repeatedly on multiple pairwise combinations of the datasets.

6. RELATED WORK

Cong et al. [8] studied the problem of mining top-k covering rule groups from binary classes of gene expression data. Unlike our work dealing with statistically significant equivalence classes, the notion of top-k rule groups is limited to only association rules which are ranked by the rules' confidence and support level. A rule group γ_1 is ranked higher than rule group γ_2 if $\gamma_1.conf > \gamma_2.conf$ or $\gamma_1.supp > \gamma_2.supp$ when $\gamma_1.conf = \gamma_2.conf$. By this ranking criteria, top-k rule groups can contain much redundancy. For example, in situations where $\gamma_1.conf = \gamma_2.conf$, γ_1's superset closed patterns could be all in the top-k patterns. Such a top-k rule group contains a very high redundancy because $k - 1$ of them are totally subsumed to the first one in terms of their transaction set. Second, the mining of generators by Cong et al. is a post-mining process after mining closed



(a) adult (b) chess (c) connect-4

(d) mushroom (e) BMS-POS (f) pumsb

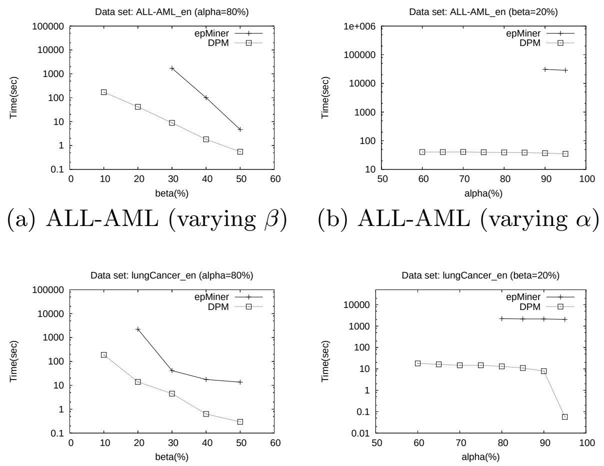
Figure 4: Performance by DPMiner, DPMiner-close, LCM3, and FPclose.

patterns, and it takes a breadth-first search strategy. However, DPMiner takes a parallel depth-first approach to mining both closed patterns and generators. Our algorithm is an order of magnitude faster than Cong et al on many benchmark datasets. (Detailed comparison results are not shown here due to space constraint.)

Our previous work by [15] is also related to this work. The previous one was focused on a theoretical study on a concise representation of odds ratio patterns and relative risk patterns from binary classes of data, by using the concept of "plateaus". However, the current work does not use the concept of "plateaus". Instead, we present new data structures, search strategies, and algorithm details to mine statistically important equivalence classes in general, and mine non-redundant δ-discriminative equivalence classes in particular. This work also provides new ideas to unify the method suitable for broad range of test statistics and multiple classes of transactions.

Recent work on mining *contrast sets* by STUCCO [5, 29] is relevant to our work. Unlike emerging patterns signifying the support ratio of an itemset between two classes of transactions, a contrast set is an itemset emphasizing the absolute support difference. They are a kind of statistical itemsets under absolute risk reduction test [28, 30]. Therefore, contrast sets can be considered as a special case of this work. Also, STUCCO uses a breadth-first search framework. Although it can nicely handle datasets containing two classes of transactions, it may become inconvenient when the data contain multiple classes of transactions as its pruning and filtering ideas become not that straightforward.

Our algorithm and results will be useful for the traditional machine learning field, especially when the application involves multiple classes of data. This is because the traditional learning methods [14, 3, 31, 22] have to organize the multiple classes of data through two typical pairwise coupling approaches. One is the *one-vs-all* approach—constructing all possible data pairwise combinations of one class against all the rest. The other is the *all-vs-all* approach—constructing all possible pairs of single classes. Then, the base learning algorithms are repeated many times to find patterns characterizing every class—k times in the one-vs-all method, and $k(k-1)/2$ times in the all-vs-all method, where k is the number of classes. However, our mining algorithm can totally avoid such a heavy computational redundancy no matter how many classes are involved.



(a) ALL-AML (varying β) (b) ALL-AML (varying α)

(c) Lung Cancer (varying β) (d) Lung Cancer (varying α)

Figure 5: Running time comparison between DPMiner and epMiner.

7. CONCLUSION

In this paper, we have investigated how to efficiently mine a concise representation of statistically important equivalence classes. Our key idea is to mine the frequent generators, their corresponding closed patterns, and the class label distribution information of the itemsets. We have proposed DPMiner that takes a depth-first search strategy for mining both closed patterns and generators simultaneously under a revised, often smaller, FP-tree data structures. DPMiner-close (and DPMiner itself) is much faster than the best closed pattern mining algorithms LCM3 and FPclose. We have also refined DPMiner to efficiently produce a subtype of equivalence classes called non-redundant δ-discriminative equivalence classes. Experiment results show that this refined algorithm is magnificently faster than epMiner, the most recent algorithm for the same mining tasks. As a future work, we will take advantage of the high efficiency for supervised classification problems where the classical statistical tests are of strong interests.

Acknowledgement

We thank to James Bailey for the source code of epMiner, and to Jian Pei and Guozhu Dong for their comments.

8. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. A new framework for itemset generation. In *PODS*, pages 18–24, 1998.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[3] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[4] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, 2000.

[5] S. D. Bay and M. J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5:213–246, 2001.

[6] T. Brijs, K. Vanhoof, and G. W. G. Defining interestingness for association rules. *International journal of information theories and applications*, 10 (4):370–376, 2003.

[7] T. Calders and B. Goethals. Depth-first non-derivable itemset mining. In *SDM*, 2005.

[8] G. Cong, K.-L. Tan, A. K. H. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In *SIGMOD*, pages 670–681, 2005.

[9] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD*, pages 43–52, 1999.

[10] W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *KDD 2001*, pages 67–76, 2001.

[11] H. Fan and K. Ramamohanarao. Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers. *IEEE TKDE*, 18(6):721–737, 2006.

[12] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE TKDE*, 17(10):1347–1362, 2005.

[13] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidates generation. In *SIGMOD*, pages 1–12. May 2000.

[14] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *The annals of statistics*, 2:451–471, 1998.

[15] H. Li, J. Li, L. Wong, M. Feng, and Y.-P. Tan. Relative risk and odds ratio: A data mining perspective. In *PODS*, pages 368–377, 2005.

[16] J. Li, H. Li, L. Wong, J. Pei, and G. Dong. Minimum description length principle: Generators are preferable to closed patterns. In *AAAI*, pages 409–414, 2006.

[17] J. Li, K. Ramamohanarao, and G. Dong. The space of jumping emerging patterns and its incremental maintenance algorithms. In *ICML*, pages 551–558, 2000.

[18] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.

[19] G. Liu, H. Lu, W. Lou, and J. X. Yu. On computing, storing and querying frequent patterns. In *KDD*, pages 607–612, 2003.

[20] E. Loekito and J. Bailey. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In *KDD*, pages 307–316, 2006.

[21] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416, 1999.

[22] R. M. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[23] R. Rymon. Search through systematic set enumeration. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 539–550, 1992.

[24] A. Soulet, B. Cremilleux, and F. Rioult. Condensed representation of emerging patterns. In *PAKDD*, pages 127–132, 2004.

[25] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *KDD*, pages 32–41, 2002.

[26] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proc. of the ACM SIGKDD Open Source Data Mining Workshop on Frequent Pattern Mining Implementations*, 2005.

[27] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *KDD*, pages 236–245, 2003.

[28] S. Wassertheil-Smoller. *Biostatistics and Epidemiology*. Springer Verlag, 2004.

[29] G. I. Webb, S. Butler, and D. Newlands. On detecting differences between groups. In *KDD*, pages 256–265, 2003.

[30] K. M. Weiss. *Genetic Variation and Human Disease: Principles and Evolutionary Approaches*. Cambridge University Press, 1993.

[31] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

[32] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *SDM*, 2002.