Towards Efficient Discovery of Coverage Patterns in Transactional Databases

R. Uday Kiran[†] Masashi Toyoda[†] Masaru Kitsuregawa[†‡]

[†] Institute of Industrial Science, the University of Tokyo, Japan
[‡] National Institute of Informatics, Japan
{uday_rage, toyoda, kitsure}@tkl.iis.u-tokyo.ac.jp

ABSTRACT

Coverage pattern mining is an important model in data mining. It provides useful information pertaining to the sets of items that have coverage interesting to the users in a transactional database. The coverage patterns do not satisfy the anti-monotonic property. This increases the search space in the itemset lattice, which in turn increases the computational cost of mining these patterns. An Apriori-like algorithm known as CMine has been proposed in the literature to discover the patterns. It uses a pruning technique to reduce the search space. We have observed that there exists further scope for reducing the search space effectively. In this paper, we theoretically analyze different measures used in the pattern model and introduce a novel pruning technique to reduce the search space. Furthermore, we extend this technique to CMine to improve its performance. We call the algorithm as CMine++. Experimental results show that the proposed algorithm is efficient.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining.

General Terms

Algorithms

Keywords

Data mining, knowledge discovery, coverage set and pattern mining.

1. INTRODUCTION

Coverage patterns (or itemsets) are an important class of regularities that exist in a database. In many real-world applications, they provide useful information pertaining to the sets of items that have coverage interesting to the users in a database [1, 2]. The model of coverage patterns is as follows.

SSDBM '13, July 29 - 31 2013, Baltimore, MD, USA

Let $I = \{i_1, i_2, \cdots, i_n\}$ be the set of items. A transaction $t = (tid, Y)$ is a tuple, where tid represents a transactional-id and $Y \subseteq I$ is a pattern (or an itemset). A pattern that contains k items is known as a k-pattern. A transactional database DB over I is a set of transactions such that $T = \{t_1, t_2, \cdots, t_m\}$, where m represents the total number of transactions. Let $TID^{DB} = \{1, 2, \cdots, m\}$ be the set of all $tids$ in the database. If an item (or 1-pattern) $i_j \subseteq Y$, $1 \leq j \leq n$, then it is said that i_j occurs in Y or Y contains i_j and such tid is denoted as $tid_k^{i_j}$, $k \leq m$. Therefore, $TID^{i_j} = \{tid_p^{i_j}, \cdots, tid_q^{i_j}\}$, $1 \leq p \leq q \leq m$, be the set of $tids$ in which i_j occurs in DB. The *relative frequency* of i_j in DB, denoted as $RF(i_j) = \frac{|TID^{i_j}|}{|TID^{DB}|}$, where $|TID^{i_j}|$ and $|TID^{DB}|$ denote the total number of transactions containing i_j in DB and database size, respectively. An item $i_j \in I$ is said to be a **frequent item** if $RF(i_j) \geq minRF$, where $minRF$ is the user-defined minimum relative frequency threshold. Let $X = \{i_p, i_q, \cdots, i_r\} \subseteq I$, $1 \leq p \leq q \leq r \leq n$, be an itemset (or a pattern). A pattern containing k number of items is called a k-pattern. The coverage set of a pattern X, denoted as TID^X, represents the set of distinct $TIDs$ containing an item $i_j \in X$ in DB. That is, $TID^X = \{TID^{i_p} \cup TID^{i_q} \cup \cdots \cup TID^{i_r}\}$. In a pattern X, if items are ordered in descending order of their relatively frequencies, i.e., $RF(i_p) \geq RF(i_q) \geq \cdots \geq RF(i_r)$, then its *overlap ratio*, denoted as $OR(X)$, is the ratio of number of transactions common in $X - i_r$ and i_r to the number of transactions in i_r. That is, $OR(X) = \frac{|TID^{X-i_r} \cap TID^{i_r}|}{|TID^{i_r}|}$. A pattern X is said to be a **non-overlap pattern** if $\forall i_j \in X$, $RF(i_j) \geq minRF$ and $OR(X) \leq maxOR$, where $maxOR$ is the user-defined maximum overlap ratio. The *coverage-support* of a pattern X, denoted as $CS(X)$, represents the ratio of the size of coverage set of X to the database size. That is, $CS(X) = \frac{|TID^X|}{|TID^{DB}|}$. A pattern X is said to be a **coverage pattern** if $\forall i_j \in X$, $RF(i_j) \geq minRF$, $OR(X) \leq maxOR$ and $CS(X) \geq minCS$, where $minCS$ is the user-defined minimum coverage support.

Table 1: Transactional database.

TID	Items	TID	Items
1	*abcf*	6	*bdg*
2	*acef*	7	*bd*
3	*aceg*	8	*beg*
4	*acd*	9	*beh*
5	*bdf*	10	*abh*

EXAMPLE 1. *Consider the transactional database shown in Table 1. It contains 10 transactions and the set of items,*

$I = \{a, b, c, d, e, f, g, h\}$. *The set of all tids in the database, i.e.,* $TID^{DB} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. *The TIDs containing the item 'a' are 1, 2, 3, 4 and 10. Therefore,* $TID^a = \{1, 2, 3, 4, 10\}$ *and* $RF(a) = \frac{|TID^a|}{|TID^{DB}|} = \frac{4}{10} = 0.4$. *Similarly, for the item 'b',* $TID^b = \{1, 5, 6, 7, 8, 9, 10\}$ *and* $RF(b) = 0.7$. *If the user-defined* $minRF = 0.4$, *then both 'a' and 'b' are frequent items. The set of items 'b' and 'a', i.e.,* $\{b, a\}$ *is a pattern. For the simplicity purpose, we represent this pattern as 'ba'. It is a 2-pattern because it has only two items. The coverage set of 'ba', i.e.,* $TID^{ba} = TID^b \cup TID^a = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. *The overlap ratio of 'ba', i.e.,* $OR(ba) = \frac{|TID^b \cap TID^a|}{|TID^a|} = \frac{2}{5} = 0.4$. *If the user-defined* $maxOR = 0.5$, *then it is a non-overlap pattern because* $RF(b) \geq minRF$, $RF(a) \geq minRF$ *and* $OR(ba) \leq maxOR$. *The coverage support of 'ba',* $CS(ba) = \frac{|TID^{ba}|}{|TID^{DB}|} = \frac{|\{1,2,3,4,5,6,7,8,9,10\}|}{|\{1,2,3,4,5,6,7,8,9,10\}|} = 1$. *If the user-defined* $minCS = 0.7$, *then the non-overlap pattern 'ba' is a coverage pattern as* $CS(ab) \geq minCS$. *Please note that we will be using the same threshold values to illustrate different concepts that were discussed in later parts of this paper.*

The problem definition of coverage patterns is as follows. Given the transactional database (DB), set of items (I) and the user-defined minimum relative frequency ($minRF$), maximum overlap ratio ($maxOR$) and minimum coverage support ($minCS$) thresholds, discover the complete set of coverage patterns that satisfy the $minRF$, $maxOR$ and $minCS$ thresholds.

The coverage patterns do not satisfy the anti-monotonic property [1]. That is, all non-empty subsets of a coverage pattern may not be coverage patterns. This increases the search space in the itemset lattice, which in turn increases the computational cost of mining the patterns. An Apriori-based Granular Computational Technique (Apriori-GCT) algorithm [3, 4] known as CMine has been introduced to discover the complete set of coverage patterns [2]. It uses a pruning technique to reduce the search space and computational cost of mining the patterns. We have observed that there exists further scope for reducing the search space effectively.

With this motivation, this paper makes an effort to discover the patterns effectively. To reduce the search space, this paper introduces a novel pruning technique by theoretically analyzing the relationship between the *relative frequency*, *overlap ratio* and *coverage* measures. We also extend the proposed technique to improve the performance of CMine. We call the algorithm as CMine++. By conducting experiments on different datasets, we show that CMine++ is efficient than CMine.

The rest of this paper is organized as follows. Section 2 describes the working of CMine. Section 3 describes the performance issue of CMine and introduces the proposed CMine++ algorithm. Section 4 reports the experimental results conducted on CMine and CMine++ algorithms. Finally, we conclude the paper with future research directions.

2. THE CMINE ALGORITHM

CMine is an Apriori-like algorithm [3] to discover the coverage patterns. Since the coverage patterns do not satisfy the anti-monotonic property, it follows a slightly different approach (as compared with Apriori) to discover the patterns. The approach is based on the following pruning technique to reduce the search space.

> "**If X is a coverage pattern, then $\exists Y \subset X$ such that $|Y| = |X| - 1$ and $OR(Y) \leq maxOR$.**"

The working of CMine using this pruning technique is as follows: Discover the set of coverage k-patterns (denoted as L_k) and non-overlap k-patterns (denoted as NO_k) from the set of candidate k-patterns (denoted as C_k). Next, generate C_{k+1} by performing $NO_k \bowtie NO_k$. Repeat this process until $NO_k = \emptyset$ or no more candidate patterns can be generated.

A performance problem of an Apriori-like algorithm is that it requires multiple scans on the database. To confront this problem, granular computational technique (GCT) was proposed in [4]. The CMine employs GCT to discover the patterns with a single scan on the database. It is as follows. The bit string of a pattern X, denoted as $B^X = \{B^{i_p} \vee B^{i_q} \vee \cdots \vee B^{i_r}\}$, records the information pertaining to the occurrence of an $i_j \in X$ in distinct transactions of a database. Each bit $b \in B^X$ uniquely represents a transaction in the database. If an item $i_j \in X$ occurs in a transaction, the corresponding bit is represented as 1, otherwise it is represented as 0. The relative frequency of an item $i_j \in I$, i.e., $RF(i_j) = \frac{card(B^{i_j})}{|DB|}$. Similarly, $CS(X) = \frac{card(B^X)}{|DB|}$ and $OR(X) = \frac{card(B^{X-i_r} \wedge B^{i_r})}{card(B^{i_r})}$. Figure 1 shows the step-by-step working of CMine using GCT.

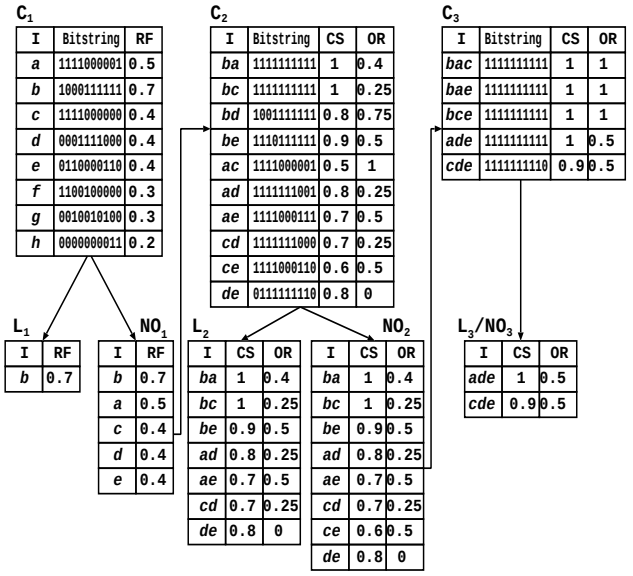

Figure 1: The working of CMine algorithm on Table 1. The term 'I' is an acronym for pattern.

3. PROPOSED ALGORITHM

In this section, we first describe the performance problem of CMine. Next, we introduce the basic idea and proposed CMine++ algorithm.

3.1 The Performance Issue of CMine

The pruning technique of CMine uses all non-overlap $(k-1)$-patterns to generate candidate k-patterns. Unfortunately

this technique is not efficient because some of the non-overlap $(k-1)$-patterns can never generate a coverage k-pattern.

> **Example 2:** Since the coverage support of '*ba*' is 1 (see Figure 1), it is straight forward to show that the overlap ratio of all its supersets will be equal to 1, which is more than the user-defined $maxOR$ threshold. In other words, any superset of '*ba*' will never be a coverage pattern, and therefore, it should not be considered for generating candidate 3-patterns. The CMine algorithm considers '*ba*' for generating candidate 3-patterns (see Figure 1). Therefore, the pruning technique of CMine is not efficient.

3.2 Basic Idea: Potential Coverage Patterns

We propose the following technique to prune those non-overlap $(k-1)$-patterns that cannot generate coverage patterns at higher-order.

> "**If X is a coverage pattern, then $X - i_r$ must be non-overlap pattern (i.e., $OR(X - i_r) \leq maxOR$) with $CS(X - i_r) \leq 1 - (maxOR \times minRF)$.**"

The correctness of the technique is shown in Lemma 2. Now, we introduce the following concept using this technique.

DEFINITION 1. *(**Potential coverage pattern.**) A pattern X is said to be a potential coverage pattern if*

$$\begin{aligned} CS(X) &\leq 1 - (maxOR \times minRF), \\ OR(X) &\leq maxOR \ \ and \\ \forall i_j \in X, RF(i_j) &\geq minRF. \end{aligned}$$

The potential coverage patterns satisfy the *convertible anti-monotonic property* [5] (see Definition 2). The correctness is straight forward to prove from Property 1. Pei et al [5] have theoretically shown that this property is same as the anti-monotonic property if the items within a pattern are considered as an ordered set with respect to their relative supports. As a result, mining coverage patterns using potential coverage patterns is no more a computationally expensive process.

LEMMA 2. *If X is a coverage pattern, then $CS(X - i_r) \leq 1 - (maxOR \times minRF)$.*

PROOF. If X is a coverage pattern, then $\frac{|TID^{X-i_r} \cap TID^{i_r}|}{|TID^{i_r}|} \leq maxOR$. That is,

$$= |TID^{X-i_r} \cap TID^{i_r}| \ \leq \ maxOR \times |TID^{i_r}|. \quad (1)$$

Since $TID^{X-i_r} \subseteq TID^{DB}$, Equation 1 is possible if and only if

$$\begin{aligned} &|TID^{X-i_r}| \leq |TID^{DB}| - maxOR \times |TID^{i_r}| \\ &= \frac{|TID^{X-i_r}|}{|TID^{DB}|} \leq \frac{|TID^{DB}|}{|TID^{DB}|} - \frac{maxOR \times |TID^{i_r}|}{|TID^{DB}|} \\ &= CS(X - i_r) \leq 1 - (maxOR \times RF(i_r)). \quad (2) \end{aligned}$$

According to the definition of coverage pattern, it turns out that $RF(i_r) \geq minRF$. Therefore, Equation 2 can also be expressed as $CS(X - i_r) \leq 1 - (maxOR \times minRF) \leq 1 - (maxOR \times RF(i_r))$. Hence proved. □

DEFINITION 2. *(**The convertible anti-monotonic property of a potential coverage pattern.**) Let $X = \{i_1, i_2, \cdots, i_k\}$, $k \geq 2$, be a sorted potential coverage k-pattern such that $RF(i_1) \geq RF(i_2) \geq \cdots \geq RF(i_k) \geq minRF$. If $OR(X) \leq maxOR$ and $CS(X) \leq 1 - (maxOR \times minRF)$, then $\forall Y \subset X$, $|Y| > 1$ and $i_k \in Y$, $OR(Y) \leq maxOR$ and $CS(Y) \leq 1 - (maxOR \times minRF)$.*

PROPERTY 1. *If $X \supset Y$, then $TID^X \supseteq TID^Y$.*

3.3 The CMine++ Algorithm

We extend the above pruning technique to improve the performance of CMine. We call the algorithm as CMine++. The level-wise search of CMine++ is as follows: Discover the set of coverage k-patterns (denoted as L_k) and potential coverage k-patterns (denoted as PC_k) from the set of candidate k-patterns (denoted as C_k). Next, generate C_{k+1} by performing $PC_k \bowtie PC_k$. Repeat this process until $PC_k = \emptyset$ or no more candidate patterns can be generated.

The CMine++ algorithm is shown in Algorithm 1, and described as follows: Line 1 performs the database scan and constructs a bit string for every candidate 1-pattern in C_1. Line 2 and 3 discover the set of coverage 1-patterns (L_1) and potential coverage 1-patterns (PC_1), respectively. Line 4 sorts PC_1 in ascending order of their relative frequencies. Lines 5 to 10 are used to generate C_k in order to find L_k and PC_k for $k \geq 2$. Figure 2 shows the step-by-step working of CMine++ algorithm. It can be observed that the non-overlap 2-patterns, '*ba*,' '*bc*' and '*be*,' which are considered by CMine to generate candidate 3-patterns (see Figure 1) have not been considered by CMine++ to perform the same.

Algorithm 1 CMine++ (DB: database, $minRF$: minimum relative frequency, $maxOR$: maximum overlap ratio, $minCS$: minimum coverage support)

1: Scan the database, DB, and generate the set of candidate 1-patterns, C_1. Simultaneously, generate the bit string for every $c \in C_1$. Let $B(c)$ denote the bit string of $c \in C_1$. Let RF denote the relative frequency of $c \in C_1$ such that $RF(c) = \frac{card(B(c))}{|DB|}$. Sort C_1 in ascending order of their relative frequencies.
2: $L_1 = \{< c > | c \in C_1, RF(c) \geq max(minRF, minCS)\}$;
3: $PC_1 = \{< c > | c \in C_1, RF(c) \geq minRF\}$;
4: Sort items in PC_1 in ascending order of their relative frequencies. That is, Sort(PC_1);
5: **for** $k = 2$, $PC_{k-1} \neq \emptyset$, $k++$ **do**
6: $C_k = candidateGen(PC_{k-1})$;
7: For each $c \in C_k$, construct bit string $B^c = \{\bigcup B^{i_j} | \forall i_j \in c\}$ and measure its coverage support ($CS(c)$) and overlap ratio ($OR(c)$).
8: $L_k = \{< c > | c \in C_k,\ CS(c) \geq minCS \ and \ OR(c) \leq maxOR\}$;
9: $PC_k = \{< c > | c \in C_k,\ OR(c) \leq maxOR \ and \ CS(C) \leq 1 - (maxOR \times minRF)\}$;
10: **end for**

4. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of CMine and CMine++ algorithms using synthetic and real-world datasets.

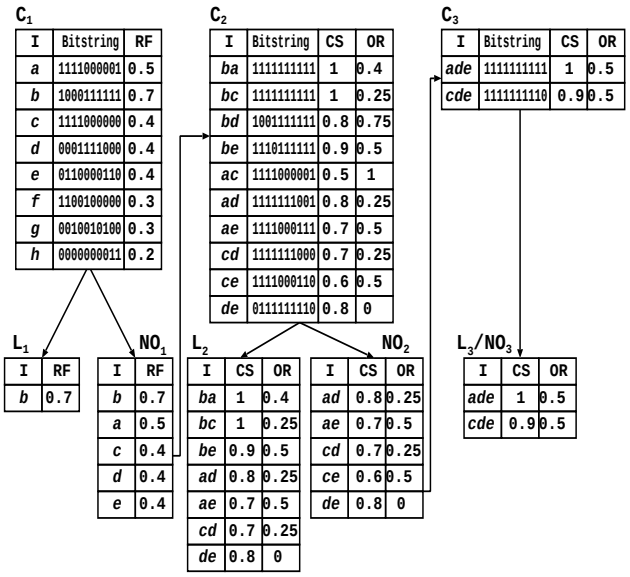

Figure 2: Working of CMine++ algorithm.

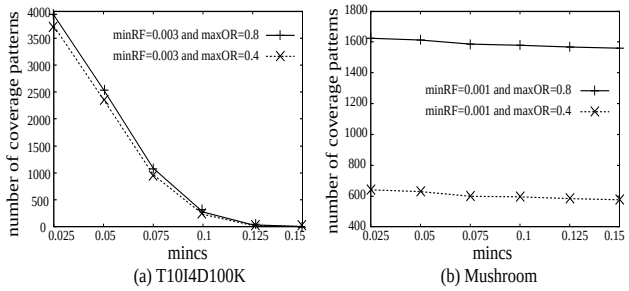

Figure 3: The number of coverage patterns discovered in different datasets.

Both algorithms are written in Java and run with Ubuntu on a 2.66 GHz machine with 2GB memory. The experiments were pursued on the following datasets: (*i*) The T10I4D100K [3] is a sparse synthetic dataset containing 100,000 transactions and 941 distinct items. (*ii*) The Mushroom dataset is a dense real-world dataset containing 8,124 transactions and 119 distinct items. The datasets are available at Frequent Itemset MIning repository (`http://fimi.ua.ac.be/data/`).

4.1 Generation of Coverage Patterns

Figure 3(a) and (b) show the number of coverage patterns discovered in T10I4D100K and Mushroom datasets by varying $minCS$ and $maxOR$ thresholds. The $minRF$ threshold in T10I4D100K and Mushroom datasets were set at 0.003 and 0.001, respectively. The following observations can be drawn from these figures: (*i*) At a fixed $minRF$ and $maxOR$ thresholds, increase in $minCS$ threshold has decreased the number of coverage patterns. The reason is as follows. *To satisfy a high minCS threshold, a pattern has to increase its coverage support by combining with other items. However, such an approach also increases the overlap ratio. As a result, although many supersets of a pattern were able to satisfy the minCS threshold, they fail to have overlap ratio within the maxOR threshold.* (*ii*) At a fixed $minRF$ and $minCS$ thresholds, increase in $maxOR$ has increased the number of coverage patterns. It is because many patterns were able to satisfy a $minCS$ threshold with increase in $maxOR$.

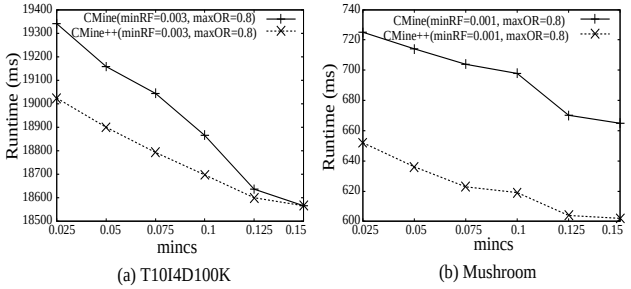

Figure 4: The runtime comparison of CMine and CMine++ algorithms.

4.2 Runtime Comparison the Algorithms

Figure 4 (a) and (b) show the runtime (in milli seconds) consumed by CMine and CMine++ algorithms in T10I4D100K and Mushroom datasets by varying $minCS$ threshold. The changes on the $minCS$ threshold show the similar effect on runtime for both CMine and CMine++ algorithms as of the generation of coverage patterns. More important, the CMine++ algorithm has taken relatively less runtime than the CMine algorithm irrespective of the $minCS$ threshold and dataset type (i.e., sparse or dense).

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have made an effort to reduce the search space and computational cost of mining the coverage patterns which do not satisfy the anti-monotonic property. In particular, we have introduced a pruning technique by exploiting the relationship between the $minRF$, $maxOR$ and $minCS$ thresholds and proposed an Apriori-like algorithm known as CMine++. The experimental results on both synthetic and real-world datasets have shown that the CMine++ is efficient.

As a part of future work, we would like to investigate the pattern-growth-like algorithms to discover the coverage patterns effectively.

6. REFERENCES

[1] Sripada, B., Reddy, P. K., Kiran, R. U.: Coverage patterns for efficient banner advertisement placement. In: WWW Companion Volume, pp. 131–132 (2011)

[2] Srinivas, P. G., Reddy, P. K., Sripada, B., Kiran, R. U., Kumar, D. S.: Discovering Coverage Patterns for Banner Advertisement Placement. In: PAKDD (2) 2012, pp. 133–144 (2012)

[3] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: VLDB, pp. 487–499 1994)

[4] Lin, T.Y. and Xiaohua Hu and Louie, E.: A Fast Association Rule Algorithm based on Bitmap and Granular Computing. In: 12th IEEE International Conference on Fuzzy Systems, pp. 678–683 (2003)

[5] Pei, J., Han, J.: Constrainted Frequent Pattern Mining: A Pattern-Growth View. SIGKDD Explor. Newsl. Vol. 4, No. 1, pp. 31–39 (2002).