CHAPTER II

RELATED WORK

The association rule mining problem has been extensively studied from various aspects over the past decade. As mentioned in the previous chapter, association rule mining consists of 2 steps: frequent itemsets mining and association rules generation. Most of previous works focus on frequent itemset mining which is the most time consuming step. They have been proposed to extend frequent itemsets mining for many purposes. One of interesting approaches is to control the number of itemsets to be mined (Fu et al., 2000), called top-*k* significant itemsets mining approach. Besides, many researchers try to find more interesting patterns (itemsets) by using other criteria instead of using only a support threshold. Recently, the regularity measure have been devised to discover itemsets that occur very frequent and regularly in transactional databases. Thus, this chapter surveys on previous works on the frequent itemsets mining, top-*k* significant itemsets mining and regular-frequent itemsets mining.

2.1 Frequent itemsets mining

The frequent itemsets and association rule mining is first introduced by (Agrawal et al., 1993). Most of association rule mining algorithms adopt the two-phase approach and focus on the frequent itemset mining in the context of transaction databases. A transaction database is a database containing a set of transactions and each transaction is associated with a transaction-id. The basic terms needed for describing association rules and frequent itemsets mining are given by using the formalism of (Agrawal and Srikant, 1994).

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items, that have been used as information units in an application domain and TDB be a database which is a set of transactions, where a transaction t is a subset of I ($t \subseteq I$). Each transaction is identified by a transaction-id tid. A set $X = \{i_{j_1}, \ldots, i_{j_l}\} \subseteq I$ is called an *itemset* or a *l-itemset* (an itemset of size l). If $X \subseteq Y$, it is said that t_q contains X (or X occurs in t_q). The support of an itemset X in a database is denoted by s^X, and is defined as $s^X = |\{t_q | 1 \leq q \leq TDB, t_q \in TDB \text{ and } X \subseteq t_q\}|/|TDB|$. An itemset X is called a *frequent itemset* if its support is greater than or equal to a support threshold specified by the user, otherwise the itemset is not frequent. An association rule is an expression of the form $X \rightarrow Y$, where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \phi$. Note that each of X and Y is a frequent itemset which contains a set of one or more items and the quantity of each item is not considered.

The itemset X is referred to as the antecedent of the rule and the itemset Y as the consequence. An example of association rule is the statement that 80% of transactions that purchase A also purchase B and 10% of all transactions contain both of them. Here, 10% is the support of the itemset A, B and 80% is the confidence of the rule $A \rightarrow B$. An association with the confidence greater than or equal a confidence threshold is considered as a valid association rule.

In association rules mining, two types of database layouts are employed: horizontal and vertical databases. As shown in Table 2.1, the traditional horizontal database contains a set of transactions and each transaction consists of a set of items. Most Apriori-like algorithms use this type of layout. On the other hand, as illustrated in Table 2.2, each item in the vertical database layout maintains a set of transaction-ids (denoted by tidset) where it occurs. Based on the vertical representation of database, various algorithms were devised to mine the results such as Eclat (Zaki et al., 1997), VIPER (Shenoy et al., 2000) and Mafia (Burdick, 2001). Lastly, as pointed out in (Zaki et al., 1997) and (Shenoy et al., 2000), they have been shown the trade-off between both layouts. They claimed that the vertical layout performs generally better than the horizontal format.

Table 2.1: Horizontal representation

tid	items
1	*a b d e*
2	*c d e*
3	*b c f g*
4	*a b d f g*
5	*c e g*
6	*a b c d g*
7	*a b c d*
8	*a b c e*
9	*b c d*
10	*a c e g*
11	*a b f*
12	*a b d g*

A large number of efficient algorithms to mine frequent itemsets have been developed over the years. The strategies developed to speed up frequent itemset mining process can be divided into two approaches. The first is based on the candidate generation-and-test approach. The Apriori algorithm and its several variations belong to this category. Apriori employs a bottom-up, breadth-first search that enumerates every single frequent itemset. It also provides the Apriori property also known as anti-monotone property that any subset of a frequent itemset must be a frequent. In this approach, a set of candidate itemsets of length $n + 1$ is generated from the set of frequent itemsets of length n and then each candidate itemset is checked to see if it meets the support threshold. Some algorithms adopt an Apriori-like method, and are focused on reducing

Table 2.2: Vertical Tidset representation

items	*tidset*
a	1 4 6 7 8 10 11 12
b	1 3 4 6 7 8 9 11 12
c	2 3 5 6 7 8 9 10
d	1 2 4 6 7 9 12
e	1 2 5 8 10
f	3 4 11
g	3 4 5 6 10 12

the number of candidate itemsets, which in turn reduce the time required for scanning databases, are briefly described as follows. Park et al. (Park et al., 1995) proposed an efficient Direct Hashing and Pruning (DHP) algorithm to control the number of candidate 2-itemsets and prune the size of the database by utilizing a hash technique. The inverted hashing and pruning (IHP) algorithm (Holt and Chung, 2002) was proposed. It is similar to the Direct Hashing and Pruning (DHP) algorithm (Park et al., 1995) in the sense that both use hash tables to prune candidate itemsets. In DHP, every k-itemset within each transaction is hashed into a hash table. In IHP, the transaction identifiers of each item of the transactions that contain the item are hashed into a hash table associated with the item. The Tree-Based Association Rule (TBAR) algorithm (Berzal et al., 2001) employs an effective data-tree structure to store all itemsets to reduce the time required for database scans. Further, Cheung et al. proposed Fast Distributed Mining (FDM) of association rules (Cheung et al., 1996) to efficiently discover frequent itemsets, which is a parallelization of the Apriori algorithm. At each level, a database scan is independently performed. In 1997, Brin et al. proposed the Dynamic Itemset Count (DIC) (Brin et al., 1997b) algorithm to locate frequent item sets, which uses fewer passes over the database than classic algorithms, and fewer candidate itemsets than methods based on sampling. Agarwal et al. presented the TreeProjection method (Agarwal et al., 2001) using the hierarchical structure of a lexicographic tree to project transactions at each node of the tree, and matrix counting on this reduced set of transactions for mining frequent itemsets. Another efficient method (Tsay and Chang-Chien, 2004) uses the techniques of clustering transactions and decomposing larger candidate itemsets for mining frequent itemsets. Tsay et al. proposed the Cluster-Based Association Rule (CBAR) method(Tsay and Chiang, 2005), which uses cluster tables to load the database into a main memory that requires only a single scan of the database. Its support count is performed on cluster tables, and thus, does not need to rescan the whole database. The Efficient Dynamic Database Updating Algorithm (EDUA) (Zhang et al., 2007) is designed for mining dynamic databases when some data are deleted. A special database structure BitTableFI (Dong and Han, 2007) is used horizontally and vertically to compress the database for quickly generating candidate itemsets and counting support.

Apriori-inspired algorithms show good performance with sparse datasets such as market basket data, where the frequent patterns are very short. However, with dense datasets such as telecommunications and census data, where there are many, long frequent patterns, the performance of these algorithms degrades incredibly. This degradation is due to the following reasons: these algorithms perform as many passes over the database as the length of the longest frequent pattern. Secondly, they have to generate and test the huge number of candidate itemsets. Thirdly, a frequent pattern of length l implies the presence of $2^l - 2$ additional frequent patterns as well, each of which is explicitly examined by such algorithms. When l is large, the frequent itemset mining methods become CPU bound rather than I/O bound. In other words, it is practically unfeasible to mine the set of all frequent patterns for other than small l. On the other hand, in many real world problems (e.g., patterns in biosequences, telecommunications data, census data, etc.) finding long itemsets of length 30 or 40 is not uncommon (Bayardo, 1998).

The second approach of pattern-growth has been proposed more recently. It also uses the Apriori property, but instead of generating candidate itemsets, it recursively mines patterns in the database to count the support for each pattern. Han et al. (Han et al., 2004) proposed the FP-growth method to avoid generating candidate itemsets by building a FP-tree with only two scans over the database. This milestone development of frequent itemsets mining avoids the costly candidate itemsets generation phase, which overcomes the main bottlenecks of the Apriori-like algorithms. Some algorithm analogy for FP-growth without generating candidate itemsets is briefly described as follows. The H-mine method (Pei et al., 2001b) uses a memory-based hyper structure to store a sparse database in the main memory, and builds an H-structure to invoke FP-growth in mining dense databases. An inverted matrix approach uses an inverted matrix to store the transactions in a special layout, then builds and mines relatively small structures, which are called COFI-trees (El-hajj and Zaiane, 2003). The CFP-tree structure (Liu et al., 2007) is designed to store pre-computed frequent itemsets on a disk to save space. A CFP-tree stores discovered frequent itemsets, but a FP-tree stores transactions. They both use prefix and suffix sharing in the CFP-tree, but only prefix sharing in the FP-tree. Maximum length frequent itemsets are generated by adapting a pattern fragment growth methodology (Hu et al., 2008) based on the FP-tree structure. For most data sets, these algorithms perform better than Apriori. Among the existing pattern-growth algorithms, H-Mine runs faster than FP-Growth on several commonly used test data sets.

2.2 Top-*k* significant itemsets mining

From mining frequent itemsets, a major problem is that user has to define a support threshold. However it is quite difficult for users to set a definite support threshold if they have no special knowledge in advance. If the threshold is set too small, there will be a large number of itemsets having been found, which not only consumes more time and space resource, but also brings much burden to users on analyzing the mining results. On the contrary, if the threshold is set too large, there might be much few even no frequent itemsets, which implies some interesting patterns are hidden owing to the improper determination of support threshold. In some cases of application, it is natural for user to specify a simple threshold on the amount of mining results, say the most 100 frequent itemsets should be found.

Therefore it is of interest to mine the most k frequent itemsets over transactional databases with the highest supports.

Definition 2.1 *An itemset X is a top-k frequent itemset if there exist no more than $(k-1)$ itemsets whose support is higher than that of X.*

To focus on mining top-*k* patterns, Chueung et al. proposed N-most interesting itemsets mining (Cheung and Fu, 2002)(Cheung and Fu, 2004). This task mines only the N *k*-itemsets with highest supports for 1 up to a certain k_{max}, where k_{max} is the upper bound of the length of itemsets, and N is the desired number of *k*-itemsets. Three algorithms were proposed for this mining: *LOOPBACK, BOLB, and BOMO*. All three algorithms are adapted from the *FP-tree* approach. *BOMO* has two phases. First, it builds the complete *FP-tree* with all items in the database to find minimum support threshold of each *k*-itemset. Then, it mines itemsets. During the mining process, the support threshold of all itemsets is increased by considering the minimum value among the support of the N^{th} most frequent *k*-itemset discovered. It is used to prune unpromising itemsets. *LOOPBACK* builds the *FP-tree* and initializes the support threshold to be the support of the N^{th} sorted largest *1*-frequent. If the number of any *k*-itemsets is less than N, the tree will be rebuilt to find the smaller support in order to mine more itemsets in the mining phase. *BOLB* is a hybrid approach of *BOMO* and *LOOPBACK*. Like *BOMO*, it builds the complete *FP-tree* only once. The mining process is applied from the technique of *LOOPBACK*. Wang et al. proposed mining top-*k* frequent closed itemsets of length no less than min_l (Han et al., 2002) (Wang et al., 2005), where k is the desired number of frequent closed itemsets to be mined, and min_l is the minimal length of closed itemsets. *TFP* starts with minimum support threshold at 0. It constructs an *FP-tree* to raise the threshold and uses the threshold to prune the tree. It may take a long time to construct the *FP-tree* to find the final threshold and to prune the tree if the database contains many transactions and long patterns. Moreover, *TFP* has to maintain candidates to ensure that they are really closed. Top-k closed itemset mining was extended to mine top-k closed sequence in (Tzvetkov et al., 2005). Pietracaprina et al. proposed *TopKMiner*

(Pietracaprina and Vandin, 2007) to mine top-*k* closed itemsets without considering the minimal length of them. From this mining, it can allow a user to dynamically raise the value k with no need to restart the computation. *TopKMiner* mines top-*k* closed itemsets by combining the *LCM* algorithm (Uno et al., 2004a)(Uno et al., 2004b) and priority queue to avoid closed checking. In addition, it adopts best first search to generate closed itemsets with highest support first. This idea is supported by C. Wu (Wu, 2006). He proved that a heuristic algorithm is preferred over an exact algorithm to solve top-*k* closed itemset mining.

2.3 Regular-frequent itemsets mining

Mining frequent patterns, periodic pattern (Elfeky et al., 2005)(Maqbool et al., 2006)(Lee et al., 2006) and cyclic patterns (Özden et al., 1998) in static database have been well-addressed over the last decade. Periodic pattern (also called regular pattern) mining problem in time-series data focuses on the cyclic behavior of patterns either in the whole (Elfeky et al., 2005) or at some point (Lee et al., 2006) of time-series. Such pattern mining has also been studied as a wing of sequential pattern mining (Maqbool et al., 2006)(Lee et al., 2006) in recent years. However, although periodic pattern mining is closely related with regular-frequent pattern mining, it cannot be directly applied for finding regular patterns from a transactional database because of two primary reasons. First, it considers either time-series or sequential data. Second, it does not consider the support threshold which is the only constraint to be satisfied by all frequent patterns. Tanbeer (Tanbeer et al., 2009) proposed the regular-frequent pattern mining technique, on the other hand, introduces a new interesting measure of regularity and provides the set of patterns that satisfy both of the regularity and support thresholds in a transactional database.

Ozden et. al. (Özden et al., 1998) proposed a method to discover the association rules (Agrawal et al., 1993) occurring cyclically in a transactional database. It outputs a set of rules that maintains a cyclic behavior in appearance among a predefined non-overlapping database segments. The main limitation of this method is segmenting the database into a series of fixed sized segments, which may suffer from border effect. That is, if the sufficient number of occurrences of a pattern (to become frequent) occurs in the borders of two consecutive segments, the pattern might be ignored to generate association rules.

The problem of regular-frequent itemsets mining which has similar definition to (Tanbeer et al., 2009) is defined as follows:

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of $n \geq 1$ literals, called items. A set $X = \{i_1, \ldots, i_k\} \subseteq I$ is called an itemset (or a pattern), or a k-itemset if it contains k items. A transactional database

$TDB = \{t_1, t_2, \ldots, t_m\}$ over I is a set of $m = |TDB|$ transactions. Each transaction $t_j = (tid, Y)$ is a tuple where $tid = j$ represents the transaction-id and $Y \subseteq I$ is an itemset. If $X \subseteq Y$, it is said that t_j contains X (or X occurs in t_j) and is denoted as t_j^X. Therefore, $T^X = \{t_j^X, \ldots, t_k^X\}$, where $j, k \in [1, m]$ and $j \leq k$, is the set of all *ordered tids*, called *tidset*, where X occurs. The support of an itemset X in TDB, denoted as $Sup(X) = |T^X|$, is the number of transactions in TDB that contains X.

Let t_j^X and t_k^X be two consecutive tids in T^X, i.e. where $j < k$ and there is no transaction t_i, $j < i < k$, such that t_i contains X. Thus, $rtt^X = t_k^X - t_j^X$ is the regularity value which represents the number of missing transaction of X between two consecutive transactions t_j^X and t_k^X. We denote as $RTT^X = \{rtt_1^X, rtt_2^X, \ldots, rtt_z^X\}$, the set of all regularities of X between each pair of two consecutive transactions. Then, the regularity of X can be defined as $r^X = max(rtt_1^X, rtt_2^X, \ldots, rtt_z^X)$.

Therefore, an itemset X is called a *regular-frequent itemset* if (i) its regularity is no greater than a user-given maximum regularity threshold (σ_r); (ii) its support is no less than a user-given minimum support threshold (σ_s). Thus, the regular-frequent itemsets mining problem is to discover the complete set of regular-frequent itemsets from *TDB* with two user-given thresholds: minimum support and maximum regularity threshold which are defined in the percentage of $|TDB|$.

However, as pointed out before it is quite difficult for users to set a definite support threshold if they have no special knowledge in advance. In addition, in some cases, it is natural for user to specify a simple threshold on the amount of regular-frequent patterns, say the most 100 frequent patterns with regularity less than $1,000$ transactions*(i.e. occur at least once in every* $1,000$ *transactions)*. It is thus of interest to mine the most frequent *k* regular patterns over transactional databases without the minimum support threshold requirement.

2.4 Benchmark datasets

To validate the performance of the variant of association rule mining algorithms, several real (*i.e.* accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb*, and retail) and synthetic database benchmarks (*i.e.* T10I4D100K, T20I6D100K, and T40I10D100K), publicly available from IBM Almaden (http://www.almaden.ibm.com/cs/quest/syndata.html), FIMI repository (http://fimi.cs.helsinki.fi/data/), and UC-Irvine Machine Learning Database Repository (http://www.ics.uci.edu/ mlearn/MLRepository.html), are utilized.

Data set accidents contains (anonymous) traffic accident data from the National Institute of Statistics (NIS) for the region of Flanders (Belgium) for the period 1991-2000. The BMS-POS dataset is a real world dataset containing several years worth of point-of-sale data from a large electronic retailer, aggregated at the product category level. The connect and chess datasets are derived from UCI Machine Learning Repository. Each transaction in connect and chess contains legal 8-ply positions in the game where neither player has won yet and the next move is not forced. The kosarak contains click-stream data of a hungarian on-line news portal. The mushroom database consists of records describing the characteristics of various mushroom species. The PUMS datasets (pumsb and pumsb*) contain census data. Pumsb* is the same as pumsb without items with 80% or more support. The retail market basket dataset is obtained from Belgian retail supermarket store from December 1999 to November 2000. The synthetic datasets (T10I4D100K, T20I6D100K and T40D100K), using the IBM generator, mimic the transactions in a retailing environment.

The classical characteristics of datasets were studies in (Gouda and Zaki, 2001). According to (Gouda and Zaki, 2001), the density was used to categorize the characteristics of datasets. A dataset is dense when it produces many long frequent itemsets for all values of support threshold. The authors studied on seven datasets, (*i.e.* chess, connect, mushroom, pumsb, pumsb*, T10I4D100K, and T40I4D100K) and then categorized these datasets according to the density. The density is estimated by using the characteristics of maximal frquent itemsets and more precisely their distribution.

However, as pointed out in (Flouvat et al., 2010), there are two limitations of Gouda's classification. First, its variability with respect to support threshold values. Second, there is no clear relationship between the proposed classification and algorithms performance. Therefore, (Flouvat et al., 2010) proposed a new classification which differs from the classification of (Gouda and Zaki, 2001). It takes into account both the negative border and the positive border of itemsets. The positive border of frequent itemsets is the set of all maximal frequent itemsets w.r.t. set inclusion. The negative border of frequent itemsets is the set of all minimal unfrequent itemsets w.r.t. set inclusion.

These different types of datasets have been identified by taking advantage of the "distance"between positive and negative borders distributions of frequent itemsets. As a consequence, the authors introduced a new classification of datasets made of three types:

- Type I datasets contain long itemsets in the positive border and a negative border closed to the positive border, *i.e.* the mean of the negative border curve is not far from the mean of the positive border curve. In other words, most of the itemsets in the two borders have approximately the same size.

- Type II datasets contain long itemsets in the positive and a large distance between the two borders distributions. In other words, the itemsets in the negative border are much smaller than those of the positive border.
- Type III is a very special case of type O: the two distribution are very close, but they are concentrated in very low levels. This type allows to catch the notion of sparseness.

The Table 2.3 summarizes this new classification and shows how this study could also used for FIMI.

Table 2.3: Datasets classification from (Flouvat et al., 2010)

Type	Type I	Type II	Type III
Distance between the borders	Small	Large	Small
Itemsets size	Long	Long	Small
Examples of datasets	accidents, chess, pumsb	connect, mushroom, pumsb*	BMS-POS, kosarak, retail, T10I4D100K, T20I6D100K, T40I10D100K

Based on the classification of (Flouvat et al., 2010), the Table 2.4 shows the characteristics of the real and synthetic datasets used in the evaluation of this dissertation. It shows the number of items, the average transaction length, and the number of records in each database. The table additionally shows the tpye of datasets that are classified.

Table 2.4: Database characteristics

Database	#items	Avg.length	#Transactions	type
accidents	468	338	340, 183	dense
BMS-POS	1, 156	7.5	515, 597	sparse
chess	75	37	3, 196	dense
connect	129	43	67, 557	dense
kosarak	41, 270	8.1	990, 002	sparse
mushroom	119	23	8, 124	dense
pumsb	2, 113	74	49, 046	dense
pusmsb*	2, 088	50.5	49, 046	dense
retail	16, 469	10.3	88, 162	sparse
T10I4D100K	1, 000	10.3	100, 000	sparse
T20I6D100K	1, 000	20.2	100, 000	sparse
T40I10D100K	1, 000	40.1	100, 000	sparse