MINING FREQUENT PATTERNS FROM PRECISE AND UNCERTAIN DATA

MINERAÇÃO DE PADRÕES FREQUENTES A PARTIR DE DADOS PRECISOS E INCERTOS.

Juan J. Cameron, Carson K. Leung*

Department of Computer Science, University of Manitoba, Winnipeg, MB, Canada
{umcame33, kleung}@cs.umanitoba.ca

Abstract
Data mining has gained popularity over the past two decades and has been considered one of the most prominent areas of current database research. Common data mining tasks include finding frequent patterns, clustering and classifying objects, as well as detecting anomalies. To handle these tasks, techniques from different fields—such as database systems, machine learning, statistics, information retrieval, and data visualization—are applied to provide business intelligent (BI) solutions to various real-life problems. In this survey, we focus on the task of *frequent pattern mining*, which non-trivially extracts implicit, previously unknown and potentially useful information in the form of frequently occurring sets of items. Mined frequent patterns can be considered as building blocks for association rules, which help reveal associative relationships between items or events on the antecedent and the consequent of rules. Here, we describe some classical algorithms, as well as some recent innovative algorithms, for mining precise data (in which users are certain about the presence or absence of data items) and uncertain data (in which users are uncertain about the presence or absence of data items and they only know that data items probably occur).

Keywords: *Index Terms*—Data mining, association rule mining, frequent patterns, precise data, uncertain data, probabilistic databases.

Resumo
Mineração de Dados ganhou popularidade nas últimas duas décadas e tem sido considerada uma das mais proeminentes áreas dentro da área de Banco de Dados. Dentre as tarefas comumente realizadas em mineração de dados encontram-se busca de padrões frequentes, clusterização e classificação de objetos, como também detecção de anomalias. Para manipular estas tarefas, técnicas de diferentes campos – tais como sistemas de banco de dados, máquinas de aprendizado, estatística, recuperação de informações e visualização de dados – são aplicadas para oferecer soluções para problemas em nível de Business Intelligent (BI). Nesta pesquisa, nós focamos em tarefas relacionadas a *mineração de padrões frequentes*, que implica na extração de informações potencialmente úteis, não triviais e previamente desconhecidas, na forma de ocorrências de conjunto de itens frequentes. Mineração de padrões frequentes pode ser considerados como blocos de informações para a construção de regras de associação, os quais auxiliam na identificação de relacionamentos entre itens ou eventos que participam das partes antecedente e consequente de uma regra. Neste trabalho são descritos alguns algoritmos clássicos, como também alguns algoritmos inovadores recentes, para mineração de dados precisos (para os quais o usuário têm certeza da presença ou ausência dos itens de dados) e dados incertos (para os quais usuários tem somente uma certeza probabilística da presença ou ausência de determinados itens de dados)

Palavras-chave: Mineração de Dados; Mineração de Regras de Associação; Dados Precisos; Dados Incertos; Banco de Dados Probabilísticos.

1 INTRODUCTION: DATA MINING

In the last few years, we have witnessed a massive increase in data storage. This results in useful information being dormant in huge hard drives. As a consequence, the field of *data mining* has gained popularity as a way of discovering new opportunities and information on data that was usually doomed to be forgotten.

Some people treat data mining as a synonym for *knowledge discovery from data* (*KDD*), whereas some other people view data mining as an essential step in the KDD process. With the latter view, other steps of the KDD process include (a) *data selection* (which retrieves from databases those target data, i.e., data relevant to the analysis task), (b) *data pre-processing* (which integrates target data from various sources and cleans target data by removing noise and inconsistent data), (c) *data transformation* (which summarizes or aggregates the pre-processed data into appropriate forms for mining), as well as (d) *pattern evaluation* and *knowledge interpretation* (which identifies interesting ones from the mined patterns and represents or visualizes these interesting patterns or knowledge to users). As such, data mining refers to the systematic extraction of patterns from (transformed) data stored or captured in large databases, data streams, data warehouses, or information repositories (Han, 2009). More formally, Piatetsky-Shapiro and Frawley (1991) referred to *data mining* as the "non-trivial extraction of implicit, previously unknown, and potentially useful information from data". Data mining combines different research fields including database systems and data warehousing, machine learning and pattern recognition, statistics, information retrieval, high-performance computing, as well as more specific application fields like WWW, multimedia and bioinformatics (Han, 2009) (Han & Kamber, 2011). Common data mining tasks include association rule mining/frequent pattern mining, clustering, classification, outlier detection, and sequential mining.

Association rule mining (Agrawal, Imielinsky, & Swami, 1993) (Cheng & Han, 2009) is a data mining task that discovers interesting relationships among frequently occurring patterns in databases. The mining process consists of two key steps. The first key step is *frequent pattern mining* (Cheng & Han, 2009) (Leung, 2009), which finds *sets of items* (i.e., *itemsets*) that occur frequently in the data. Once these frequent patterns are found, the second key step of association rule mining is to generate rules that represent interesting relationships (e.g., association, correlation) between the mined frequent patterns. A very common real-life application for association rule mining/frequent pattern mining is shopping market

basket analysis, where store managers want to find hidden relations among independent shopping items. For example, with frequent pattern mining, store managers may find that bread, milk and butter are frequently purchased by customers. With association rule mining on these frequent patterns, the store managers may also discover that 90% of the customers who purchased milk also purchased bread. Similarly, bookstore staff can use association rule/frequent pattern mining to reveal the sets of books frequently bought together by readers. Web administrators can use association rule or frequent pattern mining to know the collection of web pages also viewed by web surfers who view a particular web page.

Clustering (Hartigan, 1975) (Agrawal, Gehrke, Gunopulos, & Raghavan, 1998) (Gunpulos, 2009) (Jain, 2010), is sometimes referred to as *unsupervised learning* or *segmentation*, is a data mining task that assigns data objects to groups of similar objects (i.e., clusters). These objects are grouped in such a way that those in the same cluster are very similar to each other while at the same time are very dissimilar from objects in other groups. Objects are usually grouped using partitioning, hierarchical, density-based, grid-based, or probabilistic model-based methods. The resulting clusters are very useful in getting insight into the distribution of data objects. Consider a shopping market database, in which each transaction records the items purchased by a customer. Using clustering techniques, store managers can divide customers into different groups according to the items they purchased so that customers with similar buying patterns are in the same cluster. This clustering analysis helps store managers to characterize customers based on their buying patterns and discover significant customer groups. Moreover, clustering can also be useful in applications like data reduction, hypothesis generation and testing, bioinformatics (e.g., gene expression analysis), spatial data analysis (e.g., for city planning), as well as web mining. Furthermore, clustering can be applicable to pre-processing data for other data mining tasks like classification.

Classification (Mitchell, 1997) (Liu, 2009) (Witten, 2009) (Alpaydin, 2011), is sometimes referred to as *supervised learning*, is a data mining task that predicts categorical class labels for unseen new data instances (i.e., test data) based on the previously known information (i.e., training data). Data are usually classified using decision tree induction, rule-based classification, Bayesian and neural networks, support vector machines, *k*-nearest neighbour classification, linear regression, associative classification and frequent pattern-based classification, genetic algorithms, or rough set and fuzzy set approaches. Credit approval, target marketing, and diagnosis are some real-life classification applications, in which decisions

involving judgement were made. For example, bank officers often make decisions on approving or denying new loan applications based on information about previous applicants (e.g., their age, marital status, income, education). Specifically, bank officers predict whether the new applicants are classified as high-, medium-, or low-risk applicants based on information about previous applicants. Similarly, based on the classification results, store managers can find appropriate target marketing customers and apply appropriate promotion and marketing strategy to these focused groups of customers.

Outlier detection (Hawkins, 1980) (Patcha & Park, 2007) (Papadimitriou, 2009), is sometimes referred to as *anomaly detection*, is a data mining task that finds abnormal data objects (i.e., data objects with behaviours that are very different from expectation). Note that association rule/frequent pattern mining finds frequently occurring patterns, whereas outlier detection finds rarely occurring patterns. Moreover, clustering finds the majority patterns in a dataset, whereas outlier detection captures exceptional cases that substantially deviate from the majority patterns. Exceptional data objects are usually detected by statistical, clustering-based, classification-based, distance-based, or density-based methods. The detection of outliers or anomalies is important in security-related applications such as credit card fraud detection, in which credit card companies analyze buying behaviours of card holders. Any unusual transactions trigger an alert (e.g., to contract the card holders). Other real-life applications include the monitoring of computer systems or networks (for computer security), civil infrastructures (for maintenance and safety), patients (for pervasive healthcare), and industrial production chain process (for quality control).

Sequential mining (Agrawal & Srikant, 1995) (Dong & Pei, 2007) (Wang, 2009) refers to the analysis of data objects that change over time. Sequential mining can be considered as extensions to association rule/frequent pattern mining by considering the additional dimension: time. For instance, association rule/frequent pattern mining looks for what collections of merchandise items are frequently purchased by customers. Here, the order in which the items were purchased is unimportant. In contrast, sequential mining looks for how the frequently purchased merchandise items change over time. Other real-life application of sequential mining include detection of erroneous sentences, discovery of block correlations in storage systems, as well as analysis of stock market fluctuations, web click log streams and DNA sequences.

So far, we have introduced the five common data mining tasks. An observant reader may notice that,

among these tasks, association rule/frequent pattern mining can be used for clustering and classification (as in associative clustering and associative classification). Outlier detection (which finds *rarely* occurring patterns) can be considered as dual problems of association rule/frequent pattern mining (which finds *frequently* occurring patterns). Data that substantially deviate from frequent patterns can be considered as outliers. Moreover, sequential mining can be considered as extensions to association rule/frequent pattern mining by considering the additional time dimension. Hence, in the remainder of this paper, we will focus on association rule/frequent pattern mining due to its close connection and fundamental role to the other four data mining tasks.

2 BACKGROUND FOR FREQUENT PATTERN MINING

Many existing algorithms mine frequent patterns from *precise* transaction databases, in which each transaction contains a collection of domain items. Each of these items usually takes only one of the binary states: The item is either present in, or absent from, the transaction. Since the introduction of the association rule mining problem in (Agrawal, Imielinsky, & Swami, 1993), numerous algorithms have been proposed to mine frequent patterns from precise data and to use the mined patterns to form interesting association rules. Over the past few years, several researchers have described and compared these algorithms (Cheng, Ke, & Ng, 2007) (Aggarwal, Li, Wang, & Wang, 2009) (Goethals, 2010) (Mabroukeh & Ezeife, 2010). For example, Ceglar & Roddick (2006) provided a review on association rule mining. Han et al. (2007) discussed current algorithms available for data mining, which includes association rule mining.

As we are living in an uncertain world, data may not necessarily be precise. Sometimes, data may be uncertain. This leads to the mining of frequent patterns from *uncertain* datasets, in which users are not sure about the presence of domain items in transactions of the datasets. One way to express the uncertainty is to associate each transaction item with an existential probability value which indicates its likelihood of being present in that transaction. In recent years, researchers have proposed algorithms to mine frequent patterns from uncertain data. Recently, Leung (2011) reviewed the most recent developments in mining frequent patterns from uncertain data.

In the remainder of this paper, we will give a high-level overview of some notable algorithms designed for mining frequent patterns from precise data as well as for uncertain data.

a. Frequency or Support of a Pattern

For a set of transactions *D*, each transaction can be identified by its unique transaction ID (TID). Each transaction t_i is made of a set of items so that $t_i \subseteq I$, where *I* is set of *m* items in the domain: $I = \{x_1, x_2, x_3, \dots, x_m\}$. Then, an *association rule* is a relationship between two or more items of the form $X \Rightarrow Y$, where $X, Y \subseteq I$. A *set of items* (i.e., an *itemset*) is considered to be frequent if the number of transactions that contain it is no less than a user-defined *minimum support threshold* (*minsup*).

The *actual support* (or *frequency*) of a set of items *X* is the number (or percentage) of transactions in a database that contains *X*. For example, if bread, milk and butter are purchased together in 120 of the 200 transactions in *D*, then the support of {bread, milk, butter} is 120 (or 60%).

Similarly, the actual support of a rule $X \Rightarrow Y$ is the number (or percentage) of transactions in a database that contain $X \cup Y$. The *confidence* of the rule is the percentage of transactions that contain itemset *X* that also contains itemset *Y*. For example, out of the 160 transactions in *D* (i.e., 80% of all 200 transactions) containing milk, 140 of them contain both milk and bread. Then, the support of the rule "milk ⇒ bread" is 140 (or 70%) and its confidence is 140/160 = 87.5%.

b. Uncertain Data

Data are imprecise or uncertain if users do not have a perfect or complete description of a situation in the real world (Zimanyi & Pirotte, 1997). One way to interpret these uncertain data is to use the "possible world" interpretation (Chui & Kao, 2008), which takes in account the existential probability associated with each item. Given a domain item i in a single transaction in t_i, there are two "possible words" W_1 where the item x is present in t_i and W_2 where x is absent from t_j. If the probability of W_1 to be the true world is $P_{t_i}(x)$, then the probability of W_2 to be the true world is $1 - P_{t_i}(x)$. This can be extended to datasets with more than one transaction, which usually contain more than one item per transaction. For a transaction with two independent domain items x and y, the possibility of the world that contains both of them is $P_{t_i}(x) \times P_{t_i}(y)$. In general, the number of possible worlds is given by 2^k, where k is the number of independent items in all transactions (Leung, 2011). Following the same interpretation, the *expected support* for an itemset *X* over all possible worlds in a dataset D can be computed by summing the product of existential probabilities of all items in X:

$$expSup(X, D) = \sum_{i=1}^{|D|} \prod_{x \in X} P_{t_i}(x).$$

Unlike frequent pattern mining from precise data (which finds patterns with *actual* support ≥ *minsup*), frequent pattern mining from uncertain data finds patterns with *expected* support ≥ *minsup*.

3 FREQUENT PATTERN MINING ALGORITHMS

a. Mining Precise Data

After Agrawal et al. (1993) introduced the research problem of association rule/frequent pattern mining, Agrawal and Srikant (1994) developed the classical **Apriori algorithm**. This algorithm relies on a generate-and-test approach and an important property: the *Apriori property*. This property is also known as *anti-monotone property*, and it is a basic pillar of the Apriori algorithm. It states that all non-empty subsets of a frequent itemset must be frequent. For example, if itemset $\{a, b, c\}$ is a frequent itemset, then all of its subsets $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{b, c\}$ and $\{a, c\}$ must be frequent. In other words, if an itemset is not frequent, than none of its supersets can be frequent. As a result, the list of potential frequent itemsets eventually gets smaller as mining progresses.

To find frequent patterns, Apriori makes one first pass over the database to find the frequent 1-itemsets. Once this pass is completed, the algorithm generates candidate 2-itemsets based on these frequent 1-itemsets. The algorithm then scans the database again to find frequent 2-itemsets. In the next step, the algorithm uses these frequent 2-itemsets to generate candidate 3-itemsets. The algorithm then scans the database to find frequent 3-itemsets from these candidates. This process is repeated until no larger frequent itemsets are found illustrates how Apriori finds frequent patterns from a sample database.

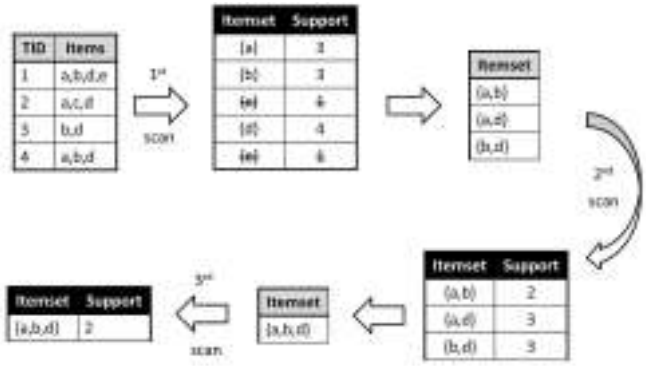

Figure 1. Example run of the Apriori algorithm with a minimum support of 2.

Although it is one of the early data mining algorithms, Apriori has the benefit of producing 100% accurate results. Moreover, it achieves good performance when dealing with very sparse databases. However, it degrades rapidly when databases get denser. The reason is that the algorithm scans databases as many times as the longest frequent pattern (Zaki & Gouda, 2003).

Park et al. (1995) observed that one potential problem of the Apriori algorithm is the huge number of candidate *k*-itemsets (especially, candidate 2-itemsets) generated and tested by the algorithm. To deal with this potential problem, they developed the **Direct Hash and Pruning (DHP) algorithm**. The DHP algorithm uses a hash table to prune away infrequent candidate *k*-itemsets. At the beginning of each level *k*, the DHP algorithm hashes each itemset to a bucket by using a hash function. Once all itemsets have been hashed, the counter at each bucket is checked. If the count is smaller than the *minsup value*, all candidates in that bucket are discarded since they cannot be frequent. As a result of having fewer candidates to check for, the hashing technique speeds up the mining process and reduces the number of candidates to be tested. Moreover, Park et al. also presented a trimming technique to reduce the size of the database as the process evolves so that successive passes search on fewer transactions and items. As the database grows in size, the bigger is the gap between the runtime of DHP and Apriori. The performance of DHP depends on the size of the hash table and of the number of infrequent itemsets being hashed into the same bucket. For example, if several distinct infrequent itemsets are being hashed into the same bucket, the count of the bucket may exceed *minsup*. Consequently, DHP cannot prune away these (infrequent) itemsets, which can be considered as false positives in the intermediate levels.

A variation of DHP is the **Perfect Hashing and Pruning (PHP) algorithm** (Özel & Güvenir, 2001), which uses perfect hashing to avoid false positives in the intermediate levels of the mining process. As a

result, each bucket shows the actual support of every itemset, and thus saves some computation.

Similarly, Pavón et al. (Pavón, Viana, & Gómez, 2006) proposed the **Matrix Apriori algorithm** to speed up the mining process. It reduces the number of candidate itemsets by utilizing matrix and vector structures.

Many Apriori-based algorithms (including DHP and PHP) require numerous database scans, which incur high I/O costs, and thus slow down the mining process. The **Partition algorithm** (Savasere, Edward, & Navathe, 1995) is another technique to improve Apriori-based algorithms by dividing the database in a number of non-overlapping segments. After the first database scan, itemsets that are frequent *locally* in each segment can be found. For an itemset to be globally frequent in the database, it must be locally frequent itemset in at least one partition (or segment). So, after gathering all local frequent itemsets, the Partition algorithm scans the database for the second and last time to check which of those local frequent itemsets are actually frequent globally in the whole database. As a result, this technique reduces drastically the number of scans needed by Apriori-based algorithms to only two. Note that the Partition algorithm depends on the data distribution and the number of segments.

To further tackle the problem associated with numerous database scans and to avoid the candidate generate-and-test process, Han et al. (2000) proposed a pattern-growth approach. The corresponding algorithm is called **Frequent Pattern Growth (FP-growth)**. It only needs two scans to get the information about the database. As contents of this database are captured in a tree structure, frequent patterns are mined from the tree structure. Consequently, FP-growth avoids the candidate generation step because the algorithm traverses the tree structure when mining frequent patterns.

Specifically, FP-growth starts by scanning the database once to find all frequent 1-itemsets. Afterwards, the algorithm makes a ranking table, in which items appear in descending frequency order. All

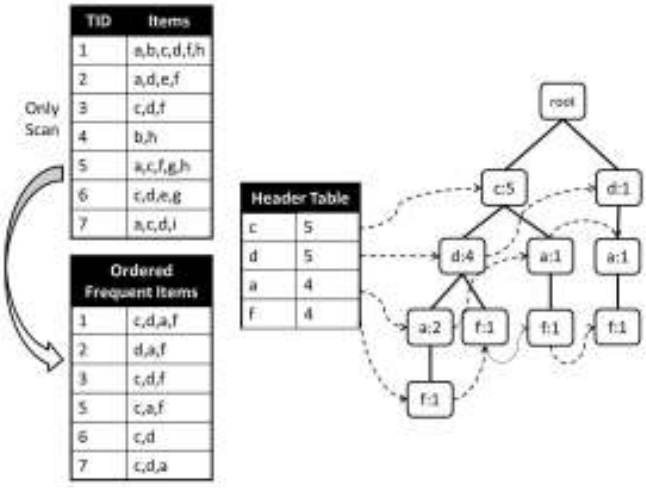

Figure 2. An FP-tree used in the FP-growth algorithm for mining frequent patterns with a minimum support of 3.

infrequent items are then discarded. During the second pass, the algorithm orders the items in each transaction according to the ranking created in the first pass. At the same time, infrequent items in the transaction are ignored. Following the rank order, frequent items are inserted in a tree structure called *FP-tree*. Since all transactions follow the same order, they can be merged if they share the same prefix. This property ensures that the FP-tree is never bigger than the database itself, and in fact, is usually a lot smaller. Once the FP-tree is built for the database, the algorithm constructs a conditional FP-tree for each frequent item so that all frequent patterns can be found by just traversing that structure shows the FP-tree that results of applying the FP-growth algorithm on a small database.

The FP-tree algorithm usually outperforms Apriori-based variations in runtime (Grahne, 2003). The worst case scenario for FP-tree occurs when mining huge but very sparse databases. In these cases, the tree becomes very big. To improve these cases, Grahne & Zhu (2003) used an array structure to reduce the number of traversals of FP-trees.

Note that arranging transaction in descending order does not guarantee the smallest tree structure possible. In fact, itemsets could be put in any order as long as each path follows the same order. Following this line of thinking, Leung et al. (2005) developed a different structure called **Canonical-order Tree (CanTree)** that takes only one database scan in total. Instead of using the descending frequency order, their algorithm reads the database once and uses a predefined order to insert transactions into the

structure so that changes in the frequency of the itemsets do not alter this order. As a consequence of this non-changing ordering property, CanTree is also very suitable for *incremental mining*, where one can insert, delete or modify transactions without the need to rebuild the tree structure.

Another tree structure developed for incremental mining is the **Compact Pattern Tree (CP-tree)** (Tanbeer, Ahmed, Jeong, & Lee, 2009). It gives similar performance as that of the CanTree, but it keeps the frequency order and rearranges the nodes in the tree by using an auxiliary list so that the process does not take too much time.

Pei et al. (2001) developed **H-Mine**, an algorithm that uses dynamic linked lists to maintain a hyperlink array structure called H-struct. The algorithm tries to improve the mining time by using this structure. Once the H-struct is built, the H-Mine algorithm just needs to maintain and update the numerous links that point from one transaction to the next that contains the same set of items. Since H-Mine keeps all transactions that contain frequent items in memory, there is no need to read the database more than once (just to construct the H-struct). From that point on, all information is extracted from the H-struct. Pei et al. (2001) showed that H-Mine outperformed Apriori by finding frequent patterns quicker and requiring less memory than FP-growth, especially with small minimum support threshold. As the minimum support gets larger, FP-growth outperforms H-Mine in terms of memory usage and their runtime is almost the same shows the early stages of the H-struct on the same database used in the Figure 2.

All the algorithms we have described so far use a horizontal representation of the datasets. As explained by Shenoy et al. (2000), the most common representation is called *horizontal item-list* (*HIL*), where each transaction has an ID and a list of items. Another horizontal representation is the *horizontal item-vector* (*HIV*) representation, which stores the database as a set of rows identified by a unique ID and a bit vector where each item is represented by a 1 or a 0 depending on whether it is present or absent, respectively.

Besides these horizontal representations, there are also vertical representations to represent the datasets. When using the *vertical tid-vector* (*VTV*), the database is stored as a set of columns, one for each item, where the transactions are represented by a 1 or a 0 whether they contain the item or not. The fourth and last representation is the *vertical tid-list* (*VTL*) which is similar to VTV. The exception is that, instead of storing 0s and 1s, it stores only the transactions IDs where the item appears shows a

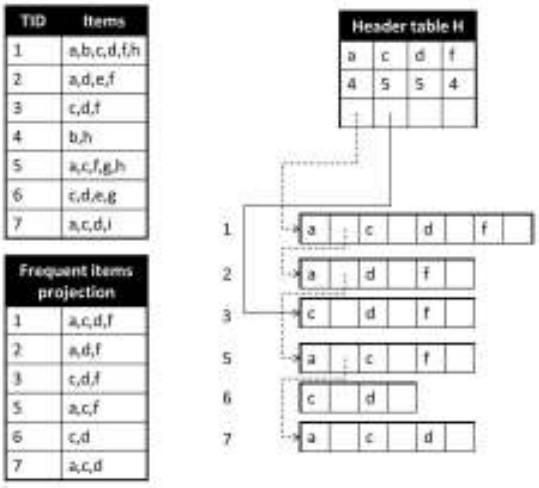

Figure 3. An H-struct used in the H-Mine algorithm for mining frequent patterns with a minimum support of 3.

comparison of these four types of data representation using the same database as in.

One of the algorithms that benefits from the vertical representation is **Vertical Itemset Partitioning for Efficient Rule-extraction (VIPER)** (Shenoy, Haritsa, Sudarshan, Bhalotia, Bawa, & Shah, 2000). VIPER uses VTV and takes advantage of the fact that this representation allows one to calculate the

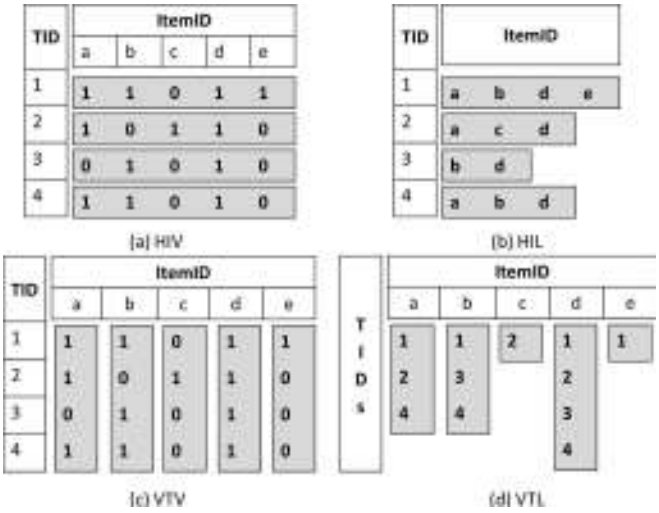

Figure 4. Four representations of datasets.

support of a given pattern by just getting the intersection of the items in it (this means using the AND operation among vectors of items contained in a given itemset). Another advantage is that operations are fast because VIPER is using only binary values, which also give one a chance to compress the vectors.

VIPER scans the database once to find the frequent 1-itemsets and builds “snakes”—binary vectors—for each itemset. A second database scan is needed to count the frequent 2-itemsets and avoid

the overhead of combining all the snakes of singletons. Afterwards, larger frequent itemsets are generated by ANDing the snakes of the items contained in the patters interesting to the user. VIPER offers a good runtime performance mainly because of the advantage of using binary values. Other algorithms using similar techniques include **CLIQUE** and **Eclat** (Zaki, Parthasarathy, Ogihara, & Li, 1997).

Although both VIPER and Eclat require short runtime and low memory consumption, their weak point comes from the costly intermediate stages of the frequent pattern generation because snakes need to be combined and created for the new ones, thus taking up more memory. As an alternative approach to remedy this problem, Zaki and Gouda (2003) developed a new technique called *diffset*, which also uses vertical mining representation but specifically VTL in this case. Instead of actually storing the whole vector for each itemset, diffset only stores the difference between the candidate patterns and the generated frequent patterns, improving on the weak points of VIPER and Eclat. They applied these concepts on Eclat, and called the new algorithm **dEclat**. In terms of memory usage: dEclat requires 80% less space than Eclat.

b. Uncertain Data

One of the earliest algorithms for uncertain data is part of the *generate-and-test* approach. Chui et al. (2007) adapted the classic Apriori algorithm for uncertain data and called it **U-Apriori**. The process is almost the same as in the original algorithm, but now the expected support of a given pattern is incremented by the product of all existential probabilities of the items in the pattern. Expecting the performance of U-Apriori to be even worse than that of the original Apriori because of the effect of multiplying small numbers several times, Chui et al. proposed a trimming strategy to reduce the database by removing items with low probability. So, the number of candidates is lower and the database is smaller, but the mining process itself is still very slow and takes a lot of space nonetheless.

Chui and Kao (2008) explained that this strategy can be counter-productive when most items have small probability values and the savings obtained may not compensate for the overhead caused. As a result, they developed a technique called *Decremental Pruning* (*DP*) to further improve the performance of U-Apriori. DP scans the database once to estimate bounds for each 1-itemset and stores this value in a decremental counter for all patterns that contain this item. As the database is scanned, this counter is updated by subtracting the corresponding “over-estimate” for each item in the pattern. If the counter gets below the minimum support, any pattern containing that item cannot be frequent and hence can be pruned. DP—with its two improvements—is a very effective technique and it improves both runtime and

memory requirements of U-Apriori. Even though it is still bounded by the generate-and-test approach limitations, the application of the decremental technique (known as **UCP-Apriori algorithm**) is a reasonable Apriori-based adaptation for uncertain data.

Leung et al. (2008) developed an adaptation of FP-growth for uncertain data, and they called the resulting algorithm **UF-growth**. The only difference of this algorithm when compared to the original one for precise data is that now it analyzes not only the item itself, but also the probability value associated to it. As a result, two nodes in the tree structure are merged only if both their probability value and the item are the same; otherwise, a new branch is created. Moreover, Leung et al. also proposed two improvements for this algorithm to speed up the mining process. The first one discretizes the probability value of each node up to k decimal places so that more nodes can be merged and less memory occupied. The second one proposes to build conditional trees only for the 1-itemsets to save even more memory. Both improvements can be implemented together.

Aggarwal et al. (2009) adapted another algorithm that is also part of the pattern-growth approach to work with uncertain data: H-Mine. The resulting **UH-Mine algorithm** uses the same basic principles as it counterpart for precise data and the only difference is that now the H-struct also stores the probability of each item besides the link and the item itself. Aggarwal et al. showed that UH-Mine outperforms UCP-Apriori in both runtime and memory usage.

A completely different approach was explored by Calders et al. (2010). Problems like the size of the data for the generate-and-test approach which affects the scalability of the algorithm and the fact that the pattern-growth techniques take a lot of memory to put the datasets in the structure is a motivation to develop a new approach that samples the database a number of times defined by the user to find the frequent itemsets with a very accurate approximation of the support. The technique works by producing a random number for each transaction and comparing it with the probability associated with each item in the transaction. If the number produced is greater than the probability of an item, then this item will be included in the currently sampled transaction. This technique is used in two algorithms: **U-Eclat** and **UFP-growth**, variations of the original Eclat and FP-growth respectively. The experimental results showed that U-Eclat outperformed UCP-Apriori and UH-Mine in terms of runtime, making it possibly the faster algorithm for finding frequent patterns using uncertain data. Its accuracy was high even for a small number of samples—between 92% and 99% for less than 5 iterations. The only issue with

this technique is that it may produce false positives or false negatives when the support of an itemset is very close to the minimum support threshold.

4 COMPARISON

a. Precise Data Algorithms

In terms of performance, the Apriori algorithm and its numerous variations are usually outperformed by algorithms that use the pattern-growth approach or vertical data representation. The main reason is the large number of candidates generated at each step that need to be tested against the database to make sure they are actually frequent. Even with improvements like Partition and DHP, these generate-and-test algorithms are bounded by the intrinsic properties of the approach. The gap is even bigger when dealing with small *minsup* values because only a small fraction of the candidates can be pruned in the early stages of the process.

The algorithms that use the pattern-growth approach work very well when dealing with precise data because many nodes can be merged every time we insert a new transaction in the tree and this property keeps a compact structure that is easily traversed to find the frequent patterns. They usually outperform generate-and-test algorithms in most situations.

Algorithms like Eclat and VIPER that use the vertical data representation are fast because they take advantage of the efficient bitwise operations (i.e., intersection of two bit vectors).

In terms of memory consumption, Apriori-based algorithms generate numerous candidates. Pattern-growth algorithms usually assume the FP-tree to be fit into the memory. With sparse databases, the FP-tree gets very bushy because new branches appear as new transactions are added. Although this is not a common case, it gives the worst case scenario for this approach and can produce worst runtime when compared with Apriori. H-Mine is usually not badly affected by sparse databases H-struct does not merge array entries that share the common prefix (cf. paths sharing the common prefix are merged in the FP-tree).

Among the algorithms that use vertical data representation, VIPER optimizes memory consumption due to the way items are represented by using only binary values. dEclat also requires small space by shrinking the “snakes” and allocating only memory that is strictly necessary, and eliminating

costly intermediate itemsets. As a result, algorithms with vertical data representations lead the pack in memory usage performance.

b. Uncertain data algorithms

Although uncertain data provides a completely different scenario and most algorithms give very different performances than their counter parts with precise data, U-Apriori inherits the problems of generating-and-testing large number of candidates. As the UCP-Apriori algorithm detects infrequent candidates they support drop below the minimum support value, its performance is better than the U-Apriori. However, it is still bounded by the Apriori intrinsic problems.

UF-growth with improvements and UH-Mine are fast enough to be on top of UCP-Apriori. Although UF-growth may suffer from the problem of having very big trees as a result of many different probability values for the same items, its improvements truncate probability values and thus merge more nodes. As a result, they decrease the chance of having very big trees and the algorithm needs small memory space. However, they require longer runtime than U-Eclat.

U-Eclat is the algorithm that requires less memory to mine frequent patterns from uncertain data when taking few database samples. The way U-Eclat samples the database to find all frequent patterns allows it to have a very good approximation of the actual support value of each itemset and makes it an appropriate algorithm when speed is the most important mining criteria. However, the more samples were taken by U-Eclat, the longer it would take to finish. This can have a great impact on the overall runtime.

In terms of accuracy, both U-Apriori and UF-growth provide 100% accuracy at the cost of long runtimes and great memory consumption. UH-Mine also provides 100% accuracy and with good memory and time trade-off. Although UF-growth with improvement may lose some precision by truncating the probability values, the algorithm is accurate enough for most common tasks and scenarios when keeping the first and second decimal values.

U-Eclat may introduce some inaccuracy in the results when aiming for fast runtime and small memory requirement. However, the more samples were taken, the more accurate were the results. This is a trade-off among accuracy, runtime, and memory consumption.

Even though we have compared these frequent pattern algorithms according to their performance in most common scenarios, the order can easily change drastically under different conditions. This is the reason why so many different algorithms exist for users to choose a suitable one for their needs.

5 CONCLUSION

Frequent pattern mining—the first step in the association rule mining—has drawn a lot of attention from researchers lately and is a very important process in the data mining field. It started as a way of analyzing shopping market basket transactions but has extended to different fields and provides useful data to the early stages of other data mining tasks such as classification and outlier detection.

In this paper, we analyzed the most well-known algorithms to find frequent patterns from both precise and uncertain data. We covered three common approaches when describing these algorithms: generate-and-test, pattern growth, and vertical mining algorithms. The classical Apriori algorithm is the referent of the first group and works by generating frequent pattern candidates and checking the database to keep those that are indeed frequent. FP-growth uses a tree structure to store the frequent items and mines the frequent patterns from it. H-Mine is similar to FP-growth but uses linked lists instead of a tree. Eclat and VIPER use vertical data representations instead of the horizontal one used by the others. Eclat uses binary values to indicate the presence or absence of each item in a given transaction, while VIPER have vectors containing the transactions in which each item appears. dEclat represents only the difference between the candidate and the actual frequent itemset in the next level. Among the uncertain data mining approaches, most were adapted from the precise data algorithms by adding the probability value to each item. Among the improvements to each classic adaptation the decremental technique for U-Apriori decrements the expected support of each itemset and prunes it away as soon as it drops below the minimum support value. UF-growth truncates the probability value after the second decimal position to avoid very big trees. U-Eclat samples the database as many times as set by the user and mines the frequent patterns from these samples.

Each of the algorithms that we have described in this paper possesses very different features, and the performance of each depends heavily on the characteristics of the databases.

REFERENCES

Aggarwal, C., Li, Y., Wang, J., & Wang, J. (2009). Frequent pattern mining with uncertain data. *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, (pp. 29–38). Paris, France.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, (pp. 487–499). Santiago de Chile, Chile.

Agrawal, R., & Srikant, R. (1995). Mining Sequential Patterns. *Proceedings of the 11th International Conference on Data Engineering (ICDE)*, (pp. 3–14). Taipei, Taiwan.

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, (pp. 94–105). Seattle, WA, USA.

Agrawal, R., Imielinsky, T., & Swami, A. (1993). Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering , 5* (6), 914–925.

Alpaydin, E. (2011). *Introduction to Machine Learning* (2nd ed.). Cambridge, MA, USA: MIT Press.

Calders, T., Garboni, C., & Goethals, B. (2010). Efficient pattern mining of uncertain data with sampling. *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, (pp. 480–487). Hyderabad, India.

Ceglar, A., & Roddick, J. F. (2006). Association mining. *ACM Computing Surveys , 38* (2), article 5.

Cheng, H., & Han, J. (2009). Frequent itemsets and association rules. In *Encyclopedia of Database Systems* (pp. 1184–1187). Springer.

Cheng, J., Ke, Y., & Ng, W. (2007). A survey on algorithms for mining frequent itemsets. *Knowledge and Information Systems , 16* (1), 1–27.

Chui, C.-K., & Kao, B. (2008). A decremental approach for mining frequent itemsets from uncertain data. *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, (pp. 64–75). Osaka, Japan.

Chui, C.-K., Kao, B., & Hung, E. (2007). Mining frequent itemsets from uncertain data. *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, (pp. 47–58). Nanjing, China.

Dong, G., & Pei, J. (2007). *Sequence Data Mining.* New York, NY, USA: Springer.

Goethals, B. (2010). Frequent set mining. In O. Maimon, & L. Rokach, *Data Mining and Knowledge Discovery Handbook* (2nd ed., pp. 321–338). Springer.

Grahne, G. (2003). Efficiently using prefix-trees in mining frequent itemsets. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI).* Melbourne, FL, USA.

Gunpulos, D. (2009). Clustering overview and applications. In *Encyclopedia of Database Systems* (pp. 383–387). Springer.
Han, J. (2009). Data mining. In *Encyclopedia of Database Systems* (pp. 595–598). Springer.

Han, J., & Kamber, M. (2011). *Data Mining, Concepts and Techniques* (3rd ed.). Waltham, MA, USA: Morgan Kaufmann.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery , 15* (1), 55–86.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, (pp. 1–12). Dallas, TX, USA.

Hartigan, J. (1975). *Clustering Algorithms.* Hoboken, NJ, USA: John Wiley & Sons.

Hawkins, D. (1980). *Identifications of Outliers.* London, UK: Chapman and Hall.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters , 31* (8), 651–666.

Leung, C. K.-S. (2009). Frequent itemset mining with constraints. In *Encyclopedia of Database Systems* (pp. 1179–1183). Springer.

Leung, C. K.-S. (2011). Mining uncertain data. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery , 1* (4), 316–329.

Leung, C. K.-S., Khan, Q. I., & Hoque, T. (2005). CanTree: A tree structure for efficient incremental mining of frequent patterns. *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*, (pp. 274–281). Houston, TX, USA.

Leung, C. K.-S., Mateo, M. A., & Brajczuk, D. A. (2008). A tree-based approach for frequent pattern mining from uncertain data. *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, (pp. 653–661). Osaka, Japan.

Liu, B. (2009). Classification by association rule analysis. In *Encyclopedia of Database Systems* (pp. 335–340). Springer.

Mabroukeh, N. R., & Ezeife, C. I. (2010). A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys , 43* (1), article 3.

Mitchell, T. M. (1997). *Machine Learning.* New York, NY, USA: McGraw-Hill.

Özel, S. A., & Güvenir, H. A. (2001). An algorithm for mining association rules using perfect hashing and database pruning. *Proceedings of the 10th Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN)*, (pp. 257–264).

Papadimitriou, S. (2009). Anomaly detection on streams. In *Encyclopedia of Database Systems* (pp. 88–90). Springer.

Park, J. S., Chen, M.-S., & Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, (pp. 175–186). San Jose, CA, USA.

Patcha, A., & Park, J.-M. (2007). An overview of anomaly detection techniques: existing solutions and latest technological trends. *Computer Networks , 51* (12), 3448–3470.

Pavón, J., Viana, S., & Gómez, S. (2006). Matrix Apriori: speeding up the search for frequent patterns. *Proceedings of the 24th IASTED International Multiconference on Databases and Applications*, (pp. 75–82). Innsbruck, Austria.

Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., & Yang, D. (2001). H-Mine: Hyper structure mining of frequent patterns in large databases. *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, (pp. 441–448). San Jose, CA, USA.

Piatetsky-Shapiro, G., & Frawley, W. J. (1991). *Knowledge Discovery in Databases.* Cambridge, MA, USA: MIT Press.

Savasere, A., Edward, O., & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases.

Proceedings of the 21st International Conference on Very Large Data Bases (VLDB), (pp. 432–444). Zürich, Switzerland.

Shenoy, P., Haritsa, J. R., Sudarshan, S., Bhalotia, G., Bawa, M., & Shah, D. (2000). Turbo-charging vertical mining of large databases. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)* , (pp. 22–33). Dallas, TX, USA.

Tanbeer, S. K., Ahmed, C. F., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient single-pass frequent pattern mining using a prefix-tree. *Information Sciences , 179* (5), 559–583.

Wang, J. (2009). Sequential patterns. In *Encyclopedia of Database Systems* (pp. 2621–2626). Springer.

Witten, I. H. (2009). Classfication. In *Encyclopedia of Database Systems* (pp. 331–335). Springer.

Zaki, M. J., & Gouda, K. (2003). Fast vertical mining using diffsets. *Proceedings of the Ninth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)* , (pp. 326–335). Washington, DC, USA.

Zaki, M., Parthasarathy, S., Ogihara, M., & Li, W. (1997). New algorithms for fast discovery of association rules. *Proceedings of the Third International Conference on Knoweldge Discovery and Data Mining (KDD)*, (pp. 283–286). Newport Beach, CA, USA.

Zimanyi, E., & Pirotte, A. (1997). Imperfect information in relational databases. In A. Motro, & P. Smets, *Uncertainty Management in Information Systems: From Needs to Solutions* (pp. 23–76). Boston, MA, USA: Kluwer Academic Publishers.