Efficient Algorithms for the Mining of Constrained Frequent Patterns from Uncertain Data

Carson Kai-Sang Leung*
Department of Computer Science
The University of Manitoba
Winnipeg, MB, Canada
kleung@cs.umanitoba.ca

Dale A. Brajczuk
Department of Computer Science
The University of Manitoba
Winnipeg, MB, Canada
umbrajcz@cs.umanitoba.ca

ABSTRACT

Mining of frequent patterns is one of the popular knowledge discovery and data mining (KDD) tasks. It also plays an essential role in the mining of many other patterns such as correlation, sequences, and association rules. Hence, it has been the subject of numerous studies since its introduction. Most of these studies find all the frequent patterns from collection of precise data, in which the items within each datum or transaction are definitely known and precise. However, there are many real-life situations in which the user is interested in only some tiny portions of these frequent patterns. Finding all frequent patterns would then be redundant and waste lots of computation. This calls for *constrained mining*, which aims to find only those frequent patterns that are interesting to the user. Moreover, there are also many real-life situations in which the data are uncertain. This calls for *uncertain data mining*. In this article, we propose algorithms to efficiently find *constrained* frequent patterns from collections of *uncertain data*.

1. INTRODUCTION

The research problem of frequent pattern mining has been the subject of numerous studies [7; 15; 16; 23; 26; 27] since its introduction. These studies can be broadly divided into the following two categories, those focused on functionality and those focused on performance.

With respect to functionality, the central question considered is *what* kind of patterns to compute. Although frequent pattern mining was first introduced as a fundamental step in mining association rules [3], studies in this category have shown that frequent pattern mining also plays an important role in the mining of various other patterns including correlation, sequences, episodes, emerging patterns, maximal itemsets, as well as closed itemsets [4; 8; 18; 19; 20; 21].

With respect to performance, the central question considered is *how* to compute the frequent patterns as efficiently as possible. Most studies in this category focused on Apriori-based algorithms [2], which depend on a generate-and-test paradigm. They find frequent patterns from the transaction database (TDB) by first generating candidates and then checking their support (i.e., their occurrences) against the TDB.

*Corresponding author: C.K.-S. Leung.

To improve efficiency of the mining process, Han et al. [17] proposed an alternative framework, namely a tree-based framework, for finding frequent patterns. The algorithm they proposed in this framework constructs an extended prefix-tree structure, called *Frequent Pattern tree* (*FP-tree*), to capture the contents of the TDB. Rather than employing the generate-and-test strategy of Apriori-based algorithms, such a tree-based algorithm focuses on frequent pattern growth—which is a restricted test-only approach (i.e., does not generate candidates, and only tests for support).

Basically, a majority of the aforementioned studies mainly considered the data mining exercise in isolation. They did not explore how data mining can best interact with the human user—a key component in the broader picture of knowledge discovery in databases. In other words, they rely on a computational model, in which the mining system does almost everything and the user is un-engaged in the mining process. Consequently, such a model provides little or no support for user focus (e.g., for limiting the computation to what interests the user). However, the support for user focus is needed in many real-life applications where the user may have some particular phenomena in mind on which to focus the mining. For example, the user may want to find some specific events occurring only on a sunny day. Without user focus, the user often needs to wait for a long time for numerous frequent patterns, out of which only a tiny fraction may be interested to the user. This calls for *constrained mining* [5; 13; 14; 22; 24; 29; 30; 31].

A common characteristic among the above frequent pattern mining algorithms (regardless whether they are for constrained mining or not) is that they handle *precise* data, such as databases of market basket transactions, Web logs, and click streams. When mining precise data, the user definitely knows whether an item (or an event) is present in, or is absent from, a transaction in the databases. However, there are situations in which the user is uncertain about the presence or absence of some items or events [9; 12; 32; 33]. For example, a physician may highly suspect (but cannot guarantee) that a patient suffers from flu. The uncertainty of such suspicion can be expressed in terms of *existential probability*. So, in this uncertain database of patient records, each transaction t_i represents a patient's visit to a physician's office. Each item within t_i represents a potential disease, and is associated with an existential probability expressing the likelihood of a patient having that disease in t_i. For instance, in t_i, the patient has a 90% likelihood of having the flu, and a 70% likelihood of having a cold regardless of having the

flu or not. With this notion, each item in a transaction t_i in traditional databases containing precise data can be viewed as an item with a 100% likelihood of being present in t_i. This calls for *uncertain data mining* [1; 6; 11; 25; 26; 28].

In recent years, uncertain data mining has become one of the popular research topics in knowledge discovery and data mining (KDD). Many of the developed algorithms for uncertain mining have been focused on data mining tasks like clustering and classification of uncertain data [9]. With respect to frequent pattern mining or association rule mining of uncertain data, Chui et al. [11] proposed an Apriori-based algorithm called *U-Apriori* and introduced a trimming strategy to reduce the number of candidates that need to be counted by U-Apriori. They [10] also proposed a decremental pruning technique to speed up the mining process. However, as an Apriori-based algorithm, U-Apriori relies on the candidate generate-and-test paradigm.

Knowing that tree-based algorithms for mining precise data (e.g., FP-growth [17]) are usually faster than their Apriori-based counterparts (e.g., Apriori [2]), we previously proposed a tree-based algorithm called *UF-growth* [28] for mining uncertain data. We then further proposed two performance enhancements [26] to this UF-growth algorithm. However, like its counterpart for mining frequent patterns from precise data (i.e., FP-growth), the UF-growth algorithm for mining frequent patterns from uncertain data also suffers from the following problem: It has not yet explored how data mining can best interact with the human user. It relies on a computational model that provides little or no support for user focus. Consequently, the algorithm finds from uncertain data *all* the frequent patterns, out of which only a tiny fraction may be interested to the user. Hence, lots of time and computation are wasted. It is important to note that the support for user focus is needed in many real-life applications where the user may have some particular phenomena in mind on which to focus the mining. For example, when analyzing laboratory test data, there are some known factors (e.g., human reaction time, measuring errors) contributing to the uncertainty of the data. Among these uncertain data, the user may be interested in some particular subsets of the data (e.g., those participated in Tests 1 and 2, those observed to have Symptoms 3 and 4). As another example, for an election, an analyst may be uncertain about which candidates the voter has cast his vote for, but the analyst may be interested in only those candidates who run for some particular posts (e.g., interested in those running for the Secretary/Treasurer position but not for the Chair position, interested in those who belong to a particular party).

In this article, our **key contribution** is the proposal and development of efficient algorithms for mining *constrained* frequent patterns (i.e., frequent patterns that satisfy user constraints) from *uncertain* data. Advantages of mining with our proposed algorithms include the following. First, the search space of frequent patterns from precise data is known to be huge, and that from uncertain data is even much bigger. By pushing the user constraints deep inside the uncertain mining process, we effectively reduce the corresponding search space. Hence, we reduce and avoid unnecessary computation. Consequently, the computation for mining is proportional to the selectivity of user constraints. Second, the algorithms generate all and only those frequent patterns that satisfy the constraints. Thus, the number of patterns generated is proportional to the selectivity of constraints. Third, although there are algorithms for constrained mining and algorithms for uncertain mining, they either mine from *precise* data for the frequent patterns that satisfy the user constraints or mine from uncertain data for *all* the frequent patterns (regardless whether they satisfy the constraints or not). In other words, they do not find constrained frequent patterns from uncertain data. In contrast, the algorithms we proposed here are capable of mining from uncertain data for all those frequent patterns that satisfy the user constraints.

This article is organized as follows. The next section presents background and related work. We propose algorithms for mining constrained frequent patterns from uncertain data in Section 3. Section 4 shows experimental results. Finally, conclusions are given in Section 5.

2. BACKGROUND AND RELATED WORK

Recall that our key contribution of this article is the proposal and development of efficient algorithms for mining constrained frequent patterns from uncertain data, which involves *user constraints* and *uncertain data*.

2.1 User Constraints

Lakshmanan, Ng and their colleagues [22; 30] proposed a *constrained frequent pattern mining* framework, within which the user can use a rich set of SQL-style constraints to guide the mining process to find only those frequent patterns—containing market basket items—that satisfy the user constraints. Examples of these constraints include the following: $C_1 \equiv min(S.Price) \geq \$10$ and $C_2 \equiv S.Type = snack$. Here, constraint C_1 says that the minimum price of all items in a pattern/set S is at least \$10; constraint C_2 says that all items in a pattern S are snack. It is important to note that, besides these market basket items, the set of constraints can also be imposed on individuals, events, or objects in other domains. The following are some examples: $C_3 \equiv max(S.Temperature) \leq 36.8^\circ$C, $C_4 \equiv S.Symptom \subseteq \{fever,\ runny\ nose\}$, $C_5 \equiv S.Day \supseteq \{Saturday,\ Sunday\}$, $C_6 \equiv min(S.Weight) \leq 23$kg, and $C_7 \equiv avg(S.Weight) \leq 23$kg. Here, constraint C_3 says that the maximum (body) temperature of all individuals in a pattern/set S is at most 36.8°C; constraint C_4 says that individuals in S suffer only from fever or runny nose. Similarly, constraint C_5 say that all the events in a pattern S must span over the weekend (Saturday and Sunday). Constraints C_6 and C_7, respectively, say that the minimum and the average weights of all the objects in S is at most 23kg.

The above constraints can be categorized into several overlapping classes according to the nice properties that they possess. One of these classes is the **succinct constraint**, and its formal definition is given below.

Definition 1 (Succinct Constraint [22; 24; 30]). *Let* Item *be the set of domain items, and let* $2^{\texttt{Item}}$ *denote the powerset of* Item*. Then, the* succinct constraint *can be defined in several steps, as follows:*

(i) *An itemset* $SS_j \subseteq$ Item *is a* succinct set *if* SS_j *can be expressed as a result of selection operation* $\sigma_p(\texttt{Item})$*, where* σ *is the usual selection operator and* p *is a selection predicate.*

(ii) *A powerset of items* $SP \subseteq 2^{\texttt{Item}}$ *is a* succinct powerset *if there is a fixed number of succinct sets* $SS_1, \ldots, SS_k$

$\subseteq$ Item *such that* SP *can be expressed in terms of the powersets of* $SS_1, \ldots, SS_k$ *using set union and/or set difference operators.*

(iii) *A constraint* C *is* **succinct** *provided that the set of patterns/itemsets satisfying* C *is a succinct powerset.* □

With the above definition, constraints C_1–C_6 are *succinct*. For example, the set of patterns/itemsets satisfying C_1 can be theoretically expressed as $2^{\sigma_{Price \geq \$10}(\texttt{Item})}$, which is a succinct powerset. Similarly, the sets of itemsets satisfying the other five succinct constraints (C_2–C_6) are also succinct powersets. Practically, one can directly generate precisely all and only those patterns satisfying these constraints by using precise "formulas"—called member generating functions [22; 24; 30]—that do not require generating and excluding patterns not satisfying the constraints. For example, patterns satisfying C_6 can be precisely generated by combining at least one object of weight $\leq$ 23kg with some optional objects (of any weight), thereby avoiding the substantial overhead of the generation and exclusion of invalid patterns. It is important to note that a majority of constraints are succinct and many non-succinct constraints can be induced into weaker constraints that are succinct (e.g., non-succinct constraint C_7 can be induced into succinct constraint C_6 because all frequent patterns satisfying C_7 must satisfy C_6).

Succinct constraints can be further divided into subclasses—such as **succinct anti-monotone (SAM) constraints** and **succinct non-anti-monotone (SUC) constraints**—based on additional properties they possessed. For instance, among the above succinct constraints, C_1–C_4 are *SAM constraints* because they possess an additional property of anti-monotonicity. With such a property, supersets of any pattern violating the SAM constraints also violate the constraints (e.g., if a pattern S contains an item with price lower than \$10, then S violates C_1 and so do any supersets of S). In contrast, succinct constraints C_5–C_6 are *SUC constraints* because they do not possess such an anti-monotonicity property. For instance, if the minimum weight of all objects within a pattern S is heavier than 23kg, then S violates C_6 but there is no guarantee that all supersets of S would violate C_6. As an example, let *x.Weight* be 30kg and *y.Weight* be 20kg. Then, $S \cup \{x\}$ and $S \cup \{y\}$ are both supersets of S. Between them, the former violates C_6 but the latter does not violate the constraint.

2.2 Uncertain Data

A key difference between precise and uncertain data is that each transaction of the latter contains items and their *existential probabilities*. The existential probability $P(x, t_i)$ of an item x in a transaction t_i indicates the likelihood of x being present in t_i. Using the *"possible world" interpretation* of uncertain data [11; 12; 26; 28], there are two possible worlds for an item x and a transaction t_i: (i) the possible world W_1 where $x \in t_i$ and (ii) the possible world W_2 where $x \notin t_i$. Although it is uncertain which of these two worlds is the true world, the probability of W_1 being the true world is $P(x, t_i)$ and that of W_2 is $1 - P(x, t_i)$.

To a further extent, there is usually more than one transaction in a transaction database (TDB). For instance, for an item x and a TDB consisting of two transactions t_1 and t_2, there are four possible worlds: (i) W_1 where x is in both t_1 and t_2, (ii) W_2 where x is in t_1 but not t_2, (iii) W_3 where x is in t_2 but not t_1, and (iv) W_4 where x is neither in t_1 nor in t_2. Let $prob(W_j)$ denote the probability of W_j to be the true world. Then, $prob(W_1) = P(x, t_1) \times P(x, t_2)$, $prob(W_2) = P(x, t_1) \times [1 - P(x, t_2)]$, $prob(W_3) = [1 - P(x, t_1)] \times P(x, t_2)$, and $prob(W_4) = [1 - P(x, t_1)] \times [1 - P(x, t_2)]$.

Similarly, there is usually more than one domain item in each transaction in a TDB. For instance, for two independent items x & y and a transaction t_i, there are also four possible worlds: (i) W_1 where both $x, y \in t_i$, (ii) W_2 where $x \in t_i$ but $y \notin t_i$, (iii) W_3 where $x \notin t_i$ but $y \in t_i$, and (iv) W_4 where both $x, y \notin t_i$. Then, $prob(W_1) = P(x, t_i) \times P(y, t_i)$, $prob(W_2) = P(x, t_i) \times [1 - P(y, t_i)]$, $prob(W_3) = [1 - P(x, t_i)] \times P(y, t_i)$, and $prob(W_4) = [1 - P(x, t_i)] \times [1 - P(y, t_i)]$.

To generalize, there are many items in each of the n transactions in a TDB (where $|TDB| = n$). Hence, the *expected support* of a pattern/itemset S in the TDB can be computed by summing the support of S in a possible world W_j (while taking in account the probability of W_j to be the true world) over all possible worlds:

$$expSup(S) = \sum_j [sup(S) \text{ in } W_j \times prob(W_j)], \quad (1)$$

where (i) $sup(S)$ in W_j denotes the support of S in a possible world W_j and (ii) $prob(W_j)$ denotes the probability of W_j to be the true world. Note that (i) $sup(S)$ in W_j can be computed by counting the number of transactions that contain S in the possible world W_j and (ii) $prob(W_j)$ can be computed by $\prod_{i=1}^{n} \left(\prod_{x \in t_i \text{ in } W_j} P(x, t_i) \times \prod_{y \notin t_i \text{ in } W_j} [1 - P(y, t_i)] \right)$, where x and y are items within a pattern/itemset S.

3. OUR PROPOSED ALGORITHMS

Recall from the previous section that existing frequent pattern mining algorithms either find constrained frequent patterns from precise data or find the all frequent patterns from uncertain data. However, these algorithms do not find constrained frequent patterns from uncertain data. Here, in this section, we propose algorithms that mine from uncertain data those frequent patterns that satisfy the user-specified succinct constraints.

3.1 A Naive Algorithm

When using the "possible world" interpretation of uncertain data, a naive algorithm for finding constrained frequent patterns from uncertain data is to find all frequent patterns first, and then checks these frequent patterns against the user constraints—as a post-processing step—to filter out the patterns that do not satisfy the constraints. Here, a pattern S is considered *frequent* if its expected support $expSup(X) \geq$ user-specified minimum support threshold *minsup*. Recall that the expected support of a pattern can be computed by summing the weighted support of S over all possible worlds. The weight reflects the probability of each possible world to be the true world. The skeleton for this algorithm can be described as follow:

1. For each possible world W_j, do the following:
   (1a) compute the support of S in W_j (i.e., $sup(S)$ in W_j) for each pattern S, and
   (1b) calculate prob(W_j) that expresses the probability of W_j to be the true world.
2. Compute the expected support $expSup(S)$ of a pattern S by summing the weighted support of S over all

possible worlds, i.e.,

$$expSup(S) = \sum_j [sup(S) \text{ in } W_j \quad \times \quad prob(W_j)] .$$

3. For each pattern S, check if S is frequent (i.e., check whether $expSup(S) \geq minsup$).
4. For each frequent pattern, check if it satisfies the user constraints.

3.2 U-FPS: Efficient Algorithms that Mine Uncertain Data for Frequent Patterns Satisfying Succinct Constraints

A problem/weakness associated with the above naive algorithm is that one needs to compute the expected support of each pattern S, regardless whether S satisfies the user constraints or not. It is important to note that support counting is orthogonal to constraint checking. As we aim to find constrained frequent patterns (i.e., frequent patterns that satisfy the user constraints), we should not waste our time and resources in computing the support of a pattern if the pattern does not satisfy the user constraints. Hence, we should check to see if the pattern satisfies the user constraints before computing its support. Pushing constraint checking earlier in the algorithm could save lots of computation, especially when the user constraints are selective (i.e., when not many frequent patterns satisfy the constraints).

Another problem/weakness associated with the naive algorithm is that one needs to compute the expected support of each pattern S by summing the support of S over all possible worlds. However, the number of possible words is well known to be huge. Fortunately, given independent items in pattern S (i.e., $s \in S$), Equation (1) can be simplified [12] to become the following:

$$expSup(S) = \sum_{i=1}^{n} \left(\prod_{s \in S} P(s, t_i) \right) . \qquad (2)$$

With this setting, we no longer need to compute the expected support of a (constrained) pattern S (i.e., $sup(S)$ in W_j) and the probability $prob(W_j)$ for every possible world W_j. Here, a (constrained) pattern S is considered *frequent* if its expected support equals or exceeds the user-specified support threshold *minsup*.

In order to count the support of a constrained pattern, one needs to represent and store the data. As with many tree-based mining algorithms, a key challenge here is how to represent and store data—in this case, uncertain data—in a tree? For precise data, each item in a TDB is implicitly associated with a definite certainty of its presence in the transaction. In contrast, for uncertain data, each item is explicitly associated with an *existential probability* ranging from a positive value close to 0 (indicating that the item has an insignificantly low chance to be present in the TDB) to a value of 1 (indicating that the item is definitely present). Moreover, the existential probability of the item can vary from one transaction to another. Different items may have the same existential probability. Inspired by the key modification made to the Apriori algorithm [2] by the U-Apriori algorithm [11] (i.e., incrementing the support values of candidate patterns by their *expected* support instead of the actual support), we propose two efficient algorithms—called **U-FPS(SAM)** and **U-FPS(SUC)**, or **U-FPS** for short—to mine from **u**ncertain data for **f**requent **p**atterns that satisfy **s**uccinct constraints. More specifically, both U-FPS(SAM) and U-FPS(SUC) algorithms mine uncertain data, but the former finds frequent patterns that satisfy SAM constraints whereas the latter finds those that satisfy SUC constraints. In general, the U-FPS algorithms consist of three key operations: (i) the constraint checking of transactions in the database of uncertain data, (ii) the construction of a tree structure that we call a *UF-tree*, and (iii) the mining of frequent patterns from the UF-tree. Note that our proposed UF-tree is a variant of the FP-tree, in which each node stores an item, its expected support, and the number of occurrences of such expected support for such an item.

3.2.1 U-FPS for Handling SAM Constraints

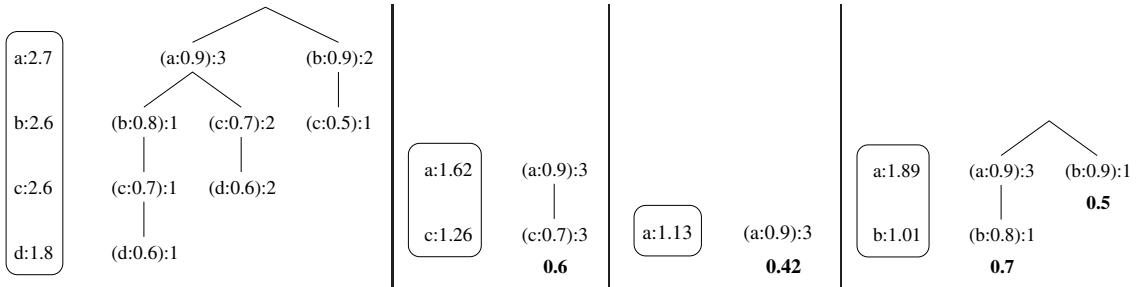
Recall from Section 2.1 that a succinct constraint can be a *succinct anti-monotone* (*SAM*) *constraint* or a *succinct non-anti-monotone* (*SUC*) *constraint*. Let us first consider a SAM constraint C_{SAM}, which possesses two nice properties: (i) anti-monotonicity (i.e., *if a pattern violates* C_{SAM}, *then all its supersets also violate* C_{SAM}) and (ii) succinctness (i.e., *one can easily enumerate all and only those patterns that are guaranteed to satisfy* C_{SAM}). Hence, any pattern S satisfying C_{SAM} must consist of only items that individually satisfy C_{SAM}. In other words, $S \subseteq \texttt{Item}^{\texttt{M}}$ (where $\texttt{Item}^{\texttt{M}}$ is the set of items that individually satisfy C_{SAM}). Due to succinctness, items in $\texttt{Item}^{\texttt{M}}$ can be efficiently enumerated.

To mine uncertain data for frequent patterns that satisfy C_{SAM}, our proposed U-FPS(SAM) algorithm first scans the TDB of uncertain data once, finds all the items that satisfy the user-specified SAM constraint (i.e., $\texttt{Item}^{\texttt{M}}$), and captures these items in a UF-tree. Note that any domain item not belonging to $\texttt{Item}^{\texttt{M}}$ can be safely discarded because any pattern containing a non-$\texttt{Item}^{\texttt{M}}$ item does not satisfy C_{SAM}. (Recall that, if a pattern violates C_{SAM}, then all of its supersets also violate C_{SAM}.) Once all the $\texttt{Item}^{\texttt{M}}$ items are found, U-FPS accumulates the expected support of each of the $\texttt{Item}^{\texttt{M}}$ items. Then, it finds all frequent items (i.e., items having expected support $\geq minsup$), and sorts them in descending order of accumulated expected support. The algorithm then scans the TDB the second time and inserts each transaction of the TDB of uncertain data into the UF-tree in a similar fashion as in the construction of an FP-tree except that (i) the new transaction is merged with a child (or descendant) node of the root of the UF-tree (at the highest support level) only if the same item *and the same expected support* exist in both the transaction and the child (or descendant) nodes and (ii) only those $\texttt{Item}^{\texttt{M}}$ items in the transaction are added. With this tree construction process, the UF-tree possesses a nice property that the occurrence count of a node is at least the sum of occurrence counts of all its children nodes.

Once the UF-tree is constructed, our proposed U-FPS(SAM) algorithm recursively mines frequent patterns that satisfy SAM constraints from this tree in a similar fashion as in the FP-growth algorithm except for the following:

- When forming a UF-tree for the projected database of a constrained pattern S, U-FPS keeps track of the *expected* support (instead of the actual support) of S.
- When computing the expected support of any extension of a constrained pattern S (say, $S \cup \{y\}$ for some item y), U-FPS *multiplies* the expected support of y in a tree path by the expected support of S (instead of just copies the actual support of S to the actual support of $S \cup \{y\}$).

By doing so, U-FPS effectively mines from uncertain data



(a) TDB (b) $\{d\}$-proj. DB (c) $\{c, d\}$-proj. DB (d) $\{c\}$-proj. DB

Figure 1: The UF-trees used in our proposed U-FPS algorithm.

all and only those frequent patterns that satisfy the user-specified SAM constraint. To get a better understanding of this algorithm, let us consider the following example.

Example 1. Consider the following database transactions consisting of uncertain data:

Transactions	Contents
t_1	{a:0.9, b:0.8, c:0.7, d:0.6, e:0.2}
t_2	{a:0.9, c:0.7, d:0.6, f:0.1}
t_3	{b:0.9, e:0.7}
t_4	{a:0.9, c:0.7, d:0.6, f:0.3}
t_5	{b:0.9, c:0.5, f:0.5}

with the following auxiliary information:

Item	a	b	c	d	e	f
Price	\$14	\$18	\$12	\$16	\$20	\$3

In the above TDB of uncertain data, each transaction contains items and their corresponding existential probabilities. For instance, there are five items a, b, c, d and e in the first transaction t_1, and the existential probabilities of these items are 0.9, 0.8, 0.7, 0.6 and 0.2 respectively. Note that the existential probabilities of the same item may vary from one transaction to another (e.g., the existential probability of item b in transaction t_1 is 0.8 whereas that in t_5 is 0.9). Different items may have the same existential probabilities (e.g., the existential probabilities of items c and f in t_5 are the same—with a value of 0.5).

Constraint checking and identification of the domain items that satisfy C_{SAM}. Let constraint C_{SAM} be the SAM constraint $C_1 \equiv min(S.Price) \geq \$10$. Our proposed U-FPS(SAM) algorithm checks each of the six domain items against the constraint C_{SAM}; it enumerates from the domain those valid items a, b, c, d & e (each having price $\geq \$10$), i.e., $\texttt{Item}^{\texttt{M}} = \{a, b, c, d, e\}$. (In other words, item $f \notin \texttt{Item}^{\texttt{M}}$.) Once U-FPS identified the domain items that satisfy C_{SAM}, these items would serve as a building block of all frequent patterns satisfying C_{SAM} because all constrained frequent patterns must comprise only those items.

Construction of the UF-tree. Let the user-specified support threshold *minsup* be set to 1.0. Our U-FPS(SAM) algorithm constructs the UF-tree as follows. First, the algorithm scans the TDB once and accumulates the expected support of each $\texttt{Item}^{\texttt{M}}$ item. Hence, it finds all frequent $\texttt{Item}^{\texttt{M}}$ items and sorts them in descending order of (accumulated) expected support. Specifically, it finds frequent $\texttt{Item}^{\texttt{M}}$ items a, b, c and d (with their corresponding accumulated expected support of 2.7, 2.6, 2.6 and 1.8), which are sorted in descending order of their expected support values. These $\texttt{Item}^{\texttt{M}}$ items and their expected support are represented as a:2.7, b:2.6, c:2.6 and d:1.8. The expected support of each of these frequent $\texttt{Item}^{\texttt{M}}$ items $\geq minsup = 1.0$. On the other hand, the $\texttt{Item}^{\texttt{M}}$ item e having accumulated expected support of $0.9 < minsup$ is removed because it is infrequent. Item f can be ignored as it does not belong to $\texttt{Item}^{\texttt{M}}$.

Then, our U-FPS algorithm scans the TDB the second time and inserts each transaction into the UF-tree. The algorithm first inserts the content of the first transaction t_1 into the tree, and results in a tree branch $\langle$(a:0.9):1, (b:0.8):1, (c:0.7):1, (d:0.6):1$\rangle$. It then inserts the content of the second transaction t_2 into the UF-tree. Since the expected support of a in t_2 is the same as the expected support of a in an existing branch (i.e., the branch for t_1), this node can be shared. So, the algorithm increments the occurrence count for the tree node (a:0.9) to 2, and adds the remainder of t_2—namely, $\langle$(c:0.7):1, (d:0.6):1$\rangle$—as a child of the node (a:0.9):2. As a result, we get the tree branch $\langle$(a:0.9):2, (c:0.7):1, (d:0.6):1$\rangle$. Afterwards, our algorithm inserts the content of the third transaction t_3 as a new branch $\langle$(b:0.9):1$\rangle$ because the node (b:0.9):1 cannot be shared with the node (a:0.9):2. Transactions t_4 and t_5 are then inserted into the UF-tree in a similar fashion. For t_4, nodes (a:0.9):2, (c:0.7):1 and (d:0.6):1 are all incremented by 1; for t_5, the node (b:0.9) is incremented to 2, and $\langle$(c:0.5):1$\rangle$ is added to this branch to get $\langle$(b:0.9):2, (c:0.5):1$\rangle$. Consequently, at the end of the tree construction process, we get the UF-tree shown in Figure 1(a) capturing the content of the above TDB of uncertain data.

Mining of constrained frequent patterns from the UF-tree. Once the UF-tree is constructed, our proposed U-FPS(SAM) algorithm recursively mines constrained frequent patterns from this tree with $minsup = 1.0$ as follows. It starts with item d (with $expSup(\{d\}) = 1.8$). The algorithm extracts from two tree paths—namely, (i) $\langle$(a:0.9), (b:0.8), (c:0.7)$\rangle$ with the occurrence count of (d:**0.6**) equal to 1 (implying that a, b, c & d occur together once in the original database) and (ii) $\langle$(a:0.9), (c:0.7)$\rangle$ with the occurrence count of (d:**0.6**) equal to 2 (implying that a, c & d occur together twice)—and forms the $\{d\}$-projected database. The expected support of $\{a, d\}$ = $(1 \times \mathbf{0.6} \times 0.9) + (2 \times \mathbf{0.6} \times 0.9) = 1.62$. Similarly, the expected support of $\{c, d\} = (1 \times \mathbf{0.6} \times 0.7) + (2 \times \mathbf{0.6} \times 0.7) = 1.26$. So, both constrained patterns $\{a, d\}$ and $\{c, d\}$ are frequent. However, $\{b, d\}$ is infrequent because $expSup(\{b, d\}) = 1 \times \mathbf{0.6} \times 0.8 = 0.48 < minsup$. Thus, b is removed from the $\{d\}$-projected database, which then consists of a single path **0.6** $\times$ $\langle$(a:0.9):3, (c:0.7):3$\rangle$. The UF-tree for such a projected database is shown in Figure 1(b).

Then, the algorithm extracts from the UF-tree for the $\{d\}$-projected database to form the $\{c, d\}$-projected database, which consists of $\{a\}$ (representing the constrained frequent

pattern $\{a, c, d\}$) with $expSup(\{a, c, d\}) = 3 \times \mathbf{0.42} \times 0.9 \approx 1.13$, where 0.42 represents $expSup(\{c, d\})$ for each of the 3 occurrences of $\{c, d\}$. See Figure 1(c).

Next, U-FPS deals with item c. It extracts from three tree paths—namely, (i) $\langle(a{:}0.9), (b{:}0.8)\rangle{:}1$, (ii) $\langle(a{:}0.9)\rangle{:}2$ and (iii) $\langle(b{:}0.9)\rangle{:}1$—and forms the $\{c\}$-projected database. Note that items in the first two paths are both associated with the same item and the same existential probability (i.e., $c{:}\mathbf{0.7}$), whereas items in the last path are associated with $c{:}\mathbf{0.5}$. All this information is captured by a UF-tree shown in Figure 1(d). From this tree, U-FPS finds frequent patterns $\{a, c\}$ and $\{b, c\}$, where $expSup(\{a, c\}) = 3 \times \mathbf{0.7} \times 0.9 = 1.89$ and $expSup(\{b, c\}) = (1 \times \mathbf{0.7} \times 0.8) + (1 \times \mathbf{0.5} \times 0.9) = 1.01$. The constrained pattern $\{a, b, c\}$ is infrequent.

Finally, the U-FPS algorithm deals with item b by extracting from tree paths and forming the $\{b\}$-projected database consisting of $\mathbf{0.8} \times \langle(a{:}0.9)\rangle{:}1$. The constrained pattern $\{a, b\}$ is infrequent because $expSup(\{a, b\}) = 1 \times \mathbf{0.8} \times 0.9 = 0.72 < minsup$.

To summarize, by applying our proposed U-FPS algorithm to the UF-tree (shown in Figure 1) that captures the content of uncertain data in Example 1, we found constrained frequent patterns $\{a\}{:}2.7$, $\{b\}{:}2.6$, $\{c\}{:}2.6$, $\{d\}{:}1.8$, $\{a, c\}{:}1.89$, $\{a, d\}{:}1.62$, $\{a, c, d\}{:}1.13$, $\{b, c\}{:}1.01$ and $\{c, d\}{:}1.26$. □

3.2.2 U-FPS for Handling SUC Constraints

Recall from Section 2.1 that succinct constraints can be further divided into two subclasses: SAM constraints and SUC constraints. So far, we have discussed how our proposed U-FPS(SAM) algorithm mines from uncertain data for those frequent patterns that satisfy SAM constraints. Here, let us discuss how our proposed U-FPS(SUC) algorithm mines from uncertain data for those frequent patterns that satisfy SUC constraints. Note that, although SUC constraints possess the succinctness property (i.e., one can easily enumerate all and only those patterns that are guaranteed to satisfy C_{SUC}), they do not possess the anti-monotonicity property. So, if a pattern violates C_{SUC}, there is no guarantee that all or any of its supersets would violate C_{SUC}. Hence, not all valid patterns are composed of only mandatory items (as for SAM constraints). Instead, any pattern S satisfying C_{SUC} is composed of mandatory items (i.e., items satisfying C_{SUC}) and possibly some optional items (i.e., items *not* satisfying C_{SUC}). In other words, a valid pattern S is usually of the form $\alpha \cup \beta$ where (i) $\alpha \subseteq \texttt{Item}^{\texttt{M}}$ (where $\texttt{Item}^{\texttt{M}}$ is the set of mandatory items) such that $\alpha \neq \emptyset$ and (ii) $\beta \subseteq \texttt{Item}^{\texttt{O}}$ (where $\texttt{Item}^{\texttt{O}}$ is the set of optional items). Due to succinctness, items in $\texttt{Item}^{\texttt{M}}$ and in $\texttt{Item}^{\texttt{O}}$ can be efficiently enumerated.

To handle C_{SUC}, our proposed U-FPS(SUC) algorithm cannot apply the same procedure as we did for C_{SAM}. Some modification is needed; otherwise, we may only get a subset of (e.g., missing some of) valid frequent patterns. Specifically, the algorithm first applies constraint checking to all the domain items and divides them into two sets—$\texttt{Item}^{\texttt{M}}$ consisting of all mandatory items and $\texttt{Item}^{\texttt{O}}$ consisting of all optional items. Then, U-FPS captures items belonging to these two sets in a global UF-tree in such a way that mandatory items appear *below* optional items (i.e., mandatory items are closer to the leaves, and optional items are closer to the root). Once the global UF-tree is constructed with this item-ordering scheme, the algorithm extracts appropriate paths to form the projection of each $x \in \texttt{Item}^{\texttt{M}}$. Note that U-FPS does not need to form projections for any $y \in \texttt{Item}^{\texttt{O}}$ because all patterns satisfying C_{SUC} must be "extensions" of items from $\texttt{Item}^{\texttt{M}}$ (i.e., all valid patterns must be grown from $\texttt{Item}^{\texttt{M}}$ items). When forming each $\{x\}$-projection and constructing its UF-tree, U-FPS does not need to distinguish those $\texttt{Item}^{\texttt{M}}$ items from $\texttt{Item}^{\texttt{O}}$ items. Such a distinction is only needed for the global UF-tree but not the projected UF-trees because optional items can come from $\texttt{Item}^{\texttt{M}}$ or $\texttt{Item}^{\texttt{O}}$ once we found at least one mandatory item for C_{SUC}. After constructing the projected UF-tree for each $x \in \texttt{Item}^{\texttt{M}}$, our proposed U-FPS(SUC) algorithm mines from uncertain data for all frequent patterns that satisfy C_{SUC} in the same manner as U-FPS(SAM) mines for those satisfying C_{SAM}. To get a better understanding of how U-FPS(SUC) handles SUC constraints, let us consider the following example.

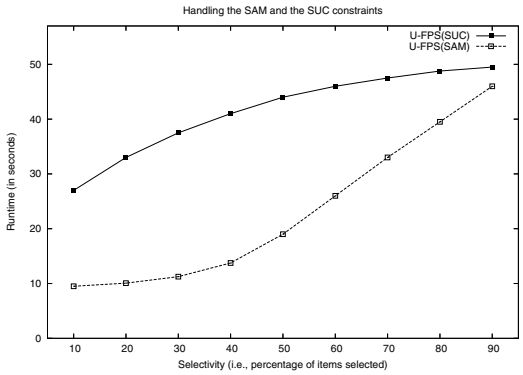
Example 2. Reconsider the uncertain TDB shown in Example 1 with the following auxiliary information:

Object	a	b	c	d	e	f
Weight	34kg	36kg	32kg	13kg	17kg	30kg

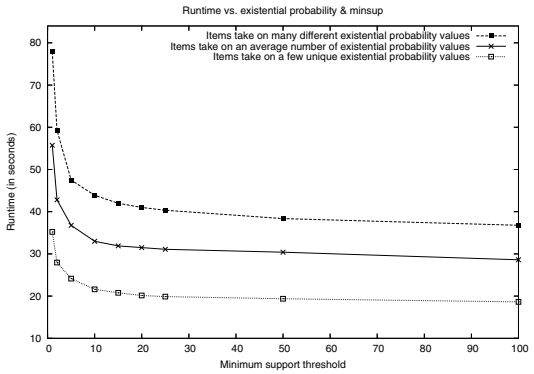
Constraint checking and identification of the mandatory & optional items for C_{SUC}. Let constraint C_{SUC} be the SUC constraint $C_6 \equiv min(S.Weight) \leq 23$kg. Our proposed U-FPS(SUC) algorithm checks each of the six domain items against the constraint C_{SUC}; it enumerates from the domain those mandatory items d & e (each having weight ≤ 23kg) as well as optional items a, b, c & f (each having weight > 23kg), i.e., $\texttt{Item}^{\texttt{M}} = \{d, e\}$ and $\texttt{Item}^{\texttt{O}} = \{a, b, c, f\}$. Once U-FPS divided the domain items into $\texttt{Item}^{\texttt{M}}$ and $\texttt{Item}^{\texttt{O}}$ for C_{SUC}, these items would play different roles in forming frequent patterns satisfying C_{SUC} because constrained frequent patterns must contain some $\texttt{Item}^{\texttt{M}}$ items and may contain some $\texttt{Item}^{\texttt{O}}$ items.

Construction of the UF-tree. Let the user-specified support threshold *minsup* be set to 1.0. Our U-FPS(SUC) algorithm constructs the UF-tree in a similar fashion as U-FPS(SAM) does for C_{SAM} except the following. After scanning the TDB and accumulating the expected support of each domain item, U-FPS(SUC) finds the frequent $\texttt{Item}^{\texttt{M}}$ item d and frequent $\texttt{Item}^{\texttt{O}}$ items a, b & c. (Items e and f are removed because they are infrequent.) Then, the algorithm scans the TDB the second time and inserts each transaction into the UF-tree. Note that, along each tree path, the $\texttt{Item}^{\texttt{M}}$ item is placed below any $\texttt{Item}^{\texttt{O}}$ items (i.e., $\texttt{Item}^{\texttt{M}}$ item is closer to the leaves and $\texttt{Item}^{\texttt{O}}$ items are closer to the root). Among $\texttt{Item}^{\texttt{O}}$ items, they are arranged in descending order of (accumulated) expected support. At the end of this UF-tree construction process, we get the UF-tree shown in Figure 1(a) capturing the content of the above TDB of uncertain data.

Mining of constrained frequent patterns from the UF-tree. Once the UF-tree is constructed, our proposed U-FPS(SUC) algorithm recursively mines constrained frequent patterns from this tree with $minsup = 1.0$ in a fashion similar—but not identical—to that in Example 1. The key difference is that, for C_{SUC}, our proposed U-FPS(SUC) algorithm only forms projected databases for $\texttt{Item}^{\texttt{M}}$ items (i.e., the item d in this example). Hence, U-FPS only needs to form the $\{d\}$-projected database (as shown in Figure 1(b)) and its subsequent $\{c, d\}$-projected database (as shown in Figure 1(c)). Note that U-FPS does not form the $\{c\}$-projected database as $c \notin \texttt{Item}^{\texttt{M}}$. By applying our proposed



(a) Changing the constraint selectivity



(b) Changing the existential probability & *minsup*

Figure 2: Runtime for the U-FPS algorithms.

U-FPS algorithm, we found constrained frequent patterns $\{d\}$:1.8, $\{a, d\}$:1.62, $\{a, c, d\}$:1.13 and $\{c, d\}$:1.26. □

4. EXPERIMENTAL RESULTS

The experimental results cited below are based on data generated by the program developed at IBM Almaden Research Center [2]. The database contains 1M records with an average transaction length of 10 items, and a domain of 1,000 items. We assigned an existential probability from the range (0,1] to every item in each transaction. In addition, we also conducted experiments using other databases including real-life databases from the UC Irvine Machine Learning Depository (e.g., mushroom data) as well as those from the Frequent Itemset Mining Implementation (FIMI) Dataset Repository. The results were consistent with those using the IBM synthetic data. So, to avoid distraction, we only show the results for the IBM data below.

All experiments were run in a time-sharing environment in a 1 GHz machine. The reported figures are based on the average of multiple runs. Runtime includes CPU and I/Os; it includes the time for constraint checking, UF-tree construction, and frequent pattern mining steps (if appropriate). In the experiments, we measured different aspects of our proposed U-FPS algorithm, which was implemented in C.

In the first set of experiments, we evaluated the *functionality* of our proposed U-FPS algorithm (when compared with unconstrained algorithms). We used a database of uncertain data and a constraint with 100% selectivity (which selects every item). We compared the frequent patterns returned by our U-FPS algorithms with those returned by existing uncertain mining algorithms (e.g., UF-growth [28]). The experimental results showed that both U-FPS and UF-growth returned the same set of frequent patterns. However, it is important to notice that UF-growth is confined to finding frequent patterns from uncertain data when using the user-specified constraint of 100% selectivity. In contrast, our U-FPS algorithms are capable of finding frequent patterns from uncertain data when using user-specified constraints of *any* selectivity.

In the second set of experiments, we also evaluated the *functionality* of our proposed U-FPS algorithm (but comparing with algorithms that mine precise data). Here, we used a user-specified constraint of some non-100% selectivity (which selects some but not all items) and a database of uncertain data containing items with existential probability of 1. We compared the frequent patterns returned by our U-FPS algorithms with those returned by existing constrained mining algorithms (e.g., FPS [29]). The experimental results showed that both U-FPS and FPS returned the same set of frequent patterns. However, it is important to notice that FPS is confined to finding frequent patterns from data with all items having existential probability of 1 (i.e., all items are guaranteed to be present). In contrast, our U-FPS algorithms are capable of finding frequent patterns from uncertain data containing items with *various* existential probability values (which can range from 0 to 1).

In the third set of experiments, we evaluated the effectiveness of constrained mining (*selectivity of constraints*). We compared the two U-FPS algorithms mentioned in this article. The y-axis of Figure 2 shows the runtime. The x-axis of Figure 2(a) shows the *selectivity* of the succinct constraints. A constraint with *pct*% selectivity means *pct*% of items is selected. The higher the *pct* value, the more is the number of selected items. As the selectivity of the SAM constraint C_{SAM} decreased (i.e., fewer items are selected), the runtime remained unchanged for the naive algorithm. It is because such an algorithm ignores C_{SAM} at the early stage of the mining process and finds all (valid and invalid) frequent patterns. In contrast, the runtimes for our U-FPS(SAM) algorithm decreased gradually as the selectivity decreased. This shows that the runtimes required by U-FPS(SAM) depend on selectivity. To elaborate, the number of valid patterns depends on the selectivity of C_{SAM}. *The computation for mining is proportional to the selectivity of the SAM constraint.* As for the the SUC constraint C_{SUC}, our U-FPS(SUC) algorithm also checks the constraints early at the initial step instead of as a post-processing step. Hence, *the computation for mining is also proportional to the selectivity of the SUC constraint* as well.

In the fourth set of experiments, we tested the effect of the *distribution of item existential probability*. Theoretically, when items take on many different existential probability values, UF-trees (for the original TDB, projected databases for singletons as well as for non-singletons) become larger and times for both UF-tree construction and frequent pattern mining become longer. On the other hand, when items take on a few unique existential probability values, the runtime becomes shorter. This is confirmed by experimental results shown in Figure 2(b).

In the fifth set of experiments, we tested the effect of *minsup*. Theoretically, the runtime decreases when *minsup* increases. Experimental results shown in Figure 2(b) confirmed that,

when *minsup* increased, fewer patterns had expected support $\geq$ *minsup*, and thus shorter runtimes were required.

In the sixth set of experiments, we tested *scalability* of our proposed U-FPS algorithms. Theoretically, the algorithms should be scalable with respect to the number of transactions. Experimental results confirmed that mining with our proposed algorithms had linear scalability.

To summarize, the above results showed the importance and the benefits of using our algorithms for efficient mining of frequent patterns that satisfy the user-specified succinct constraints from databases of uncertain data.

5. CONCLUSIONS

Frequent pattern mining plays an essential role in various knowledge discovery and data mining (KDD) tasks such as the mining of patterns like correlation, sequences, and association rules. Hence, it has been the subject of numerous studies since its introduction. Most of these studies find all the frequent patterns from collection of precise data, in which the items within each datum or transaction are definitely known and precise. However, there are many real-life situations in which the user is interested in only some tiny portions of these frequent patterns. Finding all frequent patterns would then be redundant and waste lots of computation. This calls for *constrained mining*, which aims to find only those frequent patterns that are interesting to the user. Moreover, there are also many real-life situations in which the data are uncertain. This calls for *uncertain data mining*.

A key contribution of this article is our design and development of efficient and effective algorithms, called U-FPS, to mine uncertain data for frequent patterns that satisfy the user-specified succinct constraint. By pushing succinct constraints deep inside the mining process, the amount of computation and item storage in the UF-tree is proportional to the selectivity of constraints. The U-FPS algorithms efficiently find *constrained* frequent patterns from *uncertain data*.

As ongoing work, we are developing algorithms for mining constrained frequent patterns when items in the uncertain database are not fully independent (e.g., some correlated items). Moreover, we are also developing algorithms to mine uncertain data for frequent patterns that satisfy other user-specified constraints (i.e., other than the succinct ones).

Acknowledgements. This project is partially supported by NSERC (Canada) in the form of research grants.

6. REFERENCES

[1] C.C. Aggarwal et al. Frequent pattern mining with uncertain data. In *Proc. KDD 2009*, pp. 29–37.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB 1994*, pp. 487–499.

[3] R. Agrawal et al. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD 1993*, pp. 207–216.

[4] J. Bailey et al. Fast algorithms for mining emerging patterns. In *Proc. PKDD 2002*, pp. 187–208.

[5] R.J. Bayardo Jr. (ed.) Special issue on constraints in data mining. *SIGKDD Explorations*, **4**(1), 2002.

[6] T. Bernecker et al. Probabilistic frequent itemset mining in uncertain databases. In *Proc. KDD 2009*, pp. 119–127.

[7] G. Buehrer et al. Out-of-core frequent pattern mining on a commodity PC. In *Proc. KDD 2006*, pp. 86–95.

[8] L. Cerf et al. Data-Peeler: constraint-based closed pattern mining in n-ary relations. In *Proc. SDM 2008*, pp. 37–48.

[9] R. Cheng et al. Probabilistic verifiers: evaluating constrained nearest-neighbor queries over uncertain data. In *Proc. IEEE ICDE 2008*, pp. 973–982.

[10] C.-K. Chui and B. Kao. A decremental approach for mining frequent itemsets from uncertain data. In *Proc. PAKDD 2008*, pp. 64–75.

[11] C.-K. Chui et al. Mining frequent itemsets from uncertain data. In *Proc. PAKDD 2007*, pp. 47–58.

[12] X. Dai et al. Probabilistic spatial queries on existentially uncertain data. In *Proc. SSTD 2005*, pp. 400–417.

[13] R. Ge et al. Constraint-driven clustering. In *Proc. KDD 2007*, pp. 320–329.

[14] G. Grahne et al. Efficient mining of constrained correlated sets. In *Proc. IEEE ICDE 2000*, pp. 512–521.

[15] R. Gupta et al. Quantitative evaluation of approximate frequent pattern mining algorithms. In *Proc. KDD 2008*, pp. 301–309.

[16] J. Han and J. Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explorations*, **2**(2), pp. 14–20, 2000.

[17] J. Han et al. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD 2000*, pp. 1–12.

[18] J. Hipp et al. Algorithms for association rule mining—a general survey and comparison. *SIGKDD Explorations*, **2**(1), pp. 58–64, 2000.

[19] M. Hu et al. Permu-pattern: discovery of mutable permutation patterns with proximity constraint. In *Proc. KDD 2008*, pp. 318–326.

[20] Y. Ke et al. Mining quantitative correlated patterns using an information-theoretic approach. In *Proc. KDD 2006*, pp. 227–236.

[21] A.J. Knobbe and E.K.Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *Proc. KDD 2006*, pp. 237–244.

[22] L.V.S. Lakshmanan, C.K.-S. Leung, and R. Ng. Efficient dynamic mining of constrained frequent sets. *ACM TODS*, **28**(4), pp. 337–389, 2003.

[23] L.V.S. Lakshmanan, C.K.-S. Leung, and R.T. Ng. The segment support map: scalable mining of frequent itemsets. *SIGKDD Explorations*, **2**(2), pp. 21–27, 2000.

[24] C.K.-S. Leung. Frequent itemset mining with constraints. In *Encyclopedia of Database Systems*, Springer, 2009.

[25] C.K.-S. Leung and B. Hao. Mining of frequent itemsets from streams of uncertain data. In *Proc. IEEE ICDE 2009*, pp. 1663–1670.

[26] C.K.-S. Leung et al. A tree-based approach for frequent pattern mining from uncertain data. In *Proc. PAKDD 2008*, pp. 653–661.

[27] C.K.-S. Leung et al. CanTree: a tree structure for efficient incremental mining of frequent patterns. In *Proc. IEEE ICDM 2005*, pp. 274–281.

[28] C.K.-S. Leung et al. Efficient mining of frequent patterns from uncertain data. In *Proc. IEEE ICDM Workshops 2007*, pp. 489–494.

[29] C.K.-S. Leung et al. Exploiting succinct constraints using FP-trees. *SIGKDD Explorations*, **4**(1), pp. 40–49, 2002.

[30] R.T. Ng et al. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. ACM SIGMOD 1998*, pp. 13–24.

[31] J. Pei and J. Han. Constrained frequent pattern mining: a pattern-growth view. *SIGKDD Explorations*, **4**(1), pp. 31–39, 2002.

[32] C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proc. KDD 2006*, pp. 730–735.

[33] Q. Zhang et al. Finding frequent items in probabilistic data. In *Proc. ACM SIGMOD 2008*, pp. 819–832.