Expert Systems With Applications 112 (2018) 274–287

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Probabilistic frequent itemset mining over uncertain data streams☆

Haifeng Li*, Ning Zhang, Jianming Zhu, Yue Wang, Huaihu Cao

School of Information, Central University of Finance and Economics, Beijing 100081, China

ARTICLE INFO

Article history:
Received 23 January 2018
Revised 17 June 2018
Accepted 18 June 2018
Available online 19 June 2018

Keywords:
Probabilistic frequent itemset
Uncertain data stream
Uncertain database
Data stream mining

ABSTRACT

This paper considers the problem of mining probabilistic frequent itemsets in the sliding window of an uncertain data stream. We design an effective in-memory index named *PFIT* to store the data synopsis, so the current probabilistic frequent itemsets can be output in real time. We also propose a depth-first algorithm, *PFIMoS*, to bottom-up build and maintain the *PFIT* dynamically. Because computing the probabilistic support is time consuming, we propose a method to estimate the range of probabilistic support by using the support and expected support, which can greatly reduce the runtime and memory usage. Nevertheless, massive probabilistic supports have to be computed when the minimum support is low over dense data, which may result in a drastic reduction of computing speed. We further address this problem with a heuristic rule-based algorithm, *PFIMoS+*, in which an error parameter is introduced to decrease the probabilistic support computing count. Theoretical analysis and experimental studies demonstrate that our proposed algorithms can efficiently reduce computing time and memory, ensure fast and exact mining of probabilistic data streams, and markedly outperform the state-of-the-art algorithms *TODIS-Stream* (Sun et al., 2010) and *FEMP* (Akbarinia & Masseglia, 2013).

1. Introduction

With the widely increasing use of data stream applications (Babcock, Babu, Datar, Motwani, & Widom, 2002), more researches have begun to focus on stream mining. A data stream is a new data type, that posed new challenges: Because the stream arrives continuously with a high speed, the data cannot be stored in secondary storage but only in memory; also, only one scan or limited-number scans can be done as the data arrives. In such an environment, stream mining algorithms should run in an incremental manner.

Uncertainty is often generated in real world. In an event detection system, a wireless sensor network may generate abnormal data because of the out-of-order sensor nodes or a communication delay (Chen, Yu, Gu, Jia, & Wang, 2011). Global Position System(*GPS*) can calculate positioning over only a fuzzy area because of privacy concerns or trajectory noise (Dallachiesa, Palpanas, & Ilyas, 2014).In a radar-controlled traffic monitoring application, the nearby environment will cause errors when radar is detecting vehicle speeds (Cheqing, Ke, Lei, Jeffrey, & Lin, 2009). In an animal park, the limited-pixel cameras cannot distinguish two similar animals with 100% accuracy while detecting their habits. Consequently, giving a probability to each detected value to show the occurrence will help users make a more accurate decision than if they just ignore the uncertainty. The uncertain data, however, cannot be discovered with the traditional "exist or not exist" frequency itemset mining algorithms (Cheng, Ke, & Ng, 2008).

1.1. Challenges and our contributions

The probabilistic support of each itemset has to be updated with a high cost when the data is changed, which brings many new challenges in discovering probabilistic frequent itemsets over streams: On the one hand, computing probabilistic support is time consuming, and can be improved by an incremental mining method, which, however, has to maintain a mass of historic data information for each generated itemset; this is not effective in stream-oriented applications because the streams are supposed to be unlimited but the memory is limited. On the other hand, not recording the information of each itemset saves the memory, but the runtime cost is significantly increased, which is also infeasible because the mining rate will be less than that of the arriving stream; thus, the itemsets will not be generated in real time. As a result, the main challenge is how to design an in-memory struc-

☆ This research is supported by the National Natural Science Foundation of China(61100112,61309030), Beijing Higher Education Young Elite Teacher Project(YETP0987), National Key R&D Program of China under Grant (2017YFB1400701), Discipline Construction Foundation of Central University of Finance and Economics, CUFE Young Teacher Foundation(QJJ1706).

* Corresponding author.
E-mail addresses: mydlhf@cufe.edu.cn (H. Li), zhangning75@sina.com (N. Zhang), tyzjm65@163.com (J. Zhu), yuelwang@163.com (Y. Wang), caohhu@163.com (H. Cao).

https://doi.org/10.1016/j.eswa.2018.06.042
0957-4174/

ture and a computing method to time-efficiently and memory-efficiently do mining.

Traditional algorithms can achieve the probabilistic frequent itemsets but they had the very high computing cost and memory cost, which were hard to use in the real applications, and our proposed methods can get the results quickly with low memory cost; thus, can be applied in real industry, business, finance environments and give better experience to the users. As basic data mining tools, our algorithms can be extensively applied to the expert and intelligent systems of different areas. For an example, in a financial expert system application, the real-time stock patterns can be discovered quickly and help the users to predict the stock changing trend at any time.

In this paper, we focus on the problem of how to discover probabilistic frequent itemsets over uncertain data streams, and propose the efficient algorithms to improve the performance of computer software and data analyzing tools. Our research belongs to the basic investigation of data mining, and the contributions are as follows:

1. We use a sliding window model to discover the patterns over uncertain data streams, and then introduce a concise in-memory data structure, *PFIT*(**P**robabilistic **F**requent **I**temset **T**ree), to store the data synopsis of probabilistic frequent itemsets in a bottom-up manner, which can efficiently reduce the search space.
2. We present an algorithm, *PFIMoS*(**P**robabilistic **F**requent **I**temset **M**ining **o**ver **S**treams), to incrementally discover and maintain the probabilistic frequent itemsets. *PFIMoS* uses a probabilistic support estimating method that can compute the upper and lower bounds of probabilistic support at a much lower cost. The method can reduce most of the probabilistic support computing cost, which, when used with *PFIT*, yields a much better performance.
3. We further present an improved algorithm, *PFIMoS+*, that uses a heuristic rule to reduce the count of the probabilistic support computing that is not pruned by the bounds. The heuristic rule is very effective when mining over streams, and can greatly reduce the runtime cost when the minimum support is low or the data are dense.
4. We evaluate our algorithms on 2 synthetic datasets and 4 real-life datasets. The experimental results show that the proposed algorithms are effective and efficient, and can considerably outperform the 2 state-of-the-art algorithms *TODIS-Stream* (Sun, Cheng, Cheung, & Cheng, 2010) and *FEMP* (Akbarinia & Masseglia, 2013).

1.2. Paper organization

The rest of this paper is organized as follows: In Section 2 we introduce the related works. Section 3 presents a formalized definition of uncertain data and frequent itemset mining, and then define the problem addressed in this paper. Section 4 describes current solutions to the problem. Section 5 presents the data structures, and illustrates our algorithm in detail. Section 6 describes the heuristic method. Section 7 evaluates the performance with theoretical analysis and experiment results. Finally, Section 8 concludes this paper.

2. Related works

To explore the patterns over uncertain data, 2 new definitions, the expected frequent itemset (Chui, Kao, & Hung, 2007) and the probabilistic frequent itemset (Bernecker, Kriegel, Renz, Verhein, & Zuefle, 2009), were proposed. For the expected frequent itemset, the expected support can be computed with $O(n)$ time complexity and $O(1)$ space complexity. Many algorithms have been presented to discover the expected frequent itemsets, which were mainly based on the a priori rules (Chui & Kao, 2008) and traditional data structures like FP-tree (Cuzzocrea & Leung, 2016; Leung & MacKinnon, 2014) and H-struct (Aggarwal, Li, Wang, & Wang, 2009). Calders, Garboni, and Goethals (2010b) tried to use a sampling method to achieve approximate results. In addition, some works extended the expected itemset definition and discovered the constrained expected frequent itemsets (Leung & Brajczuk, 2009), the high-utility expected itemsets (Lin, Gan, Fournier-Viger, Hong, & Tseng, 2017), the univariate itemsets (Liu, 2012), and fuzzy association rules (Pei, Zhao, Chen, Zhou, & Chen, 2013). The expected support computing is also efficient over streams because it can be done incrementally. Recently, Leung et al. proposed the *UF-streaming* (Leung & Hao, 2009), *TUF-streaming* (Leung & Jiang, 2011), and *XTUF-streaming* (Leung, Cuzzocrea, & Jiang, 2013) based on the expected frequent itemset.

Nevertheless, the expected frequent itemset cannot show the total probabilistic characteristics of the data (Zhang, Li, & Yi, 2008). In comparison, mining probabilistic frequent itemsets can better discover the probabilistic features (Zhang et al., 2008). Bernecker et al., proposed *ProApriori* (Bernecker et al., 2009) using the apriori rule to discover the probabilistic frequent itemsets, and proposed *ProFPGrowth* (Bernecker, Kriegel, Renz, Verhein, & Züfle, 2012) with *ProFP-tree* to maintain the itemsets. Sun et al. regarded the probability computation as the convolution of 2 vectors and thus used *DC* method (Sun et al., 2010) to conduct mining, in which a fast Fourier transform can reduce the computing complexity from $O(n^2)$ to $O(nlog^2n)$, so the cost was substantially reduced. The probabilistic frequent itemset and the expected frequent itemset were proved having relationships in Wang, Cheung, Cheng, Lee, and Yang (2012) based on a Poisson distribution and in Calders, Garboni, and Goethals (2010a) based on a standard normal distribution, and thus 2 approximate algorithms were proposed. Tong et al. surveyed all the methods in Tong, Chen, Cheng, and Yu (2012). Recently, Tang et al. began to study mining focused on probabilistic frequent closed itemsets and presented the exact algorithm *PFCIM* (Tang & Peterson, 2011) and the approximate algorithm *A-PFCIM* (Peterson & Tang, 2012). Tong et al. introduced a new definition of a probabilistic frequent closed itemset when the items in one transaction had the same probability, and proposed the *MPFCI* method (Tong, Chen, & Ding, 2012). Liu et al. continued to research mining compressed itemsets, which was called probabilistic representative frequent itemsets: One proposed method, *P-RFP* (Liu, Chen, & Zhang, 2013a), used a distance measure to combine the itemsets, and another method *APM* (Liu, Chen, & Zhang, 2013b) summarized the frequent itemsets by approximately joining the support probabilities, which estimated the probability that one itemset represents another. In addition, Tong et al. studied the problem of mining probabilistic frequent itemset from correlated uncertain databases (Tong, Chen, & She, 2015).

The probabilistic frequent itemset mining methods over data streams were also proposed. Akbarinia et al. presented the *FEMP* algorithm (Akbarinia & Masseglia, 2013) to discover the probabilistic frequent itemsets, in which a basic incremental probabilistic support computing method was used. Liu et al. proposed algorithm *PFIMUDS* (Lixin, Xiaolin, & Huanxiang, 2014) and stored the false positive results in a PFTree structure. Both of them employed the a priori rule to reduce computing cost, but they cannot improve the performance significantly since they use the dynamical programming method. In addition, the algorithm *TODIS* proposed in Sun et al. (2010) can also be used in streams because of its better computing cost, which will be described more clearly in Section 4.

3. Preliminaries and problem definition

This section briefly reviews the relevant concepts of uncertain databases and frequent itemset mining, and then defines the problem addressed in this paper.

3.1. Preliminaries

Given a set of distinct items $\Gamma = \{i_1, i_2, \cdots, i_{|\Gamma|}\}$ where $|\Gamma|$ is the size of Γ, a subset $X \subseteq \Gamma$ is called an itemset; supposing that each item $x_t (0 < t \leq |X|)$ in X is associated with an occurrence probability $p(x_t)$, we call X an uncertain itemset, which is denoted as $X = \{x_1, p(x_1); x_2, p(x_2); \cdots; x_{|X|}, p(x_{|X|})\}$. An uncertain transaction UT is an uncertain itemset with an ID, and $p_{UT}(X)$ is denoted as the probability of itemset X in UT. An uncertain database UD is a collection of uncertain transactions $\{UT_1, UT_2, \cdots, UT_{|UD|}\}$. An uncertain data stream US is an unlimited UD, in which the uncertain transactions continuously arrive at a high speed. Using a "possible world" semantics model, an uncertain database can be converted to exact databases. A possible world is an exact database, denoted as $PW = \{T_1, T_2, \cdots, T_{|UD|}\}$, in which $T_s \subseteq UT_s (0 < s \leq |UD|)$. Note here that we are using $T \subseteq UT$, which means that the items in UT cover the items in T without considering the probabilities of items in UT.

Assuming that the uncertain transactions are independent, a possible world has an occurrence probability $p(PW)$, which can be computed by multiplying the occurrence probability of each item $x \in PW$ if x is in T_s and UT_s, as well as the occurrence probability of each item $\bar{x}$ if x is in UT_s but not T_s, denoted as:

$$p(PW) = \prod_{T_s \in UD} \left(\prod_{x \in T_s} p(x) \right) \left(\prod_{x \notin T_s} p(\bar{x}) \right) \tag{1}$$

We denote all the possible worlds generated from UD as Ψ. The scale of Ψ is, however, exponentially increased after the original uncertain database size. That is, if we suppose that an uncertain database includes n transactions, each of which covers m_n items, then Ψ includes $2^{\sum_{i=1}^{n} m_i}$ possible worlds.

In an exact database D, given a user specified threshold(also called the minimum support) $\lambda (1 < \lambda < |D|)$, a frequent itemset is an itemset that occurs in at least λ transactions; we can also say that an itemset X is frequent if the support $\Lambda_D(X)$ is not smaller than λ. In an uncertain database UD, the frequent itemset can be defined as follows. An itemset X in each possible world PW has a support $\Lambda_{PW}(X)$ with an occurrence probability $p(PW)$, denoted as a 2-tuple $< \Lambda_{PW}(X), p(PW) >$. X in UD has $2^{\sum_{i=1}^{m} n_i}$ such tuples, which are reorganized and represented with the probability density function(PDF), denoted as $PDF(X)$. The PDF is used to define the expected frequent itemset and the probabilistic frequent itemset as follows.

Definition 1. (Expected frequent itemset (Chui et al., 2007)) For an uncertain database UD, given the minimum support λ, an itemset X is the λ-expected frequent itemset if its expected support $\Lambda^E(X)$ is not smaller than λ. Here

$$\Lambda^E(X) = \sum_{PW \in \Psi} \Lambda_{PW}(X) p(PW) \tag{2}$$

Definition 2. (Probabilistic frequent itemset (Tang & Peterson, 2011)) For an uncertain database UD, given the minimum support λ and the minimum probabilistic parameter τ, an itemset X is the probabilistic frequent itemset if the probabilistic support $\Lambda_\tau^P(X) \geq \lambda$, in which

$$\Lambda_\tau^P(X) = Max\{i | P_{\Lambda(X) \geq i} > \tau\} \tag{3}$$

where

Table 1
Summary of notations.

Notation	Description
UD	uncertain database
UT	uncertain transaction
n	uncertain database size
Ψ	possible worlds
US	uncertain data stream
η	sliding window size
λ	minimum support
λ_r	relative minimum support, $\frac{\lambda}{\eta}$
τ	minimum probabilistic parameter
$p_{UT}(X)$	the probability of itemset X in UT
$\Lambda(X)$	the support of itemset X (the frequency X occurring in UD)
$\Lambda^E(X)$, ε	the expected support of X
$PDF(X)$	the probability density function of X
$P_{\Lambda(X) \geq \lambda}(X)$	the probability of X's support that not smaller than λ
$\Lambda_\tau^P(X)$	the probabilistic support of X
$ub(\Lambda_\tau^P(X))$	upper bound of the probabilistic support
$lb(\Lambda_\tau^P(X))$	lower bound of the probabilistic support
ξ	possibility to compute the probabilistic support in $PFIMoS$
ω	error parameter

$$P_{\Lambda(X) \geq i}(X) = \sum_{PW \in \Psi, \Lambda_{PW}(X) \geq i} p(PW) \tag{4}$$

The expected frequent itemset focuses on using the expectation to measure the uncertainty. The expectation is a well-known statistic, but it cannot reflect well the uncertainty of an itemset (Bernecker et al., 2009). On the other hand, the probabilistic frequent itemset has the complete probability distribution of the support of an itemset. The minimum probabilistic parameter τ lets the user determine how possible it is for the mining itemsets to be frequent.

Lemma 1. *Given 2 itemsets X and Y in the uncertain database UD, and the minimum probabilistic parameter τ, if $X \subseteq Y$, then $\Lambda_\tau^P(X) \geq \Lambda_\tau^P(Y)$.*

Lemma 2. *Given an itemset X in the uncertain database UD, for 2 minimum probabilistic parameters τ_1 and τ_2, if $\tau_1 < \tau_2$, then $\Lambda_{\tau_1}^P(X) \geq \Lambda_{\tau_1}^P(X)$.*

Lemma 2 suggests that the larger the minimum probabilistic parameter, the smaller the possibility that an itemset becomes frequent.

3.2. Problem definition

We present our addressed problem in this paper as follows. Given an uncertain data stream US, and the minimum support λ, the minimum probabilistic parameter τ, we will explore the probabilistic frequent itemsets from the most recent arrived η transactions in US using the sliding window model. Table 1 summarizes the important symbols used in this paper.

Example 1. For an uncertain stream US in Fig. 1, the transactions in sliding window 1 are the initial data, and the sliding window size is 8. When the sliding window continues, transaction 9 is added at the head of the sliding window, and transaction 1 is deleted.

4. Naive methods

Based on the traditional stream frequent itemset mining algorithms, it is easy to find an intuitive way to address this problem. Initially, obtain the probabilistic frequent itemsets of the initial sliding window by a static divide-and-conquer(DC) mining method; while the sliding window continues, the information of the existing frequent itemsets for each transaction insertion and

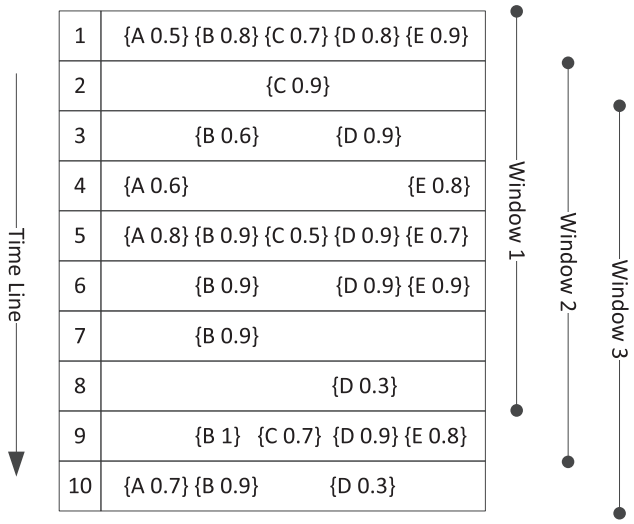


Fig. 1. A running example of sliding window.

Table 2
Complexities of the two strategies .

	TODIS-Stream	FEMP
Time complexity	$O(\eta log^2 \eta)$	$O(\eta)$
Space complexity	$O(1)$	$O(\eta)$

deletion is updated; then certain existing itemsets may be deleted, and some new itemsets will be generated. In such a method, 2 strategies can be used to maintain the in-memory data synopsis. Given the size of the sliding window, η, Table 2 shows the complexities (Sun et al., 2010) of these 2 strategies when we compute the probabilistic support of an itemset.

4.1. Memory efficient strategy(TODIS-Stream)

Regarding each update of the sliding window as a new database, we ran a static mining algorithm continuously to explore the new probabilistic frequent itemsets. During this process, a *PDF* of each existing itemset did not need to be maintained, so memory usage was greatly reduced. In our implementation, we used the state-of-the-art static mining algorithm *TODIS* (Sun et al., 2010), calling this strategy the *TODIS-Stream*. In this strategy, the space complexity of maintaining an itemset is only $O(1)$. On the other hand, the time complexity is $O(\eta log^2 \eta)$. When data streams are being mined, the continuously transactions updated in the sliding window will resulting in a very high computing cost.

Example 2. In the data stream in Fig. 1, we use itemset $\{B\}$ as an example. Initially, assuming that sliding window 1 is a static uncertain database, the probabilistic support can be computed with the *DC* method (Sun et al., 2010). After that, the *PDF* is removed and only the probabilistic support is stored; when new transaction 9 arrives and transaction 1 is deleted, sliding window 2 is regarded as the new static database, and the probabilistic support of $\{B\}$ is recomputed using the *DC* method with $O(\eta log^2(\eta)))$ time complexity.

4.2. Computing efficient strategy(FEMP)

Another strategy is to maintain a *PDF* for each existing itemset, which is actually the idea of the *FEMP* algorithm proposed in Akbarinia and Masseglia (2013). Initially, the sliding window is scanned and the *PDFs* of the itemsets are recorded in memory. When a new transaction arrives or an existing transaction is removed, the *PDFs* are updated using the dynamic programming(DP) method (Bernecker et al., 2009). During this process, the time complexity to compute the probabilistic support is $O(\eta)$. But on the other hand, it will take $O(\eta)$ space complexity to store the *PDF* of each itemset, which, when the amount of the frequent itemsets is large, will lead to a markedly increased memory usage.

Example 3. We again use itemset $\{B\}$ in Fig. 1 as an example. After the probabilistic support is computed in sliding window 1, the *PDF* will be maintained in memory for incremental computing, which has a space cost of $O(\eta)$; when new transaction 9 arrives and transaction 1 is deleted, the original *PDF* will be used to compute the new *PDF* with the time complexity $O(\eta)$, and then the probabilistic support of $\{B\}$ will be computed by scanning the *PDF*.

5. Probabilistic frequent itemset stream mining

As discussed in Section 4, simply using the traditional methods will result in either a very large computing cost or expensive memory usage. Because stream mining must be done with limited memory, we considered improving the *TODIS-Stream* method to reduce computing cost. Fig. 2 shows our idea, that is, finding a range of probabilistic support with a much lower computing cost rather than directly computing the probabilistic support. The range has the lower bound and the upper bound, which will be computed with the Chernoff-bound based method. As shown in Fig. 2(a), when the upper bound is lower than the minimum support, then the probabilistic support is lower than the minimum support, then the itemset is infrequent; On the other hand, if the lower bound is larger than the minimum support, then the probabilistic support is larger than the minimum support, then the itemset is frequent(Fig. 2(b)). Since the Chernoff-bound based method can be done with $O(1)$ time complexity, the computing cost can be reduced. Based on this idea, we introduce the data structures on which the proposed algorithm is based.

5.1. The bounds of probabilistic support

For an n-transactions uncertain database, an efficient way to compute the exact probabilistic support is the *DC* method; however, the runtime cost and memory usage are still high. Even though in Calders et al. (2010a) an approximate method was proposed to estimate the probabilistic support by the expected support, the precision depends largely on the data distribution. In this section, we try to discover an exact range of the probabilistic support with a low cost.

Theorem 1. *For an itemset X in the uncertain database UD, given the minimum probabilistic parameter τ, if X occurs in transaction UT with probability p, then the probabilistic support $\Lambda_\tau^P(X)$ is monotonically consistent with the change in p.*

Theorem 2. *For an itemset X in the uncertain dataset UD, given the minimum support λ and the minimum probabilistic parameter τ, then $\Lambda_\tau^P(X) \leq \Lambda(X)$.*

Theorem 2 shows that the support of an itemset can be regarded as an upper bound of the probabilistic support. For an itemset X, if its support $\Lambda(X)$ is smaller than the minimum support λ, then $\Lambda_\tau^P(X) < \lambda$, and it will be pruned directly.

Theorem 3. *For an itemset X in the uncertain dataset UD, given the minimum probabilistic parameter τ, we can obtain the lower bound and upper bound of the probabilistic support $\Lambda_\tau^P(X)$, denoted $lb(\Lambda_\tau^P(X))$ and $ub(\Lambda_\tau^P(X))$, as follows.*

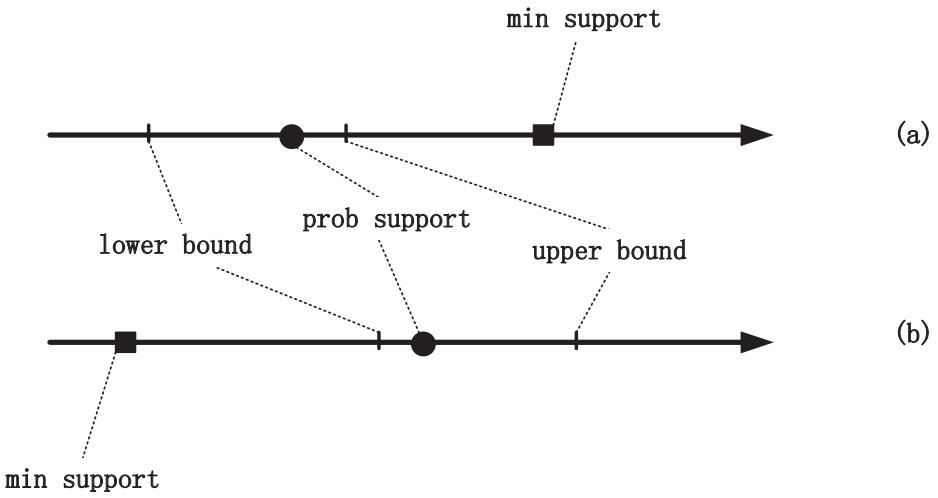


Fig. 2. The idea of our method.

$$\begin{cases} lb(\Lambda_\tau^P(X)) = max(lb'(\Lambda_\tau^P(X)), 0) \\ ub(\Lambda_\tau^P(X)) = min(ub'(\Lambda_\tau^P(X)), \Lambda(X)) \end{cases} \quad (5)$$

where

$$\begin{cases} lb'(\Lambda_\tau^P(X)) = \Lambda^E(X) - \sqrt{-2\Lambda^E(X)ln(1-\tau)} \\ ub'(\Lambda_\tau^P(X)) = \frac{2\Lambda^E(X) - ln\tau + \sqrt{ln^2\tau - 8\Lambda^E(X)ln\tau}}{2} \end{cases} \quad (6)$$

In Theorem 3, $lb'(\Lambda_\tau^P(X))$ and $ub'(\Lambda_\tau^P(X))$ need to be computed with a given minimum probabilistic parameter, so they are more flexible. In comparison to them, 0 and $\Lambda(X)$ are fixed, and the possibility that they will become the final bounds is low: 1) On condition that $\Lambda^E(X) - \sqrt{-2\Lambda^E(X)ln(1-\tau)} < 0$, that is, $\Lambda^E(X) < -2ln(1-\tau)$, 0 is the lower bound. 2) Providing that $\frac{2\Lambda^E(X) - ln\tau + \sqrt{ln^2\tau - 8\Lambda^E(X)ln\tau}}{2} > \Lambda(X)$, that is, $\Lambda^E(X) < \Lambda(X) - \frac{ln(\tau) + \sqrt{ln^2\tau - 8\Lambda(X)ln\tau}}{2}$, $\Lambda(X)$ is the upper bound.

Theorem 3 provides 2 pruning strategies. For an itemset X, if the upper bound is smaller than the minimum support λ, then X is a probabilistic infrequent itemset. Also, if the lower bound is greater than λ, then X is definitely a probabilistic frequent itemset. Thus, for an uncertain database with size n, if the minimum support λ is not within this range, one can successfully hit the target. Note that a looser upper bound can be found in Sun et al. (2010), which can only prune the infrequent itemsets. In comparison to it, our proposed upper bound is novel, and together with the lower bound can prune itemsets more efficiently.

Computing the lower and upper bounds depends on the computing of expected support and support which, over an n-transactions database, has $O(n)$ time complexity and $O(1)$ space complexity; further, updating the bounds over streams has $O(1)$ time complexity and $O(1)$ space complexity.

Example 4. We use the sliding window 1 in Fig. 1 as an example. If we set the minimum support $\lambda = 1$ and the minimum probabilistic parameter $\tau = 0.1$, then for itemset $\{C\}$, the expected support is 2.1 and the lower bound is 1.4, which is greater than 1, thus itemset $\{C\}$ is a frequent itemset. On the other hand, if we set the minimum support $\lambda = 3$ and the minimum probabilistic parameter $\tau = 0.9$, then for itemset $\{C\}$, the upper bound is 2.8, which is smaller than 3, and thus itemset $\{C\}$ is an infrequent itemset.

5.2. Probabilistic frequent itemset tree

We designed an in-memory index named *PFIT* to quickly search and incrementally maintain the itemsets. In *PFIT*, each node n_X is an itemset X. n_X is a 6-tuple $<item, sup, esup, psup, lb, ub>$, in which *item* is the last item of the current itemset X and is sorted by the lexicographical order under the same parent; *sup* is the support of X; *esup* is the expected support; *psup* is the probabilistic support; and *lb* and *ub* represent the lower and upper bounds respectively. Except for the root node, each node has a pointer to its parent. In *PFIT*, if node n_X is the parent of node n_Y, then itemset Y is the superset of itemset X. Both the root node and the intermediate nodes denote the frequent itemset, and the leaf nodes denote the infrequent itemsets or the frequent itemsets without a superset.

Regarding the initial sliding window on the stream as a static database, we can build *PFIT* with a non-incremental breath-first algorithm *BUILDTREE*. Algorithm 1 presents the details. First, the root node is generated. Second, each child node n_X, which represents the distinct items X is generated. In addition, the support, expected support, and upper and lower bounds are computed; if the bounds can determine the frequency of X, then the probabilistic support will not be computed. Finally, the descendent nodes of n_X are generated by calling *BUILDTREE* recursively.

Algorithm 1 BUILDTREE algorithm.

Require: n_X: node of *PFIT* denote itemset X;*US*: Initial Sliding Window of Uncertain Stream;λ: minimum support;τ: minimum probabilistic parameter;

```
1: if X is not frequent then
2:   return;
3: for each n_X's right sibling node n_Y do
4:   if Y is frequent then
5:     generate the child node n_{X∪Y} of n_X;
6:     compute the support Λ(X), the expected support Λ^E(X);
7:     compute the lower bound lb(Λ^P_τ(X)) andthe upper bound
       ub(Λ^P_τ(X)) (Theorem 3);
8:     if λ ∈ (lb(Λ^P_τ(X)), ub(Λ^P_τ(X))] then
9:       compute the probabilistic support Λ^P_τ(X);
10: for each new generated node n_{X∪Y} do
11:   call BUILDTREE(n_{X∪Y},US,λ, τ);
```

Example 5. Fig. 3 is the *PFIT* generated from sliding window 1. In the *PFIT*, each rectangle denotes a node and presents the itemset, the support, the expected support, the probabilistic support, and the lower and upper bounds sequentially. If the probabilistic support is not necessary to compute, it is presented with a "?". As an

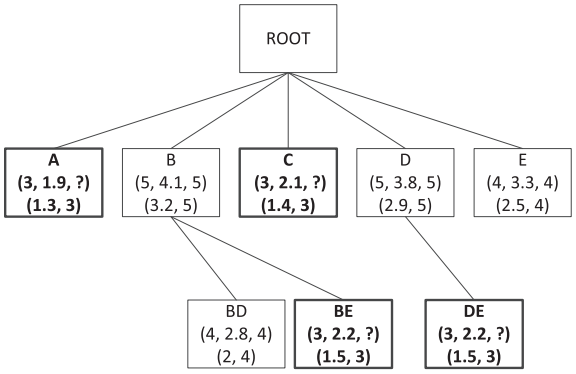


Fig. 3. PFIT corresponding to sliding window 1 for $\lambda = 4$, $\tau = 0.1$.

example, for itemset $\{B\}$, the support is 5, the expected support is 4.1, the probabilistic support is 5, the lower bound is 3.2, and the upper bound is 5. If the minimum support is 4, and the minimum probabilistic parameter is 0.1, itemsets $\{A\}$, $\{C\}$, $\{BE\}$, and $\{DE\}$ can be determined to be definitely infrequent without computing the probabilistic support, which is because their upper bounds are less than 4.

5.3. PFIMoS algorithm

After the initial *PFIT* was constructed, we used an incremental mining algorithm named *PFIMoS* to maintain the *PFIT* for the continuously updated sliding window. This section discusses the operations of adding and deleting an uncertain transaction.

5.3.1. Adding an uncertain transaction

When a new transaction *UT* arrives, it is added in the sliding window; thus, the parts of the *PFIT* that are related to transaction *UT* are traversed. For each related node n_X, the support, the expected support, the lower and upper bounds are updated. If the minimum support falls into the range between the lower and upper bounds, the probabilistic support is computed. If X is new frequent, the *BUILDTREE* function to explore all its children. The pseudo code is presented in Algorithm 2.

Algorithm 2 ADDTRANS algorithm.

Require: n_X: node of *PFIT* denote itemset X;*US*: Sliding Window of Uncertain Stream;*UT*: The added transaction;λ: minimum support;τ: minimum probabilistic parameter;

```
1: if n_X has no children nodes then
2:   return;
3: for each n_X's child node n_Y do
4:   if Y ⊆ UT then
5:     update the support Λ(Y), the expected support Λ^E(Y);
6:     update the lower bound lb(Λ^P_τ(Y)) and the upper bound
       ub(Λ^P_τ(Y)) ;
7:     if λ ∈ (lb(Λ^P_τ(Y)), ub(Λ^P_τ(Y))] then
8:       compute the probabilistic support Λ^P_τ(Y);
9: for each n_X's child node n_Y do
10:  if n_Y is new frequent node then
11:    call BUILDTREE(US,n_{X∪Y}, λ, τ);
12:  if n_Y is frequent node and Y ⊆ UT then
13:    for each n_Y's right sibling node n_Z do
14:      if n_Z is new frequent node then
15:        generate the child node n_{Y∪Z} of n_Y;
16:    call ADDTRANS(n_Y,US,UT,λ, τ);
```

Because the probabilistic support may not be computed, the key problem is how to determine the node type change by other values. In lines 10 and 14, the type change of the node is evaluated according to 6 values; that is, the original lower bound(*OLB*), the original upper bound(*OUB*), the original probabilistic support(*OPS*), the updated lower bound(*ULB*), the updated upper bound(*UUB*), and the updated probabilistic support(*UPS*). Table 3 shows the type change under different conditions, in which f denotes that a node is frequent, and i denotes that a node is infrequent. For example, at row 9, when the *OUB* is smaller than minimum support λ and the *UUBs* are greater than λ, the new probabilistic support is computed; if the *OUB* is greater than λ, the node type is regarded as having changed from infrequent to frequent, and this node is a new frequent node. Table 3 provides the pruning method when the sliding window is updated, that is, if the updated node type can be decided by computing only the bounds, the new probabilistic support is not computed. Note that adding a transaction can only increase the probabilistic support; thus, it is impossible that a node will change from frequent to infrequent.

Table 3
Type change based on different conditions(adding an uncertain transaction) .

	OLB	OUB	ULB	UUB	OPS	UPS	
1	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$\geq\lambda$		$f\rightarrow f$
2	$<\lambda$	$\geq\lambda$	$\geq\lambda$		$\geq\lambda$		
3	$\geq\lambda$						
4	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$<\lambda$	$<\lambda$	$i\rightarrow i$
5		$<\lambda$		$<\lambda$			
6		$<\lambda$		$\geq\lambda$		$<\lambda$	
7	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$i\rightarrow f$
8	$<\lambda$	$\geq\lambda$	$\geq\lambda$		$<\lambda$		
9		$<\lambda$		$\geq\lambda$		$\geq\lambda$	

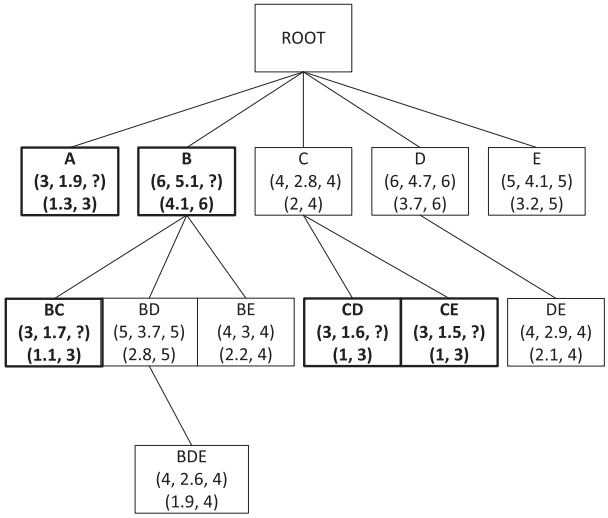


Fig. 4. Adding the ninth transaction.

Example 6. Fig. 4 is the updated *PFIT* after appending the ninth transaction {{B 1}{C 0.7}{D 0.9}{E 0.8}} to sliding window 1. 1) Itemset $\{A\}$ is the "$i\rightarrow i$" type and is ignored because it is not covered by the transaction. 2) Itemset $\{B\}$ is the "$f\rightarrow f$" type, and because the *ULB* is greater than 4, it can be determined to be a frequent itemset without recomputing the probabilistic support. 3) Itemsets $\{D\}$, $\{E\}$, and $\{BD\}$ are also the "$f\rightarrow f$" type, but the new probabilistic supports should be computed because the bounds cannot decide their frequencies. 4) Itemsets $\{C\}$, $\{BE\}$, and $\{DE\}$ are the "$i\rightarrow f$" type; their probabilistic supports should be computed, and the *BUILDTREE* function will be called for them. 5) Itemsets $\{BC\}$, $\{CD\}$, $\{CE\}$, and $\{BDE\}$ are newly generated, and only $\{BDE\}$ is required to compute the probabilistic support.

Table 4
Type change based on different conditions(deleting an uncertain transaction) .

	OLB	OUB	ULB	UUB	OPS	UPS	
1	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$\geq\lambda$	$\geq\lambda$	$f\rightarrow f$
2	$\geq\lambda$		$\geq\lambda$				
3	$\geq\lambda$		$<\lambda$			$\geq\lambda$	
4	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$<\lambda$		$i\rightarrow i$
5	$<\lambda$	$\geq\lambda$		$<\lambda$	$<\lambda$		
6		$<\lambda$					
7	$<\lambda$	$\geq\lambda$	$<\lambda$	$\geq\lambda$	$\geq\lambda$	$<\lambda$	$f\rightarrow i$
8	$<\lambda$	$\geq\lambda$		$<\lambda$	$\geq\lambda$		
9	$\geq\lambda$		$<\lambda$			$<\lambda$	

5.3.2. Deleting an uncertain transaction

An earliest arrived transaction *UT* in the sliding window will be removed when a new transaction is added. Again, the *PFIT* should be updated. Like adding an uncertain transaction, the *PFIT* that is related to the deleted transaction is also traversed. As shown in Algorithm 3, for each related node the support, the expected support, the lower and upper bounds are updated, and then the probabilistic support(Line 4–Line 8) is conditionally computed. When a node changes from frequent to infrequent, all its children nodes(Line 10–Line 14) will be deleted. Similarly, the change of the node type is determined based on the lower and upper bounds and the probabilistic support; the details are shown in Table 4.

Algorithm 3 DELTRANS algorithm.

Require: n_X: node of *PFIT* denote itemset X;*US*: Sliding Window of Uncertain Stream;*UT*: The deleted transaction;λ: minimum support;τ: minimum probabilistic parameter;

1: **if** n_X has no children nodes **then**
2: return;
3: **for** each n_X's child node n_Y **do**
4: **if** $Y \subseteq UT$ **then**
5: update the support $\Lambda(Y)$, the expected support $\Lambda^E(Y)$;
6: update the lower bound $lb(\Lambda^P_\tau(Y))$ and the upper bound $ub(\Lambda^P_\tau(Y))$;
7: **if** $\lambda \in (lb(\Lambda^P_\tau(Y)), ub(\Lambda^Y_\tau(Y))]$ **then**
8: compute the probabilistic support $\Lambda^P_\tau(Y)$;
9: **for** each n_X's child node n_Y **do**
10: **if** n_Y is new infrequent node **then**
11: delete the children nodes of n_Y;
12: **if** n_Y is frequent node and $Y \subseteq UT$ **then**
13: **for** each n_Y's right sibling node n_Z **do**
14: **if** n_Z is new infrequent node **then**
15: delete the child node $n_{Y\cup Z}$ and its children nodes;
16: call DELTRANS(n_Y,US,UT,λ, τ);

Example 7. Fig. 5 is the updated *PFIT* after deleting the first transaction {{A 0.5}{{B 0.8}{C 0.7}{D 0.8}{E 0.9}}. 1) Itemset {A} is still the "$i\rightarrow i$" type because the *OUB* is smaller than the minimum support 4. 2) Itemsets {B}, {D}, {E}, and {BD} are the "$f\rightarrow f$" type; the new probabilistic supports should be computed because the bounds cannot decide their frequencies. 3) Itemsets {C}, {BE}, and {DE} are the "$f\rightarrow i$" type, and their probabilistic supports are unnecessary to compute because the upper bounds are all smaller than the minimum support. 4) Itemsets {CD}, {CE}, and {BDE} are deleted because their parent nodes become infrequent.

5.4. Discussion

The *BUILDTREE* algorithm is the most time-consuming function because it will recursively generate the nodes and compute the support, the expected support, or the probabilistic support by scanning transactions in the sliding window. The time complexity of the *BUILDTREE* algorithm depends on the size of the sliding window, the minimum support, the minimum probabilistic parameter, and the count of nodes in the *PFIT*.

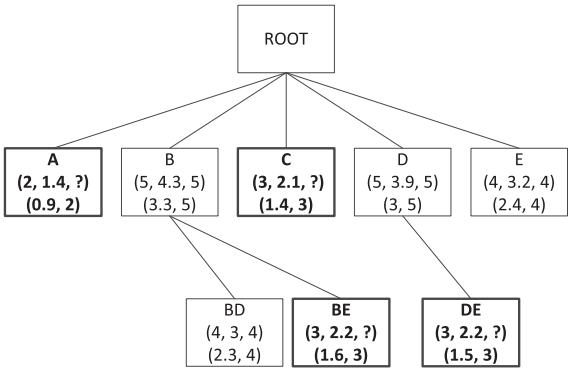


Fig. 5. Deleting the first transaction.

In the *ADDTRANS* algorithm, *BUILDTREE* is still the most time-consuming operation. Nevertheless, as we determined by experiments, the number of such invocations is very small, because an itemset may change its type only when the probabilistic support is $\lambda - 1$. Clearly, most insertions will not change the node types. On the other hand, the *DELTRANS* algorithm does not generate the new nodes; in most cases, its asymptotic time complexity is linear to the number of related nodes. Usually, it is faster to do a deletion than an addition.

Both the *ADDTRANS* and the *DELTRANS* algorithms have the bottlenecks; that is, when the minimum support is between the lower bound and the upper bound, the probabilistic support must be continuously be recomputed, which is very time consuming.

6. PFIMoS+ algorithm

Even though the probabilistic support is computing costly, the *PFIMoS* algorithm is still efficient since the possibility of computing the probabilistic support is small.

Theorem 4. *Given a stream sliding window with size η, the minimum support λ, and the minimum probabilistic parameter τ, if the expected support of an itemset X is ε, and the support is $\Lambda(X)$, then the possibility of computing the probabilistic support of X is* $\xi = min(\frac{2\sqrt{-2\varepsilon ln(1-\tau)}-ln\tau+\sqrt{ln^2\tau-8\varepsilon ln\tau}}{2\eta}, \frac{2\varepsilon-ln\tau+\sqrt{ln^2\tau-8\varepsilon ln\tau}}{2\eta}, \frac{\Lambda(X)-\varepsilon+\sqrt{-2\varepsilon ln(1-\tau)}}{\eta}, \frac{\Lambda(X)}{\eta})$.

Theorem 4 shows that the possibility ξ of computing the probabilistic support is mainly related to the expected support, the minimum probabilistic parameter and the sliding window size.

Lemma 3. *Given a stream sliding window with size η, the minimum support λ, and the minimum probabilistic parameter τ, if the expected support of an itemset X is ε, and the support is $\Lambda(X)$, then the asymptotic time complexity of determining the frequency of X is $O(\sqrt{\varepsilon}log^2\eta)$.*

Lemma 3 shows that when the sliding window size is large enough, the minimum probabilistic parameter has almost no effect on the runtime cost, which can be verified in Example 8. In addition, the smaller the expected support or the smaller the sliding window size, the more efficient the proposed method is. This also shows that the proposed algorithm may be more efficient over a sparse dataset. On the other hand, when the sliding window size increases, the proposed method can achieve a much higher speed than the *TODIS-Stream* algorithm.

Table 5
Possibility of computing the probabilistic support(η=100,000).

τ	$lb(\Lambda_\tau^p(X))$	$ub(\Lambda_\tau^p(X))$	Possibility
0.999	9628	10,004	0.38%
0.99	9697	10,014	0.32%
0.9	9785	10,046	0.26%
0.5	9882	10,118	0.24%
0.1	9954	10,216	0.26%
0.01	9986	10,306	0.32%
0.001	9996	10,375	0.38%

Example 8. For an itemset X, assuming that the sliding window size is 100000, and the expected support is 10000, then Table 5 shows the possibilities of computing the probabilistic support when the minimum probabilistic parameter τ changes. As can be seen, when $\tau = 0.5$, the possibility is the lowest, and then it will become higher no matter whether the τ change is large or small. Nevertheless, even though on the extreme condition, that $\tau = 0.001$ or $\tau = 0.999$, the possibility is only 0.38%. That is to say, at least 250 speedups can be theoretically achieved. In addition, it can be seen that the possibility changes slightly with regard to τ; that is, from 0.24% to 0.38%, which suggests the change of τ may have little effect on performance.

Theorem 4 shows that the possibility of computing the probabilistic support is small; thus, the runtime cost may be affected slightly for a static database. Nevertheless, when mining streams, once the probabilistic support is computed, it will be recomputed for the rest of the updated sliding window with a large possibility, which will significantly slow the mining. For a typical example, in Fig. 3, itemset $\{D\}$ is a frequent itemset, whose probabilistic support will be computed because the range is [2.7,5]; then, in Figs. 4 and 5, the probabilistic support must be recomputed continuously. As will be shown in the experiments, we found that over a dense dataset or under much less minimum support, this continuous computing resulted in a high time cost. In this section, we introduce a heuristic rule to address this problem and present the *PFIMoS+* algorithm to reduce the computing cost.

Theorem 5. *Suppose the current probabilistic support of an itemset X is ϕ, the minimum support is λ, and the minimum probabilistic parameter is τ. 1) If X is frequent, then the probabilistic support must be recomputed after at least $\omega = \phi - \lambda + 1$ transactions that cover X are deleted. 2) If X is infrequent, then the probabilistic support must be recomputed after at least $\omega = \lambda - \phi$ transactions that cover X are added.*

We call ω in Theorem 5 the error parameter. Theorem 5 supplies a heuristic rule, which can "wisely" detect when an itemset will change its type. That is, after the probabilistic support is computed, an itemset will not change its type until the sliding window is ω updated.

Based on the heuristic rule, we present a pruning method in our algorithm *PMFIoS+* as follows. For an itemset X, once the probabilistic support is computed, it is updated with assuming that each uncertain transaction is added or deleted on the extreme condition: If X is infrequent, its probabilistic support will be increased by 1 when the adding transaction covers X, and this operation will be done instead of directly computing the probabilistic support until the probabilistic support is greater than the minimum support. If X is frequent, its probabilistic support will be decreased by 1 when the deleting transaction covers X, and the operation continues until the probabilistic support is smaller than the minimum support. Note that when the probabilistic support is updated, the probabilistic support is not the exact value, and it can only reflect whether or not an itemset is frequent.

As can be seen, neither of the 2 operations will compute the probabilistic support until they possibly reach minimum support. In this process, the evaluation of updating the probabilistic support of each itemset is not $O(\eta log^2 \eta)$ but only $O(1)$. This improvement is mostly determined by the error parameter ω. A larger ω allows us to ignore more probabilistic support computing. On the worst condition, that is, ω is always 0, the *PMFIoS+* algorithm will have the same performance as the *PMFIoS* algorithm.

Example 9. As shown in Fig. 3, itemset $\{D\}$ is frequent in sliding window 1, and its probabilistic support is 5; that is to say, its probabilistic support may be lower than the minimum support 4 until 2 transactions covering $\{D\}$ are deleted from the sliding window. During this process, the probabilistic support is simply decreased by 1. In Fig. 1, transactions 1 and 3 cover $\{D\}$. Then, the exact probabilistic support of $\{D\}$ in the sliding window can be directly computed beginning at transaction 5.

7. Experiments

We performed the experiments to evaluate the performance of *PFIMoS* and *PFIMoS+*. The state-of-the-art algorithms *TODIS-Stream* and *FEMP* were used as the evaluation methods. The main factors that may have affected the performance were the sliding window size, the minimum support, and the minimum probabilistic parameter; as a result, they were used to evaluate the algorithms for runtime and memory cost.

7.1. Running environment and datasets

All the algorithms were implemented with C++, compiled with Visual Studio 2010 running on Microsoft Windows 7 and ran on a PC with a 3.60 GHZ Intel Core i7-4790M processor and 4GB of main memory.

Because there were no public real-world uncertain datasets, we used the datasets from Tong, Chen, Cheng, et al. (2012), which assigned a probability of Gaussian distribution to each item; this has been widely accepted in uncertain data mining research. We used 2 synthetic datasets and 4 real-life datasets. The *T25I15D100K* and *T40I10D100K* datasets were generated with an IBM synthetic data generator. The *KOSARAK* dataset contained the click-stream data of a Hungarian online news portal. The *ACCIDENTS* dataset was a report of multiple accidents. The *GAZELLE* dataset contained e-commerce click-stream data, and the *CONNECT4* dataset contained all legal 8-ply positions in the game of Connect Four in which neither player has won yet, and the next move is not forced. To each item we assigned a probability generated from a Gaussian distribution, which, to our knowledge, is widely accepted by current research. The detailed data characteristics are shown in Table 6.

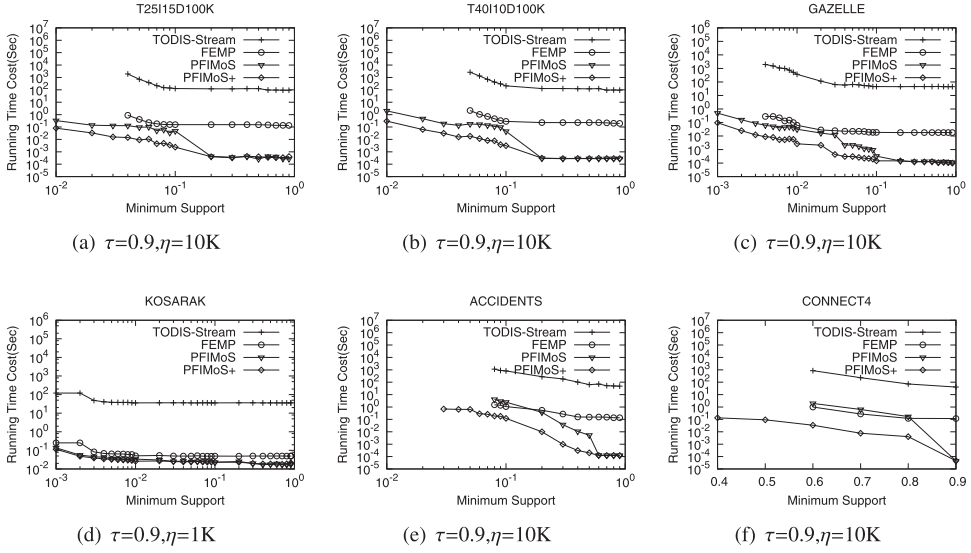
In a dataset, given the distinct item number μ and the average transaction length ν, an item will occur once in at most $\frac{\mu}{\nu}$ transactions on average. Thus, we can demonstrate the approximate correlation among transactions with $\frac{\mu}{\nu}$. For small values of $\frac{\mu}{\nu}$, the correlation of the transactions is high, and the dataset is dense.

We also determined the mean and the variance of each dataset. As can be seen, the *T25I15D100K* and *GAZELLE* datasets have the high mean and low variance, the *T40I10D100K* and *CONNECT4* datasets have high mean and high variance, the *ACCIDENTS* dataset has low mean and high variance, and the *KOSARAK* dataset has low mean and low variance.

When we ran the algorithms, we used the first η transactions as the initial sliding window to build the *PFIT*, and the rest of the transactions were regarded as the arriving stream data. Our experimental figures show the average runtime cost with regard to the sliding window size, and the maximal memory usage for each sliding window update.

Table 6
Uncertain dataSet characteristics.

DataSet	Trans count	Average size	Min size	Max size	Items count	Mean	Variance	Trans corr
T25I15D100K	100 000	26	4	67	1000	0.88	0.27	38
T40I10D100K	100 000	39	4	77	1000	0.79	0.61	25
KOSARAK	990 002	8	1	2498	41 270	0.5	0.28	5159
ACCIDENTS	340 183	33	18	51	468	0.5	0.58	14
CONNECT4	67 557	43	43	43	129	0.78	0.65	3
GAZELLE	59 602	3	2	268	497	0.94	0.08	166



(a) τ=0.9,η=10K (b) τ=0.9,η=10K (c) τ=0.9,η=10K

(d) τ=0.9,η=1K (e) τ=0.9,η=10K (f) τ=0.9,η=10K

Fig. 6. Effect of relative minimum support(runtime cost).

7.2. Effect of relative minimum support

Because the sliding window sizes were different for each dataset, instead of the minimum support, we used the relative minimum support λ_r in mining. We set the fixed minimum probabilistic parameter and sliding window size for each dataset, and compared the performance of the 4 algorithms when the relative minimum support was changed.

7.2.1. Running time cost

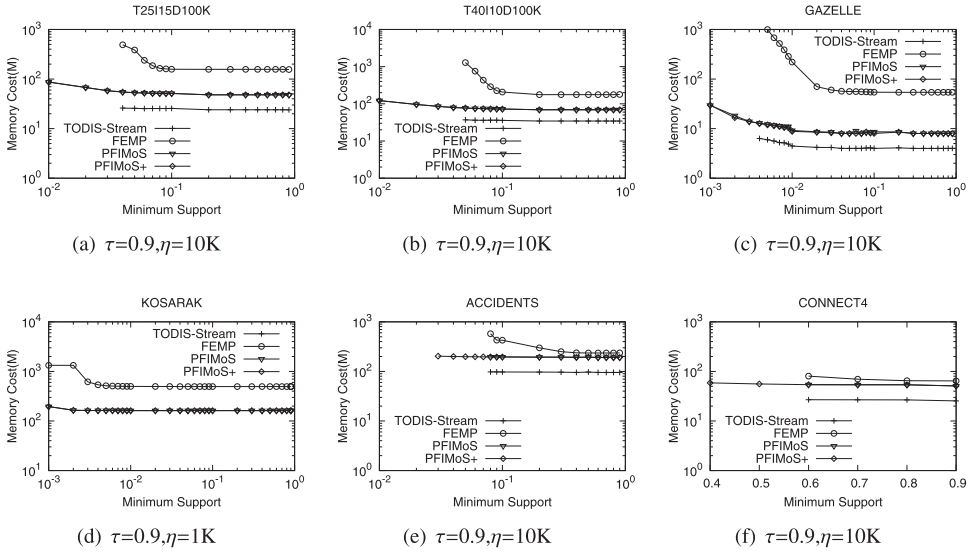
We evaluated the runtime cost in Fig. 6 of the 4 algorithms over 6 datasets. As can be seen, the runtime cost of all 4 algorithms reduced linearly with the increment of the relative minimum support; this is consistent with traditional mining. Clearly, *TODIS-Stream* was the most time-consuming algorithm; in comparison to it, the other 3 algorithms could achieve speeds of at least several hundred time faster. On the extreme condition, when the relative minimum support was high, the runtime of the proposed algorithm *PFIMoS+* was 6 orders of magnitude faster than *TODIS-Stream*. It can also be seen that *PFIMoS* was much more efficient than *FEMP* when the relative minimum support was high. Nevertheless, when the relative minimum support was low, the runtime of *FEMP* could be similar to that of the *PFIMoS* algorithm. In particular, over 2 dense datasets, *CONNECT4* and *ACCIDENTS, FEMP* was slightly more efficient than *PFIMoS*. This was due to its incremental mining manner, which has a stable $O(\eta)$ time complexity. However, both *FEMP* and *PFIMoS* could hardly be done when the relative minimum support further decreased, which we consider is reasonable. For *FEMP*, the exponential increase in the count of frequent itemsets will use more computing time; for *PFIMoS*, the probabilistic supports of more itemsets cannot be estimated by our method.

Table 7
Average probabilistic support computing count.

Dataset	λ_r	PSCC/Sliding Window		
		TODIS-Stream	PFIMoS	PFIMoS+
	0.1	199	0.983	0.052
T40I10D100K	0.08	287	1.813	0.136
τ = 0.9	0.06	1083	2.651	0.158
η = 10K	0.04	1456	2.051	0.145
	0.02	4280	7.208	0.762
CONNECT4	0.9	0.93	0	0
τ = 0.9	0.8	7	2.785	0.137
η = 10K	0.7	32	10.289	0.178
	0.6	294	36.08	0.848

In addition, we compared our 2 algorithms and noticed that when the relative minimum support was high, the algorithms had a similar runtime cost, and then the *PFIMoS+* became more efficient than *PFIMoS* when the relative minimum support became lower, especially over dense dataset. As shown in Fig. 6(f), *PFIMoS+* performed nearly 600 times as fast as *PFIMoS* when the relative minimum support changed to 0.6. This is because in most cases, *PFIMoS+* can find out the appropriate time to compute the probabilistic supports, and this can greatly reduce the count of probabilistic support computing.

Table 7 shows the average probabilistic support of computing count(*PSCC*) for each sliding window, which in our theoretical analysis was the most time-consuming operation. It can be seen that over 2 datasets, the *TODIS-Stream* must do much more probabilistic support computing than the proposed algorithms. Here we do not present the computing of *FEMP* because it is a completely incremental algorithm that did not need probabilistic support computing. Nevertheless, that did not mean that *FEMP* was the most



(a) τ=0.9,η=10K (b) τ=0.9,η=10K (c) τ=0.9,η=10K

(d) τ=0.9,η=1K (e) τ=0.9,η=10K (f) τ=0.9,η=10K

Fig. 7. Effect of relative minimum support(memory cost).

Table 8
Average itemset count.

Dataset	λ_r	IC	AIC/Trans	DIC/Trans
	0.09	1086	0.157	0.15
T25I15D100K	0.07	1744	0.299	0.259
τ=0.9	0.05	6796	0.693	0.784
η=10K	0.03	37,066	2.647	2.628
	0.01	210,451	13.656	14.625
	0.009	4104	1.271	1.234
GAZELLE	0.007	7912	1.309	1.573
τ=0.9	0.005	12,813	1.889	2.25
η=10K	0.003	22,690	3.083	4.173
	0.001	50,345	23.766	29.191

efficient one, because the incremental computing still required a $O(n)$ time complexity.

7.2.2. Running memory cost

We also evaluated the running memory cost. As shown in Fig. 7, similar to the runtime cost, the memory usage decreased as the relative minimum support increased. On memory cost, *TODIS-Stream* and *FEMP* performed opposite to their runtime cost. As can be seen, even though the runtime was very high, *TODIS-Stream* was the most memory-efficient algorithm; on the contrary, *FEMP* used a substantial amount of memory to maintain the data synopsis when the relative minimum support was low, which was also the reason that *FEMP* could not perform even though the computing cost was low. *PFIMoS* and *PFIMoS+* had a memory cost that was between those 2 algorithms. It can be accepted because their memory cost was only approximately twice as high as that of *TODIS-Stream*, which, however, was too time consuming. Note that both *PFIMoS* and *PFIMoS+* had very similar memory costs, this is because both of them used *PFIT* to maintain the in-memory data.

Table 8also shows the average maintained itemset count(*IC*) in a sliding window, the average added itemset count(*AIC*) when a new transaction arrived, and the average deleted itemset count(*DIC*) when an existing transaction was removed. As can be seen, when the minimum support became smaller, the count of the maintained itemsets increased slightly, as well as that of the added and deleted itemsets. This also verified the changing trend of the memory cost with regard to the minimum support.

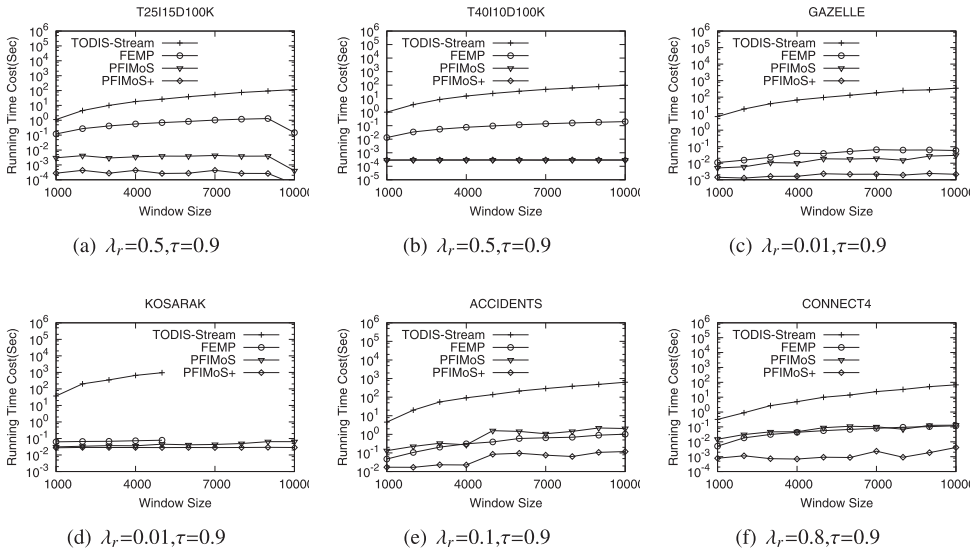
7.3. Effect of sliding window size

We evaluated the scalability of these 4 algorithms, that is, we performed the algorithms with different sliding window sizes, and set the relative minimum support and the minimum probabilistic parameter at fixed values. Please note that even though the relative minimum support was fixed, the minimum support changed because the sliding window size changed.

Fig. 8 shows the average running times. When the sliding window size increased, the runtime cost of *TODIS-Stream* increased significantly, because computing the probabilistic support is a data-size sensitive operation; that is, the time complexity is $O(\eta log^2 \eta)$; so a larger sliding window resulted in a higher computing cost. The *FEMP* algorithm, with a $O(\eta)$ time complexity, increased very smoothly. The window size had little effect on the proposed algorithm *PFIMoS* because the time complexity was $O(\sqrt{\varepsilon} log^2 \eta)$; in addition, the expected support ε also affected the mining speed, which is the reason that *PFIMoS* was more stable when running over sparse datasets, such as *GAZELLE*(Fig. 8(c)) and *KOSARAK*(Fig. 8(d)). In comparison, *PFIMoS+* was the most stable algorithm because in most cases the computing cost was $O(1)$. Note that over the *T25I15D320K* dataset, the runtime cost reduced markedly when the sliding window size became 10,000, which seemed to conflict with Lemma 3. We considered that this was reasonable because a larger sliding window may reduce the count of the probabilistic frequent itemsets, and thus result in a decreased runtime cost. This also can explain why the runtime cost may have fluctuated irregularly when the window size changed. We can also see that in comparison to the *TODIS-Stream* algorithm, the increasing sliding window size resulted in a greater improvement of the proposed algorithms, which is consistent with our theoretical analysis.

Fig. 9 shows the memory cost and verifies that the memory cost of the proposed algorithms scaled well. In comparison, even though *FEMP* had good scalability in runtime, its memory usage increased significantly with the increase in the sliding window size. This is because *FEMP* maintained the *PDF* for each itemset to guarantee that the probabilistic support could be incrementally computed. Unfortunately, the *PDF* is data-size sensitive.

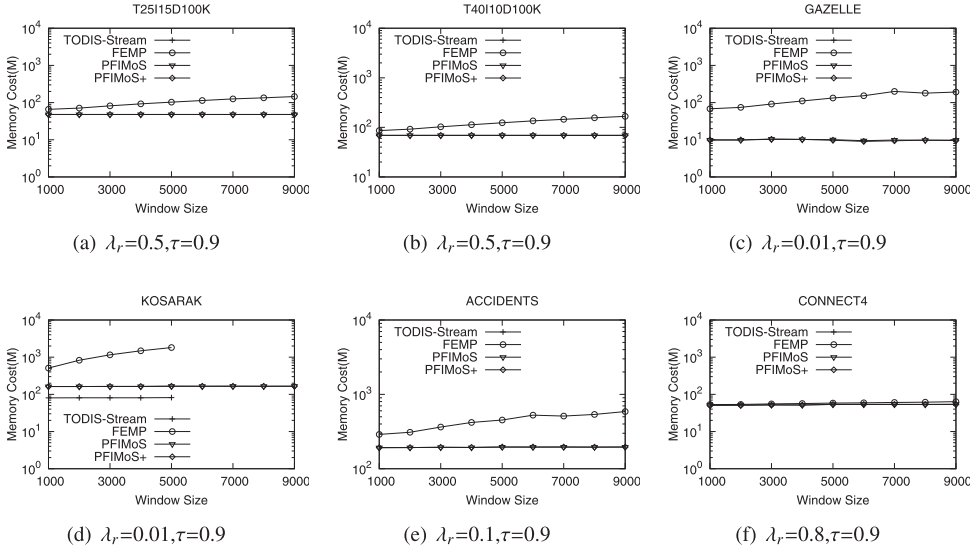
We also noticed that over the sparse *KOSARAK* dataset, *TODIS-Stream* had an expensive runtime if the sliding window size increased, and *FEMP* also had a significantly increased memory cost.



(a) λ_r=0.5,τ=0.9 (b) λ_r=0.5,τ=0.9 (c) λ_r=0.01,τ=0.9

(d) λ_r=0.01,τ=0.9 (e) λ_r=0.1,τ=0.9 (f) λ_r=0.8,τ=0.9

Fig. 8. Effect of sliding window size(runtime cost).



(a) λ_r=0.5,τ=0.9 (b) λ_r=0.5,τ=0.9 (c) λ_r=0.01,τ=0.9

(d) λ_r=0.01,τ=0.9 (e) λ_r=0.1,τ=0.9 (f) λ_r=0.8,τ=0.9

Fig. 9. Effect of sliding window size(memory cost).

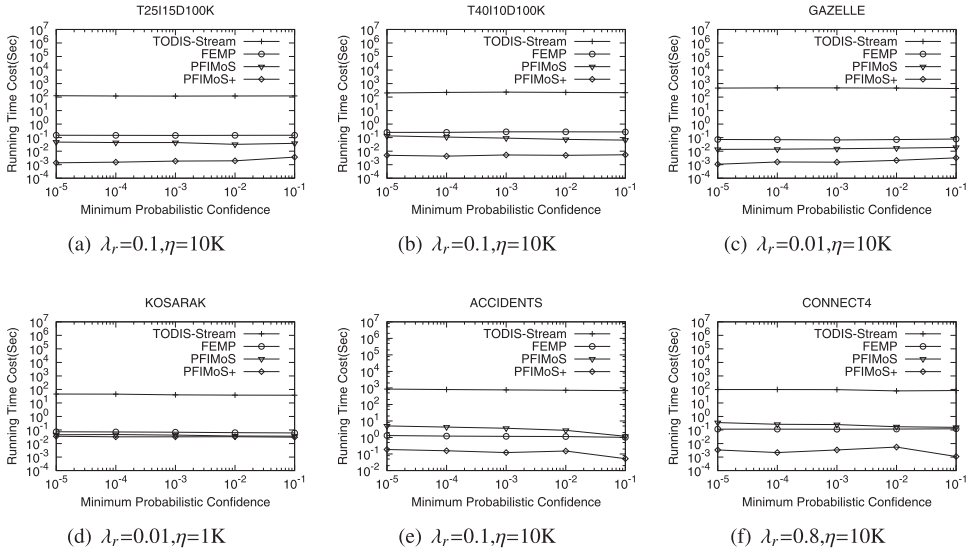
This was because *KOSARAK* had numerous distinct items. *TODIS-Stream* had to continuously compute the probabilistic supports of all the items when the sliding window was updated; thus, the computing cost was very high. Similarly, *FEMP* had to store a *PDF* for all the distinct items, which also resulted in greatly increased memory usage.

7.4. Effect of minimum probabilistic confidence

Figs. 10 and 11 show the runtime costs and memory usages respectively for different minimum probabilistic parameters. When the minimum probabilistic parameter increased, both the runtime and the memory cost of the 4 algorithms remained almost unchanged. This shows that the minimum probabilistic parameter had little effect on the performance of any of these 4 algorithms, which is in line with our theoretical analysis in Section 5.4.

7.5. Experimental summary

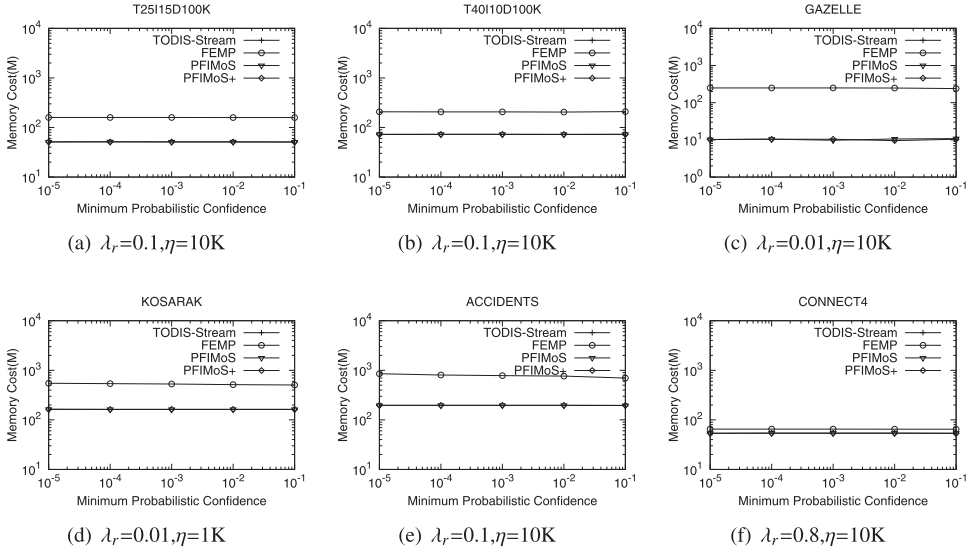
From our experimental results, we can conclude the following: First, all the algorithms were sensitive to the relative minimum support. The *FEMP* was the most memory-consuming algorithm; the proposed 2 algorithms could achieve much less memory usage, and *TODIS-Stream* was slightly more efficient. However, *TODIS-Stream* was also the most time-consuming algorithm, compared to which *FEMP* could achieve thousands-fold speedups. The *PFIMoS* algorithm was hundreds of times as fast as *FEMP* when the relative minimum support was high, but could achieve only similar or slightly lower efficiency when the relative minimum support was low. *PFIMoS+* was the most efficient algorithm, and achieved substantial speedups over *PFIMoS* with a similar memory cost. Second, due to the use of pruning methods, the proposed algorithms scaled well when the sliding window size changed. In comparison, *TODIS-Stream* also scaled well in memory cost, but its runtime drastically increased in line with the sliding window size. The runtime of *FEMP* remained almost unchanged with regard to the sliding window size, but its memory usage increased significantly. Third, both



(a) λ_r=0.1,η=10K (b) λ_r=0.1,η=10K (c) λ_r=0.01,η=10K

(d) λ_r=0.01,η=1K (e) λ_r=0.1,η=10K (f) λ_r=0.8,η=10K

Fig. 10. Effect of minimum probabilistic confidence(runtime cost).



(a) λ_r=0.1,η=10K (b) λ_r=0.1,η=10K (c) λ_r=0.01,η=10K

(d) λ_r=0.01,η=1K (e) λ_r=0.1,η=10K (f) λ_r=0.8,η=10K

Fig. 11. Effect of minimum probabilistic confidence(memory cost).

TODIS-Stream and *FEMP* were sensitive to the distinct items count; the proposed algorithms performed oppositely. Finally, the minimum probabilistic parameter had almost no effect on any the 4 algorithms.

8. Conclusions

In this research, we studied efficient algorithms for mining probabilistic frequent itemsets over the sliding window model of uncertain streams. An effective in-memory data structure named *PFIT* was used to record all the probabilistic frequent itemsets in the sliding window. We propose an algorithm, *PFIMoS*, to dynamically maintain the data synopsis in *PFIT*. In *PFIMoS*, a Chernoff-bound based method is presented to estimate the range of probabilistic support, thus reducing the computing time complexity from $O(\eta log^2\eta)$ to $O(\sqrt{\varepsilon} log^2\eta)$. To address the low efficiency of *PFIMoS* when the data are dense or the minimum support is low, we propose another algorithm, *PFIMoS+*, based on a heuristic rule, to further reduce the count of probabilistic support computing. Our experimental results show that the proposed algorithm *PFIMoS* significantly outperformed 2 state-of-the-art algorithms *TODIS-Stream* and *FEMP*. Additionally, the *PFIMoS+* algorithm achieved ten-fold speedups over *PFIMoS* with a similar memory cost in most cases.

Appendix

Proof of Lemma 2. Suppose that $\Lambda^P_{\tau_1}(X) < \Lambda^P_{\tau_1}(X)$ when $\tau_1 < \tau_2$. Let $\Lambda^P_{\tau_1}(X) = t_1$ and $\Lambda^P_{\tau_2}(X) = t_2$, then $t_1 < t_2$. According to Eq. (3), $P_{\Lambda(X)\geq t_1} > \tau_1$, and $P_{\Lambda(X)\geq t_2} > \tau_2$, and also $P_{\Lambda(X)\geq t_1+1} \leq \tau_1$, $P_{\Lambda(X)\geq t_2+1} \leq \tau_2$. Because $t_1 < t_2$, then $t_1 + 1 \leq t_2$, and then $P_{\Lambda(X)\geq t_1+1} \geq P_{\Lambda(X)\geq t_2}$(Lemma 14 in Bernecker et al., 2009). Consequently, we can get that $\tau_2 < P_{\Lambda(X)\geq t_2} \leq P_{\Lambda(X)\geq t_1+1} \leq \tau_1$, which contradicts $\tau_1 < \tau_2$. As a result, if $\tau_1 < \tau_2$, then $\Lambda^P_{\tau_1}(X) \geq \Lambda^P_{\tau_1}(X)$. □

Proof of Theorem 1. Without loss of generality, we randomly select a transaction $UT_t(t = 1 \cdots |UD|)$, and X occurs in UT_t with probability p_t. Since $P_{\Lambda_{UD}(X)\geq i} = P_{\Lambda_{UD-T_t}(X)\geq i-1} \times p_t + P_{\Lambda_{UD-T_t}(X)\geq i} \times (1 - p_t)$, in which $P_{\Lambda_{UD-T_t}(X)\geq i-1}$ is larger than $P_{\Lambda_{UD-T_t}(X)\geq i}$(Lemma

14 in Bernecker et al., 2009), then $P_{\Lambda_{UD}(X)\geq i}$ will change in line with p_t. Again, without loss of generality, we suppose p_t increases, then $P_{\Lambda_{UD}(X)\geq i}$ increases, which will result in a larger $\Lambda_\tau^P(X)$, the maximal value of i, for keeping $P_{\Lambda_{UD}(X)\geq i}$ larger than τ. □

Proof of Theorem 2. From Lemma 2 and Theorem 1, the smaller the τ, or the larger probability X occurs in transactions, the larger its probabilistic support $\Lambda_\tau^P(X)$. Then, when $\tau = 0$, and any probability p_t that itemset X occurs in transaction UT_t is 1, $\Lambda_\tau^P(X)$ reaches to its upper bound. In such a case, the probability that X has support $\Lambda(X)$ is 1, otherwise 0; according to Eq. (3), $\Lambda_\tau^P(X) = \Lambda(X)$. Thus, for any $p_t < 1$, $\Lambda_\tau^P(X) < \Lambda(X)$. □

Proof of Theorem 3. For the itemset X, we use ε to denote the expected support $\Lambda^E(X)$, also, we use t to denote the probabilistic support $\Lambda_\tau^P(X)$, which, according to Definition 2, satisfies the following equations.

$$\begin{cases} P_{\Lambda(X)\geq t} > \tau \Leftrightarrow P_{\Lambda(X)>t-1} > \tau \\ P_{\Lambda(X)\geq t+1} \leq \tau \Leftrightarrow P_{\Lambda(X)>t} \leq \tau \end{cases} \tag{7}$$

1) If we set $t = (1+\xi)\varepsilon$, i.e., $\xi = \frac{t}{\varepsilon} - 1$, where $\xi \geq 0$, that is, $t \geq \varepsilon$, then based on the Chernoff-bound, $P_{\Lambda(X)\geq t} = P_{\Lambda(X)\geq(1+\xi)\varepsilon} \leq e^{-\frac{\xi^2\varepsilon}{2+\xi}}$; based on the first inequality of Eq. (7), we can get $\tau < e^{-\frac{\xi^2\varepsilon}{2+\xi}} = e^{-\frac{(t-\varepsilon)^2}{t+\varepsilon}}$; that is to say, when $t \geq \varepsilon$, $\frac{2\varepsilon - ln\tau - \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2} < t < \frac{2\varepsilon - ln\tau + \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2}$. Since $\frac{2\varepsilon - ln\tau - \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2} \leq \varepsilon$, we can obtain the following inequality.

$$t < \frac{2\varepsilon - ln\tau + \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2} \qquad if \quad t \geq \varepsilon \tag{8}$$

2) If we set $t = (1-\xi')\varepsilon$, i.e., $\xi' = 1 - \frac{t}{\varepsilon}$, where $\xi' \geq 0$, that is, $t \leq \varepsilon$, then based on the Chernoff-bound, $P_{\Lambda(X)>t} = P_{\Lambda(X)>(1-\xi')\varepsilon} > 1 - e^{-\frac{\xi'^2\varepsilon}{2}}$; based on the second inequality of Eq. (7), we can get $\tau > 1 - e^{-\frac{\xi'^2\varepsilon}{2}}$, i.e., $-\sqrt{-\frac{2ln(1-\tau)}{\varepsilon}} < \xi' < \sqrt{-\frac{2ln(1-\tau)}{\varepsilon}}$, then when $t \leq \varepsilon$, $\varepsilon - \sqrt{-2\varepsilon ln(1-\tau)} < t < \varepsilon + \sqrt{-2\varepsilon ln(1-\tau)}$. Since $\varepsilon \leq \varepsilon + \sqrt{-2\varepsilon ln(1-\tau)}$, we can get the following inequality.

$$t > \varepsilon - \sqrt{-2\varepsilon ln(1-\tau)} \qquad if \quad t \leq \varepsilon \tag{9}$$

From Eq. (8) and (9), we can conclude that no matter t is larger or smaller than the ε, it is definitely within the range of $(lb'(\Lambda_\tau^P(X)), ub'(\Lambda_\tau^P(X)))$, where $lb'(\Lambda_\tau^P(X)) = \varepsilon - \sqrt{-2\varepsilon ln(1-\tau)}$, $ub'(\Lambda_\tau^P(X)) = \frac{2\varepsilon - ln\tau + \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2}$.

Furthermore, from Theorem 2, $0 \leq t \leq \Lambda(X)$. As a result, the lower bound is $max(lb'(\Lambda_\tau^P(X)), 0)$, and the upper bound is $min(ub'(\Lambda_\tau^P(X)), \Lambda(X))$. □

Proof of Theorem 4. Theorem 3 presents the lower bound $lb(\Lambda_\tau^P(X))$ of the probabilistic support is $\max(\varepsilon - \sqrt{-2\varepsilon ln(1-\tau)}, 0)$, and the upper bound $ub(\Lambda_\tau^P(X))$ is $\min(\frac{2\varepsilon - ln\tau + \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2}, \Lambda(X))$. Unless the minimum support λ falls into this range, the probabilistic support will not be computed. Since λ can be set from 1 to η, the probability that λ in this range is $\frac{ub(\Lambda_\tau^P(X)) - lb(\Lambda_\tau^P(X))}{\eta}$. □

Proof of Lemma 3. When the stream sliding window size η increases, the support $\Lambda(X)$ and the expected support ε also become larger, and then the minimum probabilistic parameter τ, as a given constant, has less impact on ξ. In such a case, $ub(\Lambda_\tau^P(X))$ approaches $\frac{2\varepsilon - ln\tau + \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2}$, and $lb(\Lambda_\tau^P(X))$ approaches $\varepsilon - \sqrt{-2\varepsilon ln(1-\tau)}$; thus, the possibility ξ is $\frac{2\sqrt{-2\varepsilon ln(1-\tau)} - ln\tau + \sqrt{ln^2\tau - 8\varepsilon ln\tau}}{2\eta}$, which can be denoted $\frac{c_1\sqrt{\varepsilon}}{\eta}$, where c_1 is a constant computed from τ. Consequently, computing the probabilistic support, with time complexity $O(\eta log^2\eta)$, has the possibility $\frac{c_1\sqrt{\varepsilon}}{\eta}$; otherwise, the possibility is $1 - \frac{c_1\sqrt{\varepsilon}}{\eta}$ when computing the bounds with time complexity $O(1)$. As a result, the expected computing cost is $\frac{c_1\sqrt{\varepsilon}}{\eta} \times c_2\eta log^2\eta + (1 - \frac{c_1\sqrt{\varepsilon}}{\eta}) \times c_3$ (c_2 and c_3 are constants); thus, the expected asymptotic time complexity is $O(\sqrt{\varepsilon} log^2\eta)$. □

Proof of Theorem 5. Without loss generality, we suppose that the itemset X is infrequent, only adding transaction can change X to frequent. On the extreme condition, assuming the added transaction covers X with probability 1, and it will increase the probabilistic support by 1. As a result, adding at least $\lambda - \phi$ such transactions would possibly lead X to frequent. □

References

Aggarwal, C. C., Li, Y., Wang, J., & Wang, J. (2009). Frequent pattern mining with uncertain data. In *Proceedings of the 15th acm sigkdd international conference on knowledge discovery and data mining* (pp. 29–38). ACM.

Akbarinia, R., & Masseglia, F. (2013). Fast and exact mining of probabilistic data streams. In *Machine learning and knowledge discovery in databases* (pp. 493–508). Springer.

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first acm sigmod-sigact-sigart symposium on principles of database systems* (pp. 1–16). ACM.

Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., & Zuefle, A. (2009). Probabilistic frequent itemset mining in uncertain databases. In *Proceedings of the 15th acm sigkdd international conference on knowledge discovery and data mining* (pp. 119–128). ACM.

Bernecker, T., Kriegel, H.-P., Renz, M., Verhein, F., & Züfle, A. (2012). Probabilistic frequent pattern growth for itemset mining in uncertain databases. In *Scientific and statistical database management* (pp. 38–55). Springer.

Calders, T., Garboni, C., & Goethals, B. (2010a). Approximation of frequentness probability of itemsets in uncertain data. In *Data mining (icdm), 2010 ieee 10th international conference on* (pp. 749–754). IEEE.

Calders, T., Garboni, C., & Goethals, B. (2010b). Efficient pattern mining of uncertain data with sampling. In *Advances in knowledge discovery and data mining* (pp. 480–487). Springer.

Chen, M., Yu, G., Gu, Y., Jia, Z., & Wang, Y. (2011). An efficient method for cleaning dirty-events over uncertain data in wsns. *Journal of Computer Science and Technology, 26*(6), 942–953.

Cheng, J., Ke, Y., & Ng, W. (2008). A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems, 16*(1), 1–27.

Cheqing, J., Ke, Y., Lei, C., Jeffrey, X. Y., & Lin, X. (2009). Sliding-window top-k queries on uncertain streams. *The Vldb Journal, 19*(3), 411–435.

Chui, C.-K., & Kao, B. (2008). A decremental approach for mining frequent itemsets from uncertain data. In *Advances in knowledge discovery and data mining* (pp. 64–75). Springer.

Chui, C.-K., Kao, B., & Hung, E. (2007). Mining frequent itemsets from uncertain data. In *Advances in knowledge discovery and data mining* (pp. 47–58). Springer.

Cuzzocrea, A., & Leung, C. K. (2016). Computing theoretically-sound upper bounds to expected support for frequent pattern mining problems over uncertain big data. In *International conference on information processing and management of uncertainty in knowledge-based systems* (pp. 379–392). Springer.

Dallachiesa, M., Palpanas, T., & Ilyas, I. F. (2014). Top-k nearest neighbor search in uncertain data series. *Proceedings of the VLDB Endowment, 8*(1), 13–24.

Leung, C. K.-S., & Brajczuk, D. A. (2009). Efficient algorithms for mining constrained frequent patterns from uncertain data. In *Proceedings of the 1st acm sigkdd workshop on knowledge discovery from uncertain data* (pp. 9–18). ACM.

Leung, C. K.-S., Cuzzocrea, A., & Jiang, F. (2013). Discovering frequent patterns from uncertain data streams with time-fading and landmark models. *Transactions on Large-Scale Data-and Knowledge-Centered Systems, 8*, 174–196.

Leung, C. K.-S., & Hao, B. (2009). Mining of frequent itemsets from streams of uncertain data. In *Data engineering, 2009. icde'09. ieee 25th international conference on* (pp. 1663–1670). IEEE.

Leung, C. K.-S., & Jiang, F. (2011). Frequent pattern mining from time-fading streams of uncertain data. In *Data warehousing and knowledge discovery* (pp. 252–264). Springer.

Leung, C. K.-S., & MacKinnon, R. K. (2014). Blimp: A compact tree structure for uncertain frequent pattern mining. In *Data warehousing and knowledge discovery* (pp. 115–123). Springer.

Lin, J. C.-W., Gan, W., Fournier-Viger, P., Hong, T.-P., & Tseng, V. S. (2017). Efficiently mining uncertain high-utility itemsets. *Soft Computing-A Fusion of Foundations, Methodologies and Applications, 21*(11), 2801–2820.

Liu, C., Chen, L., & Zhang, C. (2013a). Mining probabilistic representative frequent patterns from uncertain data,.

Liu, C., Chen, L., & Zhang, C. (2013b). Summarizing probabilistic frequent patterns: a fast approach. In *Proceedings of the 19th acm sigkdd international conference on knowledge discovery and data mining* (pp. 527–535). ACM.

Liu, Y.-H. (2012). Mining frequent patterns from univariate uncertain data. *Data & Knowledge Engineering, 71*(1), 47–68.

Lixin, L., Xiaolin, Z., & Huanxiang, Z. (2014). Mining of probabilistic frequent itemsets over uncertain data streams. In *Web information system and application conference (wisa), 2014 11th* (pp. 231–237). IEEE.

Pei, B., Zhao, S., Chen, H., Zhou, X., & Chen, D. (2013). Farp: Mining fuzzy association rules from a probabilistic quantitative database. *Information Sciences, 237*, 242–260.

Peterson, E. A., & Tang, P. (2012). Fast approximation of probabilistic frequent closed itemsets. In *Proceedings of the 50th annual southeast regional conference* (pp. 214–219). ACM.

Sun, L., Cheng, R., Cheung, D. W., & Cheng, J. (2010). Mining uncertain data with probabilistic guarantees. In *Proceedings of the 16th acm sigkdd international conference on knowledge discovery and data mining* (pp. 273–282). ACM.

Tang, P., & Peterson, E. A. (2011). Mining probabilistic frequent closed itemsets in uncertain databases. In *Proceedings of the 49th annual southeast regional conference* (pp. 86–91). ACM.

Tong, Y., Chen, L., Cheng, Y., & Yu, P. S. (2012). Mining frequent itemsets over uncertain databases. *Proceedings of the VLDB Endowment, 5*(11), 1650–1661.

Tong, Y., Chen, L., & Ding, B. (2012). Discovering threshold-based frequent closed itemsets over probabilistic data. In *Data engineering (icde), 2012 ieee 28th international conference on* (pp. 270–281). IEEE.

Tong, Y.-X., Chen, L., & She, J. (2015). Mining frequent itemsets in correlated uncertain databases. *Journal of Computer Science and Technology, 30*(4), 696.

Wang, L., Cheung, D. W.-L., Cheng, R., Lee, S. D., & Yang, X. S. (2012). Efficient mining of frequent item sets on large uncertain databases. *IEEE Transactions on Knowledge and Data Engineering, 24*(12), 2170–2183.

Zhang, Q., Li, F., & Yi, K. (2008). Finding frequent items in probabilistic data. In *Proceedings of the 2008 acm sigmod international conference on management of data* (pp. 819–832). ACM.