Chapter 1

FREQUENT PATTERN MINING IN DATA STREAMS

Ruoming Jin
Computer Science Department
Kent State University
jin@cs.kent.edu

Gagan Agrawal
Department of Computer Science and Engineering
The Ohio State University
agrawal@cse.ohio-state.edu

Keywords:

Abstract Frequent pattern mining is a core data mining operation and has been extensively studied over the last decade. Recently, mining frequent patterns over data streams have attracted a lot of research interests. Compared with other streaming queries, frequent pattern mining poses great challenges due to high memory and computational costs, and accuracy requirement of the mining results.

In this chapter, we overview the state-of-art techniques to mine frequent patterns over data streams. We also introduce a new approach for this problem, which makes two major contributions. First, this one pass algorithm for frequent itemset mining has deterministic bounds on the accuracy, and does not require any out-of-core summary structure. Second, because the one pass algorithm does not produce any false negatives, it can be easily extended to a two pass accurate algorithm. The two pass algorithm is very memory efficient.

1. Introduction

Frequent pattern mining focuses on discovering frequently occurring patterns from different types of datasets, including unstructured ones, such as transaction and text datasets, semi-structured ones, such as XML datasets, and structured ones, such as graph datasets. The patterns can be itemsets, sequences, subtrees, or subgraphs, etc., depending on the mining tasks and targeting datasets. Frequent patterns can not only effectively summarize the underlying datasets, providing key sights into the data, but also serve as the basic tool for many other data mining tasks, including association rule mining, classification, clustering, and change detection among others [Inokuchi et al., 2000, Zaki and Aggarwal, 2003, Huan et al., 2004, Jin and Agrawal, 2005].

Many efficient frequent pattern algorithms have been developed in the last decade [Agrawal et al., 1996, Goethals and Zaki, 2003, Han et al., 2000, Zaki et al., 1997, Kuramochi and Karypis, 2001, Yan and Han, 2002, Zaki, 2002]. These algorithms typically require datasets to be stored in persistent storage and involve two or more passes over the dataset. Recently, there has been much interest in data arriving in the form of continuous and infinite data streams. In a streaming environment, a mining algorithm must take only a single pass over the data [Babcock et al., 2002]. Such algorithms can only guarantee an approximate result.

Compared with other stream processing tasks, the unique challenges in discovering frequent patterns are in three-fold. First, frequent pattern mining needs to search a space with an exponential number of patterns. The cardinality of the answering set itself which contains all frequent patterns can be very large too. In particular, it can cost much more space to generate an approximate answering set for frequent patterns in a streaming environment. Therefore, the mining algorithm needs to be very memory-efficient. Second, frequent pattern mining relies on the down-closure property to prune infrequent patterns and generate the frequent ones. This process (even without the streaming constraint) is very compute-intensive. Consequently, keeping up the pace with high-speed data streams can be very hard for a frequent pattern-mining task. Given these challenges, a more important issue is the quality of the approximate mining results. The more accurate results usually require more memory and computations. What should be the acceptable mining results to a data miner? To deal with this problem, a mining algorithm needs to provide users the flexibility to control the accuracy of the final mining results.

In the last several years, several new mining algorithms have been proposed to find frequent patterns over data streams. In the next chapter, we will overview these new algorithms.

2. Overview

2.1 Frequent Pattern Mining: Problem Definition

Let the dataset D be a collection of objects, i.e. $D = \{o_1, o_2, \cdots, o_{|D|}\}$. Let P be the set of all possible (interesting) patterns occurring in D, g be the counting function $g : P \times O \rightarrow N$, where O is the set of objects, and N is the set of nonnegative integers. Given parameters $p \in P$, and $o \in O$, $g(p, o)$ returns the number of times p occurs in o. The support of a pattern $p \in P$ in the dataset D is defined as

$$supp(p) = \sum_{j=0}^{j=|D|} I(g(p, o_j))$$

where, I is an *indicator* function: if $g(p, o_j) > 0$, $I(g(p, o_j)) = 1$; otherwise, $I(g(p, o_j)) = 0$. Given a support level θ, the frequent patterns of P in D is the set of patterns in P which have support greater than or equal to the θ.

The first and arguably the most important frequent pattern-mining task is *frequent itemsets mining*, proposed by Rakesh Agrawal *et.al.* in 1993 [Agrawal et al., 1993]. In this setting, the objects in the dataset D are transactions or sets of items. Let *Item* be the set of all possible items in the dataset D. Then the dataset D can be represented as $D = \{I_1, \cdots, I_{|D|}\}$, where $I_j \subseteq Item, \forall j, 1 \leq j \leq |D|$. The set of all possible patterns P is the power-set of *Item*. Note that the set of all possible objects O is the same as P in this setting. The counting function g is defined upon on the set containing ($\subseteq$) relationship. In other words, if the itemset p is contained in I_j ($p \subseteq I_j$), the function $g(p, I_j)$ returns 1; otherwise, it returns 0. For instance, given a dataset D={{A,B,D,E}, {B,C,E},{A,B,E}, {A,B,C}, {A,C}, {B,C}}, and a support level θ = 50%, the frequent patterns are $\{A\}, \{B\}, \{C\}$, and $\{B, C\}$.

The majority of work in mining frequent patterns over data streams focuses on frequent itemsets mining. Many techniques developed in this setting can be served as a basis for mining other more complicated pattern mining tasks, such as graph mining [Inokuchi et al., 2000]. To simplify the discussion, this chapter will focus on mining frequent itemsets over data streams.

2.2 Data Streams

In a data stream, transactions arrive continuously and the volume of transactions can be potentially infinite. Formally, a data stream D can be defined as a sequence of transactions, $D = (t_1, t_2, \cdots, t_i, \cdots)$, where t_i is the i-th arrived transaction. To process and mine data streams, different window models are often used. A window is a subsequence between i-th and j-th arrived transactions, denoted as $W[i,j] = (t_i, t_{i+1}, \cdots, t_j), i \leq j$. A user can ask different types of frequent pattern-mining questions over different type of window models.

Landmark window: In this model, we are interested in, from a starting timepoint i to the current timepoint t, what are the frequent itemsets. In other words, we are trying to find the frequent itemsets over the window $W[i,t]$. A special case of the landmark window is when $i = 1$. In this case, we are interested in the frequent itemsets over the entire data stream. Clearly, the difficulty in solving the special case is essentially the same as the more general cases, and all of them require an efficient single-pass mining algorithm. For simplicity, we will focus on the case where the *Entire Data Stream* is the target.

Note that in this model, we treat each time-point after the starting point equally important. However, in many cases, we are more interested in the *recent* time-points. The following two models focus on such cases:

Sliding window: Given the length of the sliding window w and current timepoint t, we are interested in the frequent pattern time in the window $W[t-w+1, t]$. As time changes, the window will keep its size and move along with the current time point. In this model, we are not interested in the data which arrived before the timepoint $t - w + 1$.

Damped window model: This model assigns more weights to the recently arrived transactions. A simple way to do that is to define a *decay rate* [Chang and Lee, 2003], and use this rate to update the previously arrived transactions (by multiplication) as a new transaction arrives. Correspondingly, the count of an itemset is also defined based on the weight of each transaction.

In the next subsection, we will overview the algorithms for mining frequent itemsets on these three different window models over data streams. In addition, we would like to point out that besides the three windows we introduced above, Jiawei Han *et. al.* proposed another model called *tilted-time window* model. In this model, we are interested in frequent itemsets over a set of windows. Each window corresponds to different time granularity based on their recency. For example, we are interested in every minute for the last hour, every five minutes for the previous hour, every ten minutes for the hour before that. Moreover, the trans-

actions inside each window are also weighted. Such model can allow us to pose more complicated queries over data stream. Giannella *et. al.* have developed a variant of FP-tree, called FP-stream, for dynamically updating frequent patterns on streaming data and answering the approximate frequent itemsets for even arbitrary time intervals [Giannella et al., 2002].

2.3 Mining Algorithms

	All	Closed
Entire Data Stream	Lossy Counting [Manku and Motwani, 2002] FPDM [Yu et al., 2004]	
Sliding Window		Moment [Chi et al., 2004a]
Damped Window	estDec [Chang and Lee, 2003]	

Closed: Closed frequent itemsets

Table 1.1. Algorithms for Frequent Itemsets Mining over Data Streams

Table 1.1 lists the algorithms which have been proposed for mining frequent itemsets in the last several years. Note that *All* suggests to find all of the frequent itemsets given support level θ. Closed frequent itemsets are itemsets that are frequent but have higher frequency than all of their supersets. (If an itemset p is a subset of itemset q, q is called the superset of p.) In the following, we will briefly introduce these algorithms and their basic ideas.

Mining Algorithms for Entire Data Stream Manku and Motwani proposed the first one-pass algorithm, Lossy Counting, to find all frequent itemsets over a data stream [Manku and Motwani, 2002]. Their algorithm is *false-positive oriented* in the sense that it does not allow false negatives, and has a provable bound on false positives. It uses a user-defined error parameter ϵ to control the quality of the answering set for a given support level θ. More precisely, its answering set is guaranteed to have all itemsets whose frequency exceeds θ, and contains no itemsets whose true frequency is less than $\theta - \epsilon$. In other words, the itemsets whose frequency are between $\theta - \epsilon$ and θ possibly appear in the answering set, and are the false positives.

Recently, Yu and his colleagues proposed FPDM, which is a false-negative oriented approach, for mining frequent itemsets over data streams [Yu et al., 2004]. Their algorithm does not allow false positive, and has

a high-probability to find itemsets which are truly frequent. In particular, they use a user-defined parameter δ to control the probability to find the frequent itemsets at support level θ. Specifically, the answering set does not include any itemsets whose frequency is less than θ, and include any itemsets whose frequency exceeds θ with probability of at least $1-\delta$. It utilizes the Chernoff bound to achieve such quality control for the answering set.

Both algorithms logically partitioned the data stream into equally sized segments, and find the potentially frequent itemsets for each segment. They aggregate these locally frequent itemsets and further prune the infrequent ones. However, the number of transactions in each segment as well as the method to define potentially frequent itemsets is different for these two methods. In Lossy Counting, the number of transactions in a segment is $k \times \lceil 1/\epsilon \rceil$, and an itemset which occurs more than k times in a segment is potentially frequent. In FDPM, the number of transactions in a segment is $k \times n_0$, where n_0 is the required number of observations in order to achieve Chernoff bound with the user-defined parameter δ. In this case, an itemset whose frequency exceeds $\theta - \epsilon$, where ϵ is computed by Chernoff bound in terms of δ and the number of observations ($k \times n_0$). Note that k is a parameter (batch size) to control the size of each segment.

To theoretically estimate the space requirement for both algorithms, we consider each transaction including only a single item, and the number of transactions in the entire data stream is $|D|$. Lossy Counting will take $O(1/\epsilon log(\epsilon|D|))$ to find frequent items (1-itemsets), and FPDM-1 (the simple version of FPDM on finding frequent items) will need $O((2+2ln(2/\delta))/\theta)$.

Note that different approaches have different advantages and disadvantages. For instance, for the false positive approach, if a second pass is allowed, we can easily eliminate false positives. For the false negative approach, we can have a small answering set which have almost all the frequent itemsets, but might miss some of them (with very small probability controlled by δ).

Sliding Window Chi *et al.* have studied the problem on mining closed frequent itemsets over a sliding window of a data stream [Chi et al., 2004b]. In particular, they assume the width of sliding window is not very large, therefore, the transactions of each sliding window could be held in the main memory. Clear, such assumption is very close to the problem setting of the *incremental* association rule mining [Cheung et al., 1996]. But their focus is on how to maintain the closed frequent itemsets in an efficient way.

To deal with this problem, they proposed a new mining algorithm, called MOMENT. It utilizes the heuristic that in most cases, the sets of frequent itemsets are relatively stable for the consecutive sliding windows in the data stream. Specifically, such stability can be expressed as the fact that the boundary between the frequent itemsets and infrequent itemsets, and the boundary between closed frequent itemsets and the rest of itemsets move very slowly. Therefore, instead of generating all closed frequent itemsets for each window, they focus on monitoring such boundaries. As the key of this algorithm, an in-memory data structure, the *closed enumeration tree* (CET), is developed to efficiently monitor closed frequent itemsets as well as itemsets that form the boundary between the closed frequent itemsets and the rest of the itemsets. An efficient mechanism has been proposed to update the CET as the sliding window moves so that the boundary maintains for each sliding window.

Damped Window Model Chang and Lee studied the problem to find recently frequent itemsets over data streams using the damped window model. Specifically, in their model, the weight for an existing transaction in the data stream reduces by a decay factor,d, as a new transaction arrives. For example, the initial weight of a newly arrived transaction has weight 1, and after another transaction arrives, it will be reduced as $d = (1 \times d)$.

To keep tracking down the frequent itemsets in such a setting, they propose a new algorithm, *estDec*, which process the transaction one by one. It maintains a lattice for recording the potentially frequent itemsets and their counts, and updates the lattice for each new transaction correspondingly. Note that theoretically, the count of each itemset in the lattice will change as a new transaction arrives. But by recording an additional information for each itemset p, the time-point of the most recent transaction contains p, the algorithm only needs to update the counts for the itemsets which are the subsets of newly arrived transaction. It will reduce their counts using the constant factor d, and then increases them by one. Further, it inserts the subsets of the current transaction which are potentially frequent into the lattice. It uses a method similar to Carma [Hidber, 1999] to estimate the frequency of these newly inserted itemsets.

Discussion Among the above three different problem settings, we can see that the first one, finding the frequent itemsets over the entire data stream, is the most challenging and fundamentally important one. It can often serve as the basis to solve the latter two. For example, the current sliding window model studied in MOMENT is very similar to the incremental data mining. A more difficult problem is the case when the data in a sliding window cannot be held in the main memory. Clearly,

in such case, we need single-pass algorithms for even for a single sliding window. The main difference between the damped window model and entire data stream is that the counts of itemsets need to be adjusted for each new arrival transactions in the damped window model even the itemsets do not appear in these transactions.

In the next Section, we will introduce a new mining algorithm *Stream-Mining* we proposed recently to find frequent itemsets over the entire data stream. It is a false-positive approach (similar to Lossy Counting). It has provable (user-defined) deterministic bounds on accuracy and very memory efficient. In Section 4, we will review research works closely related with the field on frequent pattern mining over data streams. Finally, we will conclude this chapter and discuss directions for future work (Section 5).

3. New Algorithm

This section describes our new algorithm for mining frequent itemsets in a stream. Initially, we discuss a new approach for finding frequent items from Karp *et al.* [Karp et al., 2002]. We then discuss the challenges in extending this idea to frequent itemset mining, and finally outline our ideas for addressing these issues.

3.1 KPS's algorithm

Our work is derived from the recent work by Karp, Papadimitriou and Shenker on finding frequent elements (or 1-itemset) [Karp et al., 2002]. Formally, given a sequence of length N and a threshold θ $(0 < \theta < 1)$, the goal of their work is to determine the elements that occur with frequency greater than $N\theta$.

A trivial algorithm for this will involve counting the frequency of all distinct elements, and checking if any of them has the desired frequency. If there are n distinct elements, this will require $O(n)$ memory.

Their approach requires only $O(1/\theta)$ memory. Their approach can be viewed as a generalization of the following simple algorithm for finding the *majority element* in a sequence. A majority element is an element that appears more than half the time in an entire sequence. We find two distinct elements and eliminate them from the sequence. We repeat this process until only one distinct element remains in the sequence. If a majority element exists in the sequence, it will be left after this elimination. At the same time, any element remaining in the sequence is not necessarily the majority element. We can take another pass over the original sequence and check if the frequency of the remaining element is greater than $N/2$.

```
FindingFrequentItems(Sequence S, θ)
 global Set P; // Set of Potentially
 P ← ∅;              // Frequent Items
 foreach (s ∈ S) // each item in S
     if s ∈ P
         s.count++;
       else
         P ← {s} ∪ P;
         s.count = 1;
         if |P| ≥ ⌈1/θ⌉
              foreach (p ∈ P)
                   p.count − −;
                   if p.count = 0
                        P ← P − {p};
 Output(P);
```

Figure 1.1. Karp *et al.* Algorithm to Find Frequent Items

The idea can be generalized to an arbitrary θ. We can proceed as follows. We pick any $1/\theta$ distinct elements in the sequence and eliminate them together. This can be repeated until no more than $1/\theta$ distinct elements remain in the sequence. It can be claimed that any element appearing more than $N\theta$ times will be left in the sequence. The reason is that the elimination can only be performed at most $N/(1/\theta) = N\theta$ times. During each such elimination, any distinct element is removed at most once. Hence, for each distinct element, the total number of eliminations during the entire process is at most $N\theta$. Any element appearing more than $N\theta$ times will remain in the sequence. Note, however, the elements left in the sequence do not necessarily appear with frequency greater than $N\theta$. Thus, this approach will provide a superset of the elements which occur more than $N\theta$ times.

Such processing can be performed to take only a single pass on the sequence, as we show in Figure 1.1. $\mathcal{P}$ is the set of potentially frequent items. We maintain a *count* for each item in the set $\mathcal{P}$. This set is initially empty. As we process a new item from a sequence, we check if it is in the set $\mathcal{P}$. If yes, its count is incremented, otherwise, it is inserted with a count of 1. When the size of the set $\mathcal{P}$ becomes larger than $\lceil 1/\theta \rceil$, we decrement the count of each item in $\mathcal{P}$, and eliminate any item whose count has now become 0. This processing is equivalent to the eliminations we described earlier. Note that this algorithm requires only $\Omega(1/\theta)$ space. It computes a superset of frequent items. To find the

precise set of frequent items, another pass can be taken on the sequence, and the frequency of all remaining elements can be counted.

3.2 Issues In Frequent Itemset Mining

In this paper, we build a frequent itemset mining algorithm using the above basic idea. There are three main challenges when we apply this idea to mining frequent itemsets, which we summarize below.

1 *Dealing with Transaction Sequences:* The algorithm from Karp *et al.* assumes that a sequence is comprised of elements, i.e., each transaction in the sequence only contains one-items. In frequent itemset mining, each transaction has a number of items, and the length of every transaction can also be different.

2 *Dealing with k-itemsets:* Karp *et al.*'s algorithm only finds the frequent items, or 1-itemsets. In a frequent itemset mining algorithm, we need to find all k-itemsets, $k \geq 1$, in a single pass.

Note that their algorithm can be directly extended to find i-itemsets in the case where each transaction has a fixed length, l. This can be done by eliminating a group of $(1/\theta) \times \binom{l}{i}$ different i-itemsets together. This, however, requires $\Omega((1/\theta) \times \binom{l}{i})$ space, which becomes extremely high when l and i are large. Furthermore, in our problem, we have to find all i-itemsets, $i \geq 1$, in a single pass.

3 *Providing an Accuracy Bound:* Karp *et al.*'s algorithm can provably find a superset of the frequent items. However, no accuracy bound is provided for the item(set)s in the superset, which we call the potential frequent item(set)s. For example, even if an item appears just a single time, it can still possibly appear in the superset reported by the algorithm. In frequent itemset mining, we will like to improve above result, and provide a bound on the frequency of the itemsets that are reported by the algorithm.

3.3 Key Ideas

We now outline how we can address the three challenges we listed above.

Dealing with k-itemsets in a Stream of Transactions: Compared with the problem of finding frequent items, the challenges in finding frequent itemsets from a transaction sequence mainly arise due to the large number of potential frequent itemsets. This also results in high memory costs. As we stated previously, a direct application of the idea from Karp *et al.* will require $\Omega((1/\theta) \times \binom{l}{i})$ space to find potential frequent

i-itemsets, where l is the length of each transaction. This approach is prohibitively expensive when l and i are large, but can be feasible when i is small, such as 2 or 3.

Recall that most of the existing work on frequent itemset mining uses the *Apriori* property [Agrawal et al., 1996], i.e., an i-itemset can be frequent only if all subsets of this itemset are frequent. One of the drawbacks of this approach has been the large number of 2-itemsets, especially when the number of distinct items is large, and θ is small.

Our idea is to use a *hybrid* approach to mine frequent itemsets from a transaction stream. We use the idea from Karp *et al.* to determine the potential frequent 2-itemsets. Then, we use the set of potential frequent 2-itemsets and the Apriori property to generate the potential i-itemsets, for $i > 2$. This approach finds a set of potential frequent itemsets, which is guaranteed to contain all the *true* frequent itemsets, in a single pass of the stream.

Also, if a second pass of the data stream is allowed, we can eliminate all the *false* frequent itemsets from our result set. The second pass is very easy to implement, and in the rest of our discussion, we will only focus on the first pass of our algorithm.

Bounding False Positives: In order to have a accuracy bound, we propose the following criteria for the reported potential frequent itemsets after the first pass. Besides reporting all items or itemsets that occur with frequency more than $N\theta$, we want to report only the items or itemsets which appear with frequency at least $N\theta(1-\epsilon)$, where $0 < \epsilon \leq 1$. This criteria is similar to the one proposed by Manku and Motwani [Manku and Motwani, 2002].

We can achieve this goal by modifying the algorithm as shown in Figure 1.2. In the *first* step, we invoke the algorithm from Karp *et al.* with the frequency level $\theta\epsilon$. This will report a superset of items occurring with frequency more than $N\theta\epsilon$. We also record the number of eliminations, c, that occur in this step. Clearly, c is bounded by $N\theta\epsilon$. In the *second* step, we remove all items whose reported frequency is less than $N\theta - c \geq N\theta(1-\epsilon)$.

We have two claims about the above process: 1) it reports all items that occur with frequency more than $N\theta$, and 2) it only reports items which appear with frequency more than $N\theta(1-\epsilon)$. The reason for this is as follows. Consider any element that appears with frequency $N\theta$. After the first step, it will be reported in the superset with a frequency greater than $c, c \leq N\theta\epsilon$. Therefore, it will remain in the set after the second step also. Similarly, consider any item that appears with frequency less than $N\theta(1-\epsilon)$. If this item is present in the superset reported after the first

```
FindingFrequentItemsBounded(Sequence S, θ, ε)
 global Set P;
 P ← ∅;
 c ← 0; // Number of Elimination
 foreach (s ∈ S)
      if s ∈ P
           s.count++;
        else
           P ← {s} ∪ P;
           s.count = 1;
           if |P| ≥ ⌈1/(θε)⌉
                c++; // Count Eliminations
                foreach (p ∈ P)
                     p.count--;
                     if p.count = 0
                          P ← P − {p};
 foreach (p ∈ P)
      if p.count ≤ (Nθ − c)
           P ← P − {p};
 Output(P);
```

Figure 1.2. Improving Algorithm with An Accuracy Bound

step, it will be removed during the second step since $N\theta - c \geq N\theta(1-\epsilon)$. This idea can be used for frequent itemset mining also.

In the next Section, we introduce our algorithm for mining frequent itemsets from streaming data based on the above two ideas.

3.4 Algorithm Overview

We now introduce our new algorithm in three steps. In Subsection 3.5, we describe an algorithm for mining frequent itemsets from a data stream, which assumes that each transaction has the same length. In Subsection 3.6, we extend this algorithm to provide an accuracy bound on the potential frequent itemsets computed after one pass. In Subsection 3.7, we further extend the algorithm to deal with transactions of variable length.

Before detailing each algorithm, we first introduce some terminology. We are mining a stream of transactions $\mathcal{D}$. Each transaction t in this stream comprises a set of *items*, and has the length $|t|$. Let the number of transactions in $\mathcal{D}$ be $|\mathcal{D}|$. Each algorithm takes the support level θ as one parameter. An itemset in $\mathcal{D}$ to be considered frequent should occur more than $\theta|\mathcal{D}|$ times.

To store and manipulate the candidate frequent itemsets during any stage of every algorithm, a lattice $\mathcal{L}$ is maintained.

$$\mathcal{L} \;=\; \mathcal{L}_1 \cup \mathcal{L}_2 \cup \ldots \cup \mathcal{L}_k$$

where, k is largest frequent itemset, and $\mathcal{L}_i, 1 \leq i \leq k$ comprises the potential frequent i-itemsets. Note that in mining frequent itemsets, the size of the set $\mathcal{L}_1$, which is bound by the number of distinct items in the dataset, is typically not very large. Therefore, in order to simplify our discussion, we will not consider $\mathcal{L}_1$ in the following algorithms, and assume we can find the exact frequent 1-itemsets in the stream $\mathcal{D}$. Also, we will directly extend the idea from Karp *et al.* to find the potential frequent 2-itemsets.

As we stated in the previous section, we deal with all k-itemsets, $k > 2$, using the Apriori property. To facilitate this, we keep a buffer $\mathcal{T}$ in each algorithm to store the recently received transactions. The buffer will be accessed several times to find the potential frequent k-itemsets, $k > 2$.

3.5 Mining Frequent Itemsets from Fixed Length Transactions

The algorithm we present here mines frequent itemsets from a stream, under the assumption that each transaction has the same length $|t|$.

```
StreamMining-Fixed(Stream D, θ )
 global Lattice L;
 local Buffer T;
 local Transaction t;
 L ← ∅; T ← ∅;
 f ← |t| * (|t| − 1)/2;
 foreach (t ∈ D)
      T ← T ∪ {t};
      Update(t, L, 2);
      if |L_2| ≥ ⌈1/θ⌉ · f
           ReducFreq(L, 2);
           {* Deal with k − itemsets, k > 2 *}
           i ← 2;
           while L_i ≠ ∅
                i + +;
                foreach (t ∈ T)
                     Update(t, L, i);
                ReducFreq(L, i);
           T ← ∅;
 Output(L);
```

Figure 1.3. StreamMining-Fixed: Algorithm Assuming Fixed Length Transactions

```
Update(Transaction t, Lattice L, i )
 for all i subsets s of t
      if s ∈ L_i
           s.count + +;
        else if i ≤ 2
           L_i.insert(s);
        else if all i − 1 subsets of s ∈ L_{i−1}
           L_i.insert(s);

ReducFreq(Lattice L, i)
 foreach i itemsets s ∈ L_i
      s.count − −;
      if s.count = 0
           L_i.delete(s);
```

Figure 1.4. Subroutines Description

The algorithm has two interleaved phases. The *first* phase deals with 2-itemsets, and the *second* phase deals with k-itemsets, $k > 2$. The main algorithm and the associated subroutines are shown in Figures 1.3 and 1.4, respectively. Note that the two subroutines, *Update* and *ReducFreq*, are used by all the algorithms discussed in this section.

The first phase extends the Karp *et al.*'s algorithm to deal with 2-itemsets. As we stated previously, the algorithm maintains a buffer $\mathcal{T}$ which stores the recently received transactions. Initially, the buffer is empty. When a new transaction t arrives, we put it in $\mathcal{T}$. Next, we call the *Update* routine to increment counts in $\mathcal{L}_2$. This routine simply updates the count of 2-itemsets that are already in $\mathcal{L}_2$. Other 2-itemsets that are in the transaction t are inserted in the sets $\mathcal{L}_2$.

When the size of $\mathcal{L}_2$ is beyond the *threshold*, $\lceil 1/\theta \rceil f$, where f is the number of 2-itemsets per transaction, we call the procedure *ReducFreq* to reduce the count of each 2-itemsets in $\mathcal{L}_2$, and the itemsets whose count becomes zero are deleted. Invoking *ReducFreq* on $\mathcal{L}_2$ triggers the *second* phase.

The second phase of the algorithm deals with all k-itemsets, $k > 2$. This process is carried out level-wise, i.e, it proceeds from 3-itemsets to the largest potential frequent itemsets. For each transaction in the buffer $\mathcal{T}$, we enumerate all i-subsets. For any i-subset that is already in $\mathcal{L}$, the process will be the same as for a 2-itemset, i.e, we will simply increment the count. However, an i-subset that is not in $\mathcal{L}$ will be inserted in $\mathcal{L}$ only if all of its $i-1$ subsets are in $\mathcal{L}$ as well. Thus, we use the *Apriori* property.

After updating i-itemsets in $\mathcal{L}$, we will invoke the *ReducFreq* routine. Thus, the itemsets whose count is only 1 will be deleted from the lattice. This procedure will continue until there are no frequent k-itemsets in $\mathcal{L}$. At the end of this, we clear the buffer, and start processing new transactions in the stream. This will restart the first phase of our algorithm to deal with 2-itemsets.

We next discuss the correctness and the memory costs of our algorithm. Let $\mathcal{L}_i^\theta$ be the set of frequent i-itemsets with support level θ in $\mathcal{D}$, and $\mathcal{L}_i$ be the set of potential frequent i-itemsets provided by this algorithm.

THEOREM 1 *In using the algorithm StreamMining-Fixed on a set of transactions with a fixed length, for any $k \geq 2$, $\mathcal{L}_k^\theta \subseteq \mathcal{L}_k$.*

LEMMA 1.1 *In using the algorithm StreamMining-Fixed on a set of transactions with a fixed length, the size of $\mathcal{L}_2$ is bounded by $(\lceil 1/\theta \rceil + 1)\binom{|t|}{2}$.*

The proofs for the Theorem 1 and the Lemma 1.1 are available in a technical report [Jin and Agrawal, 2004]. Theorem 1 implies that any

```
StreamMining-Bounded(Stream D, θ, ε )
 global Lattice L;
 local Buffer T;
 local Transaction t;
 L ← ∅; T ← ∅;
 f ← |t| * (|t| − 1)/2;
 c ← 0; // Number of ReducFreq Invocations
 foreach (t ∈ D)
      T ← T ∪ {t};
      Update(t, L, 1);
      Update(t, L, 2);
      if |L_2| ≥ ⌈1/θε⌉ · f
           ReducFreq(L, 2);
           c++;
           i ← 2;
           while L_i ≠ ∅
                i++;
                foreach (t ∈ T)
                     Update(t, L, i);
                ReducFreq(L, i);
           T ← ∅;
 foreach s ∈ L
      if s.count ≤ θ|D| − c
           L_i.delete(s);
 Output(L);
```

Figure 1.5. StreamMining-Bounded: Algorithm with a Bound on Accuracy

frequent k-itemset is guaranteed to be in the output of our algorithm. Lemma 1.1 provides an estimate of the memory costs for $\mathcal{L}_2$.

3.6 Providing an Accuracy Bound

We now extend the algorithm from the previous subsection to provide a bound on the accuracy of the reported results. As described in Subsection 3.3, the bound is described by an user-defined parameter, ϵ, where $0 < \epsilon \leq 1$. Based on this parameter, the algorithm ensures that the frequent itemsets reported do occur more than $(1 - \epsilon)\theta|\mathcal{D}|$ times in the dataset.

The basic idea for achieving such a bound on frequent items computation was illustrated in Figure 1.2. We can extend this idea to finding frequent itemsets. Our new algorithm is described in Figure 1.5. Note that we still assume that each transaction has the same length.

This algorithm provides the new bound on accuracy in two steps. In the *first* step, we invoke the algorithm in Figure 1.3 with the frequency level $\theta\epsilon$. This will report a superset of itemsets occurring with frequency more than $N\theta\epsilon$. We record the number of invocations of *ReducFreq*, c, in the first step. Clearly, c is bounded by $N\theta\epsilon$. In the *second* step, we remove all items whose reported frequency is less than $N\theta - c \geq N\theta(1-\epsilon)$. This is achieved by the last *foreach* loop.

The new algorithm has the following property: 1) if an itemset has frequency more than θ, it will be reported. 2) if an itemset is reported as a potential frequent itemset, it must have a frequency more than $\theta(1-\epsilon)$. Theorem 2 formally states this property, and its proof is available in a technical report [Jin and Agrawal, 2004].

THEOREM 2 *In using the algorithm StreamMining-Bounded on a set of transactions with a fixed length, for any* $k \geq 2$, $\mathcal{L}_k^\theta \subseteq \mathcal{L}_k \subseteq \mathcal{L}_k^{(1-\epsilon)\theta}$.

Note that the number of invocations of *ReducFreq*, c, is usually much smaller than $N\theta\epsilon$ after processing a data stream. Therefore, an interesting property of this approach is that it produces a very small number of false frequent itemsets, even with relatively large ϵ. The experiments in [Jin and Agrawal., 2005] also support this observation.

The following lemma claims that the memory cost of $\mathcal{L}_2$ is increased by a factor proportional to $1/\epsilon$.

LEMMA 1.2 *In using the algorithm StreamMining-Bounded on a set of transactions with a fixed length, the size of* $\mathcal{L}_2$ *is bounded by* $(\lceil 1/\theta\epsilon \rceil + 1)\binom{|t|}{2}$.

3.7 Dealing with Variable Length Transactions

In this subsection, we present our final algorithm, which improves upon the algorithm from the previous subsection by dealing with variable length transactions. The algorithm is referred to as *StreamMining* and is illustrated in Figure 1.6.

When each transaction has a different length, the number of 2-itemsets in each transaction also becomes different. Therefore, we cannot simply maintain f, the number of 2-itemsets per transaction, as a constant. Instead, we maintain f as a weighted average of the number of 2-itemsets that each transaction processed so far. This weighted average is com-

StreamMining(Stream $\mathcal{D}$, θ, ϵ)
global *Lattice* $\mathcal{L}$;
local *Buffer* $\mathcal{T}$;
local *Transaction* t;
$\mathcal{L} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \emptyset$;
$f \leftarrow 0$; // *Average* $2-$ *itemset per transaction*
$c \leftarrow 0$;
foreach $(t \in \mathcal{D})$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$;
    $Update(t, \mathcal{L}, 1)$;
    $Update(t, \mathcal{L}, 2)$;
    $f \leftarrow TwoItemsetPerTransaction(t)$;
    if $|\mathcal{L}_2| \geq \lceil 1/\theta\epsilon \rceil \cdot f$
        $ReducFreq(\mathcal{L}, 2)$;
        $c++$;
        $i \leftarrow 2$;
        while $\mathcal{L}_i \neq \emptyset$
            $i++$;
            foreach $(t \in \mathcal{T})$
                $Update(t, \mathcal{L}, i)$;
            $ReducFreq(\mathcal{L}, i)$;
        $\mathcal{T} \leftarrow \emptyset$;
foreach $s \in \mathcal{L}$
    if $s.count \leq \theta|D| - c$
        $\mathcal{L}_i.delete(s)$;
$Output(\mathcal{L})$;

TwoItemsetPerTransaction(Transaction t)
global X; // *Number of 2 Itemset*
global N; // *Number of Transactions*
local f;
$N++$;
$X \leftarrow X + \begin{pmatrix} |t| \\ 2 \end{pmatrix}$;
$f \leftarrow \lceil X/N \rceil$;
if $|\mathcal{L}_2| \geq \lceil 1/\theta\epsilon \rceil \cdot f$
    $N \leftarrow N - \lceil 1/\theta\epsilon \rceil$;
    $X \leftarrow X - \lceil 1/\theta\epsilon \rceil \cdot f$;
return f;

Figure 1.6. StreamMining: Final Algorithm

puted by giving higher weightage to the recent transactions. The details are shown in the pseudo-code for the routine *TwoItemsetPerTransaction*.

To motivate the need for taking such a weighted average, consider the natural alternative, which will be maintaining f as the average number of 2-itemsets that each transaction seen so far has. This will not work correctly. For example, suppose there are 3 transactions, which have the length 2, 2, and 3, respectively, and θ is 0.5. The first two transactions will have a total of two 2-itemsets, and the third one has 6 2-itemsets. We will preform an elimination when the number of different 2-itemsets is larger than or equal to $(1/\theta) \times f$. When the first two transactions arrive, an elimination will happen (assuming that the two 2-itemsets are different). When the third one arrives, the average number of 2-itemsets is less than 3, so another elimination will be performed. Unfortunately, a frequent 2-itemset that appears in both transactions 1 and 3 will be deleted in this way.

In our approach, the number of invocations of *ReducFreq*, c, is less than $|\mathcal{D}|(\theta\epsilon)$, where $|\mathcal{D}|$ is the number of transactions processed so far in the algorithm. Lemma 1.3 formalizes this, and its proof is available in a technical report [Jin and Agrawal, 2004].

LEMMA 1.3 *$c < |\mathcal{D}|(\theta\epsilon)$ is an invariant in the algorithm StreamMining.*

Note that by using the Lemma 1.3, we can deduce that the property of the Theorem 2 still holds for mining a stream of transaction with variable transaction lengths. Formally,

THEOREM 3 *In using the algorithm StreamMining on a stream of transactions with variable lengths, for any $k \geq 2$, $\mathcal{L}_k^\theta \subseteq \mathcal{L}_k \subseteq \mathcal{L}_k^{(1-\epsilon)\theta}$.*

An interesting property of our method is that in the situation where each transaction has the same length, our final algorithm, *StreamMining* will work in the same fashion as the algorithm previously shown in Figure 1.5.

Note, however, that unlike the case with fixed length transactions, the size of $\mathcal{L}_2$ cannot be bound by a closed formula. Also, in all the algorithms discussed in this section, the size of sets $\mathcal{L}_k$, $k > 2$ also cannot be bound in any way. Our algorithms use the Apriori property to reduce their sizes.

Finally, we point out that the new algorithm is very memory efficient. For example, Lossy Counting utilizes an out-of-core (disk-resident) data structure to maintain the potentially frequent itemsets. In comparison, we do not need any such structure. On the T10.I4.N10K dataset used in their paper, we see that with 1 million transactions and a support level

of 1%, Lossy Counting requires an out-of-core data-structures on top of even a 44 MB buffer. For datasets ranging from 4 million to 20 million transactions, our algorithm only requires 2.5 MB main memory based summary. In addition, we believe that there are a number of advantages of an algorithm that does not require an out-of-core summary structure. Mining on streaming data may often be performed in mobile, hand-held, or sensor devices, where processors do not have attached disks. It is also well known that additional disk activity increases the power requirements, and battery life is an important issue in mobile, hand-held, or sensor devices. Also, while their algorithm is shown to be currently computation-bound, the disparity between processor speeds and disk speeds continues to grow rapidly. Thus, we can expect a clear advantage from an algorithm that does not require frequent disk accesses.

4. Work on Other Related Problems

In this section, we look at the work on problems that are closely related with the frequent pattern-mining problem defined in Section 2.

Scalable Frequent Pattern Mining Algorithms: A lot of research effort has been dedicated to make frequent pattern mining scalable on very large disk-resident datasets. These techniques usually focus on reducing the passes of the datasets. They typically try to find a superset or an approximate set of frequent itemsets in the first pass, and then find all the frequent itemsets as well as the counts in another one or few passes [Savasere et al., 1995], [Toivonen, 1996], [Hidber, 1999]. However, the first pass algorithms either do not have appropriate guarantees on the accuracy of frequent itemsets [Toivonen, 1996], or produces a large number of false positives [Savasere et al., 1995, Hidber, 1999]. Therefore, they are not very suitable for the streaming environment.

Mining Frequent Items in Data Streams: Given a potentially infinite sequence of items, this work tries to identify the items which have higher frequencies than a given support level. Clearly, this problem can be viewed a simple version of frequent pattern mining over the entire data stream, and indeed most of the mining algorithms discussed in this chapter are derived from these work. Algorithms in mining frequent items in data streams use different techniques, such as random sketches [Charikar et al., 2002, Cormode et al., 2002] or sampling [Toivonen, 1996, Manku and Motwani, 2002], and achieve the different space requirement. They also have either false-positive or false-negative properties. Interested user can look at [Yu et al., 2004] for more detailed comparison.

Finding Top-k Items in Distributed Data Streams: Assuming we have several distributed data streams and each item might carry different

weights at each of its arrival, the problem is to find the k items which has the highest global weight. Olston and his colleagues have studied this problem, which they call *top-k monitoring queries* [Babcock et al., 2003, Manjhi et al., 2005]. Clearly, in order to maintain the global top-k items in a distributed streaming environment, frequent communication and synchronization is needed. Therefore, the focus of their research is on reducing the communication cost. They have proposed a method to achieve such goal by constraining each individual data streams with an arithmetic condition. The communication is only necessary when the arithmetic condition is violated.

Finding Heavy Hitters in Data Streams: Cormode *et. al.* studied the problem to efficiently identify *heavy hitters* in data streams [Cormode et al., 2003]. It can be looked as an interesting variation of frequent-items mining problem. In this problem, there is a hierarchy among different items. Given a frequency level ϕ, the count of an item i in the hierarchy include all the items which are descendants of i, and whose counts are less than ϕ. An item whose counts exceeds ϕ is called Hierarchy Heavy Hitter (HHH), and we want to find all HHHs in a data stream. They have presented both deterministic and randomized algorithms to find HHHs.

Frequent Temporal Patterns over Data Streams: Considering a sliding window moves along a data stream, we monitor the counts for an itemset at each time point. Clearly, this sequence of counting information for a given itemset can be formulated as a time series. Inspired by such observation, Teng *et. al.* have developed an algorithm to find frequent patterns in sliding window model and collect sufficient statistics for a regression-based analysis of such time series [Teng et al., 2003]. They have showed such a framework is applicable in answering itemsets queries with flexible time-intervals, trend identification, and change detection.

Mining Semi-Structured Data Streams: Asai *et. al.* developed an efficient algorithm for mining frequent rooted ordered trees from a semi-structured data stream [Asai et al., 2002]. In this problem setting, they model a semi-structured dataset as a tree with infinite width but finite height. Traversing the tree with a left-most order generates the data stream. In other words, each item in the data stream is a node in the tree, and its arriving order is decided by the left-most traversing of the tree. They utilize the method in Carma [Hidber, 1999] for candidate subtree generation.

5. Conclusions and Future Directions

In this chapter, we gave an overview of the state-of-art in algorithms for frequent pattern mining over data streams. We also introduced a new approach for frequent itemset mining. We have developed a new one-pass algorithm for streaming environment, which has deterministic bounds on the accuracy. Particularly, it does not require any out-of-core memory structure and is very memory efficient in practice.

Though the existing one-pass mining algorithms have been shown to be very accurate and faster than traditional multi-pass algorithms, the experimental results show that they are still computationally expensive, meaning that if the data arrives too rapidly, the mining algorithms will not able to handle the data. Unfortunately, this can be the case for some high-velocity streams, such as network flow data. Therefore, new techniques are needed to increase the speed of stream mining tasks. We conclude this chapter with a list of future research problems to address this challenge.

mining maximal and other condensed frequent itemsets in data streams: Maximal frequent itemsets (MFI), and other condensed frequent itemsets, such as the $\delta - cover$ proposed in [Xin et al., 2005], provide good compression of the frequent itemsets. Mining them are very likely to reduce the mining costs in terms of both computation and memory over data streams. However, mining such kinds of compressed pattern set poses new challenges. The existing techniques will logically partition the data stream into segments, and mine potentially frequent itemsets each segment. In many compressed pattern sets, for instance, MFI, if we just mine MFI for each segment, it will be very hard to find the global MFI. This is because the MFI can be different in each segment, and when we combine them together, we need the counts for the itemsets which are frequent but not maximal. However, estimating the counts for these itemsets can be very difficult. The similar problem occurs for other condensed frequent itemsets mining. Clearly, new techniques are necessary to mine condensed frequent itemsets in data steams.

Online Sampling for Frequent Pattern Mining: The current approaches involve high-computational cost for mining the data streams. One of the main reasons is that all of them try to maintain and deliver the potentially frequent patterns at any time. If the data stream arrives very rapidly, this could be unrealistic. Therefore, one possible approach is to maintain a sample set which best represents the data stream and provide good estimation of the frequent itemsets.

Compared with existing sampling techniques [Toivonen, 1996, Chen et al., 2002, Bronnimann et al., 2003] on disk-resident datasets for frequent itemsets mining, sampling data streams brings some new issues. For example, the underlying distribution of the data stream can change from time to time. Therefore, sampling needs to adapt to the data stream. However, it will be quite difficult to monitor such changes if we do not mine the set of frequent itemsets directly. In addition, the space requirement of the sample set can be an issue as well. As pointed by Manku and Motwani [Manku and Motwani, 2002], methods similar to concise sampling [Gibbons and Matias, 1998] might be helpful to reduce the space and achieve better mining results.

References

[Agrawal et al., 1996] Agrawal, R., Mannila, H., Srikant, R., Toivonent, H., and Verkamo., A. Inkeri (1996). Fast discovery of association rules. In Fayyad, U. and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA.

[Agrawal et al., 1993] Agrawal, Rakesh, Imielinski, Tomasz, and Swami, Arun (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 207–216.

[Asai et al., 2002] Asai, Tatsuya, Arimura, Hiroki, Abe, Kenji, Kawasoe, Shinji, and Arikawa, Setsuo (2002). Online algorithms for mining semi-structured data stream. In *ICDM*, pages 27–34.

[Babcock et al., 2002] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and Issues in Data Stream Systems. In *Proceedings of the 2002 ACM Symposium on Principles of Database Systems (PODS 2002) (Invited Paper)*. ACM Press.

[Babcock et al., 2003] Babcock, B., Chaudhuri, S., and Das, G. (2003). Dynamic Sampling for Approximate Query Processing. In *Proceedings of the 2003 ACM SIGMOD Conference*. ACM Press.

[Bronnimann et al., 2003] Bronnimann, Herve;, Chen, Bin, Dash, Manoranjan, Haas, Peter, and Scheuermann, Peter (2003). Efficient data reduction with ease. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68.

[Chang and Lee, 2003] Chang, Joong Hyuk and Lee, Won Suk (2003). Finding recent frequent itemsets adaptively over online data streams. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*.

[Charikar et al., 2002] Charikar, Moses, Chen, Kevin, and Farach-Colton, Martin (2002). Finding frequent items in data streams. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming.*

[Chen et al., 2002] Chen, Bin, Haas, Peter, and Scheuermann, Peter (2002). A new two-phase sampling based algorithm for discovering association rules. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–468.

[Cheung et al., 1996] Cheung, D., Han, J., NG, V., and Wong, C. (1996). Maintenance of discovered association rules in large databases : an incremental updating technique. In *ICDE.*

[Chi et al., 2004a] Chi, Yun, Wang, Haixun, Yu, Philip S., and Muntz, Richard R. (2004a). Moment: Maintaining closed frequent itemsets over a stream sliding window. In *ICDM*, pages 59–66.

[Chi et al., 2004b] Chi, Yun, Yang, Yirong, and Muntz, Richard R. (2004b). Hybridtreeminer: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms. In *The 16th International Conference on Scientific and Statistical Database Management (SSDBM'04).*

[Cormode et al., 2002] Cormode, G., Datar, M., Indyk, P., and Muthukrishnan, S. (2002). Comparing Data Streams Using Hamming Norms. In *Proceedings of Conference on Very Large Data Bases (VLDB)*, pages 335–345.

[Cormode et al., 2003] Cormode, Graham, Korn, Flip, Muthukrishnan, S., and Srivastava, Divesh (2003). Finding hierarchical heavy hitters in data streams. In *VLDB*, pages 464–475.

[Giannella et al., 2002] Giannella, C., Han, Jiawei, Pei, Jian, Yan, Xifeng, and Yu, P. S. (2002). Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In *Proceedings of the NSF Workshop on Next Generation Data Mining.*

[Gibbons and Matias, 1998] Gibbons, Phillip B. and Matias, Yossi (1998). New sampling-based summary statistics for improving approximate query answers. In *ACM SIGMOD*, pages 331–342.

[Goethals and Zaki, 2003] Goethals, Bart and Zaki, Mohammed J. (2003). Workshop Report on Workshop on Frequent Itemset Mining Implementations (FIMI).

[Han et al., 2000] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*.

[Hidber, 1999] Hidber, C. (1999). Online Association Rule Mining. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 145–156. ACM Press.

[Huan et al., 2004] Huan, Jun, Wang, Wei, Bandyopadhyay, Deepak, Snoeyink, Jack, Prins, Jan, and Tropsha, Alexander (2004). Mining protein family-specific residue packing patterns from protein structure graphs. In *Eighth International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 308–315.

[Inokuchi et al., 2000] Inokuchi, Akihiro, Washio, Takashi, and Motoda, Hiroshi (2000). An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Knowledge Discovery and Data Mining (PKDD2000)*, pages 13–23.

[Jin and Agrawal., 2005] Jin, R. and Agrawal., G. (2005). An algorithm for in-core frequent itemset mining on streaming data. In *ICDM*.

[Jin and Agrawal, 2004] Jin, Ruoming and Agrawal, Gagan (2004). An algorithm for in-core frequent itemset mining on streaming data. Technical Report OSU-CISRC-2/04-TR14, Ohio State University.

[Jin and Agrawal, 2005] Jin, Ruoming and Agrawal, Gagan (2005). A systematic approach for optimizing complex mining tasks on multiple datasets. In *Proceedings of ICDE*.

[Karp et al., 2002] Karp, Richard M., Papadimitrious, Christos H., and Shanker, Scott (2002). A Simple Algorithm for Finding Frequent Elements in Streams and Bags. Available from http://www.cs.berkeley.edu/ christos/iceberg.ps.

[Kuramochi and Karypis, 2001] Kuramochi, Michihiro and Karypis, George (2001). Frequent subgraph discovery. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 313–320.

[Manjhi et al., 2005] Manjhi, Amit, Shkapenyuk, Vladislav, Dhamdhere, Kedar, and Olston, Christopher (2005). Finding (recently) frequent items in distributed data streams. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 767–778.

[Manku and Motwani, 2002] Manku, G. S. and Motwani, R. (2002). Approximate Frequency Counts Over Data Streams. In *Proceedings of Conference on Very Large DataBases (VLDB)*, pages 346 – 357.

[Savasere et al., 1995] Savasere, A., Omiecinski, E., and S.Navathe. (1995). An efficient algorithm for mining association rules in large databases. In *21th VLDB Conf.*

[Teng et al., 2003] Teng, Wei-Guang, Chen, Ming-Syan, and Yu, Philip S. (2003). A regression-based temporal pattern mining scheme for data streams. In *VLDB*, pages 93–104.

[Toivonen, 1996] Toivonen, H. (1996). Sampling large databases for association rules. In *22nd VLDB Conf.*

[Xin et al., 2005] Xin, Dong, Han, Jiawei, Yan, Xifeng, and Cheng, Hong (2005). Mining compressed frequent-pattern sets. In *VLDB*, pages 709–720.

[Yan and Han, 2002] Yan, Xifeng and Han, Jiawei (2002). gspan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 721.

[Yu et al., 2004] Yu, Jeffrey Xu, Chong, Zhihong, Lu, Hongjun, and Zhou, Aoying (2004). False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, Toronto, Canada.

[Zaki et al., 1997] Zaki, M.J., Parthasarathy, S., Ogihara, M., and W.Li (1997). Parallel algorithms for fast discovery of association rules. *Data Mining and Knowledge Discovery: An International Journal*, 1(4):343–373.

[Zaki, 2002] Zaki, Mohammed J. (2002). Efficiently mining frequent trees in a forest. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.

[Zaki and Aggarwal, 2003] Zaki, Mohammed J. and Aggarwal, Charu C. (2003). Xrules: an effective structural classifier for xml data. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–325.