Rare Pattern Mining from Data Streams Using SRP-Tree and Its Variants

David Tse Jung Huang(✉), Yun Sing Koh, and Gillian Dobbie

Department of Computer Science, University of Auckland,
Auckland, New Zealand
{dtjh,ykoh,gill}@cs.auckland.ac.nz

Abstract. There has been some research in the area of rare pattern mining where the researchers try to capture patterns involving events that are unusual in a dataset. These patterns are considered more useful than frequent patterns in some domains, including detection of computer attacks, or fraudulent credit transactions. Until now, most of the research in this area concentrates only on finding rare rules in a static dataset. There is a proliferation of applications which generate data streams, such as network logs and banking transactions, and applying techniques that mine static datasets is not practical for data streams. We propose a novel approach called Streaming Rare Pattern Tree (SRP-Tree) and its variations, which finds rare rules in a data stream environment using a sliding window, and show that it both finds the complete set of itemsets and runs with fast execution time.

Keywords: Rare pattern mining · FP-Growth · Data stream · Sliding window

1 Introduction

Traditionally pattern mining techniques focus on finding frequent patterns within a dataset. The early works in pattern mining revolve around the Apriori-like candidate generation-and-test approach. The Apriori algorithm [2] was widely used and studied with several variations proposed [1,15,22]. The introduction of FP-Tree by Han et al. [12] directed focus in frequent pattern mining from Apriori approaches to tree-structured approaches. Along with the introduction of FP-Tree was the simultaneous proposal of FP-Growth in [12]. Since then FP-Growth have been one of the most widely used tree mining algorithms in the recent decade.

Even though frequent patterns are widely considered to be both informative and useful, in some scenarios rare patterns may be more interesting as they represent infrequent events. Rare patterns are patterns that do not occur frequently within the dataset and can be considered as exceptions. The discovery of rare patterns also has possible application in several different areas of research such as Wireless Sensor Networks [8] and Auction Fraud Detection [30]. An example of a useful rare pattern in practice could be the association of certain occurrences of symptoms to diseases. For instance, Meningitis is the inflammation of the

A. Hameurlain et al. (Eds.): TLDKS XXI, LNCS 9260, pp. 140–160, 2015.
DOI: 10.1007/978-3-662-47804-2_7

protective membranes covering the brain and spinal cord. Symptoms of Meningitis include headache, fever, vomiting, neck stiffness, and altered consciousness. Most of the symptoms commonly occur with influenza except for neck stiffness. By discovering the rare occurrence of all symptoms including neck stiffness, we are capable of flagging a patient possibly suffering from Meningitis out of a pool of patients suffering from common influenza. In recent years, the problem of extracting rare patterns from static datasets has been addressed. These proposed algorithms often follow either the Apriori algorithm or the FP-Tree algorithm. However, as the capability of generating data streams increases, the ability to capture useful information from streaming data becomes more important.

Ever since its inception, data stream mining has remained one of the more challenging problems within the data mining discipline. This is mainly due to the nature of data streams being continuous and unbounded in size as opposed to traditional databases where data is static and stable. Therefore, techniques developed have focused on improving the execution time and storage efficiency [7]. Recently the term "Big Data" has been widely used and has close relations to high performance data mining. Some of the important aspects of Big Data is introduced in [9] and the problem of devising models and algorithms for such high performance data mining tasks is further explained in [6].

Data from a wide variety of application areas ranging from online retail applications such as online auctions and online bookstores, telecommunications call data, credit card transactions, sensor data, wireless sensor networks and climate data are a few examples of applications that generate vast quantities of data on a continuous basis. Data produced by such applications are highly volatile with new patterns and trends emerging on a continuous basis. The unbounded size of data streams is considered the main obstacle when processing data streams. As it is unbounded, it makes it infeasible to store the entire data on disk. Furthermore, the processing of data streams should ideally be near real time. This raises two issues. Firstly, a multi-pass algorithm cannot be used because the entire dataset would need to be stored before mining can commence. This restriction limits the use of the majority of current database-based techniques as most of them require multiple scans of the entire dataset which needs the entire dataset to be stored. Secondly, obtaining the exact set of rules that includes both frequent and rare rules from the data streams is too expensive. In theory, most frequent pattern mining techniques that uses a minimum support threshold can be used to find rare patterns as these thresholds can be lowered to find both frequent and rare rules then later the frequent rules can be pruned out leaving only rare rules. However, this approach is extremely inefficient as a large number of unnecessary rules are generated and then discarded. In light of this situation, the need for an efficient algorithm that finds only rare rules in data streams without having to generate frequent rules is evident. In this paper we propose a technique that achieves this aim.

Contributions. Our contribution in this paper is the proposal of a novel technique called Streaming Rare Pattern Tree (SRP-Tree), which captures the complete set of rare rules in data streams using only a single pass scanning of the dataset and adapts a sliding window approach. We also propose two variations of the technique where the first one involves using a novel data structure called the Connection Table which allows us to efficiently constrain the search in our tree, and keep track of items within the window. The second variation improves upon the first and requires the tree to go through restructuring before mining with FP-Growth.

Paper Structure. The paper is organized as follows. In Sect. 2 we look at related work in the area of rare association rule mining. In Sect. 3 we present preliminary concepts and definitions for rare pattern mining in data streams. In Sect. 4 we describe our SRP-Tree approach, and in Sect. 5 we describe and discuss our experimental results. Finally, Sect. 6 concludes the paper.

2 Related Work

In this section we will discuss the related work in the area of pattern mining, specifically, frequent pattern mining and rare pattern mining. We look at the previous approaches that performs these tasks in both traditional static databases and in stream environments.

2.1 Frequent Pattern Mining

In the area of frequent pattern mining, the first algorithm was Apriori [2] and was proposed in the year 1994. The Apriori algorithm does multiple database scans and generates a large number of possible rules which are later found to be infrequent. This repeated candidate generation-and-test approach was extensively studied and used until in the year 2000 when the FP-Tree algorithm was proposed. The FP-Tree [12] is one of the most widely known techniques, and unlike Apriori it uses a tree structure to find frequent patterns in a database. FP-Tree solves the efficiency problem of Apriori-like approaches when finding patterns and rules and proposes the FP-Growth algorithm as the mining algorithm for the tree. FP-Tree proves to be efficient at finding the complete set of frequent patterns from a database and the FP-Growth algorithm was later adopted by many and considered as an efficient pattern mining algorithm. FP-Tree, being a landmark technique, is designed as a multi-pass approach which is not ideal in the data stream environment where data can only be read once. In addition, it is also memory intensive and unfitting for stream environments as streams are considered to be unbounded in data and working algorithms are required to have an efficient memory use.

Recently, more research that considers mining frequent patterns from data streams have been proposed. Giannella et al. [11] proposed a technique called FP-Stream which adopts the structure of FP-Tree and finds the approximate set of frequent patterns from data streams. FP-Stream mines time-sensitive frequent patterns and incrementally maintains only the historical information of frequent patterns. Overall, FP-Stream proposed efficient algorithms for constructing, maintaining, and updating pattern structures over data streams. However, FP-Stream only finds the approximate set of frequent patterns and discovering an approximate set of patterns is a disadvantage of the technique as in most cases, the user would want to find the complete set of patterns.

The technique DSTree was proposed by Leung et al. [20] and it is also a tree-structured algorithm. Unlike FP-Stream, DSTree finds the complete set of frequent patterns from a data stream. Unlike FP-Tree which builds its tree based on frequency-descending order, DSTree incrementally builds the tree based on a canonical ordering. DSTree is unaffected by changes in item frequency and therefore has the attractive property that it is easily updated and maintained as data is removed and added into the tree. DSTree has a wider applicability and can be used to find maximal, closed, and constrained itemsets.

Tanbeer et al. [27] proposed the CPS-Tree which is an efficient tree-structured algorithm that also finds the complete set of frequent patterns from a data stream. Unlike DSTree which uses a canonical ordering, CPS-Tree builds a compact tree in frequency-descending order by performing multiple sorting operations on the tree before mining of the tree. Resorting of the tree was proven to be more cost effective in the experiments when compared to other tree-based frequent mining techniques.

Most pattern mining algorithms like the DSTree and CPS-Tree adopt a sliding window model. However, for some applications, other models such the time-fading model and the landmark model may be more appropriate. In [19] the authors proposed a tree-based mining algorithm that mines frequent patterns from data streams using the time-fading model and the landmark model. The two different models were applied to streams of uncertain data were it is more sensible than using the sliding window model and the proposed algorithms were shown to be efficient through evaluation.

There has been a lot of other works in the area of data stream mining and most of them look at finding frequent patterns from data streams [3–5,13,16,18,21,24].

2.2 Rare Pattern Mining

There has also been a lot of work in the area of rare pattern mining, including many recent works such as [17,23,25]. However all current research in this area is designed for static datasets and is not able to handle a data stream environment. Currently there are two different types of rare pattern mining approaches: level-wise and tree based, like that of frequent pattern mining. Current itemset mining approaches, which are based on level-wise exploration of the search space are similar to the Apriori algorithm [2]. In Apriori, k-itemsets (itemsets of cardinality k) are used to generate $k+1$-itemsets. These new $k+1$-itemsets are pruned using the downward closure property, which states that the superset of a non-frequent itemset cannot be frequent. Apriori terminates when there are no new $k+1$-itemsets remaining after pruning. MS-Apriori [22], Rarity [28], ARIMA [26], AfRIM [1] and Apriori-Inverse [15] are five algorithms that detect rare itemsets. They all use level-wise exploration similar to Apriori, which have candidate generation and pruning steps.

MS-Apriori [22] uses a bottom-up approach similar to Apriori. In MS-Apriori, each item can be assigned a different minimum item support value (MIS). Rare items can be assigned a low MIS, so that during candidate pruning, itemsets that include rare items are more likely to be retained and participate in rule generation. Apriori-Inverse [15] is used to mine perfectly rare itemsets, which are itemsets that only consist of items below a maximum support threshold (maxSup).

Szathmary et al. [26] proposed two algorithms that can be used together to mine rare itemsets: MRG-Exp and ARIMA. They defined three types of itemsets: minimal generators (MG), which are itemsets with a lower support than its subsets; minimal rare generators (MRG), which are itemsets with non-zero support and whose subsets are all frequent; and minimal zero generators (MZG), which are itemsets with zero support and whose subsets all have non-zero support. The first algorithm, MRG-Exp, finds all MRG by using MGs for candidate generation in each layer in a bottom up fashion. The MRGs represent a border that separates the frequent and rare itemsets in the search space. All itemsets above this border must be rare according to the antimonotonic property. The second algorithm, ARIMA, uses these MRGs to generate the complete set of rare itemsets. ARIMA stops the search for non-zero rare itemsets when the MZG border is reached, which represents the border above which there are only zero rare itemsets.

Adda et al. [1] proposed AfRIM, which begins with the itemset that contains all items found in the database. Candidate generation occurs by finding common k-itemset subsets between all combinations of rare $k+1$-itemset pairs in the previous level. Troiano et al. proposed the Rarity algorithm that begins by identifying the longest transaction within the database and uses it to perform a top-down search for rare itemsets, thereby avoiding the lower layers that contain only frequent itemsets.

All of the above algorithms use the fundamental generate-and-test approach used in Apriori, which has potentially expensive candidate generation and pruning steps. In addition, these algorithms attempt to identify all possible rare itemsets, and as a result require a significant amount of execution time. RP-Tree algorithm was proposed by Tsang et al. [29] as a solution to these issues. RP-Tree avoids the expensive itemset generation and pruning steps by using a tree data structure, based on FP-Tree [12], to find rare patterns. However it uses a multi-pass approach, which is not suitable in a data stream environment.

More recently Lavergn et al. [17] proposed a rare pattern mining technique using itemset trees and relative support called TRARM-RelSup. As opposed to traditional techniques that used the minimum support threshold, TRARM-RelSup used the relative support measure with the goal of capturing some additional rare rules that are missed out when using the minimum support threshold. The algorithm combines the efficiency of targeted association mining querying with the capabilities of rare rule mining resulting in the discovery of more rare rules for the user.

Even though there has been many algorithms that perform rare pattern mining, up until now there has been no research into rare pattern mining in data streams.

3 Preliminaries

In this section, we provide definitions of key terms that explain the concepts of frequent pattern mining in a data stream. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called items, that represent a unit of information in an application domain. A set $X = \{i_l, \ldots, i_m\} \subseteq I$ and $l, m \in [1, n]$, is called a itemset, or a k-itemset if it contains k items. A transaction $t = (tid, Y)$ is a tuple where tid is a transaction-id and Y is a pattern. If $X \subseteq Y$, it is said that t contains X or X occurs in t. Let size(t) be the size of t, i.e., the number of items in Y.

Definition 1. A data stream $\mathcal{DS}$ is an infinite sequence of transactions $\mathcal{DS} = [t_1, t_2, \ldots, t_m)$, where $t_i, i \in [1, m]$ is the i th transaction in the data stream.

Definition 2. A window $\mathcal{W}$ is a set of all transactions between the ith and jth (where $j > i$) transactions and the size of $\mathcal{W}$ is $|\mathcal{W}| = j - i$.

Definition 3. Window $\mathcal{W}$ consists of blocks where $\mathcal{W} = \{B_1, B_2, \cdots, B_n\}$. A block is also a set of transactions like $\mathcal{W}$.

Definition 4. The count of an itemset X in $\mathcal{W}$, $count_{\mathcal{W}}(X)$, is the number of transactions in $\mathcal{W}$ that contain X.

Definition 5. The support of an itemset X in $\mathcal{W}$ is the count of an itemset divided by the size of $\mathcal{W}$

$$sup_{\mathcal{W}}(X) = \frac{count_{\mathcal{W}}(X)}{|\mathcal{W}|}$$

An association rule is an implication $X \rightarrow Y$ such that $X \cup Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$. X is the antecedent and Y is the consequent of the rule. The *support* of $X \rightarrow Y$ in $\mathcal{W}$ is the proportion of transactions in $\mathcal{W}$ that contains $X \cup Y$. The *confidence* of $X \rightarrow Y$ is the proportion of transactions in $\mathcal{W}$ containing X that also contains Y.

3.1 Rare Itemsets

We adopted the rare itemsets concept from Tsang et al. [29]. We consider an itemset to be rare when its support falls below a threshold, called the minimum frequent support (minFreqSup) threshold. One difficulty when generating rare itemsets is differentiating noisy itemsets from the actual rare itemsets. As the support of the itemset is low, the potential of pushing unrelated items together increases as well, thus producing noisy itemsets. To overcome this problem, we define a noise filter threshold to prune out the noise called the minimum rare support (minRareSup) threshold. Typically minRareSup is set to a very low level, e.g., 0.01 %.

Definition 6. An itemset X is a *rare itemset* in a window $\mathcal{W}$ iff $sup_{\mathcal{W}}(X) \leq$ minFreqSup and $sup_{\mathcal{W}}(X) >$ minRareSup.

However not all rare itemsets that fulfill these properties are interesting. Furthermore, rare itemsets can be divided into two types: *rare-item itemsets* and *non-rare item itemsets.*

Rare item itemsets refer to itemsets which are a combination of only rare items and itemsets that consist of both rare and frequent items. Given 4 items $\{a, b, c, x\}$ with supports $a = 0.70$, $b = 0.45$, $c = 0.50$, and $x = 0.10$, with minFreqSup = 0.15 and minRareSup = 0.01, the itemset $\{a, x\}$ would be a rare item itemset assuming that the support of $\{a, x\} > 0.01$, since the itemset includes the rare item x.

Definition 7. An itemset X is a *rare-item itemset* iff X is a *rare itemset* **and**

$$\exists x \in X, sup_{\mathcal{W}}(x) \leq \text{minFreqSup}$$

Non-rare item itemsets only has frequent items which fall below the minimum frequent support threshold. Given 4 items $\{a, b, c, x\}$ with supports $a = 0.70$, $b = 0.45$, $c = 0.50$, and $x = 0.10$, with minFreqSup = 0.15 and minRareSup = 0.01, and the itemset $\{a, b, c\}$ with a support of 0.09, then this itemset would be a non-rare item itemset as all items within the itemset are frequent, and its support is between minFreqSup and minRareSup.

Definition 8. An itemset X is a *non-rare item itemset* iff X is a *rare itemset* **and**

$$\forall x \in X, sup_{\mathcal{W}}(x) > \text{minFreqSup}$$

4 SRP-Tree: Rare Pattern Tree Mining for Data Streams

Current tree based rare pattern mining approaches follow the traditional FP-Tree [12] approach. It is a two-pass approach and is affordable when mining a static dataset. However in a data stream environment, a two-pass approach is not suitable. To process a static environment, non-streaming rare pattern techniques such as RP-Tree order each item within a transaction according to its frequency before inserting it into the tree. Using these algorithms the frequency of the items are obtained during the first pass through the dataset.

In data streams we can only look at the transactions within the stream once, thus, a one-pass approach is necessary. This rules out the possibility of building a tree based on the frequency of items within the data stream. Furthermore, frequency of an item may change as the stream progresses. There are four scenarios which we need to consider:

Scenario 1. A frequent item x at a particular time T_1 may become rare at time T_2.
Scenario 2. A rare item x at a particular time T_1 may become frequent at time T_2.
Scenario 3. A frequent item x at a particular time T_1 may remain frequent at time T_2.
Scenario 4. A rare item x at a particular time T_1 may remain rare at time T_2.

T_1 represents a point in time, and T_2 represents a future point in time after T_1.

To find rare patterns within data streams, we propose a new algorithm called SRP-Tree that mines rare patterns in a data stream using a sliding window and tree based approach. We discuss the details of the algorithm and introduce the two variations of the algorithm: SRP-Tree with Connection Table and SRP-Tree with Restructuring.

4.1 Approach 1: SRP-Tree with Connection Table

In this approach, items from incoming transactions in the stream are inserted into a tree based on a canonical ordering. A canonical ordering allows us to capture the content of the transactions from the data stream and organise tree nodes according to a particular order. We use the lexicographic ordering of the items as the canonical ordering here. When we use a canonical order to build the tree, the ordering of items is unaffected by the changes in frequency caused by incremental updates. There has been other work carried out using a canonical ordering to build trees for a data stream mining environment [10,20], but these research has been tailored to find frequent patterns.

In our tree built using canonical ordering, the frequency of a node in the tree is at least as high as the sum of frequencies of its children. However, this does not guarantee the downward closure property which exists in a tree ordered in frequency-descending order. The downward closure property in a traditional rare pattern tree mining algorithm, whereby, *rare-items will never be the ancestor of a non-rare item in the initial tree due to the tree construction process* is violated. Hence, we propose a novel item list called the Connection Table which keeps track of each unique item in the window and the items they co-occur with along with their respective frequencies.

The Connection Table used in this approach captures in the transactions only items that have a lower canonical ordering. For example, given a transaction in a canonical order of $\{a, b\}$ we store in the table that a is connected to b with a frequency of 1 but it does not store b is connected to a. This is because of the properties of the canonical ordering in the constructed tree: item a will always be the ancestor of item b. Since mining is carried out using a bottom-up approach, by mining item b, item a is also mined. The opposite does not hold since mining item a does not guarantee that item b is mined. Therefore, by using the Connection Table to keep track of connected items and adding them as arguments to FP-Growth in the mining phase, the complete set of rare-item itemsets can then be captured. The Connection table is designed using a hash map which allows for $O(1)$ access. In worst case scenarios, the table could reach size of $\frac{x(x+1)}{2}$ where x is the total number of items; however, in reality this is highly unlikely.

At any point of time should we decide to mine the current window, the initial tree of the current window is used to construct conditional pattern bases and conditional trees for each rare-item and their connected items in the Connection Table. We trigger the mining step when a block is filled. In this example the block size will be equal to the window size of 12. Note that only connection items with an occurring frequency greater than or equal to the minRareSup are included. Each conditional tree and corresponding item are then used as arguments for FP-Growth. The threshold used to prune items from the conditional trees is minRareSup. The union of the results from each of these calls to FP-Growth is a set of itemsets that contains a rare-item, or rare item itemsets.

Table 1. Set of transactions in a given window $\mathcal{W}$ of size 12

Tid	Transaction	Tid	Transaction	Tid	Transaction
1	{a, g, h}	5	{c, f, h}	9	{c, d, h}
2	{a, g, h, i}	6	{a, e, g, h}	10	{b, f}
3	{b, c, d, f}	7	{g}	11	{a, h}
4	{b, d, j}	8	{h}	12	{a}

Table 2. Connection table using the window of transactions listed in Table 1

Item	Items Co-occurred	Item	Items Co-occurred
a	{(e:1), (g:3), (h:4), (i:1)}	f	{(h:1)}
b	{(c:1), (d:2), (f:2), (j:1)}	g	{(h:1), (i:1)}
c	{(d:2), (f:2), (h:2)}	h	{(i:1)}
d	{(f:1), (h:1), (j:1)}	i	{∅}
e	{(g:1), (h:1)}	j	{∅}

Example. Given the dataset in Table 1, we show how the Connection Table is built in Table 2. The left column in Table 2 list the unique items in the window, whereas the right column lists the set of co-occurring items along with the co-occurrence frequency of that particular item to the item in the right column. For example, item c co-occurs twice with items d, f, and h.

SRP-Tree with Connection Table Algorithm. Our SRP-Tree with connection table algorithm is shown in Algorithm 1. This approach essentially performs in one pass the counting of item frequencies and the building of the initial tree. Therefore, in a given window $\mathcal{W}$, for each incoming transaction t, SRP-Tree first updates the list of item frequencies according to the transactions contained in the current window. We refer to this as updateItemFreqList(t) method, where we increment the counts of items contained in the new transaction and decrement the counts of items contained in the oldest transcations to be discarded from the window. SRP-Tree then updates the tree structure according to the transactions contained in the current window in a similar fashion, which is referred to as updateTree(t) method in Algorithm 1. We mine the tree after a particular number of transactions also known as a block. We refer to a preset block size as

Algorithm 1. SRP-Tree (Connection Table)

```
1.  Input: $DS$, $\mathcal{W}$, $\mathcal{B}$, $minRareSup, minFreqSup$;
2.  Output: $results$ (Set of rare item itemsets);

3.  while exist($DS$) do
4.     $t \leftarrow$new incoming transaction from $DS$;
5.     currentBlockSize $\leftarrow$ currentBlockSize + 1;
6.     updateItemFreqList($t$);
7.     updateConnectionTable($t$);
8.     $tree \leftarrow$ updateTree($t$);
9.     Mining at the end of block
10.    if $\mathcal{B}$ == currentBlockSize then
11.       currentBlockSize $\leftarrow$ 0;
12.       $results = \emptyset$;
13.       $\mathcal{I} \leftarrow$ {all unique items in $\mathcal{W}$};
14.       $\mathcal{R} \leftarrow \{i \in \mathcal{I} \mid sup_{\mathcal{W}}(i) \geq minRareSup \wedge sup_{\mathcal{W}}(i) < minFreqSup\}$;
15.       $\mathcal{C} \leftarrow \{k \in \mathcal{R}, j \in connectionTable(k) \mid sup_{\mathcal{W}}(k) \geq minRareSup\}$;
16.       for item $a$ in $tree$ do
17.          if $a \in \mathcal{R}$ or $a \in \mathcal{C}$ then
18.             construct $a$'s conditional pattern-base and then $a$'s conditional FP-Tree
                $Tree_a$;
19.             $results \leftarrow results \cup$ FP-Growth($Tree_a, a$);
20.          end if
21.       end for
22.    end if
23. end while
24. return $results$;
```

$\mathcal{B}$. In this algorithm, $\mathcal{R}$ refers to the set of rare items and $\mathcal{C}$ refers to the set of items that co-occur with a particular rare item.

We would also like to point out, that another difference between SRP-Tree and a static rare pattern mining approach is that in SRP-Tree, the tree is built using all the transactions in the window, whereas in a static rare pattern mining approach, only transactions with rare items are used to build the tree. A static approach has the luxury of looking at the dataset twice and discarding items which it is not interested in before the tree is even built. In our case, we simply cannot know which transactions contain rare items until we decide to mine.

SRP-Tree with Connection Table Example. Applying SRP-Tree to the window $\mathcal{W}$ of the 12 transactions in Table 1, the support ordered list of all items is $\langle(h{:}6), (a{:}5), (g{:}4), (b{:}3), (c{:}3), (d{:}3), (f{:}2), (e{:}1), (i{:}1), (j{:}1)\rangle$. Using minFreqSup = 4 and minRareSup = 2, only the items $\{b, c, d, f\}$ are rare, and included in the set of rare items, $\mathcal{R}$.

The initial SRP-Tree constructed for window $\mathcal{W}$ of size 12 is shown in Fig. 1. To find the rare-item itemsets, the initial SRP-Tree is used to build conditional pattern bases and conditional SRP-Trees for each rare item $\{b, c, d, f\}$ <u>and</u> any additional items in the Connection Table that is connected with a rare item that

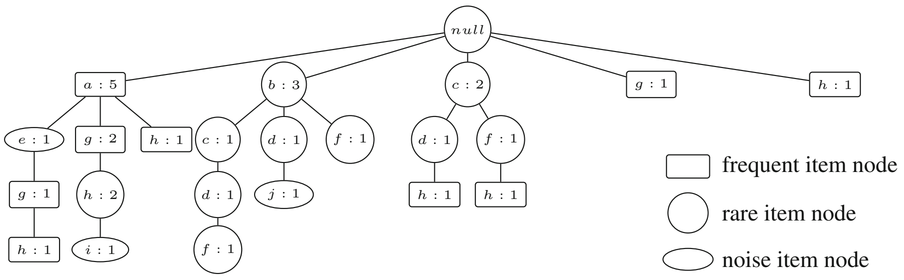

Fig. 1. Pattern tree constructed from window $\mathcal{W}$ using SRP-Tree

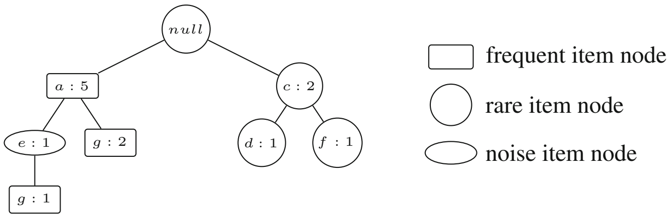

Fig. 2. Conditional tree, $Tree_h$

has a frequency greater than the minRareSup, in this example, item h. The conditional tree for item h is shown in Fig. 2. Each of the conditional SRP-Trees and the conditional item are then used as parameters for the FP-Growth algorithm.

4.2 Approach 2: SRP-Tree with Restructuring

In the previous approach we build the tree based on a canonical ordering and proposed the Connection Table that enables the capture of the complete set of rare rules from the canonically ordered tree. In this approach we improve upon the first and use an additional tree restructuring step. The additional restructuring step replaces the Connection Table and still allows the algorithm to find the complete set of rare rules.

The Connection Table proposed in the previous approach was used because when building the tree with a canonical ordering, the downward closure property is violated (see Sect. 4.1). In this second approach, instead of using the Connection Table, we perform a tree restructuring at the end of each block before mining the tree with FP-Growth. The additional tree restructuring effectively re-orders the nodes in the tree at each block into a strict frequency-descending order for that block. After restructuring, the tree will maintain the downward closure property and performing FP-Growth on the rare items themselves will yield the complete set of rare rules.

SRP-Tree with Restructuring Algorithm. The algorithm for the restructuring approach of our SRP-Tree is shown in Algorithm 2. The building and

Algorithm 2. SRP-Tree (Restructuring)

```
1. Input: $DS$, $\mathcal{W}$, $\mathcal{B}$, $minRareSup, minFreqSup$;
2. Output: $results$ (Set of rare item itemsets);

3. while exist($DS$) do
4.    $t$ ←new incoming transaction from $DS$;
5.    currentBlockSize ← currentBlockSize + 1;
6.    updateItemFreqList($t$);
7.    $tree$ ← updateTree($t$);
8.    Mining at the end of block
9.    if $\mathcal{B}$ == currentBlockSize then
10.      currentBlockSize ← 0;
11.      $tree$ ← restructureTree();
12.      $results = \emptyset$;
13.      $\mathcal{I} \leftarrow$ {all unique items in $\mathcal{W}$};
14.      $\mathcal{R} \leftarrow \{i \in \mathcal{I} \mid sup_{\mathcal{W}}(i) \geq minRareSup \wedge sup_{\mathcal{W}}(i) < minFreqSup\}$;
15.      for item $a$ in $tree$ do
16.         if $a \in \mathcal{R}$ then
17.            construct $a$'s conditional pattern-base and then $a$'s conditional FP-Tree
               $Tree_a$;
18.            $results \leftarrow results \cup$ FP-Growth($Tree_a, a$);
19.         end if
20.      end for
21.   end if
22. end while
23. return $results$;
```

updating of the item frequency list and the tree is similar to the previous approach and explained in the earlier sections. The main difference is the absence of maintaining the Connection Table and also an additional restructuring method at line 11.

Restructuring Technique. A tree is usually restructured by rearranging the nodes of an existing tree built based on a previous ordering to another different desired ordering. In our case we are restructuring the tree built based on a canonical ordering to a frequency-descending ordering. This operation involves sorting the item frequencies list and shifting the position of the nodes in the tree. We adopt the restructuring technique used in [16]. The branch sorting method (BSM) is an efficient tree restructuring technique that fits the purpose of our algorithm. In summary, BSM performs tree restructures by going through several steps. First the item-frequency-list of the items in the current block is rearranged into frequency-descending order. Then, the technique iterates through each unsorted path in the tree and sorts them based on the reordered item-frequency-list. BSM is an array-based technique whereby for each unsorted branch (path) in the tree, it first removes the path, sorts the path based on the frequency-descending ordering, then adds the path back into the tree.

Restructuring is complete when all the branches are processed and sorted which produces the final sorted tree.

SRP-Tree with Restructuring Example. Consider the same example as given by the transactions in Table 1. The support ordered list of all items is $\langle(h{:}6), (a{:}5), (g{:}4), (b{:}3), (c{:}3), (d{:}3), (f{:}2), (e{:}1), (i{:}1), (j{:}1)\rangle$. Using minFreqSup $= 4$ and minRareSup $= 2$, only the items $\{b, c, d, f\}$ are rare, and included in the set of rare items, $\mathcal{R}$.

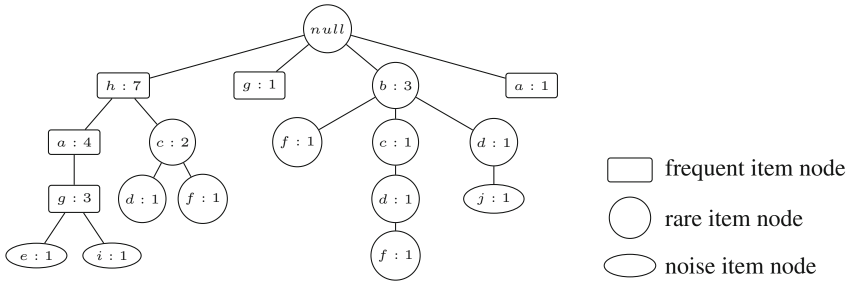

Fig. 3. Pattern tree from window $\mathcal{W}$ after restructuring into frequency descending order

In this approach, instead of mining the tree (shown in Fig. 1) built with canonical ordering $a, b, c, d, e, f, g, h, i, j$, we mine the restructured tree based on the support ordered list of items in the window W with the ordering $h, a, g, b, c, d, f, e, i, j$. The tree after restructuring is shown in Fig. 3.

Recall that the downward closure property in a traditional rare pattern tree mining algorithm states that *rare-items will never be the ancestor of a non-rare item in the initial tree due to the tree construction process.* From this restructured tree built based on frequency-descending support ordering, we can observe that all the rare-items (circular nodes) are indeed never the ancestor of a non-rare item (rectangular nodes). Therefore, we can prove that the downward closure property is satisfied in this case.

We also observe that because of the change in the ordering of the tree, it has become more compact (containing fewer nodes). In general, the compactness of a tree will have an influence on the mining time of FP-Growth. Therefore, the more compact the tree is, the faster the mining time of FP-Growth.

5 Experimental Results

In this section we present the evaluation and results performed on the two variations of our algorithm SRP-Tree. In Sect. 5.1 we describe in detail the algorithm we used to compare our SRP-Tree against. In Sect. 5.2 we evaluate the algorithms on real-world datasets and compare their performance at finding rare-item itemsets. Lastly in Sect. 5.3 we present a case study on the T40I10D100K dataset and perform additional evaluations by varying the minimum frequent support threshold and the block size of the techniques.

All algorithms were implemented in Java and executed on a machine equipped with an Intel Core i5-2400 CPU @ 3.10 GHz with 4 GB of RAM running Windows 7 x 64.

5.1 Streaming Canonical Tree (SC-Tree)

SRP-Tree is the very first attempt at mining rare patterns in a data stream environment so there are no other similar techniques that mine rare patterns in a data stream to compare to for evaluation. Therefore, in these experiments we compared the performance of our SRP-Tree to the Streaming Canonical Tree (SC-Tree), which is a modified version of DSTree [20] with pruning of frequent itemsets. This technique that we use for comparison, SC-Tree with pruning, finds rare item itemsets in an unoptimized brute-force manner. The SC-Tree is a one pass technique which stores the transactions from the stream in a tree using canonical ordering and stores/updates item frequencies similar to DSTree. To find all rare item itemsets, SC-Tree finds and generates all itemsets that meet the minRareSup threshold, then removes all other itemsets except rare item itemsets with an extra pruning step. SC-Tree produces the same itemsets and generates the same rules as SRP-Tree (Table) and SRP-Tree (Restructure).

5.2 Real-World Dataset

In this section we present the time and relative time taken for itemset generation of SRP-Tree (both approaches) and SC-Tree on real-world datasets. The time is reported in seconds and the relative time is calculated by setting SC-Tree to 1.00 and SRP-Tree relative to that of SC-Tree.

$$\text{relative time} = \frac{\text{Time taken by SRP-Tree}}{\text{Time taken by SC-Tree}}$$

We used a window size and block size of 25 K for all datasets except for the Mushroom dataset where we used 2 K due to its smaller size. The minFreqSup and minRareSup thresholds are shown in Table 3 for each dataset. The thresholds are user-defined through examining the distribution of item frequencies in each of the datasets. We acknowledge that given a data stream environment, this is not the most suitable way of defining thresholds and in the future we will be looking at a way to adapt the thresholds to the change in distribution and drift of the transactions in the stream.

We have tested the algorithms on 6 datasets obtained from the FIMI (Frequent Itemset Mining Implementations) repository[1]. The datasets are: Mushroom, Retail, BMS-POS, T10I4D100K, T40I10D100K, and Kosarak (250K).

Here are brief descriptions of each dataset:

[1] http://fimi.ua.ac.be/data/.

- The Mushroom dataset is a dense dataset with a relatively larger number of possible rules and patterns. It contains a total of 8124 instances and 22 attributes that includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family.
- The Retail dataset, compared to Mushroom, is a much more sparse dataset that contains anonymized retail market basket data from an Belgian retail store over approximately 5 months. The total number of instances is 88163.
- The BMS-POS dataset contains several years worth of point-of-sale data from a large electronics retailer. Each transaction represents customer purchases of items at one time. It contains a total of 515597 instances and the average transaction size is 6.5.
- The T10I4D100K and T40I10D100K originates from the same source, the IBM Almaden Quest market basket data generator. They both contain a total of 100000 instances.
- The Kosarak dataset contains anonymized click-stream of a hungarian online news portal. Kosarak contains a larger number of different transactions with different characteristics (i.e. the dataset contains transactions that vary largely in average transaction size and items).

Table 3. Comparison between SRP-Tree and SC-Tree

Dataset	$\mathcal{B}$	MinRareSup	MinFreqSup	# Itemsets	SRPT (Table)		SRPT (Restructure)		SC-Tree	
					Time (s)	Rel. time	Time (s)	Rel. time	Time (s)	Rel. time
Mushroom	2K	0.01	0.05	14443674	1131	0.86	86	0.07	1321	1.00
Retail	25K	0.0001	0.0005	572673	239	0.71	9	0.03	339	1.00
BMS-POS	25K	0.0002	0.0005	1426	58	0.02	25	0.01	3783	1.00
T10I4D100K	25K	0.0001	0.0005	1161	12	0.63	6	0.32	19	1.00
T40I10D100K	25K	0.003	0.05	4734806	301	0.79	94	0.25	380	1.00
Kosarak(250K)	25K	0.001	0.15	35623519	703	0.10	591	0.08	7213	1.00

Table 3 shows, for each dataset, the block size, minRareSup, and minFreqSup used to run the algorithms then the comparison between SRP-Tree (Table), SRP-Tree (Restructure) and SC-Tree of the itemsets generated, time it took to run the algorithm, and the relative time. The objective of this comparison is to look at the efficiency of each algorithm under the same conditions. SRP-Tree (Table), SRP-Tree (Restructure) and SC-Tree generate the same number of itemsets because both SRP-Tree approaches only generate rare-item itemsets and SC-Tree generates all itemsets that meet the minRareSup threshold then the extra pruning step removes all other itemsets that are not rare-item itemsets. Therefore, the final number of itemsets generated by both algorithms is the same.

In all datasets of varying item frequency distribution, both SRP-Tree approaches run faster than SC-Tree where SRP-Tree (Restructure) runs significantly faster than the other two techniques. The time taken to run the various datasets are highly dependent on the tree structure built from the transactions in the datasets and generally have a positive correlation with the number of itemsets generated. The number of itemsets generated also has a great variation and is highly dependent on the composition/nature of the respective dataset. For datasets that are more sparse in nature like Retail, BMS-POS, and T10I4D100K, the number of itemsets generated are usually smaller than a dense dataset like Mushroom and the run-time is usually faster.

5.3 Case Study: T40I10D100K

The T40I10D100K dataset is generated using the generator from the IBM Almaden Quest research group. Figures 4 and 5 shows the item frequency distribution of the T40I10D100K dataset. When we compared the distribution of the T40I10D100K dataset, to other datasets in the FIMI repository, we observed that the T40I10D100K dataset is more sparse in nature compared to the Mushroom dataset, but more dense than datasets like Retail and BMS-POS. It is also important to note that T40I10D100K contains a high proportion of items with a frequency of less than 0.1 (approximately 90 % of the total items). This particular distribution contains a greater number of prospective rare items and rules to be mined from the dataset.

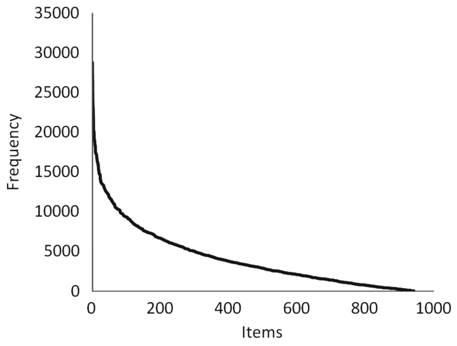

Fig. 4. Item frequency distribution

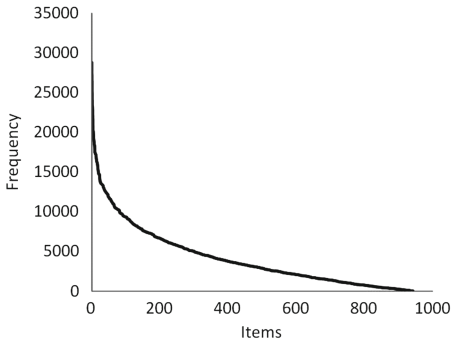

Fig. 5. Normalized item frequency distribution

In Table 4 we show the difference in itemsets generated and the time it took to generate items of varying minFreqSup on the T40I10D100K dataset. We aim to look at the fluctuations in run-time and the number of itemsets generated caused by varying the minimum frequent support threshold. As we increase the minFreqSup, the number of itemsets generated increases and the relative time also increases. It is important to note that the real time taken for SRP-Tree (Table) to generate itemsets decreases as we increase the minFreqSup. This is because at a lower minFreqSup for this particular distribution of the dataset, there is a high co-occurrence of rare items to other items with similar item frequency (indicating that these other items are also likely to have rare properties).

Table 4. Varying MinFreqSup for T40I10D100K

MinFreqSup	Itemset	SRPT (Table)		SRPT (Restructure)		SC-Tree	
		Time(s)	Relative Time	Time(s)	Relative Time	Time(s)	Relative Time
0.04	4397517	333	0.93	76	0.21	357	1.00
0.05	4734806	301	0.79	94	0.25	380	1.00
0.06	5028947	278	0.66	116	0.28	421	1.00
0.10	5105892	246	0.55	148	0.33	446	1.00
0.15	5136904	238	0.52	160	0.35	454	1.00

The mining of these additional highly connected items with rare association patterns incurred a larger overhead in maintaining the Connection Table. As the minFreqSup increases and most of the highly connected potential rare items are accounted for, the overhead decreases and this results in a faster runtime.

However, for SRP-Tree (Restructure) the time taken to generate itemsets increases as we increase the minFreqSup like that of SC-Tree. This is because this approach does not use the Connection Table structure to mine for itemsets and that the time taken is generally positively correlated with the number of itemsets. Table 5 shows the difference in execution time when the block size varies. Similar to the above experiment, we vary one variable in the algorithms while keeping the others constant. Here we aim to compare the behavior of the algorithms when the block size variable is changed. Overall we observed that the execution time is increased as the block size increases due to the increased size of the tree being built and SRP-Tree (Restructure) is still the fastest running technique of the three.

Table 5. Execution Time based on Varying Block Sizes for T40I10D100K

Block size	SRPT (Table)		SRPT (Restructure)		SC-Tree	
	Avg time / window (s)	Relative time	Avg. time / window (s)	Relative time	Avg. time / window (s)	Relative time
10K	41	0.59	19	0.27	70	1.00
25K	75	0.79	24	0.25	95	1.00
50K	142	0.71	19	0.09	201	1.00

6 Discussion

Rare pattern mining is often a more difficult and computationally expensive task than frequent pattern mining. Frequent patterns occur with a high frequency and frequent itemsets will only contain frequent items, therefore, frequent pattern mining techniques generally need to only consider a smaller subset of items out of the total set when mining. Strategies such as top-k items or minimum frequent support threshold are used to determine what items are frequent. The natural property of frequent items virtually constrains the search space of the mining techniques and effectively helps guarantee mining efficiency. In contrast, rare patterns occur with a low frequency and rare itemsets may contain both frequent and rare items. When mining rare patterns, the search space is considered to be much larger compared to mining frequent patterns, making rare pattern mining a more computationally expensive task. We remedy this by better constraining the initial search space of the patterns using SRP-Tree. We compared our two variations of SRP-Tree against SC-Tree, a modified version of DSTree that finds rare patterns using a brute-force strategy. Using SC-Tree, the search space included both frequent and rare patterns. It is only through an additional pruning step that SC-Tree removes frequent patterns from the final discovered set leaving the desired rare patterns. The effects of constraining the initial search space using SRP-Tree is evident from the experimental evaluation showing a significant improvement in execution time.

Traditionally pattern mining techniques are used in databases to find a specific set of useful patterns. In dynamically changing data streams, the set of patterns can vary and change over-time. Our SRP-Tree which works in a stream environment can be incorporated with drift detection methods [14] to provide an effective way of discovering useful rare patterns at various points of the data stream. Drift detection signals points where patterns have possibly changed, then instead of using a block size variable to mine patterns periodically, the drift points found by the detection method is used as points when SRP-Tree mines patterns. This incorporated approach will allow the user to discover more meaningful rare patterns and also during times when patterns do not change, saves on the unnecessary mining steps.

7 Conclusions and Future Work

We present a new algorithm for mining rare patterns using a tree structure in a data stream environment with two different variations. To the extent of our knowledge, this is the first algorithm that looks at mining rare patterns in a data stream. Our technique is a one-pass only strategy which is capable of mining rare patterns in a static database or in a dynamic data stream. In the case of mining data streams, our technique is also capable of mining at any given point of time in the stream and with different window and block sizes. One of the contributions of this algorithm is a novel approach using a Connection Table which keeps track of related items in a sliding window and reduces the mining space during itemset generation. In our evaluations on six different datasets, both variations of the SRP-Tree algorithm are capable of generating itemsets in a more efficient manner compared to the SC-Tree.

In the future we intend to investigate dynamically adapting the minRareSup and minFreqSup thresholds on-the-fly because data streams are volatile and neither setting fixed thresholds for all data nor defining the thresholds prior based on distribution is deemed suitable. We will also look at the possibility of dynamically adjusting the window size to reflect the density of incoming data in the stream. For example, if the new transactions in the window contained uninteresting or duplicate itemsets and rules, we could (through varying the window size) decide not to mine until more interesting itemsets and rules are captured. It will also be interesting to investigate the limitations of the tree with respect to the different characteristics and intensity of the data stream.

References

1. Adda, M., Wu, L., Feng, Y.: Rare itemset mining. In: Proceedings of the Sixth International Conference on Machine Learning and Applications, ICMLA 2007, pp. 73–80. IEEE Computer Society, Washington, DC (2007)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pp. 487–499. Morgan Kaufmann, Santiago (1994)
3. Cheng, J., Ke, Y., Ng, W.: Maintaining frequent closed itemsets over a sliding window. J. Intell. Inf. Syst. **31**, 191–215 (2008)
4. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Moment: maintaining closed frequent itemsets over a stream sliding window. In: Proceedings of the Fourth IEEE International Conference on Data Mining, ICDM 2004, pp. 59–66. IEEE Computer Society, Washington, DC (2004)
5. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. Knowl. Inf. Syst. **10**, 265–294 (2006)
6. Cuzzocrea, A.: Models and algorithms for high-performance distributed data mining. J. Parallel Distrib. Computi. **73**(3), 281–283 (2013)
7. Cuzzocrea, A., Furfaro, F., Masciari, E., Saccà, D., Sirangelo, C.: Approximate query answering on sensor network data streams. In: GeoSensor Networks, vol. 49, pp. 53–72 (2004)
8. Cuzzocrea, A., Papadimitriou, A., Katsaros, D., Manolopoulos, Y.: Edge betweenness centrality: a novel algorithm for qos-based topology control over wireless sensor networks. J. Network Comput. Appl. **35**(4), 1210–1217 (2012). http://dx.doi.org/10.1016/j.jnca.2011.06.001
9. Cuzzocrea, A., Saccà, D., Ullman, J.D.: Big data: a research agenda. In: Proceedings of the 17th International Database Engineering & #38; Applications Symposium, IDEAS 2013, pp. 198–203. ACM, New York (2013)
10. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows: (extended abstract). In: Proceedings of theThirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 635–644 (2002)
11. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.S.: Mining frequent patterns in data streams at multiple time granularities (2002)
12. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD 2000, pp. 1–12. ACM, New York (2000)

13. Huang, D.T.J., Koh, Y.S., Dobbie, G., Pears, R.: Kernel-tree: mining frequent patterns in a data stream based on forecast support. In: Thielscher, M., Zhang, D. (eds.) AI 2012. LNCS, vol. 7691, pp. 614–625. Springer, Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-35101-3_52
14. Huang, D.T.J., Koh, Y.S., Dobbie, G., Pears, R.: Detecting changes in rare patterns from data streams. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., Kao, H.-Y. (eds.) PAKDD 2014, Part II. LNCS, vol. 8444, pp. 437–448. Springer, Heidelberg (2014). http://dx.doi.org/10.1007/978-3-319-06605-9_36
15. Koh, Y.S., Rountree, N.: Finding sporadic rules using apriori-inverse. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 97–106. Springer, Heidelberg (2005)
16. Koh, Y.S., Dobbie, G.: Efficient single pass ordered incremental pattern mining. In: Hameurlain, A., Küng, J., Wagner, R., Cuzzocrea, A., Dayal, U. (eds.) TLDKS VIII. LNCS, vol. 7790, pp. 137–156. Springer, Heidelberg (2013). http://dx.doi.org/10.1007/978-3-642-37574-3_6
17. Lavergne, J., Benton, R., Raghavan, V.V.: TRARM-RelSup: targeted rare association rule mining using itemset trees and the relative support measure. In: Chen, L., Felfernig, A., Liu, J., Raś, Z.W. (eds.) ISMIS 2012. LNCS, vol. 7661, pp. 61–70. Springer, Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-34624-8_7
18. Lee, C.H., Lin, C.R., Chen, M.S.: Sliding window filtering: an efficient method for incremental mining on a time-variant database. Inf. Syst. **30**(3), 227–244 (2005)
19. Leung, C.K.-S., Cuzzocrea, A., Jiang, F.: Discovering frequent patterns from uncertain data streams with time-fading and landmark models. In: Hameurlain, A., Küng, J., Wagner, R., Cuzzocrea, A., Dayal, U. (eds.) TLDKS VIII. LNCS, vol. 7790, pp. 174–196. Springer, Heidelberg (2013). http://dx.doi.org/10.1007/978-3-642-37574-3_8
20. Leung, C.K.S., Khan, Q.I.: Dstree: A tree structure for the mining of frequent sets from data streams. In: Proceedings of the Sixth International Conference on Data Mining, ICDM 2006, pp. 928–932. IEEE Computer Society, Washington, DC (2006)
21. Li, H.F., Lee, S.Y.: Mining frequent itemsets over data streams using efficient window sliding techniques. Expert Syst. Appl. **36**, 1466–1477 (2009)
22. Liu, B., Hsu, W., Ma, Y.: Mining association rules with multiple minimum supports. In: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 337–341 (1999)
23. Luna, J., Romero, J., Ventura, S.: On the adaptability of g3parm to the extraction of rare association rules. Knowl. Inf. Syst. **38**, 391–418 (2013). http://dx.doi.org/10.1007/s10115-012-0591-9
24. Mozafari, B., Thakkar, H., Zaniolo, C.: Verifying and mining frequent patterns from large windows over data streams. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, pp. 179–188. IEEE Computer Society, Washington, DC (2008). http://dl.acm.org/citation.cfm?id=1546682.1547157
25. Okubo, Y., Haraguchi, M., Nakajima, T.: Finding rare patterns with weak correlation constraint. In: 2010 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 822–829 (2010)
26. Szathmary, L., Napoli, A., Valtchev, P.: Towards rare itemset mining. In: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2007, vol. 01, pp. 305–312. IEEE Computer Society, Washington, DC (2007)
27. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., Lee, Y.K.: Sliding window-based frequent pattern mining over data streams. Inf. Sci. **179**(22), 3843–3865 (2009)

28. Troiano, L., Scibelli, G., Birtolo, C.: A fast algorithm for mining rare itemsets. In: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, ISDA 2009, pp. 1149–1155. IEEE Computer Society, Washington, DC (2009)
29. Tsang, S., Koh, Y.S., Dobbie, G.: RP-tree: rare pattern tree mining. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2011. LNCS, vol. 6862, pp. 277–288. Springer, Heidelberg (2011)
30. Tsang, S., Koh, Y.S., Dobbie, G., Alam, S.: SPAN: finding collaborative frauds in online auctions. Knowl.-Based Syst. **71**, 389–408 (2014). http://dx.doi.org/10.1016/j.knosys.2014.08.016