Finding Frequent Itemsets in High-Speed Data Streams

Xingzhi Sun   Maria E. Orlowska   Xue Li
School of Information Technology and Electrical Engineering
The University of Queensland, Australia
{sun, maria, xueli}@itee.uq.edu.au

Abstract

Finding frequent itemsets from data streams is one of important tasks of stream data mining. In a time-varying data stream, when the significant change on the frequent itemsets is detected, it is ideal to compute the new frequent itemsets as quickly as possible. Traditional stream data mining algorithms mainly focus on finding the frequent itemsets in one scan, which saves the disk access time. However, the computation cost of processing data is also an important factor in the stream management. In this paper, we propose a new approach that can avoid disk access and meanwhile simplify the computation of data processing. The basic idea is: during the mining of the frequent itemsets, we make every data pass on the significant amount of new coming data rather than scan the old data multiple times. Preliminary experiment results are reported to demonstrate the performance of our approach.

1 Introduction

Nowadays, a growing number of applications generate data streams [2, 6], such as telecom call records, web click-streams, data collected in sensor networks and etc. Generally, the data streams have the following characteristics: 1) data are coming continuously and at a very rapid rate, 2) the size of data is unbounded, and 3) data (including the patterns hidden in the data) are time-varying. Because the traditional database management system is designed for finite and persistent datasets, the tasks of managing data streams provide researchers with many new challenges and opportunities. In general, the stream data mining research targets on extracting *approximate* knowledge/patterns by only scanning the data stream *once* so that the mining results can be obtained as quickly as possible.

Finding frequent itemsets is one of the most important data mining tasks. In some applications, one may be only interested in the frequent itemsets in the recent period of time. For example, in the telecommunication network fault analysis, it is important to know what are the frequent itemsets happening before the faults on the network during the recent period of time. In a time-varying data stream, the frequency of itemsets may change with time. This will invalid some old frequent itemsets and also introduce new ones. When changes are detected (e.g., a large fraction of old frequent itemsets are found no longer valid during the test against new arriving data), the frequent itemsets need to be re-computed to reflect the features of new data and the result should be available as quickly as possible.

In this paper, we discuss how to find frequent itemset from a high speed data stream. The research problem is specified in Figure 1. Let S be a data stream in which each transaction record comes with a timestamp at a rapid rate. The interval $[t_s, t_e]$ gives the range of data for computing frequent itemsets, where t_s and t_e are two timestamps. We assume that data in $[t_s, t_e]$ cannot fit in the main memory. Traditionally, these data are stored into the hard disk. After timestamp t_e, certain algorithm can be applied to compute frequent itemsets and the result set R can be obtained at the timestamp t_r. In this paper, to meet the real time requirement, we try to propose an approach to *gradually* compute the frequent itemsets when data arrive such that the *approximate* result set R' can be obtained at t'_r where $t'_r - t_e \ll t_r - t_e$.

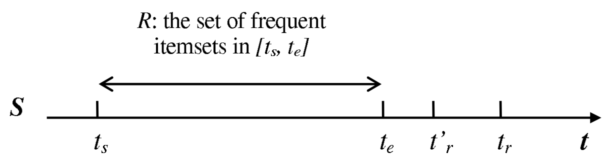

Figure 1. A fragment of sequence

To shorten the response time $t'_r - t_e$, previous studies [8, 5, 9] focus on avoiding disk access. That is, data are not stored into the hard disk and are only buffered in the main memory. After being processed, historic data in the main memory are replaced by new arriving data and only the summary of historic data is maintained. Finally, the approximate knowledge is derived from the summary. The research driven by the above basic idea targets on finding an appropriate summary such that 1) the size of summary is far less than that of original data, 2) approximate knowledge can be derived from the summary, and 3) the summary can be maintained incrementally.

Generally, the response time is related to the disk access time and the data processing time. In some cases, only controlling the disk access time cannot be sufficient to process the data stream. For example, if data come with a very high speed and the data processing speed is relatively slow due to the complexity of the algorithm, when the data buffered in the main memory are computed, there are already large amounts of new data accumulated. In this case, the new arriving data cannot be buffered into the main memory due to the size limit. Then, the data have to be stored into hard disk, or the sampling technique needs to be applied. Apparently, both will have negative impact on the stream processing.

In this paper, we investigate the problem of finding frequent itemsets in data streams from a different angle. That is, besides avoiding disk access, we also try to reduce the data processing time. It is well-known that finding accurate frequent itemsets needs multiple scans on the data. Our basic idea is: in a data stream, when *enough* amount of data are observed, we compute the part of mining result from one scan on these data, then we make the subsequent scans on the new coming data until the complete set of frequent itemsets is discovered. Especially, we use Chernoff bound [3] to find how many transactions are significant enough to show the statistical features of the data stream. Then, using this number, we can partition the streaming data into fragments and the *k*-th fragment corresponds to a data scan required for finding the *k*-lengthed frequent itemsets. Thus we only need to scan the whole data stream once and the computation cost is dramatically reduced because in each fragment we only compute a set of fixed-lengthed frequent itemsets.

The rest of this paper is organized as follows. Section 2 gives the background of finding frequent itemsets in data streams. Our approach is proposed and discussed in Section 3. The preliminary experiment results are shown in Section 4. Finally, we conclude this paper in Section 5.

2 Background

In this section, we give the background of our study. First, we review the problem of mining frequent itemsets and then discuss some related work of finding frequent itemsets on the context of data streams.

The frequent itemset mining is introduced in [1] by Agrawal and Srikant. To facilitate our discussion, we give the formal definitions as follows.

Let $I = \{i_1, \ldots, i_m\}$ be a set of *items*. An *itemset* X is a subset of I. X is called k-itemset if $|X| = k$, where k is the *size* (or *length*) of the itemset. A *transaction* T is a pair (tid, X), where tid is a unique identifier of a transaction and X is an itemset. A transaction (tid, X) is said to *contain* an itemset Y iff $X \supseteq Y$. A *dataset* D is a set of transactions.

Given a dataset D, the *support of an itemset* X, denoted as $Supp(X)$, is the fraction of transactions in D that contain X. An itemset X is *frequent* if $Supp\,(X)$ is no less than a given threshold s_0. An important property of the frequent itemsets, called the *Apriori property*, is that every nonempty subset of a frequent itemset must also be frequent.

The problem of finding frequent itemsets can be specified as: given a dataset D and a support threshold s_0, to find any itemset whose support in D is no less than s_0. In this paper, the dataset D corresponds to the segment of data stream S in interval $[t_s, t_e]$.

The algorithm *Apriori* is introduced in [1] to find the frequent itemsets based on the Apriori property. Basically, $(k+1)$-candidate frequent itemsets are generated by k-frequent itemsets to reduce the searching space. The algorithm consists of the following steps.

1. Candidate generation: Generating candidate frequent itemsets, each of which has k items (k is initialized as 1 when the algorithm starts).

2. Support counting: Scanning the dataset to find the support of each candidate itemset and prune those whose support is less than the threshold.

3. $k = k + 1$, go to step 1.

It is clear that the Apriori algorithm needs at most $l + 1$ scans if the maximum size of frequent itemset is l. On the context of data streams, to avoid disk access, previous studies focus on finding the approximation of frequent itemsets with a bound of space complexity.

In [8], the authors propose two algorithms, called *Stick-Sampling* and *Lossy-counting*, to support (ϵ, δ) approximation for the frequent *items* problem with a bound of space complexity. That is, given an error bound ϵ and a probabilistic bound δ, both algorithms guarantee that with confidence $1-\delta$, 1) all items (not itemsets) whose true support exceeds s_0 are output, 2) no item whose true support is less than $s_0 - \epsilon$ is output, and 3) computed support is less than the true support by at most ϵ. Specially, with (ϵ, δ) approximation, the space complexity of Stick-Sampling and Lossy-counting are $O\left(\frac{1}{\epsilon}\log\left(s_0^{-1}\delta^{-1}\right)\right)$ and $O\left(\frac{1}{\epsilon}\log\left(\epsilon N\right)\right)$ respectively, where N is the total number of transactions. In [8], the authors also propose a Buffer-Trie-SetGen approach to extend finding frequent items to finding frequent itemsets. In this approach, when a transaction (a set of items) arrives, all of its non-empty subsets are enumerated and treated as “items”. However, this leads to the difficulty of determine the upper bound of the number of entries.

The work [9] also applies the Buffer-Trie-SetGen approach to compute the frequent itemsets. However, the proposed algorithm is false-negative oriented, i.e., some frequent itemsets may not be included in the mining result. This algorithm is effective to reduce the memory consumption because a large number of infrequent itemsets (false-positive ones) are excluded from the mining result. Experiments shows that the quality of mining results can be guarantied by controlling the predefined parameters.

In [5], data stream is partitioned into data fragments, each of which can fit in the main memory. Given a support error $0 < \epsilon < s_0$, to guarantee the computed support in the range of $[Supp(X) - \epsilon, Supp(X)]$, the itemsets whose support is no less than ϵ (called *semi-frequent itemsets*) are found in each data fragments. All semi-frequent itemsets are incrementally maintained in a compact data structure, which is a pattern tree with tilt time window frame embedded on its nodes. At time t_e, given a window size T, the approximate frequent itemsets in $[t_e - T, t_e]$ can be derived from the pattern tree.

We can see that previous studies mainly focus on avoiding disk access but disregard the cost of data processing. Considering the work [5], suppose that time interval Δt is required to find the semi-frequent itemsets in a data fragment. If the streaming data arrive at a very rapid rate r_N, during the period of computing the data fragment in the buffer, $r_N * \Delta t$ amount of new arriving data are accumulated. These data may not be buffered due to the limited size of memory. From the above discussion, we believe that for data stream management, besides the attempt of avoiding disk access, data processing time should also be considered.

Our consideration for simplifying the computing process is: instead of scanning the entire data multiple times, we make each data pass on a fraction of data stream and always perform the next scan on the new arriving data. This idea is inspired by the previous work [4], which studies the decision tree induction in a high-speed data stream. In [4], when enough data (determined by the Hoeffding bound) are observed, the root attribute of the decision tree is selected. Then subsequent data are passed through the induced portion to further choose a split attribute until the decision tree is built completely. In this paper, we apply the similar idea on the context of frequent itemset mining.

3 Our Proposed Approach

In this section, we propose a new approach to solve the problem defined in Section 1. Remember that as shown in Figure 1, our goal is to quickly find approximate frequent itemsets from the data segment of data stream S in the interval $[t_s, t_e]$. To minimize the response time $t'_r - t_e$, our approach should satisfy the following requirements during the mining process: 1) no disk access is needed and 2) the data in the buffer can be processed very quickly.

The new proposed approach, called S_Apriori, follows the basic structure of the Apriori algorithm. As stated in Section 2, the Apriori algorithm can find frequent itemsets by scanning the dataset at most $l + 1$ times, where l is the maximum size of frequent itemsets. The pseudo code of S_Apriori is shown in Figure 2. Let n_0 be the *enough* number of transactions that can reflect the true support of itemsets (we will discuss how to determine n_0 later). At the beginning stage, we find 1-frequent itemsets F_1 from the first n_0 transactions. Candidate 2-frequent itemsets are then generated based on F_1 and their supports are computed by scanning the next n_0 transactions. Such process continues until there are no candidate frequent itemsets generated. Apparently, if the number of actual scans is l_0, the data used in computing frequent itemset is $l_0 * n_0$. Note that although different lengthed frequent itemsets are found from different data fragments, due to the candidate generation process, the Apriori property still holds for the mining result. That is, our mining result ensures that any subset of frequent itemsets is frequent.

```
Algorithm
Input: A data stream $S$ and the support threshold $s_0$.
Output: Approximate frequent itemsets $F$ in a segment of $S$
Method:
    Determine $n_0$;
    Scan $n_0$ transactions and find $F_1 = \{$frequent 1-itemset$\}$;
    for($k = 2; F_{k-1} \neq \phi; k++$) do
      $C_k$ =Candidate k-itemsets generated from $F_{k-1}$;
      Scan the next $n_0$ transactions and
        compute $Supp(X)$ for any $X \in C_k$
      $F_k = \{X \in C_k | (Supp(X)) \geq s_0\}$;
    Approximate frequent itemsets $F = \bigcup_k F_k$;
```

Figure 2. S_Apriori algorithm

After introducing our basic idea, we further discuss our approach by answering the following three questions.

1. How to determine n_0 (where n_0 is the number of transactions that is significant enough to reflect the support of itemsets)?

2. Given n_0 and the number of scans l_0, theocratically, the proposed approach needs $l_0 * n_0$ transactions to compute frequent itemsets. How can we link $l_0 * n_0$ transactions to the data in the interval $[t_s, t_e]$?

3. Since our approach provides the approximate mining result, can the result R' be a good approximation of the accurate result set R (where R is the set of frequent itemsets in the segment of S in $[t_s, t_e]$)?

First, to determine the number of transactions that can reflect the true support of itemsets, we use a statistical result Chernoff bound [3]. This statistical tool has been successfully applied in the previous studies [4, 7, 9] of stream data management. Based on the n observations of the independently generated data, the Chernoff bound can provide certain probabilistic guaranty on the estimation of statistic features of the entire data. As stated in [9], Chernoff bound can be applied into the problem of finding frequent itemsets in data streams as follows.

Given n transactions observed and the support threshold s_0, the Chernoff bound ensures that for any itemset X, with confidence $1-\delta$, the true support of X ($Supp(X)$) is in the range of $\left[\widetilde{Supp(X)}-\epsilon, \widetilde{Supp(X)}+\epsilon\right]$, where $\widetilde{Supp(X)}$ is the computed support of X in these n transactions and $\epsilon = \sqrt{\frac{2s_0 \ln(\frac{2}{\delta})}{n}}$.

Given the error bound ϵ, probabilistic bound δ, and the minimum support s_0, we can use the above formula to compute the number n_0. For example, if $s_0 = 0.01$, $\epsilon = 0.002$ and $\delta = 0.1$, with Chernoff bound, we have $n = 14,979$. This means that for an itemset X, after 14,979 transactions are observed, its true support is in the range of $\left[\widetilde{Supp(X)} - 0.002, \widetilde{Supp(X)} + 0.002\right]$ with high probability 0.9. Note that $\widetilde{Supp(X)}$ gives the support of X in these 14,979 transactions.

Now we discuss how the number of transactions required by our approach links to the number of transactions in the interval $[t_s, t_e]$. Suppose that the data arriving rate is fixed, denoted as c_0. The number N of transactions in $[t_s, t_e]$ is $(t_e - t_s) * c_0$. In addition, assume that we know the number of scans (l_0) required to find all frequent itemsets. To match the number of transactions required in S_Apriori to N, it is ideal that each fragment has $\left\lceil \frac{N}{l_0} \right\rceil$ transactions. Also remember that to achieve the (ϵ, δ) approximation, the number of transaction required is $\frac{2s_0 \ln\left(\frac{2}{\delta}\right)}{\epsilon^2}$, which is determined by Chernoff bound. Thus, we can adjust n_0 as $Max\{\left\lceil \frac{N}{l_0} \right\rceil, \frac{2s_0 \ln\left(\frac{2}{\delta}\right)}{\epsilon^2}\}$.

In practice, the number of data scans l_0 is unknown to the user before the mining, however, according to the Apriori algorithm, we know that $l_0 \in \{l, l+1\}$ where l is the maximum size of frequent itemsets. In a time-varying data streams, we often need to recompute frequent itemsets if any significant change is detected. In this case, we could estimate l_0 based on old frequent itemsets. Then, n_0 can be decided based on the estimated size l'_0. If the actual size $n_0 * l_0$ is not well-matched the total number of transactions N in the interval $[t_s, t_e]$ (due to the bad estimation of l_0), we have the following two situations:

- $n_0 * l_0 > N$: It means that the number of transactions required for computation is more than the number of transactions in the interval. In this case, we can use the new arriving data to complete the computation. Or we can buffer the last n transactions, on which further scans are made.

- $n_0 * l_0 < N$: the computation is completed before some data in the interval are used. One way to solve this problem is: after the frequent itemsets are found, we use the rest data to do the support checking, based on which the mining result could be adjusted. The other way is to ignore the data if the amount of unused data is not large.

In our approach, we don't make strict constraint on the matching between actual data required by S_Apriori and the data in window $[t_s, t_e]$. This is because the data range of the mining is also an unknown factor, which is determined by domain experts. In practice, often multiple window sizes are attempted. So, our approach would be working if the estimation makes the data range sensible to the application domain. In this case, the actual interval for computing is relaxed to $[t_s, t'_e]$, where $|t_e - t'_e|$ should be much less than $(t_e - t_s)$.

Finally, we discuss whether the mining result is close to the accurate result in $[t_s, t'_e]$. Based on Algorithm S_Apriori, we have the following two observations.

1. If data are uniformly distributed in the interval $[t_s, t'_e]$, each n_0 transactions carry the same statistical information. Then, our proposed approach can find the result that is a close approximation of the accurate result.

2. If data contain distribution changes on the frequent itemsets, our approach may make the considerable difference between the mining result and the true result. Suppose that after the set F_1 of 1-frequent itemsets is found in the first n_0 transactions, there is a sudden change on the distribution of the subsequent data. Then, some frequent itemsets will be missing if any item they contained is not included in F_1.

Observation 1 gives the situation in which our approach can work perfectly. From Observation 2, we know that the performance of our approach depends on the data. The reason is that the computations on different fragment (n_0 transactions) are tightly coupled because the candidate frequent itemsets are generated by the previously computed frequent itemsets. Theoretically, if there is any fragment providing incorrect result, the computation on the subsequent fragments may become meaningless.

To sum up, the advantages of our proposed algorithm are 1) data are only scan once and 2) because each scan is made on different data fragment, the computation cost is distributed and only the simple support counting is required for the fixed-lengthed candidate frequent items during each scan. The disadvantage is that the performance of the algorithm can be affected by the data distribution.

4 Preliminary Experiment Results

In this section, we show the preliminary experiment results to test the effectiveness of our proposed approach and discuss how to improve the accuracy of the mining result.

Consider the algorithm S_Apriori, for any arriving transaction, only a set of fixed-lengthed candidate frequent itemsets are tested, which makes the data processing very simple. Also, during the scan of the k-th data fragment, only the shorter-lengthed frequent itemsets (i.e., $F_1 \cup F_2 \cdots \cup F_{k-1}$) and k-lengthed candidate frequent itemsets (i.e., C_k) are kept in the main memory. From this analysis, we can see that the S_Aprior algorithm is efficient to the stream processing due to its time complexity and physical complexity. So, the focus of the experiment part is to discuss the effectiveness of our approach, i.e., whether the output of S_Apriori is a good approximation of the accurate result.

We show some preliminary experiment results on both artificial dataset and real dataset. Because our experiment goal is to demonstrate the effectiveness of our approach, all datasets are treated as static. The artificial dataset $T10I4D100K$ is created by IBM synthetic data generator [1]. Two real datasets are derived from the fault-related data from an Australian telecommunication company. In the real datasets, each "transaction" is a set of events that occurred within 24 hours before a fault on the telecommunication network. Especially, dataset $Telecom\_1$ consists of such transactions that are collected from a single network node in one month. To deliberately make the distribution change, we create $Telecom\_2$ by combining two half-month data collected from two different nodes, i.e., the first half transactions are collected in fifteen days from the node A and the rest data are from the other node B in the same fifteen days. The characteristics of the above-mentioned datasets are given in Table 1.

Table 1. Datasets

Dataset	# of items	# of transactions
T10I4D100K	*1000*	*100K*
Telecom_1	*189*	*52K*
Telecom_2	*190*	*54K*

In our experiment, we always set $s_0 = 0.01$, $\epsilon = 0.002$, and $\delta = 0.1$. The number of transactions determined by Chernoff bound N_c is 14.98K. According to the IBM data generator, we can estimate that the number of scans required for finding frequent itemsets in dataset *T10I4D100K* is about 4. So, we set the fragment size n_0 as 25K (i.e., $Max\{\frac{100K}{4}, 14.98K\}$) for the dataset *T10I4D100K*. For the telecommunication datasets, we use the number 14.98K (N_c) since we have no idea how many scans required in the mining process.

As discussed before, the total number N_1 of transactions required for the algorithm S_Apriori may not well match the number N of transactions in the dataset. If $N_1 > N$, we make the rest scans on the last N_c transactions in the dataset, where N_c is the number determined by the Chernoff bound. And the accurate mining result R is the set of frequent itemsets computed from N transactions. Otherwise if $N > N_1$, we ignore the rest $N - N_1$ transactions. In this case, R is the set of frequent itemsets found in N_1 transactions.

Table 2. Experiment results

Dataset	Fragment Size n_0	Recall	Precision
T10I4D100K	$25K$	0.99	0.99
Telecom_1	$14.98K$	0.86	0.92
Telecom_2	$14.98K$	0.61	0.80

Let the output of the algorithm S_Apriori be R'. We compare R' and the accurate result R by using recall and precision, which are defined as $\frac{|R \cap R'|}{|R|}$ and $\frac{|R \cap R'|}{|R'|}$ respectively. The comparison results are shown in Table 2. Because in the dataset $T10I4D100K$, data are generated with the same distribution, our approach works very effectively. In the real dataset $Telecom\_1$, there is no guarantee on the consistence of the data distribution. So the recall and precision are relatively low compared with the artificial dataset. In the third dataset $Telecom\_2$, because the data distribution is deliberately changed in the dataset, R' cannot be an appropriate approximation of R. These preliminary results also verify the two observations that we made in Section 3.

One reason why the performance of our approach depends on the data distribution is that the computations on different fragments are dependant. Since this dependency lies in the candidate frequent itemset generation, we may test more candidate itemsets during each scan to relax this dependency. Based on this consideration, in the mining process, we could lower the support threshold from s_0 to s_1. Then a larger amount of frequent itemsets are discovered and consequently more candidate frequent itemsets are generated and tested. After the mining process complete, we can recover the threshold s_0, i.e., select the itemsets that are frequent in terms of s_0 as the mining result. Note that according to the above mining process, the frequent $(k+1)$-itemset (in terms of s_0) found in $(k+1)$-th fragment can be generated by the semi-frequent itemset (i.e., the itemset whose support is in the range $[s_1, s_0)$) in the k-th fragment. Thus, the Apriori property may be violated in the mining result. So, in the final step, we need to use the following two rules to adjust frequent itemsets to ensure the Apriori property.

1. Adjustment starts from the largest-sized frequent item-

sets to 1-frequent itemset.

2. If a frequent itemset X is of size k and any $(k-1)$-sized subset Y of X satisfies the condition $Supp(X) > Supp(Y)$, the support of Y is updated as $Supp(X)$[1].

We will test the effectiveness of the above-mentioned approach in the future work.

5 Conclusion

In this paper, we have investigated the problem of finding frequent itemsets in high-speed data streams. To meet the real time requirement, previous research targets on avoiding disk access to reduce the response time. Due to the fact that the response time is related to disk access time and the data process time, we made our attempt from a different angle, i.e., to lower the computation cost of data processing. Based on this consideration, we have proposed a new approach which can avoid disk access and meanwhile simplify the computation of data processing. The basic idea is that data stream is partitioned into fragments (with certain size), from each of which part of mining result is generated. Finally, we report and analyze the preliminary experiment results of our proposed algorithm.

One weakness of the proposed approach is that the level of approximation between computed result and accurate result can be affected by the data distribution. In the future work, we will investigate how to guarantee the good approximation of mining result when the data distribution changes.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th VLDB*, pages 487–499, 1994.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS '02*, pages 1–16, 2002.

[3] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[4] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD*, pages 71–80, 2000.

[5] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*. AAAI Press/MIT Press, 2003.

[6] L. Golab and M. T. Ozsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[7] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD*, pages 97–106, 2001.

[8] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[9] J. X. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *VLDB*, pages 204–215, 2004.

[1] In the S_Apriori approach proposed in Section 3, we also use the same two rules to adjust the support of frequent itemsets to guarantee that for any two frequent itemset X and Y with $X \supset Y$, $Supp(X) \leq Supp(Y)$ always holds.