Reducing Rule Covers with Deterministic Error Bounds*

Vikram Pudi vikram@dsl.serc.iisc.ernet.in
Jayant R. Haritsa haritsa@dsl.serc.iisc.ernet.in

Database Systems Lab, SERC
Indian Institute of Science
Bangalore 560012, India

Abstract

The output of boolean association rule mining algorithms is often too large for manual examination. For dense datasets, it is often impractical to even generate all frequent itemsets. The closed itemset approach handles this information overload by pruning "uninteresting" rules following the observation that most rules can be derived from other rules. In this paper, we propose a new framework, namely, the generalized closed (or g-closed) itemset framework. By allowing for a small tolerance in the accuracy of itemset supports, we show that the number of such redundant rules is far more than what was previously estimated. Our scheme can be integrated into both levelwise algorithms (Apriori) and two-pass algorithms (ARMOR). We evaluate its performance by measuring the reduction in output size as well as in response time. Our experiments show that incorporating g-closed itemsets provides significant performance improvements on a variety of databases.

1 Introduction

The output of boolean association rule mining algorithms is often too large for manual examination. For dense datasets, it is often impractical to even generate all frequent itemsets. Among recent approaches [16, 15, 9, 6, 8, 5, 4] to manage this gigantic output, the *closed itemset* approach [16, 15] is attractive in that both the identities and supports of all frequent itemsets can be derived *completely* from the frequent closed itemsets. However, the usefulness of this approach critically depends on the presence of frequent itemsets that have supersets with *exactly the same support*. This causes the closed itemset approach to be sensitive to *noise* in the database – even minor changes can result in a significant increase in the number of frequent closed itemsets. For example, adding

* A poster of an earlier version of this paper appeared in Proc. of Intl. Conf. on Data Engineering (ICDE), March 2003, Bangalore, India.

a select 5% transactions to the mushroom dataset (from the UC Irvine Repository) caused the number of closed frequent itemsets at a support threshold of 20% to increase from 1,390 to 15,541 – a factor of 11 times! The selection of transactions was made so as to break exact equalities in itemset supports.

In order to overcome this limitation, we propose in this paper the *generalized closed (or g-closed) itemset framework*, which is more robust to the database contents. In our scheme, although we do not output the *exact* supports of frequent itemsets, we estimate the supports of frequent itemsets within a *deterministic*, user-specified "tolerance" factor. A side-effect of allowing for a tolerance in itemset supports is that the supports of some "borderline" infrequent itemsets may be over-estimated causing them to be incorrectly identified as frequent. Since our typical tolerance factors are much less than the minimum support threshold, this is not a major issue. Further, an extra (quick) database pass can always be made to check these borderline cases.

We provide theoretical arguments to show why the g-closed itemset scheme works and substantiate these observations with experimental evidence. Our experiments were run on a variety of databases, both real and synthetic, as well as sparse and dense. Our experimental results show that even for very small tolerances, we produce *exponentially fewer rules* for most datasets and support specifications than the closed itemsets, which are themselves much fewer than the total number of frequent itemsets.

Our scheme can be used in one of two ways: (1) as a post-processing step of the mining process (like in [9, 6]), or (2) as an integrated solution (like in [17, 11]). We show that our scheme can be integrated into both levelwise algorithms as well as the more recent two-pass mining algorithms. We chose the classical Apriori algorithm [2] as a representative of the levelwise algorithms and the recently-proposed ARMOR [14], as a representative of the class of two-pass mining algorithms. Integration into Apriori yields a new algorithm, g-**Apriori** and into ARMOR, yields g-**ARMOR**. Our experimental results show that these integrations often result in a significant reduction in response-time, especially for dense datasets.

We note that integration of our scheme into two-pass mining algorithms is a novel and important contribution because two-pass algorithms have several advantages over Apriori-like levelwise algorithms. These include: (1) significantly less I/O cost, (2) significantly better overall performance as shown in [12, 14], and (3) the ability to provide approximate supports of frequent itemsets at the end of the first pass itself, as in [7, 14]. This ability is an essential requirement for mining *data streams* [10] as it is infeasible to perform more than one pass over the complete stream.

2 Generalized Closed Itemsets

In addition to the standard boolean association rule mining inputs ($\mathcal{I}$, the set of database columns, $\mathcal{D}$, the set of database rows and $minsup$, the minimum support), the frequent g-closed itemset mining problem also takes as input ϵ, the user-specified tolerance factor. It outputs a set of itemsets (that we refer to as the frequent g-closed itemsets) and their supports. The frequent g-closed itemsets are required to satisfy the following properties: (1) The supports of all frequent itemsets can be derived from the output within an error of ϵ. (2) If $\epsilon = 0$, the output is precisely the frequent closed itemsets.

Note that the set of g-closed itemsets may not be unique for a given database and mining parameters. We are interested in obtaining any set of itemsets that satisfies the above properties since that would ensure that the frequent itemsets and their supports can be estimated sufficiently accurately.

2.1 ϵ-equal Support Pruning

The key concept in the g-closed itemset framework lies in the *generalized openness propagation property*, which is stated as a corollary to the following theorem[1]. Here, the supports of itemsets X and Y are said to be approximately equal or ϵ-equal (denoted as $support(X) \approx support(Y)$) iff $|support(X) - support(Y)| \leq \epsilon$. Also, we refer to the allowable error in itemset counts as *tolerance count*. The term "tolerance" is reserved for the allowable error in itemset supports and is equal to the tolerance count normalized by the database size.

Theorem 2.1 *If Y and Z are supersets of itemset X, then $support(Z) - support(Y \cup Z) \leq support(X) - support(Y)$.*

Corollary 2.1 *If X and Y are itemsets such that $Y \supseteq X$ and $support(X) \approx support(Y)$, then for every itemset $Z : Z \supseteq X$, $support(Z) \approx support(Y \cup Z)$.*

This result suggests a general technique to incorporate into mining algorithms, which we refer to as *ϵ-equal support pruning*: If an itemset X has an immediate superset Y, with ϵ-equal support, then prune Y and avoid generating any candidates that are supersets of Y. The support of any of these pruned itemsets, say W, will be ϵ-equal to one of its subsets, $(W - Y) \cup X$. The remaining unpruned itemsets are referred to as *generators*. Note that this is a generalization of the notion of generators proposed in [11] and would reduce to it when $\epsilon = 0$.

2.2 Generating g-Closed Itemsets When $\epsilon = 0$

We now present a simple technique to generate closed itemsets from their generators. This technique does not involve an additional database scan as is required in the A-Close algorithm [11] for generating frequent closed itemsets. Also, the technique will directly carry over to the g-closed itemset case. For any closed itemset Y with generator X, the following theorem enables us to determine $Y - X$ using information that could be gathered while performing ϵ-equal support pruning. We refer to $Y - X$ as $X.pruned$.

Theorem 2.2 *Let Y be a closed itemset and $X \subseteq Y$ be the generator of Y. Then for each item A in $Y - X$, $X \cup \{A\}$ would be pruned by the equal support pruning technique. No other immediate supersets of X would be pruned by the same technique.*

Therefore, in order to generate a closed itemset Y from its generator X, it is sufficient to compute $X.pruned$ while performing ϵ-equal support pruning. Note that if

[1] Proofs of theorems are available in [13]

some subset W of X had a proper subset V with equal support, then W would be pruned using the ϵ-equal support pruning technique. It would then be necessary to include all the items in $V.pruned$ in $X.pruned$. That is, the $pruned$ value of any itemset needs to be propagated to all its supersets.

2.3 Generating g-Closed Itemsets When $\epsilon > 0$

The technique outlined above for $\epsilon = 0$ will not produce correct results when $epsilon > 0$ – the supports of all frequent itemsets will not be derivable even approximately from the output. This is because Corollary 2.1 considers for any itemset X, only *one* superset Y with ϵ-equal support. If X has more than one superset (say $Y_1, Y_2, \ldots, Y_n$) with ϵ-equal support then *approximation error accumulates.* A naive interpretation of the generalized openness propagation property would seem to indicate the following: Every itemset $Z : Z \supset X \bigwedge Z \not\supseteq Y_k, k = 1...n$, also has a proper superset $\bigcup_{k=1}^{n} Y_k \cup Z$ with ϵ-equal support. Although this is valid when $\epsilon = 0$, in the general case, it is not necessarily true. However, the following theorem reveals an upper bound on the difference between the supports of $\bigcup_{k=1}^{n} Y_k \cup Z$ and Z.

Theorem 2.3 *If $Y_1, Y_2, \ldots, Y_n, Z$ are supersets of itemset X, then* $support(Z) - support(\bigcup_{k=1}^{n} Y_k \cup Z) \leq \sum_{k=1}^{n}(support(X) - support(Y_k))$.

In our approach we solve the problem of approximation error accumulation by ensuring that an itemset is pruned using the ϵ-equal support pruning technique only if the *maximum* possible cumulative error in approximation does not exceed ϵ. Whenever an itemset X, having more than one immediate superset $Y_1, Y_2, \ldots, Y_n$, with ϵ-equal support is encountered, we prune each superset Y_k only as long as the sum of the differences between the supports of each pruned superset and X is within tolerance.

While performing the above procedure, at any stage, the sum of the differences between the support counts of each pruned superset and X is denoted by $X.debt$. Recall from Section 2.2 that these pruned supersets are included in $X.pruned$. Since $X.pruned$ needs to be propagated to all unpruned supersets of X, it becomes necessary to propagate $X.debt$ as well.

For any itemset X, $X \cup X.pruned$ is referred to as its corresponding g-closed itemset and will have ϵ-equal support. This is a natural extension of the closed itemset concept because when $\epsilon = 0$, $X \cup X.pruned$ is the closed itemset corresponding to X.

2.4 Rule Generation

Given the frequent g-closed itemsets and their associated supports, it is possible to generate association rules with approximate supports and confidences. This is stated in the following theorem:

Theorem 2.4 *Given the g-closed itemsets and their associated supports, let $\hat{c}$ and $\hat{s}$ be the estimated confidence and support of a rule $X_1 \longrightarrow X_2$, and c and s be its actual confidence and support. Then, $\hat{s} - \epsilon \leq s \leq \hat{s}$; and $\hat{c} \times \lambda \leq c \leq \hat{c}/\lambda$ where $\lambda = (1 - \epsilon/minsup)$.*

Further, it has been shown earlier [15, 16] that it suffices to consider rules among adjacent frequent closed itemsets in the itemset lattice since other rules can be inferred by transitivity. This result carries over to frequent g-closed itemsets.

3 Incorporation in Levelwise Algorithms

In the previous sections we presented the g-closed itemset framework and the theory supporting it. In this section we show that the framework can be integrated into levelwise algorithms. We chose the classical Apriori algorithm as a representative of the levelwise mining algorithms. Integration of our scheme into Apriori yields g-Apriori, an algorithm for mining frequent g-closed itemsets.

The g-Apriori algorithm is obtained by combining the ϵ-equal support pruning technique described in Section 2.1 with the subset-based pruning of Apriori. The pseudo-code of the g-Apriori algorithm is shown in Figure 1 and works as follows: The code between lines 1–9 of the algorithm, excluding lines 6 and 7, consists of the classical Apriori algorithm. The SupportCount function (line 4) takes a set of itemsets as input and determines their counts over the database by making one scan over it.

Every itemset X in C_k (the set of candidate k-itemsets), G_k (frequent k-generators) and G (the frequent generators produced so far) has an associated counter, $X.count$, to store its support count during algorithm execution. Every itemset X in G has two fields in addition to its counter: (1) $X.pruned$ (described in Section 2.2). (2) $X.debt$: an integer value to check the accumulation of approximation error in itemset supports.

The Prune function is applied on G_k (line 6) before the $(k+1)$-candidates C_{k+1} are generated from it using AprioriGen (line 8). Its responsibility is to perform ϵ-equal support pruning while ensuring that approximation error in the supports of itemsets is not accumulated. The pseudo-code for this function is shown in Figure 2 and it performs the following task: it removes any itemset X from G_k if X has a subset Y with ϵ-equal count, provided $Y.debt$ remains within tolerance.

The code in lines 1–9, excluding line 7, is analogous to the A-Close algorithm [11] for generating frequent closed itemsets. At the beginning of line 10, G would contain the equivalent of the "generators" of the A-Close algorithm.

The PropagatePruned function is applied on G_k (line 7) and it ensures that the $pruned$ value of each itemset X in G_k is appended with the $pruned$ values of each immediate subset of X. The pseudo-code for this function is shown in Figure 3. The necessity for performing this function was explained in Section 2.2, where we showed that the $pruned$ value of an itemset should be propagated to all its supersets.

Finally, in lines 10–11, the g-closed itemsets are output.

4 Incorporation in Two Pass Algorithms

In this section we show that the g-closed framework can be incorporated into two-pass mining algorithms. As mentioned in the Introduction, this is a novel and important contribution because two pass algorithms are typically much faster than level-wise algorithms and also because they can be tweaked to work on data streams [10]. We selected

```
g-Apriori (D, I, minsup, tol)
Input: Database D, Set of Items I, Minimum Support minsup, Tolerance Count tol
Output: Generalized Closed Itemsets
1.      C_1 = set of all 1-itemsets;
2.      G = φ;
3.      for (k = 1; |C_k| > 0; k++)
4.          SupportCount(C_k, D); // Count supports of C_k over D
5.          G_k = Frequent itemsets in C_k
6.          Prune(G_k, G, tol);
7.          PropagatePruned(G_k, G, tol);
8.          C_{k+1} = AprioriGen(G_k);
9.          G = G ∪ G_k;
10.     for each itemset X in G
11.         Output (X ∪ X.pruned, X.count);
```

Figure 1: **The g-Apriori Algorithm**

```
Prune (G_k, G, tol)
Input: Frequent k-itemsets G_k, Generators G, Tolerance Count tol
Output: Remove non-generators from G_k
1.      for each itemset X in G_k
2.          for each (|X| − 1)-subset Y of X, in G
3.              debt = Y.count − X.count;
4.              if (debt + Y.debt ≤ tol)
5.                  G_k = G_k − {X}
6.                  Y.pruned = Y.pruned ∪ (X − Y)
7.                  Y.debt += debt
```

Figure 2: **Pruning Non-generators from G_k**

```
PropagatePruned (G_k, G, tol)
Input: Frequent k-itemsets G_k, Generators G, Tolerance Count tol
Output: Propagate pruned value to generators in G_k
1.      for each itemset X in G_k
2.          for each (|X| − 1)-subset Y of X, in G
3.              if (X.debt + Y.debt ≤ tol)
4.                  X.pruned = X.pruned ∪ Y.pruned
5.                  X.debt+ = Y.debt
```

Figure 3: **Propagate Pruned Value to Supersets**

the recently-proposed ARMOR [14] as a representative of the class of two-pass mining algorithms. Integration of the g-closed framework into ARMOR yields g-ARMOR, a two-pass algorithm for mining frequent g-closed itemsets.

4.1 The ARMOR Algorithm

We first review the overall structure of the ARMOR algorithm – its details are available in [14]. In this algorithm, the database is conceptually partitioned into disjoint blocks and data is read from disk and processed partition by partition.

In the first pass, the algorithm starts with the set of all 1-itemsets as candidates. After processing each partition, the set of candidates (denoted as C) is updated – new candidates may be inserted and existing ones removed. The algorithm ensures that at any stage, if d is the database scanned so far, then the frequent itemsets within d (also called d-frequent itemsets) are available. The algorithm also maintains the *partial counts* of these itemsets – the partial count of an itemset is its count within the database scanned so far from the point it has been inserted into C.

In the second pass, complete counts of the candidates obtained at the end of the first pass are determined. During this pass, there are no new insertions into C. However, candidates that can no longer become frequent are removed at each stage.

4.2 Details of Incorporation

In ARMOR, the complete supports of candidate itemsets are not available during the first pass. However, for each candidate X, its *partial support* is available representing its support over the portion of the database that has been processed so far starting from the point where the candidate was inserted into C. The rule that we follow in the integrating the g-closed itemset framework is simple[2]: While processing a partition during the first pass, if we find the partial support of an itemset X to be ϵ-equal to that of its superset Y, then prune every proper superset of Y from C while ensuring that the approximation error does not accumulate beyond the tolerance limit. That is, whenever an itemset X, that has more than one immediate superset $Y_1, Y_2, \ldots, Y_n$, with ϵ-equal partial support is encountered, we prune each superset Y_k only as long as the sum of the differences between the partial supports of each pruned superset and X is within ϵ.

Next, if after processing a few more partitions, the supports of X and Y are no longer ϵ-equal, then regenerate the pruned supersets of Y as follows: For every d-frequent itemset Z in C such that $Z \supset X \bigwedge Z \not\supseteq Y$, insert a new candidate $Y \cup Z$ into C. The partial support of the new candidate should be set equal to that of Z.

5 Performance Study

In the previous sections, we have described the g-closed itemset framework along with the g-Apriori and g-ARMOR algorithms. We have conducted a detailed study to assess the utility of the framework in reducing both the output size and the response time of mining operations. Our experiments cover a range of databases and mining workloads including the real datasets from the UC Irvine Machine Learning Database Repository, the synthetic datasets from the IBM Almaden generator, and the real dataset, BMS-WebView-1 from Blue Martini Software. Due to lack of space, we show only representative samples of our results. The complete results are available in [13].

[2]The resulting pseudo-code is not shown here due to lack of space, but is available in [13]

All the algorithms were coded in C++ and the experiments were conducted on a 700-MHz Pentium III workstation running Red Hat Linux 6.2, configured with 512 MB main memory and a local 18 GB SCSI 10000 rpm disk. The same data-structures (hashtrees [2]) and optimizations (using arrays to store itemset counters in the first two database passes) were used in both g-Apriori and Apriori to ensure fairness. We chose tolerance count values ranging from zero (corresponding to the exact closed itemset case) to 1000. While higher values of tolerance are uninteresting, their inclusion is useful in studying the effect of increasing tolerance on the output size.

5.1 Experiment 1: Output Size Reduction

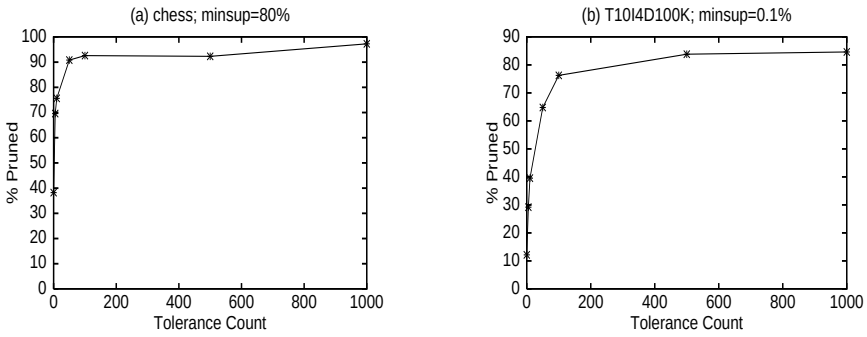

Figure 4: **Output Size Reduction**

We now report on our experimental results. In our first experiment, we measure the output size reduction obtained with the g-closed itemset framework as the percentage of frequent itemsets pruned to result in frequent g-closed itemsets. The results of this experiment are shown in Figures 4a–b. The x-axis in these graphs represents the tolerance count values, while the y-axis represents the percentage of frequent itemsets pruned.

In these graphs, we first see that the pruning achieved is significant in most cases. For example, on the chess dataset (Figure 4a) for a minimum support of 80%, the percentage of pruned itemsets is only 38% at zero tolerance (closed itemset case). For the same example, at a tolerance count of 50 (corresponding to a maximum error of 1.5% in itemset supports), *the percentage of pruned itemsets increases to 90%!*

The pruning achieved is significant even on the sparse datasets generated by the IBM Almaden generator. For example, on the T10I4D100K dataset (Figure 4b) for a minimum support of 0.1%, the percentage of pruned itemsets is only 12% at zero tolerance, whereas it increases to 41.5% at a tolerance count of 10 (corresponding to a maximum error of 0.01% in itemset supports).

An interesting trend that we notice in all cases is that the percentage of pruned itemsets increases dramatically at low tolerances and then plateaus as the tolerance is increased further. This trend is significant as it indicates that the maximum benefit attainable using the g-closed itemset framework is obtained at low tolerances. The reason for this trend is as follows:

As the length of an itemset X increases, its subsets increase exponentially in number. This means that the chance of one of the subsets being ϵ-equal to one of its subsets becomes exponentially high. Hence most of the long generators get pruned at low tolerances itself. This accounts for the initial steep rise in the curve. Shorter generators get pruned at a slower pace with regard to the increase in tolerance. This accounts for the gradual upward slope in the curve after the initial exponential increase.

5.2 Experiment 2: Response Time Reduction

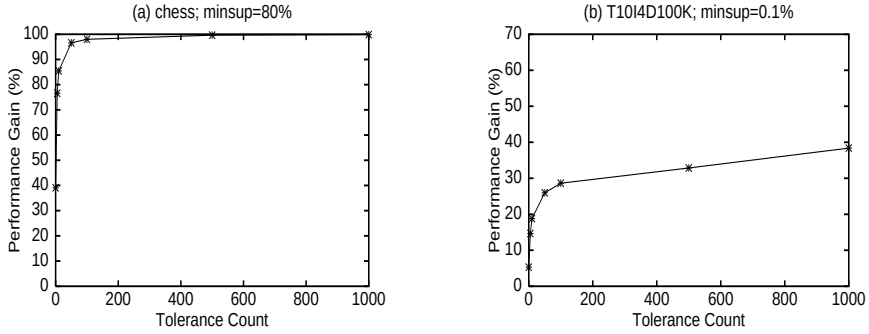

Figure 5: **Response Time Reduction**

In our second experiment, we measure the performance gain obtained from the g-closed itemset framework. This is measured as the percentage reduction in response time of g-Apriori over Apriori. The results of this experiment are shown in Figures 5a–b. The x-axis in these graphs represents the tolerance count values, while the y-axis represents the performance gain of g-Apriori over Apriori.

In all these graphs, we see that the performance gain of g-Apriori over Apriori is significant. In fact, the curves follow the same trend as in Experiment 1. This is expected because the bottleneck in Apriori (and other frequent itemsets mining algorithms) lies in the counting of the supports of candidates. Hence any improvement in pruning would result in a corresponding reduction in response-time.

5.3 Experiment 3: Response Times of g-ARMOR

In our third experiment, we measure the response times of g-ARMOR and compare them against those of Apriori. The results of this experiment are shown in Figures 6a–b. The x-axis in these graphs represents the tolerance count values, while the y-axis (plotted on a **log-scale**) represents response times in seconds.

In all these graphs, we see that the response times of g-ARMOR are over an *order of magnitude* faster than Apriori. We also notice that the response times become faster with an increase in tolerance count values. As in Experiment 2, this is expected because more candidates are pruned at higher tolerances. The reduction in response time is not

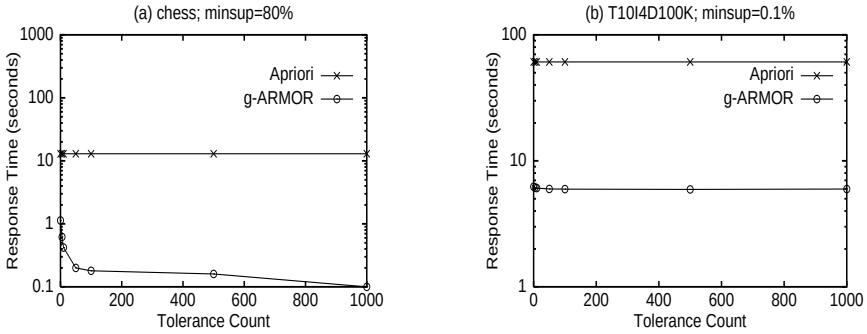

Figure 6: **Response Times of g-ARMOR**

as steep as in Experiment 2 due to the fact that g-ARMOR is much more efficient than g-Apriori and hence less responsive to a change in the number of candidates.

We do not show the response times of ARMOR in these graphs since it ran out of main memory for most of the datasets and support specifications used in our evaluation. This was because most of these datasets were dense, whereas ARMOR, as described in [14], is designed only for sparse datasets and is memory intensive.

5.4 Experiment 4: Scale-up Experiment

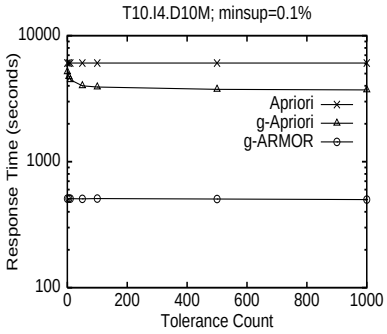

Figure 7: **Scale-up Experiment**

In our fourth (and final) experiment, we studied the scalability of g-ARMOR, g-Apriori and Apriori by measuring their response times for the T10I4D10M database having 10 million records. The results of this experiment are shown in Figure 7. These results (when analyzed along with Figures 5b and 6b) show that the performances of all three algorithms are linear w.r.t. database size. This behaviour is due to the following reasons: (1) The number of database passes in these algorithms depends only on the pattern density and not on the number of transactions. (2) The rate at which transactions are processed in each pass does not depend on the number of transactions but only on the

distribution from which the transactions are derived, the number of candidate itemsets and on the efficiency of the data-structure holding the counters of candidates.

6 Related Work

A number of *post-mining schemes* to discover "redundancy" in association rules have been proposed – [1, 3, 6, 9]. These schemes are to be applied *after* frequent itemsets have been mined. They are therefore inefficient and sometimes even infeasible because the number of frequent itemsets could be very large, especially for dense databases.

Techniques for pruning uninteresting rules during mining have been previously presented in [16, 15, 9, 6, 8, 5, 4]. In most of these studies (other than those following the closed itemset approach), it is sufficient for a rule to be considered uninteresting or redundant if it has no additional predictive power over another rule with fewer items. Techniques based on the closed itemset approach [16, 15], on the other hand, have a tighter requirement for a rule to be considered redundant: A rule is redundant only if its identity and support can be derived from another "non-redundant" rule.

In this paper, we follow the tighter approach. However, as mentioned in the Introduction, we relax the requirement of deriving exact supports – instead, it is sufficient if the supports can be estimated within a deterministic user-specified tolerance factor. This strategy of relaxing the requirement of deriving exact supports has also been considered in [5, 4]. In [5], the authors develop the notion of *freesets* along with an algorithm called *MINEX* to mine them. The bound on approximation error in the freesets approach increases linearly with itemset length in contrast to the constant bound featured in our approach. In [4], the authors do not provide any bounds on approximation error. Further, the focus in [5, 4] is only on highly correlated, i.e. "dense" data sets, whereas we show that our techniques can be profitably applied even on sparse data sets. Finally, there was no attempt in [5, 4] to incorporate their scheme into two-pass mining algorithms, which as mentioned earlier is essential for mining data streams.

The A-Close algorithm [11] for mining frequent closed itemsets is a levelwise algorithm based on Apriori. g-Apriori significantly differs from A-Close (even for the zero tolerance case) in that it does not require an additional database scan to mine closed itemsets from their respective generators. This is achieved by utilizing the technique described in Section 2.2. Our technique also bypasses the additional processing that is required in the MINEX [5] algorithm to test for "freeness".

7 Conclusions

In this paper we proposed the generalized closed itemset framework (or g-closed itemset framework) in order to manage the information overload produced as the output of frequent itemset mining algorithms. This framework provides an order of magnitude improvement over the earlier closed itemset concept. This is achieved by relaxing the requirement for exact equality between the supports of itemsets and their supersets. Instead, our framework accepts that the supports of two itemsets are equal if the difference between their supports is within a user-specified tolerance factor. We also

presented two algorithms – g-Apriori (based on the classical levelwise Apriori algorithm) and g-ARMOR (based on a recent two-pass mining algorithm) for mining the frequent g-closed itemsets. g-Apriori is shown to perform significantly better than Apriori solely because the frequent g-closed itemsets are much fewer than the frequent itemsets. Finally, g-ARMOR was shown to perform over an order of magnitude better than Apriori over all workloads used in our experimental evaluation.

References

[1] C. Aggarwal and P. Yu. Online generation of association rules. In *Intl. Conf. on Data Engineering (ICDE)*, February 1998.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of Intl. Conf. on Very Large Databases (VLDB)*, September 1994.

[3] R. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Intl. Conf. on Data Engineering (ICDE)*, February 1999.

[4] J-F. Boulicaut and A. Bykowski. Frequent closures as a concise representation for binary data mining. In *Pacifi c-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, April 2000.

[5] J-F. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of frequency queries by means of free-sets. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, September 2000.

[6] G. Dong and J. Li. Interestingness of discovered association rules in terms of neighborhood-based unexpectedness. In *Pacifi c-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 1998.

[7] C. Hidber. Online association rule mining. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 1999.

[8] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Intl. Conf. on Information and Knowledge Management (CIKM)*, November 1994.

[9] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered association rules. In *Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, August 1999.

[10] G. Manku and R. Motwani. Approximate frequency counts over streaming data. In *Proc. of Intl. Conf. on Very Large Databases (VLDB)*, August 2002.

[11] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of Intl. Conference on Database Theory (ICDT)*, January 1999.

[12] J. Pei et al. H-mine: Hyper-structure mining of frequent patterns in large databases. In *Intl. Conf. on Data Mining (ICDM)*, December 2001.

[13] V. Pudi and J. Haritsa. Generalized closed itemsets: Improving the conciseness of rule covers. Technical Report TR-2002-02, DSL, Indian Institute of Science, 2002.

[14] V. Pudi and J. Haritsa. On the efficiency of association-rule mining algorithms. In *Pacifi c-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, May 2002.

[15] R. Taouil, N. Pasquier, Y. Bastide, and L. Lakhal. Mining basis for association rules using closed sets. In *Intl. Conf. on Data Engineering (ICDE)*, February 2000.

[16] M. J. Zaki. Generating non-redundant association rules. In *Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, August 2000.

[17] M. J. Zaki and C. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SIAM International Conference on Data Mining*, 2002.