Mining and Validating Localized Frequent Itemsets with Dynamic Tolerance[*]

Olfa Nasraoui[†] Suchandra Goswami[‡]

Abstract

We cast the frequent itemset mining problem as a criterion guided optimization problem instead of one based on exact counting. This opens several interesting possibilities, including modification of the criterion function to take into account (i) error tolerance, (ii) locality, (iii) unsupervised estimation of the error tolerance, and (iv) search strategy. We also propose a new validation procedure that takes into account the completeness and accuracy of the discovered patterns. Experiments with real Web transaction data are presented.

keywords: frequent itemsets, association rules, frequent patterns, error tolerant itemsets, market basket analysis.

1 Introduction

Association rule mining [2] is a key data mining task that is most widely applied to problems in market basket analysis. Mining *frequent itemsets* is a crucial prerequisite step to mining association rules, a crucial task in some Web usage mining systems that discover association rules to build prediction models for recommender systems. Frequent itemset or *Frequent Pattern* (*FP*) mining faces several challenges that have previously been identified. Most have addressed the issue of mining and maintaining association rules in very large databases [6] and in evolving databases [7,8]. In this paper we focus our attention on three *other* issues that directly address the FP definition, in particular in the context of Transactional DataBases (TDBs) with a *large number of items/dimensionality*, *sparse* data, and *heterogenous* data distributions. Such databases are very common in the context of *e-commerce transactions* on large Websites that offer a huge number of products. Such TDBs are also very common in the context of mining web *user clickstream data*, where user sessions are the transactions and URLs are the items. Other kinds of data that satisfy these characteristics occur in the context of mining large collections of *text documents*, where the documents play the role of transactions and keywords play the role of items. Sparse data sets suffer from the fact that even though there is a large number of items, the number of non-zero entries is a very small fraction of the total number of entries in the transaction matrix.

1.1.1 Sensitivity to Support Thresholds

Very low support thresholds typically lead to generating too many spurious patterns (that are due to random correlations in data), while high support thresholds risk missing many interesting patterns that occur with low support, but have high confidence. This problem most particularly affects heterogeneous data sets, where certain itemsets may occur on only part of the data (e.g. in only some segments of a customer database depending on the geographical location), and hence will have low support. These itemsets cannot be discovered on the entire aggregate data, but have a better chance to be discovered by mining only the small portion of the data where they occur. One of the first researchers who have addressed this issue are Pei, Tung, and Han [4] who defined the notion of *Fault-Tolerant Frequent Patterns* (*FTFP*). Unlike frequent itemsets, FTFPs allow a fault tolerance equal to δ, meaning that *up to δ mismatches* in the items are allowed. So instead of finding exact patterns in data the search is for approximate and more general fault-tolerant patterns. Fault tolerant frequent pattern mining is an extension of mining frequent itemsets. Unlike frequent itemsets, FTFPs allow a fault tolerance of δ, that is upto δ mismatches in the items is allowed. So instead of finding exact patterns in data the search is for approximate and more general fault-tolerant patterns. Unfortunately, this approach actually requires *two* instead of *one* support threshold: one for the items, and another one for the itemsets. Moreover, an *additional* threshold is required for the amount of tolerance, δ.

1.1.2 Error Tolerance

When counting the support of an itemset, only transactions that completely include *all* the items of an itemset are counted. If a transaction matches an itemset in say 10 items, but fails to match *one* item, then it is completely excluded from the support count. In other words, this is an *all or nothing* counting.

Some work has addressed this issue, including Fault-Tolerant Frequent Pattern (*FTFP*) [4], and Error-tolerant Itemsets (*ETI*) [5] of two types (*strong* and *weak*). The problem with this approach is that getting away from having to pre-specify support thresholds led to getting trapped in another requirement: having to specify tolerance thresholds.

1.1.3 Locality

Data may be skewed, heterogeneous, or better modeled by several subsets, each with its own frequent itemsets, possibly with *different support* thresholds and even different levels of *tolerance*. Researchers who have addressed this issue include Aggarwal, Procopiuc, and Yu [1] who have proposed CLASD (**CL**ustering for **AS**sociation **D**iscovery) that discovers frequent patterns called *metatransactions* by aggregating the item frequencies of transactions assigned to each clusters based on maximum similarity into a frequency vector. The clusters are found using a Hierarchical Agglomerative Clustering that starts with randomly selected transactions as the initial seeds.

While this approach , it still needs to address several problems: *(i)* sensitivity to prespecified number of clusters (k) and minimum size threshold of a cluster that implicitly plays the role of metatransaction support, *(ii)* also, since transactions are assigned

[*] supported by National Science Foundation CAREER Award IIS-0133948 to O. Nasraoui
[†] Department of Computer Engineering & Computer Science, University of Louisville, Louisville, KY 40292
[‡] Department of Computer Science, University of Central Florida, Orlando, FL-32816

based only on similarity, transactions that are not very similar to any cluster will still get lumped to the closest cluster, and hence contribute to its support. In other words, the notion of *cluster size* which is equivalent to *metatransaction support* does not take into account the *level* of similarity of the transaction. This corresponds to the *opposite* extreme end of exact/intolerant support counting because, even a transaction that does not match any of the items in the candidate pattern will be counted in the support.

1.2 Contributions of this Paper

We present an approach that addresses all the identified problems *simultaneously*, and in an *integrated* manner, namely:

(1) discovering *local* frequent patterns
(2) discovering frequent patterns that are *error tolerant*,
(3) Allowing tolerance to be *dynamically* estimated depending on the underlying *local* distribution.
(4) Performing the search *without pre-fixed support or tolerance threshold* requirements. Instead, support and tolerance are allowed to adapt to the local distribution of the data.

We call the special kind of patterns that we mine: ***L**ocalized **E**rror **T**olerant **F**requent **P**attern* (*LET-FP)*. We also propose an information retrieval inspired validation procedure that attempts to answer the following crucial question: Is the data set completely and accurately summarized/represented by the mined patterns?

2 A Generalized Framework for Localized Error-Tolerant Frequent Pattern (LET-FP) Mining

2.1 Frequent Itemsets: A Similarity Based Perspective

Frequent itemsets or frequent patterns (FPs) can be considered as one way to form a *summary* of the input data. As a summary, frequent patterns represent a reduced form of the data that is at the same time, *as close as possible* to the original input data. This is compatible with the notion of support as a critical measure of goodness for a FP. *Classical support* measures the *count* of the transactions that *completely include* a FP. Therefore transactions that are very similar to a FP, but perhaps lacking a single item from the FP do not even count in its support. The first step toward including *tolerance* is to allow transactions that are very similar to a FP to count in what we term a *partial* support. For this reason, we need to consider using a *similarity* measure to capture *closeness* between an FP and a transaction. We first list the notation that will be used throughout the rest of this section, below.

P_i	ith Frequent Pattern (FP) or itemset.
$\|P_i\|$	number of items in P_i
t_j	jth transaction.
$\|t_j\|$	number of items in t_j
S_{ij}	Similarity between the ith FP and the jth transaction.

An FP should represent a *frequently* occurring trend. Hence it should be as *similar* as possible to *as many* transactions as possible. Hence, we need to assess the similarity between an FP, P_i, and each transaction t_j. Below, we explore some similarity measures in the order of increasing complexity and error tolerance.

Total similarity

(1) $$S_{ij} = \begin{cases} 1 & if \quad t_j = P_i \\ 0 & otherwise \end{cases}$$

In this case, P_i is considered a valid FP *iff* it matches most of the transactions *exactly*, hence leaving no room for tolerance in this definition. This is an *even more stringent* (or less error-tolerant) requirement than *Apriori* [2].

Complete transaction inclusion based similarity

(2) $$S_{ij} = \begin{cases} 1 & if \quad t_j \subseteq P_i \\ 0 & otherwise \end{cases}$$

In this case, a transaction t_j will contribute to the support of pattern P_i, only if it is completely included in pattern P_i, thus favoring longer patterns even if most of P_i's items are not part of the transaction.

Complete FP inclusion based similarity

(3) $$S_{ij} = \begin{cases} 1 & if \quad P_i \subseteq t_j \\ 0 & otherwise \end{cases}$$

In this case the similarity is 1 if the transaction contains *all* the items from the FP. This corresponds to the case of the *Apriori* algorithm [2], where the frequent pattern has to be completely contained in the transaction, in order for this transaction to contribute to its support. This strategy favors the smallest frequent itemset (with fewer items). Hence it is the extreme opposite of the previous approach.

FTFP based transaction inclusion based similarity

(4) $$S_{ij} = \begin{cases} 1 & if \quad |P_i| - |t_j \cap P_i| \leq \delta \\ 0 & otherwise \end{cases}$$

In this case, a transaction t_j will contribute to the support of pattern P_i, even if it does not include all of P_i's items. In particular, in this case, $|P_i|$-$|P_i \cap t_j|$ items are missing. Without imposing any constraints on the number of missing items, it is clear that a pattern could grow as large as necessary to make each transaction contribute to its support. Hence the best itemset P_i will be the set of all items. This is why a constraint is imposed on the number of missing items as follows: $|P_i|$-$|P_i \cap t_j| \leq \delta$. This is the same condition that qualifies transaction t_j to *FT-contain* the pattern P_i [4]. In this case, δ is the fault tolerance, or equivalently, the error ratio is $\varepsilon = \delta / |P_i|$.

Precision and Coverage/Recall as similarity measures

A more general and flexible similarity measure would not require that either the FP or the transaction include the other. Instead, a considerable degree of overlap between the transaction and FP is considered as follows:

$S_{ij} = |t_j \cap P_i| / f(t_j, P_i)$ where $f(t_j, P_i)$ is a generic function that can be defined in a number of ways to arrive at various familiar metrics, such as

(5) $$f(t_j, P_i) = f_1(t_j, P_i) = |P_i|$$

(6) $$f(t_j, P_i) = f_2(t_j, P_i) = |t_j|$$

(7) $$f(t_j, P_i) = f_3(t_j, P_i) = |t_j|^{1/2} \, |P_i|^{1/2}$$

Using $f_1(t_j, P_i)$, S_{ij} reduces to the *precision* given by

$$Prec_{ij} = |\, t_j \cap P_i \,| / |\, P_i \,|.$$

Precision is biased towards smaller itemsets. It is used in the case of imposing support conditions on ETIs [11]. Using $f_2(t_j, P_i)$, S_{ij} reduces to the *coverage/recall* given by

$$Cov_{ij} = |\, t_j \cap P_i \,| \,/\, |\, t_j \,|.$$

Coverage is biased towards larger itemsets. Using $f_3(t_j, P_i)$, S_{ij} reduces to the cosine similarity. It is evident from the above that the cosine similarity can be re- written in the form $cos(P_i, t_j) = (Prec_{ij})^{1/2}. (Cov_{ij})^{1/2}$, which is proportional to the geometric mean of $Prec_{ij}$ and

Cov$_{ij}$. Hence it simultaneously takes into account Precision and Coverage.

Minimum of Precision and Coverage similarity measure (MinPC)

The most demanding similarity measure to optimize would be

(8) $$S_{ij}=min(Prec_{ij},Cov_{ij})$$

This similarity measure tries to achieve a balance between precision and coverage, while at the same time penalizing the situation in which either precision or coverage becomes too high at the expense of the other. In addition to the above similarity measures, other sophisticated measures that exploit the domain knowledge such as a concept hierarchy can be used. In particular, for Web sessions, a similarity that takes into account an implicit *Website hierarchy* was presented in [3], so that related items/URLs are considered similar to a certain degree, even if distinct.

2.2 Error Tolerant Support

Let a candidate ***L**ocalized **E**rror **T**olerant **F**requent **P**attern*, henceforth referred to as *LET-FP*, be denoted as P_i, and let the transactions in a DB be denoted by t_j. Instead of *Apriori's* complete pattern inclusion based matching, we propose to use a generalized, softer matching measure. This matching measure $Sim(P_i,t_j)$ quantifies how faithfully the frequent pattern P_i serves as a summary for transaction t_j . A dissimilarity $d(P_i,t_j)$ can be defined so that it is inversely related to $Sim(P_i, t_j)$, for example,

(9) $$d(P_i,t_j) = (1- Sim(P_i,t_j))^2$$

Let the amount of tolerance ε be dynamic and defined in the same units as $d(P_i,t_j)$. Furthermore, let the tolerance be *localized*, and hence depend on the ETFP itself, i.e

$$\varepsilon_i = \varepsilon(P_i).$$

Next, let a *tolerance-normalized* dissimilarity between ETFP P_i and transaction t_j be defined as

(10) $$d_\varepsilon(P_i, t_j,\varepsilon_i) = \frac{d(P_i,t_j)}{\varepsilon_i}$$

A lower tolerance will tend to inflate the effect of dissimilarity, hence reflecting a more stringent matching process. Now that the tolerance degree, ε_i has been "absorbed" into the normalized dissimilarity $d_\varepsilon(P_i, t_j,\varepsilon_i)$, a measure of *support* that is *error-tolerant* can be defined. Let this localized error-tolerant support be defined as

$$s(P_i,t_j,\varepsilon_i) = f(d_\varepsilon(P_i,t_j,\varepsilon_i)) = f(d(P_i,t_j), \varepsilon_i),$$

where f: $\Re \rightarrow [0,1]$ is a monotonically non-increasing function. The *Total Localized Error-Tolerant support* of LET-FP P_i may be defined by summing the contributions from all transactions as follows

(11) $$Ts(P_i,\varepsilon_i) = \sum_{t_j \in T} s(P_i,t_j,\varepsilon_i)$$

Note that the total support in (11) increases monotonically with the tolerance ε_i. Because tolerance is not known in advance, this will favor higher tolerance values. To put a limit on this bias, we define a "*normalized* support" instead of an absolute support, to be used as an FP goodness criterion, i.e,

(12) $$\rho(P_i,\varepsilon_i) = \frac{Ts(P_i,\varepsilon_i)}{\varepsilon_i}$$

In this equation the tolerance degree can also be considered as a penalty factor P_i. Given the same support, LET-FPs will be rewarded if their tolerance degree is smaller and penalized if their tolerance-degree is higher.

2.3 Avoiding Fixed Support Thresholds: Mining Frequent Patterns by Support *Optimization*

Instead of searching for the FPs that exceeds a *fixed* support threshold, we propose to seek the FPs that *maximize the error tolerant support* in (12). The FP mining and tolerance search problem can be stated as an *alternating optimization problem* that boils down to two optimization steps, to determine the frequent *patterns* and the error *tolerance* respectively that optimize the error tolerant support:

```
Step1:
- Fix $\varepsilon_i$
- Solve for $P_i = Arg\ Max\ (\rho(\mathbf{P_i},\varepsilon_i))$
Step2:
- Fix $P_i$
- Solve for $\varepsilon_i = Arg\ Max\ (\rho(P_i,\boldsymbol{\varepsilon_i}))$
```

Step 2 can be solved by *analytical* optimization if $\rho(P_i,\varepsilon_i)$ is *differentiable* with respect to ε_i, and a closed *Piccard update equation* can be derived. See Eqs (13), (14), and (15) below. However $\rho(P_i,\varepsilon_i)$ is not in a form that is differentiable with respect to P_i. Therefore, a *non-analytical* optimization approach is needed for Step 1. We use a Genetic algorithm for this purpose, but do not rule out other heuristic optimization methods.

2.4 Localized ETFP Mining by Partitioning and Zooming

Some strong (i.e. highly accurate/confident) associations may lurk in small segments of a huge data set. In this case, they risk being missed because of their low support. In other words, finding frequent itemsets from the entire aggregate data may not be able to reveal patterns that are only valid in *small localized* segments of the data. Data *locality* concepts can offer several advantages in this context. We achieve locality by gradually focusing the search on smaller and smaller segments of the data set. A greedy procedure extracts the unique optimal patterns discovered in each iteration. Redundant patterns are identified based on their compatibility with a previously extracted pattern, and are therefore ignored.
Based on these extracted FPs, the dataset is gradually divided into smaller parts/clusters containing similar transactions. The steps needed to obtain localized LET-FPs may be summarized as follows:

```
Algorithm LET-FP Mining:
0. Let current transaction dataset ($D_c$) = $D$ (input data), and let the set of
   final extracted LET-FPs, $P=\varnothing$
1. Initial FP-Generation: Seed the FPs by selecting random samples from
   current transaction dataset ($D_c$).
2. Iterative LET-FP search and extraction: will result in $C$ LET-FPs $P_1,\ldots,$
   $P_c$, and tolerance values $\varepsilon_1,\ldots, \varepsilon_c$
3. Partition transactions by assigning each transaction to nearest LET-FP
   (based on one of the dissimilarity measures in Eqs (1-8). This will partition
   the dataset into $C$ subsets $T_1,\ldots,T_c$.
4. For $i = 1, \ldots, C$ {      // Step 4 is for zooming (optional)
      Let $T_i^{out-of-core} = \{t_j \mid s(P_i,t_j,\varepsilon_i) < s_{core}\}$
   Let $T_i = T_i - T_i^{out-of-core}$
   Let $T_{Zoom} = \bigcup_i T_i^{out-of-core}$
}
5. For each subset $T_i$ {
    If $\varepsilon_i>\varepsilon_{max}$ and $|T_i| > t_{max}$ Then {
   Let current transaction dataset $D_c = T_i$ .
   Go to step 1,      // repeat search on each cluster
    }
    Else $P=P\cup P_i$        // Add to final list of LET-FPs
}
6. Let current dataset $D_c = T_{Zoom}$. Go to Step 1. // Step 6 is for zooming
   (optional)
```

Step 2 can be any competent search method, preferably, one that is global, and that can benefit from randomized search to sample the huge search space of all possible LET-FPs, such as a genetic algorithm.

2.5 Doing Away with Tolerance Thresholds by Dynamic Tolerance Optimization

The normalized error-tolerant support in (12) satisfies several desiderata.
- **Localized Support:** Support is defined on increasingly smaller subsets/clusters of the data, providing a *localized* and confined counting.
- **Error-tolerance:** Data tuples that deviate slightly from candidate LET-FP will still contribute to its support, though to a lesser degree, depending on the tolerance amount.
- **Dynamic Tolerance:** Given the local support measure function

(13) $$s(P_i,t_j,\varepsilon_i) = f(d_\varepsilon(P_i,t_j,\varepsilon_i)) = e^{-d_\varepsilon(P_i,t_j,\varepsilon_i)},$$

we can analytically derive an iterative update equation for dynamic tolerance level ε_i based on optimizing the total error-tolerant support given by (12). For this purpose we set

(14) $$\nabla\rho(P_i,\varepsilon_i)/\nabla\varepsilon_i = 0$$

This can be shown to result in

(15) $$\varepsilon_i^{(t+1)} = \frac{\sum_{t_j \in T_i} f^{(t)}(d_\varepsilon(P_i,t_j,\varepsilon_i)d(P_i,t_j)}{\sum_{t_j \in T_i} f^{(t)}(d_\varepsilon(P_i,t_j,\varepsilon_i)}.$$

The main assumptions are in fixing $f(d_\varepsilon(P_i,t_j,\varepsilon_i))$ from the previous iteration (t) to derive the new $\varepsilon_i^{(t+1)}$.

2.6 Validation in an Information Retrieval Context

Frequent itemsets or patterns (FP) can be considered as one way to form a summary of the input data. As a summary, frequent patterns represent a *reduced* form of the data that is at the same time, *as close as possible* to the original input data. This description is reminiscent of an *information retrieval* scenario, in the sense that patterns that are retrieved should be as *close* as possible to the original transaction data. Closeness should take into account both *(i) precision* (a summary FP's items are all correct or included in the original input data, i.e. they include *only* the true data items) and *(ii) coverage/recall* (a summary FP's items are complete compared to the data that is summarized, i.e. they include *all* the data items). These criteria are clearly contradictory, since precision will favor only the smallest itemsets, eventually 1-itemsets, while coverage will favor the longest possible itemsets. Ideally, for perfect retrieval, *each data query should be answered by an FP that is identical to this query*. However, this is unrealistic since it corresponds to the case where the LET-FPs summary is identical to the entire input database. Therefore, it is imperative that the summary consist of the *smallest number* of LET-FPs that are *as similar as possible* to the input data. We propose a validation procedure that attempts to answer the following crucial questions: Is the data set *(a) completely* and *(b) faithfully* summarized/represented by the mined patterns? Each of the previous questions is answered by computing *coverage/recall* as an *Interestingness measure to answer part (a),* and *precision as an Interestingness measure to answer part (b)*.

First, we compute the following *Interestingness* measures for each LET-FP, letting the *Interestingness measure*, $Int_{ij} = Cov_{ij}$ *(i.e., coverage:* See Eqs. In Sec 3.1.) *to answer part (a),* and $Int_{ij} = Prec_{ij}$ *(i.e., precision:* See Eqs. In Sec 3.1.) *to answer part (b)*.

Now, if we let $T^* = \{t_j \mid \underset{i}{Max}\ (Int_{ij}) > Int_{min}\}$. Then

(17) $$Int^T = |T^*| / |\mathrm{T}|$$

When $Int_{ij} = Cov_{ij}$, we call Int^{T} the *Cumulative Coverage of Transactions*, and it answers Question a. When $Int_{ij} = Prec_{ij}$, we call Int^{T} the *Cumulative Precision of Transactions*, and it answers Question b. The above measures are computed for the different techniques over the entire range of the Interestingness threshold Int_{min} from 0% to 100% in increments of 10%, and compared.

2.7 LET-FP Search and Post-processing Options

After the completion of the LET-FP search algorithm, we partition the input transactions into as many clusters as the number, say $|P|$, of the *original* (i.e., *without post-processing*) LET-FPs, $P=\{P_1,\ldots, P_{|P|}\}$. Let these transaction clusters be denoted as $T_1,\ldots, T_{|P|}$. Then, there are several ways that we may use the LET-FPs, as listed below.

Search Options: First the search for LET-FPs can either use zooming or not.

(1) *Standard* LET-FPs: obtained by eliminating steps 4 and 6 in Algorithm LET-FP Mining (see Sec 2.4) and doing *no* post-processing
(2) *Zoomed* LET-FPs: We use steps 4 and 6 in Algorithm LET-FP Mining (see Sec 2.4) to gradually zoom into each transaction cluster by peeling off the out-of-core transactions.

Post-Processing Options: After completing the search for LET-FPs, there are several options:

(1) *Original* LET-FPs: These are the LET-FPs obtained *without* post-processing
(2) *Aggregate* LET-FPs: Frequency vectors computed by averaging the item occurrence frequencies in each cluster separately, then converting to a binary vector (1 if frequency > 0.10, 0 otherwise).
(3) *Robustified* LET-FPs: Before aggregating the LET-FP frequencies as in the previous option, we zoom into each cluster, and remove the out-of-core transactions, i.e, $T_i = T_i - \{t_j \mid s(P_i,t_j,\varepsilon_i) < s_{core}\}$.

Other options are produced by different combinations of search and post-processing options. Table 1 lists the different codes used in our experimental section that designate these different options

Table 1. Category codes corresponding to LET-FP search and post-processing options

Code	**search**	**post-processing**
spa	**S**tandard (i.e., no zooming)	**p**ost-processing**: a**ggregate
spr	**S**tandard (i.e., no zooming)	**p**ost-processing**: r**obustified
so	**S**tandard (i.e., no zooming)	**O**riginal (no post-processing)
zpa	**Z**oomed (w/ steps 4 & 6 of LET-FP Mining Algorithm)	**p**ost-processing**: a**ggregate
zpr	**Z**oomed (w/ steps 4 & 6 of LET-FP Mining Algorithm)	**p**ost-processing**: r**obustified
zo	**Z**oomed (w/ steps 4 & 6 of LET-FP Mining Algorithm)	**O**riginal (no post-processing)

3 Experimental Results

Experimental results are obtained by using the LET-FP Mining Algorithm described in Sec. 2.4, and we compare against the performance of *APriori* [2] with varying minimum support levels. However only the results with lowest support are shown since as expected, they result in the best quality for APriori. The proposed LET-FP Mining algorithm is validated using the different *search* strategies (with or without zooming) and different *similarity* measures. Hence, we validate all the possible combinations listed in Sec. 2.7, as well as the different similarity measure options by computing and plotting the interestingness measures described in Sec. 2.6. To avoid information overload, for each [search & post-processing] category, we report the results *only for the best performing similarity measure*. Also because of the definition of frequent itemsets in *APriori*, precision is always equal to 1. Hence we list it in tabular format, considering it as threshold of 0.9. To optimize step 1, we use a GA with population size 100, 30 generations, and binary encoding of transactions. The crossover and mutation rates were 0.9 and 0.001 respectively.

3.1 Description of the Web Transaction Data

The first dataset consists of the preprocessed web-log data of a Computer science department website. This dataset has 343 distinct URLs accessed by users. A session may be defined as a sequence of URL accesses from the same IP address within a prespecified time threshold [3]. There are a total of 1704 real user sessions. Since each URL address can be mapped to a unique integer index starting from 0 to 343, the entire dataset can be modeled as a 1704 by 343 binary matrix where the presence of an URL in a session may be encoded as 1, and its absence encoded as 0.

3.2 Results for Web Transaction Data

On this data set, *APriori* [2] generates a large number of itemsets, despite the conservative minimum support thresholds, as shown in Tables 2-3. The proposed LET-FP Mining algorithm produces a much smaller number of frequent patterns as shown in Tables 2-3 for the different search strategies (with or without zooming) and different similarity measures.

Table 2 shows the percentage of sessions well covered by FPs using the best similarity measure for LET-FPs and the best support percentage for *Apriori*. Here again, the LET-FPs perform much better than the Frequent Patterns obtained by the *Apriori* algorithm.

Table 2: Percentage of sessions well covered (Coverage > Threshold) by the best Frequent Patterns (Web transaction data)

	Threshold >=0.5	**Threshold >=0.9**
***Apriori* (support = 1%)**	9.22 % (531 itemsets)	3.908 % (531 itemsets)
so (sim: coverage)	88.086854% (19 LET-FPs)	84.565728% (19 LET-FPs)
spa (sim: web hierarchy)	92.077465% (24 LET-FPs)	**88.321596% (24 LET-FPs)**
spr (sim: precision)	89.906103% (31 LET-FPs)	87.734742% (31 LET-FPs)
zo (sim: coverage)	90.316901% (22 LET-FPs)	**88.849765% (22 LET-FPs)**
zpa (sim: precision)	87.969484% (26 LET-FPs)	85.974178% (26 LET-FPs)
zpr (sim: precision)	89.319249% (26 LET-FPs)	87.5% (26 LET-FPs)

Table 3 shows the percentage of sessions precisely retrieved by FPs using the best similarity measure for LET-FPs. It can be seen that the *zpa-LET-FPs* using coverage similarity measure gives the best results.

Table 3: Percentage of sessions precisely retrieved (Precision > Threshold) by the best Frequent Patterns

	Threshold >=0.5	**Threshold >=0.9**
***Apriori* (support = 1%)**	-	32.466 % (531 itemsets)
so (sim: coverage)	90.082160% (19 LET-FPs)	89.260563% (19 LET-FPs)
spa (sim: coverage)	100% (19 LET-FPs)	**100% (19 LET-FPs)**
spr (sim: cosine)	86.85446% (31 LET-FPs)	86.85446% (31 LET-FPs)
zo (sim: cosine)	82.570423% (31 LET-FPs)	82.570423% (31 LET-FPs)
zpa (sim: coverage)	100% (22LET-FPs)	**100% (22 LET-FPs)**
zpr (sim: precision)	88.497653% (26 LET-FPs)	88.497653% (26 LET-FPs)

3.3 Summary of Experimental Results

The FPs generated by *Apriori* do not perform better than the LET-FPs obtained by the proposed approach, on any of the chosen validation measures.
- The percentage of sessions/transactions well covered by any of the FPs obtained by *Apriori* is less than the % of sessions/transactions well covered by any of the LET-FPs obtained by our proposed approach.
- The percentage of sessions/transactions precisely retrieved by any of the FPs obtained by *Apriori* is less than or equal to the one retrieved by any of the LET-FPs obtained by the proposed approach.
- The FPs obtained by *Apriori* are larger in *number* but poorer in *quality* (see Tables 2-3).
- The LET-FPs obtained by the proposed method serve as good FPs at no additional cost. There is no problem of prespecified, fixed, global support. We obtain the optimal number of good quality LET-FPs that are comparable and better than the FPs obtained by traditional methods.
- The proposed LET-FP Mining algorithm takes roughly half the time compared to *APriori*.
- When no post-processing is done, z*ooming* has the effect of increasing coverage as shown in Table 2 (compare patterns *zo* to *so* as per Table 1) because it succeeds in discovering *more, even small localized LET-FPs*. *More* itemsets means *more complete* summary or in other words, better coverage. However, despite the fact that coverage is increased significantly, especially compared to *APriori*, *precision is not significantly affected*. In particular, precision remains higher than *APriori* in most cases. Zooming achieves this desired goal because it acts by peeling all the *out-of-core* transactions that are *not at the core* of each pattern's cluster, and then *repeating* the mining on this smaller out-of-core set.
- It is hard to compare the effect of the different similarity measures, perhaps because of the interaction between similarity and method of search and or post-processing (such zooming and robustification of patterns). However, we note that while each similarity measure is tailored to optimize its own interestingness, the *MinPC similarity* tries to optimize *both precision and coverage*, suggesting it as a potentially useful similarity measure, when one looks for a compromise.

4 Conclusions

We presented an approach addressing several problems in an *integrated* manner: *(i)* discovering *local* frequent patterns, *(ii)* discovering frequent patterns that are *error tolerant*, *(iii)* allowing tolerance to be *dynamically* estimated depending on the underlying *local* distribution, and *(iv)* performing the search *without pre-fixed support or tolerance threshold* requirements. We have also presented an information retrieval inspired *validation* procedure that measures the accuracy and completeness of discovered patterns. We have proposed zooming as an efficient search strategy, and used the *MinPC* similarity to satisfy *both* of the contradictory precision and coverage based interestingness measures. Other important issues such as scalability were not addressed in this work, but could be addressed in the future for example along the lines of [6].

References

[1] C. Aggarwal, C. Procopiuc, and P. Yu. Finding Localized associations in market basket data. IEEE Trans. Knowledge and Data Engineering, Vol 14, No. 1, Jan 2002.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conf. on Very Large Databases*, SanTiago, Chile, June 1994.

[3] O. Nasraoui, R. Krishnapuram, and A. Joshi. Mining Web Access Logs Using a Relational Clustering Algorithm Based on a Robust Estimator, 8th International World Wide Web Conference, Toronto, pp. 40-41, 1999.

[4] J. Pei, A.K.H. Tung, and J. Han, Fault tolerant frequent pattern mining: Problems and challenges, Proc. 2001 ACM-SIGMOD Int. Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'01), Santa Barbara, CA, May 2001.

[5] C. Yang, U. Fayyad, and P. Bradley, Efficient Discovery of error-tolerant frequent itemsets in high dimensions, In Proc. of seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 194-203, San Francisco, California, Aug. 2001.

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

[7] A. Veloso,W. Meira Jr., M. B. de Carvalho, B. Possas, S. Parthasarathy, and M. Zaki. Mining frequent itemsets in evolving databases. In *Proc. of the 2 SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.

[8] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Mining and monitoring evolving data. In *Proc. of the 16 Int'l Conf. on Data Engineering*, pp. 439–448, May 2000.