Quantitative Evaluation of Approximate Frequent Pattern Mining Algorithms

Rohit Gupta
Dept of Comp Sc and Engg
Univ of Minnesota, Twin Cities
Minneapolis, MN USA
rohit@cs.umn.edu

Gang Fang
Dept of Comp Sc and Engg
Univ of Minnesota, Twin Cities
Minneapolis, MN USA
gangfang@cs.umn.edu

Blayne Field
Dept of Comp Sc
Univ of Wisconsin, Madison
Madison, WI USA
bfield@cs.wisc.edu

Michael Steinbach
Dept of Comp Sc and Engg
Univ of Minnesota, Twin Cities
Minneapolis, MN USA
steinbac@cs.umn.edu

Vipin Kumar
Dept of Comp Sc
Univ of Minnesota, Twin Cities
Minneapolis, MN USA
kumar@cs.umn.edu

ABSTRACT

Traditional association mining algorithms use a strict definition of support that requires every item in a frequent itemset to occur in each supporting transaction. In real-life datasets, this limits the recovery of frequent itemset patterns as they are fragmented due to random noise and other errors in the data. Hence, a number of methods have been proposed recently to discover approximate frequent itemsets in the presence of noise. These algorithms use a relaxed definition of support and additional parameters, such as row and column error thresholds to allow some degree of "error" in the discovered patterns. Though these algorithms have been shown to be successful in finding the approximate frequent itemsets, a systematic and quantitative approach to evaluate them has been lacking. In this paper, we propose a comprehensive evaluation framework to compare different approximate frequent pattern mining algorithms. The key idea is to select the optimal parameters for each algorithm on a given dataset and use the itemsets generated with these optimal parameters in order to compare different algorithms. We also propose simple variations of some of the existing algorithms by introducing an additional post-processing step. Subsequently, we have applied our proposed evaluation framework to a wide variety of synthetic datasets with varying amounts of noise and a real dataset to compare existing and our proposed variations of the approximate pattern mining algorithms. Source code and the datasets used in this study are made publicly available.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*; I.5 [**Pattern Recognition**]: Miscellaneous

General Terms

Algorithms, Experimentation, Performance, Verification

Keywords

Association analysis, approximate frequent itemsets, error tolerance, quantitative evaluation

1. INTRODUCTION

Traditional association mining algorithms use a strict definition of support that requires every item in a frequent itemset to occur in each supporting transaction. In real-life datasets, this limits the recovery of frequent itemset patterns as they are fragmented due to random noise and other errors in the data.

Motivated by such considerations, various methods [11, 7, 8, 6, 5, 2] have been proposed recently to discover approximate frequent itemsets (often called error-tolerant itemsets (ETIs)) by allowing itemsets in which a specified fraction of the items can be missing. Please see figure 1 for a conceptual overview. The most basic approach is to require only that a specified fraction of the items in a collection of items and transactions be present. However, such a 'weak' ETI [11] provides no guarantees on the distribution of the items within this 'block,' i.e., some rows or columns could be completely empty. To address this issue, a 'strong' ETI was defined [11], which required that each row must have at most a specified fraction of items missing. The support of strong ETIs is simply the number of transactions that support the pattern, as in the traditional case, but support does not have the anti-monotone property, i.e., support can increase as the number of items increases. Thus, a heuristic algorithm for finding strong ETIs was developed.

Indeed, the lack of an anti-monotone support measure is one of the factors that has made the construction of algorithms for finding approximate itemsets very challenging. One solution that leads to an anti-monotone support measure is to allow only a fixed number of missing items in each row [7]. This approach does not enforce any constraint on the number of items missing in a column, and is unappealing in that bigger itemsets should be allowed more missing items than the smaller ones. Another potential solution is an approach that uses 'dense' itemsets [8]. The support of an approximate itemset is defined in terms of the minimum support of every subset and is anti-monotone, but one may argue whether the definition of an approximate pattern used by this approach is as appealing as some of the other definitions since different subsets of items may be supported by different transactions.

Both strong and weak ETI patterns can have empty columns. To handle this situation, approximate frequent itemsets (AFI) [5] were proposed. AFIs enforce constraints on the number of missing items in both rows and columns. One of the advantages of AFIs over weak/strong ETIs is that there is a limited version of an anti-monotone property that helps prune the search space. However, this algorithm cannot guarantee that no AFIs are overlooked since a heuristic approach is used to verify the column constraint. Although AFIs might seem like the most natural and complete definition of an approximate itemset pattern, one might argue that it is easier and perhaps more meaningful to find approximate itemsets if at least some transactions contain all the items comprising an approximate itemset. This formed the motivation for AC-CLOSE, which finds AFIs based on the notion of core itemsets [2].

Although these approaches have shown to be successful in finding the approximate frequent itemsets, a systematic and quantitative approach to evaluate the patterns they generate is still lacking. Indeed, most of the papers on approximate itemsets have provided only limited comparisons with other work: The paper that introduced AFIs [5] only presented a comparison to strong ETIs [11], while AC-CLOSE [2] only compared against AFIs [6, 5].

Contributions of this paper:

- We perform a comprehensive evaluation of algorithms for finding approximate itemsets. Building on the initial work by others, we propose an evaluation framework to compare various frequent pattern mining algorithms. We apply our evaluation framework to compare different approximate pattern mining algorithms based on their robustness to the input parameters ($minsup$, ϵ_r, and ϵ_c) and on the quality of the patterns (measured in terms of significance and redundancy).
- Our work highlights the importance of choosing optimal parameters. In general, approximate pattern mining algorithms use various parameters and given that each parameter can be set to many different values, the performance of various algorithms can be greatly affected. Thus, it is more reasonable and fair to compare each algorithm's performance using its own optimal parameters in the parameter space.
- We propose simple variations of the existing algorithms proposed in [11] and [8] by adding an additional step to check if each candidate approximate pattern also satisfies the column error threshold (ϵ_c). These modified algorithms not only generate high quality patterns but also compare well with those generated by 'AFI' ([5]) and 'AC-CLOSE' ([2]).

The source codes of all the algorithms as well as synthetic and real data sets used in this study are available at the following website: `www.cs.umn.edu/∼kumar/ETI/`.

Organization: The remainder of the paper is organized as follows. In Sections 2, we briefly review the existing algorithms for approximate frequent pattern mining and we also propose simple variations of some of them. Section 3 gives the details of our proposed evaluation methodology, including the definition of measures and how to select optimal parameters for each algorithm on a given dataset. We discuss our experimental results in Section 4 and conclude with final remarks and future work in Section 5.

2. APPROXIMATE FREQUENT PATTERN MINING ALGORITHMS

Below, we briefly define different generalizations of frequent itemsets, which can generally be viewed as a hierarchy as shown in Figure 1. It is important to note that each of these definitions

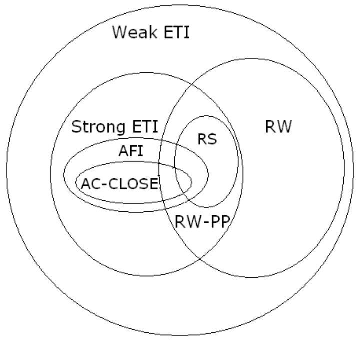

Figure 1: A conceptual hierarchy of approximate patterns/algorithms

will lead to a traditional frequent itemset if their error-tolerance is set to 0. However, first we define a set of common terminology. Assume, we have a binary transaction database consisting of a set I=$\{i_1, \ldots, i_m\}$ of items, and a set T=$\{t_1, \ldots, t_n\}$ of transactions, where each t_i has a subset of items from I. It is useful to think of such a database as a n-by-m binary matrix D, with each column corresponding to an item, and each row corresponding to a transaction. Thus $D_{j,k} = 1$ if $i_k \in t_j$, and 0 otherwise. An itemset (or pattern) I' is said to be a frequent itemset if its support (the number of transactions it appears in, $|\{T' : I' \subset T'\}|$), is more than some user-specified threshold denoted by $minsup$.

2.1 Error Tolerant Itemsets (ETIs) - GETI

The concept of weak and strong ETI was defined in [11]. An itemset I' at support $minsup$ is said to be a weak ETI with tolerance ϵ if $\exists T' \in T$ such that $|T'| \geq minsup$ and the following condition holds: $\frac{\sum_{i \in I'} \sum_{t \in T'} D_{t,i}}{|I'| * |T'|} \geq 1 - \epsilon$

It is difficult to find all weak ETIs, since we effectively have to search the entire pattern space without any pruning. Also, we can have both rows and columns included that consist entirely of zeros, since there is no constraint as to where the 0's can occur within the itemset. An itemset I' at support $minsup$ is said to be a strong ETI with tolerance ϵ if $\exists T' \in T$ such that $|T'| \geq minsup$ and the following condition holds $\forall t \in T'$: $\frac{\sum_{i \in I'} D_{t,i}}{|I'|} \geq 1 - \epsilon$

This implies that for a given set of parameters, any strong ETI is also a weak ETI (figure 1). Also, the definition of strong ETI helps to eliminate the possibility of adding spurious transactions. A greedy approach for computing strong ETIs is also proposed [11].

2.2 Recursive Weak ETIs - RW

This algorithm was developed by Seppänen and Mannila and named (somewhat misleadingly) dense itemsets [8]. The idea was to add a recursive condition to the definition of a weak ETI in order to overcome weak ETIs inherent susceptibility to spurious items. Thus an itemset I' is a recursive weak ETI if I', as well as all subsets of I', are weak ETIs. One key point to note here is the set of transactions for the subset of items do not necessarily need to be the same. While this may seem to be a drawback, it still guarantees that any set of items within the itemset are related to one another. This algorithm also has an advantage of apriori-like pruning meaning if an itemset is not a recursive weak ETI, no superset of it can possibly be a recursive weak ETI. We denote this algorithm as 'RW' in this paper (see figure 1 for its relationship to other algorithms).

2.3 Approximate Frequent Itemsets (AFI)

The concept of an Approximate Frequent Itemset (AFI) was developed in [5], although it was earlier introduced in [9]. The idea is to extend the concept of strong ETI to include separate row and column constraints (ϵ_r and ϵ_c respectively). An itemset I' at support $minsup$ is said to be an AFI with tolerance ϵ_r and ϵ_c if $\exists T' \in T$ such that $|T'| \geq minsup$ and the following two conditions hold: $\forall i \in I'$, $\frac{\sum_{t \in T'} D_{t,i}}{|T'|} \geq 1 - \epsilon_c$ and $\forall t \in T'$, $\frac{\sum_{i \in I'} D_{t,i}}{|I'|} \geq 1 - \epsilon_r$.

From the above properties, it can be seen that AFIs are a subset of strong ETIs (see figure 1). One of the advantage of AFIs over weak/strong ETIs is that a relaxed version of an anti-monotone property holds for this pattern.

2.4 AC-CLOSE

AC-CLOSE [2] uses a core pattern constraint in addition to row (ϵ_r) and column (ϵ_c) error thresholds to find frequent approximate patterns. Specifically, this algorithm uses a parameter α to control the percentage of supporting transactions that must have all the items in an itemset. This essentially further filters out patterns generated by the algorithm 'AFI' (see figure 1). In [2], an efficient top-down mining algorithm was also proposed to discover approximate frequent patterns with core patterns as the initial seeds. In this paper, we apply the additional parameter α as a post-processing step to 'AFI' patterns to obtain 'AC-CLOSE' patterns. Conceptually, the patterns should be equivalent to the original AC-CLOSE, our implementation is not as efficient as the original 'AC-CLOSE' implementation.

2.5 Error Tolerant Itemsets (ETIs) with strong post-processing - GETI-PP

In order to overcome the possibility that 'GETI' can pick spurious items, we propose a variation of 'GETI' which uses an additional parameter (ϵ_c) to make sure every item in each ETI generated by the algorithm 'GETI' also satisfies the column constraint.

2.6 Recursive Weak ETIs with strong post processing - RW-PP

In order to overcome the drawback of both strong ETIs and recursive weak ETIs, we added a post-processing step in algorithm 'RW' to make sure that all recursive weak ETIs also meets a certain column threshold (say ϵ_c). Hence, patterns generated using 'RW-PP' lie in the intersection of 'RW' and strong ETI (figure 1).

2.7 Recursive Strong ETIs - RS

Since the recursive weak definition seems to work well in practice [8], a natural step is to define a recursive strong ETI, where each subset must also be a recursive strong ETI (figure 1).

3. EVALUATION METHODOLOGY

The evaluation approach given in [3] compares the result quality of different noise-tolerant models using precision and recall. Another evaluation framework for frequent sequential pattern mining [4] considers recoverability, spuriousness, redundancy and number of extraneous items as quantitative measures and influenced the design of our evaluation framework. Building upon the definition of recoverability and spuriousness given in [4], below we describe the evaluation measures in more detail. Note, we define $B = \{b_1, \ldots, b_m\}$ to be the set of base itemsets ("true" patterns) and $F = \{f_1, \ldots, f_n\}$ to be the set of found patterns.

3.1 Recoverability

This quantifies how well an approximate pattern mining algorithm recovers the base patterns. In this definition, recoverability is similar to recall. To measure the recoverability of the true patterns using the found patterns, we create a matrix of size $|F|X|B|$, whose ij^{th} element (i^{th} row and j^{th} column) is represented as fb_{ij}, i.e, the number of common items in found pattern F_i and base pattern B_j. We consider the recoverability of any base pattern B_j to be the largest percent of the itemset found by any pattern F_i that is associated with B_j. For illustration purposes, as shown in table 1, for each base pattern (each column in table 1), we put a $*$ on an entry which is maximum in that column. If there is a tie among found patterns for the maximum value, we put $*$ on multiple entries. For computing the recoverability of the base pattern B_j, we take any entry with a $*$ on it and divide it by the size of B_j. When we have more than one true pattern in the data, we need to combine the recoverability for each pattern into one measure. This is done by taking a weighted average (bigger patterns count more than smaller patterns) over all base patterns.

$$recoverability(B) = \frac{\sum_{j=1\ldots|B|} max_{i=1\ldots|F|} fb_{ij}}{\sum_{j=1\ldots|B|} |B_j|} \tag{1}$$

3.2 Spuriousness

Although recoverability gives a good idea of what fraction of patterns are recovered by an algorithm, it does not give a complete picture. It is possible that a pattern is recovered solely because a found pattern contained a large number of items, not all necessarily related to the base pattern. Thus just as precision is complementary to recall, we need another sibling measure of recoverability (recall) that measures the quality of the found patterns. The quality of a pattern can be estimated using the spuriousness measure which computes the number of items in the pattern that are not associated with the matching base pattern (i.e., are spurious). Hence, precision of a found pattern can be defined as $1 - spuriousness$. For illustration purposes, as shown in table 1, for each found pattern (each row in table 1), we put a $\#$ on an entry which is maximum in that row. If there is a tie among base patterns for the maximum value, we put a $\#$ on multiple entries in a row. For computing the spuriousness of the found pattern F_i, we take any entry with a $\#$ on it, subtract it from $|F_i|$ and divide it by $|F_i|$. Since there are usually numerous found patterns, the spuriousness of a set of found patterns is equivalent to the number of spurious items over total items found.

$$spuriousness(F) = \frac{\sum_{i=1\ldots|F|} (|F_i| - max_{j=1\ldots|B|} fb_{ij})}{\sum_{i=1\ldots|F|} |F_i|} \tag{2}$$

F/B	B_1	$\ldots$	B_j	$\ldots$	B_m
F_1	$fb_{11}{}^{*\#}$	$\ldots$	$fb_{1j}{}^{*}$	$\ldots$	fb_{1m}
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
F_i	fb_{i1}	$\ldots$	$fb_{ij}{}^{\#}$	$\ldots$	$fb_{im}{}^{*}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
F_n	$fb_{n1}{}^{*}$	$\ldots$	fb_{nj}	$\ldots$	$fb_{nm}{}^{\#}$

Table 1: Illustration of the matrix formed by found patterns and base (or true) patterns

3.3 Significance

Based on the two measures, recoverability and spuriousness that are defined above, we define a measure called 'significance of the found patterns' that combines the two just as F-measure combines precision and recall [10].

$$significance(F) = \frac{2 * (recoverability * (1 - spuriousness))}{(recoverability + (1 - spuriousness))} \tag{3}$$

This measure balances the trade-off between useful and spurious information in the generated patterns.

3.4 Redundancy

As mentioned above, many approximate pattern mining algorithms produce huge number of patterns that are often a small variation of one another. Hence, it is important to quantify how many of the found patterns are actually useful. For example, an itemset of size 10 has 45 subsets of size 8. If we recovered all of these, we would have a recoverability of 0.8, and a spuriousness of 0, but we would quickly be overwhelmed by the number of patterns. To measure the extent of the redundancy in the found patterns, we create a matrix R of size $|F|X|F|$, whose ij^{th} element (i^{th} row and j^{th} column) is represented as ff_{ij}, i.e, the number of common items in patterns F_i and F_j. We then take the sum of the upper triangular matrix R excluding the diagonal to estimate the redundancy of a set of patterns.

$$redundancy(F) = \frac{\sum_{i,j=1\ldots|F|} ff_{ij} - \sum_{i=1\ldots|F|} ff_{ii}}{2} \tag{4}$$

Note, we do not take the average in the computation of redundancy to differentiate between the algorithms that generate unequal number of patterns but have the same average pairwise overlap of items. Hence, this definition of redundancy indirectly takes into account the number of patterns generated by a mining algorithm.

3.5 Robustness

It is also important to compare approximate pattern mining algorithms based on their sensitivity to the input parameters. Some of the algorithms are more sensitive to these parameters than others and the quality of the patterns changes drastically if the optimal parameters are not used. Another parameter that these algorithms are usually sensitive to is the percentage of noise in the data. Ideally, an evaluation framework should evaluate the algorithm not only on the basis of the quality of patterns (based on significance which is defined as the combination of recoverability and spuriousness) but also on the basis of the size of the parameter space for which this algorithm generates patterns of acceptable quality. This parameter sensitivity analysis quantifies the robustness of an algorithm, which is a very important criterion as optimal parameters for a real life dataset will not be known. To do this, we explore a reasonable three-dimensional parameter space of support threshold ($minsup$), row constraints (ϵ_r) and column constraint (ϵ_c) for each algorithm and choose the top $k\%$ combinations of parameters for which the values of the significance measure are highest. We then take the mean and variance of these top $k\%$ significance values. While the mean denotes the performance of the algorithm in terms of quality of the patterns, variance denotes how sensitive it is to the selection of parameters. Ideally, one wants the mean to be high and variance to be low. However, in real-life applications, the unavailability of ground truth makes it inconceivable to obtain optimal parameters. Hence, one would like to choose an algorithm that consistently shows low variance (high stability) in top $k\%$ significance values even at some cost of pattern quality, when tested on synthetic datasets.

3.6 Choosing optimal parameters

As mentioned earlier, different approximate pattern mining algorithms generate best results on different optimal parameters. To do a fair comparison among them, it is very important to compare the results obtained by each on its own optimal parameters setting. Depending on the application, one may use different evaluation measures to choose optimal parameters. For a given dataset, we define the optimal parameters for an algorithm to be the ones that give the best value of significance. In case there is a tie among multiple parameter combinations, any parameter combination can be used.

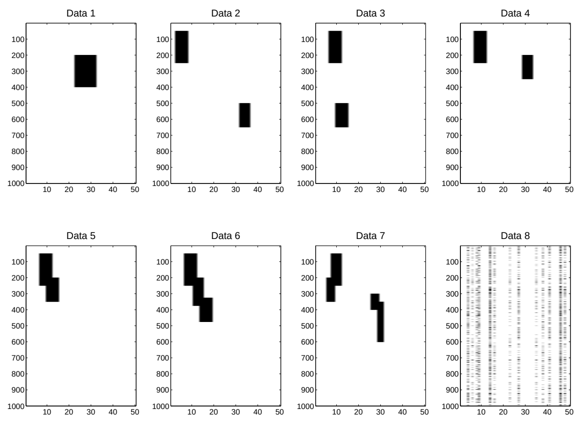

Figure 2: Images of the base (no noise) synthetic datasets used in the experiments. All datasets have 1000 transactions and 50 items.

4. EXPERIMENTAL STUDY

We implemented the algorithms 'AFI' and 'GETI' and used the publicly available version of 'RW'. We also implemented the variations of 'GETI' and 'RW' as discussed earlier. For 'AC-CLOSE', we use the patterns generated by 'AFI' and identify the ones that also have a core (completely dense) block of at least $\alpha * minsup$ support. Though we also implemented and tested the algorithm 'RS', we exclude its results because performance is generally worse than 'RW'. We then apply our proposed evaluation framework on both synthetic and real datasets to compare the performance of these approximate pattern mining algorithms. Synthetic datasets are used because it is easier to evaluate the merits/demerits of different algorithms when the ground truth is available. We also used zoo dataset [1] as an example of a real dataset since a number of previously published studies have also used zoo dataset [5, 3]. Moreover to avoid the problem of huge number of patterns, we only compare the maximal patterns generated by different algorithms.

4.1 Synthetic Datasets

Various synthetic datasets were generated keeping in mind different characteristics of a real-life dataset. These characteristics include: 1) *Noise*: Almost all real-life datasets are noisy and finding true patterns but not just the confounding groups of noisy attributes is a non-trivial task; 2) *Types of Patterns*: There are variety of patterns that a real-life data set may have depending on the application domain. For example, patterns may be overlapping (either in items or in transactions or both) or non-overlapping, of different sizes (in terms of number of items or in terms of transactions).

We generated 8 synthetic datasets (shown in figure 2) based on the above data characteristics. All the generated datasets have 50 items and 1000 transactions. Note that the datasets shown in figure 2 are only the base datasets with 0% noise level but we also generated noisy versions of each of them by uniformly adding random noise in fixed increments. Following is the brief description of each of the synthetic datasets.

Data 1 - Single embedded pattern of 10 items with a support of 200. *Data* 2 - Two non-overlapping embedded patterns of 6 and 5 items with a support of 200 and 150 respectively. *Data* 3 - Two embedded patterns of 6 items each (3 overlapping items) with a support of 200 and 150 respectively. *Data* 4 - Two embedded patterns of 6 and 5 items and with a support of 200 and 150 respectively (50 overlapping transactions). *Data* 5 - Two embedded patterns of

6 items each (3 overlapping items) with a support of 200 and 150 respectively (50 overlapping transactions). *Data* 6 - Three embedded patterns of 6, 5 and 6 items with a support of 200, 175 and 150 respectively. While patterns 1 & 2, and patterns 2 & 3 overlap in 2 items and 50 transactions, there is no overlap of either items or transactions in patterns 1 & 3. *Data* 7 - Four embedded patterns of 5, 4, 4 and 3 items with a support of 200, 150, 100 and 250 respectively. Patterns 1 & 2 overlap in 2 items and 50 transactions, patterns 3 & 4 overlap in 1 item and 50 transactions, and patterns 2 & 3 overlap in 50 transactions but no items. *Data* 8 - Similar to data 7 except that the patterns are generated in a different way. All the rows and columns in the data are randomly shuffled before a new pattern is embedded. Figure 2 shows the shuffled data after 4 patterns are embedded.

Given a base (no noise) synthetic dataset, we first add random noise by flipping its elements with a probability of $n\%$, which means that $(100 - n)\%$ is the probability of any element remaining constant. We vary the value of n to obtain noisy versions of the base synthetic dataset. We then run the algorithms on each of the noisy dataset using a wide range of parameters like support threshold ($minsup$), row (ϵ_r) and column (ϵ_c) tolerance. While we use 0, 0.05, 0.10, 0.20, 0.25 and 0.33 as 6 different values for row (ϵ_r) and column (ϵ_c) tolerance; the range of $minsup$ is selected based on the size of the implanted true patterns in the synthetic data. Moreover, as the noise is random, we repeat the complete process of adding noise and running the algorithms on all possible parameter combinations 5 times and report average results. To give an illustration of the computational complexity, consider applying the AFI, which uses both row (ϵ_r) and column (ϵ_c) tolerance, on a single dataset. To cover the parameter space defined by noise, support threshold ($minsup$), row (ϵ_r) and column (ϵ_c) threshold, we need to run this algorithm 5 (number of runs) x 5 (assuming 5 different noise levels) x 5 (assuming 5 different values of support parameter) x 6 (# of values of ϵ_r) x 6 (# of values of ϵ_c) = 4500 times. As the true patterns for all synthetic datasets are known, we run all the algorithms on a wide range of parameters to select the best combination (refered to as the optimal parameters) for each. The performance of the algorithms is then compared with each other using optimal parameters.

4.1.1 Results on synthetic datasets

Due to the space constraints, we only show results on synthetic data 6 (figure 3) and synthetic data 8 (figure 4). Results on other datasets are similar and are available on the website that contains all the source codes and the datasets (see section 1). Also, Tables 2 and 3 shows the optimal parameters selected by each algorithm for these datasets at different noise levels. Sometimes there are multiple parameter values for which the generated patterns show the same performance measured in terms of significance. In such cases, we show the parameter combinations corresponding to minimum and maximum support within such cases. Noise level is varied from 0% to 16% in increments of 4% for synthetic data 6 and from 0% to 8% in increments of 2% for synthetic data 8. Again, the maximum amount of noise level introduced in the data is governed by the size of the smallest implanted pattern.

In both figure 3 and figure 4, we can see that the performance of the 'APRIORI' algorithm (measured in terms of significance) falls most rapidly as the random noise in the data increases. Although as seen from table 2, 'APRIORI' uses low support threshold ($minsup = 100$) for all noise levels to recover true patterns, due to the rigid definition of support, overall recoverability and hence significance is low.

Generally speaking, the performance of all the algorithms falls as expected when the random noise is increased in the data. As the noise increases, recoverability goes down, spuriousness goes up and as a result, the significance of the patterns goes down. Although, every algorithm chooses optimal parameters corresponding to the best value of significance, the effect of random noise is different on each algorithm. Broadly, these algorithms could be divided into two groups: one that uses single parameter ϵ and one that uses two parameters ϵ_r and ϵ_c. Usually, single parameter algorithms 'RW' and 'GETI' pick more spurious items than those that uses two parameters. This is because these single parameter algorithms only require each supporting transaction to have at least $(1 - \epsilon_r)$ fraction of items. They do not impose any restriction on the items in the column. On the other hand, algorithms 'AFI', 'AC-CLOSE' and our proposed variations 'GETI-PP' and 'RW-PP' pick fewer spurious items and hence have a better significance values. AFI uses two parameters ϵ_r and ϵ_c and hence additionally requires each item in a pattern to be supported by at least $(1 - \epsilon_c)$ fraction of total supporting transactions. 'AC-CLOSE' further requires that pattern should have a core (completely dense) block of support at least $\alpha * minsup$, where $\alpha \in [0, 1]$. 'GETI-PP' and 'RW-PP' uses another parameter ϵ_c in addition to the parameterϵ_r used by the algorithms 'GETI' and 'RW' to check if all the items in each pattern have more than $(1-\epsilon_c)$ fraction of supporting transactions. This helps in filtering some of the patterns that have spurious items. Hence, 'GETI-PP' and 'RW-PP' finds the patterns with a flavor similar to the ones generated by 'AFI'.

As can be clearly seen from significance plots in figure 3 and figure 4, generally 'AFI', 'GETI-PP', and 'RW-PP' have similar performance. However, the optimal parameters used by these algorithms are different as shown in Tables 2 (for data 6) and 3 (for data 8). For instance at a noise level of 8% in synthetic data 6, 'GETI-PP' can find the patterns at $minsup = 150$, but 'AFI' and 'RW-PP' can only find them at $minsup = 125$. Similarly at a 6% noise level in table 3, 'RW-PP' finds same quality patterns at either parameters $minsup = 90$, $\epsilon_r = 0.25$, and $\epsilon_c = 0.05$ or at $minsup = 100$, $\epsilon_r = 0.25$, and $\epsilon_c = 0.10$. Therefore, by relaxing ϵ_c from 0.05 to 0.10, 'RW-PP' was able to find same quality patterns at higher support. All such cases, where multiple optimal parameter values are possible, are shown in the optimal parameters tables.

Our results here demonstrate that differences amongst most of these algorithms are not very large when optimal parameters are used. This finding is not consistent with some of the conclusions in previous work ([5, 2]). In [5] 'AFI' and 'GETI' were compared on a simple synthetic dataset with one embedded pattern (similar to our synthetic data 1) and 'AFI' was found to outperforms 'GETI' by a huge margin both in terms of recoverability and spuriousness. Following are some of the possible reasons for this inconsistency: (1) Parameter space ($minsup$, ϵ_r, ϵ_c) is not explored in [5] to choose the optimal parameters for each algorithm, and (2) an exact matching criterion between the found pattern and the true pattern might have been used. In [2] 'AFI' and 'AC-CLOSE' were compared on synthetic datasets generated using the IBM data generator. Ground truth is defined to be the traditional dense patterns obtained using the APRIORI algorithm in the noise-free version of the data. This truth appears to favor the algorithm 'AC-CLOSE', which requires a pattern to have a core block of support at least $\alpha * minsup$.

We also compare the algorithms based on their sensitivity to the input parameters because in real datasets where the ground truth is not available, optimal parameters cannot be estimated. Ideally for a real dataset, one would like to choose an algorithm which gives acceptable performance as measured in terms of significance and yet be less sensitive to the input parameters. Figures 3 and 4 shows the variance of the top $k\%$ significance values obtained on different

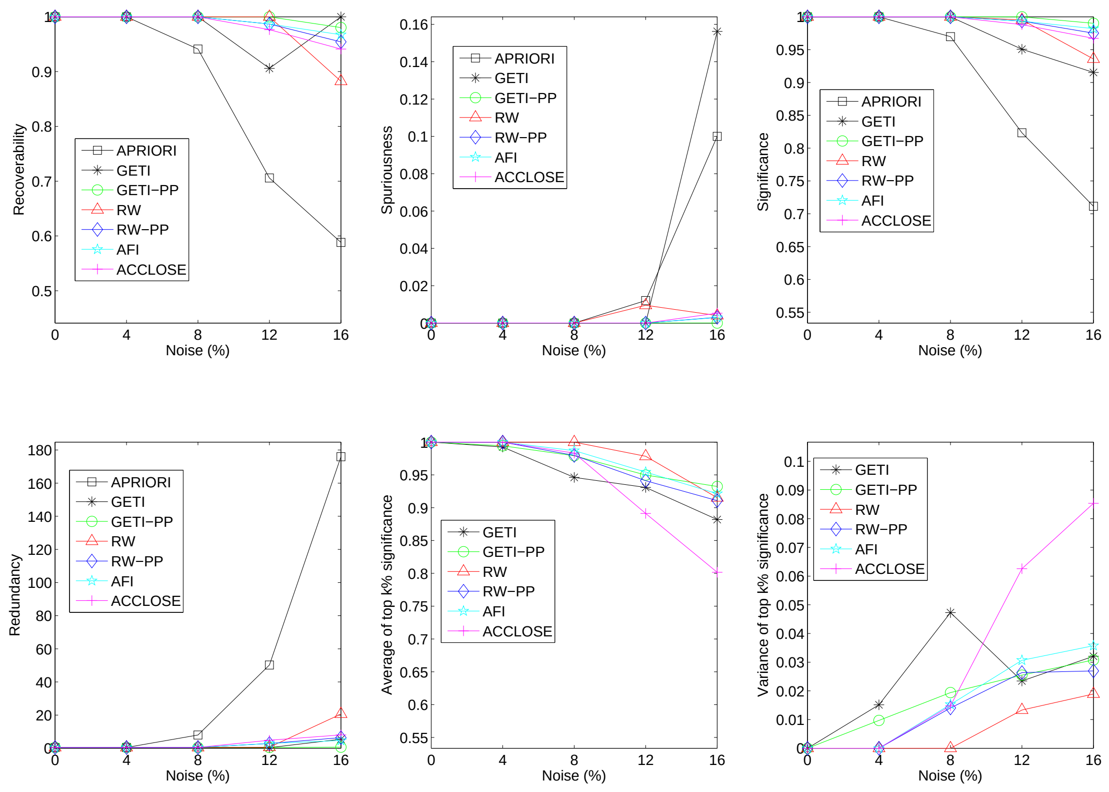

Figure 3: Results on synthetic data 6 - Comparing different algorithms in terms of performance and robustness.

Algorithms	Noise Levels (%)				
	0	4	8	12	16
APRIORI (sup)	(100) (150)	(100)	(100)	(100)	(100)
GETI (sup, ϵ_r)	(100,0) (150,0)	(100,0) (125,0.05)	(100,0.05)	(125,0.1)	(100,0.1)
GETI-PP (sup, ϵ_r, ϵ_c)	(100,0,0) (150,0,0)	(100,0,0) (150,0.2,0.05)	(100,0.05,0.05) (150,0.2,0.1)	(125,0.2,0.1)	(125,0.2,0.1)
RW (sup, ϵ_r)	(100,0) (200,0.25)	(100,0) (200,0.25)	(100,0.05) (175,0.2)	(175,0.2)	(200,0.2)
RW-PP (sup, ϵ_r, ϵ_c)	(100,0,0) (150,0,0)	(100,0,0) (125,0.2,0.05)	(100,0.2,0.05) (125,0.2,0.1)	(125,0.2,0.1)	(125,0.25,0.1)
AFI (sup, ϵ_r, ϵ_c)	(100,0,0) (150,0,0)	(100,0,0) (125,0.2,0.1)	(100,0.2,0.1) (125,0.2,0.1)	(125,0.2,0,2)	(125,0.2,0,2)
ACCLOSE (sup, ϵ_r, ϵ_c, α)	(100,0,0,1) (150,0,0,1)	(100,0,0,1)	(100,0.2,0.1,0.9)	(125,0.2,0.2,0.5)	(125,0.2,0.2,0.4)

Table 2: Optimal parameters for different algorithms on synthetic data 6.

parameter combinations for dataset 6 and 8 respectively. The mean of the top $k\%$ significance values is also shown to indicate the overall performance. Note, we do not show 'APRIORI' in the plots of mean and variance of top $k\%$ values because 'APRIORI' only uses one $minsup$ parameter while parameters ϵ_r and ϵ_c are 0 by design. Also remember, variance does not indicate the performance of the algorithm, it only indicates how consistently an algorithm generates the patterns with similar quality on different parameter settings. It is more meaningful to compare only those algorithms on this measure, which show acceptable significance values. We set $k = 16.67$ ($\frac{1}{6}$ of the total runs) in this paper. It is important to note that 'GETI' and 'RW', which require only one parameters ϵ_r apart from $minsup$ have fewer number of runs in comparison to algorithms 'AFI', 'AC-CLOSE', 'GETI-PP' and 'RW-PP', which require two parameters ϵ_r and ϵ_c apart from $minsup$ and hence have more number of runs. Figure 3 and 4 shows the mean and variance of top 4 (out of total 24) and top 24 (out of total 144) significance values for these two sets of algorithms respectively. We notice that although the difference in variance is not too high on these datasets, 'AC-CLOSE' shows relatively high variance (hence less robustness) than others. This may be because of the requirement of specifying fourth parameter α, which makes it difficult to estimate the optimal parameters.

4.2 Real Dataset: Zoo Data

In the Zoo dataset [1], there are 101 instances (animals) with 15 boolean attributes (e.g. aquatic, tail, hair, eggs etc.) and a class label (mammal, bird etc.). For approximate pattern mining, we

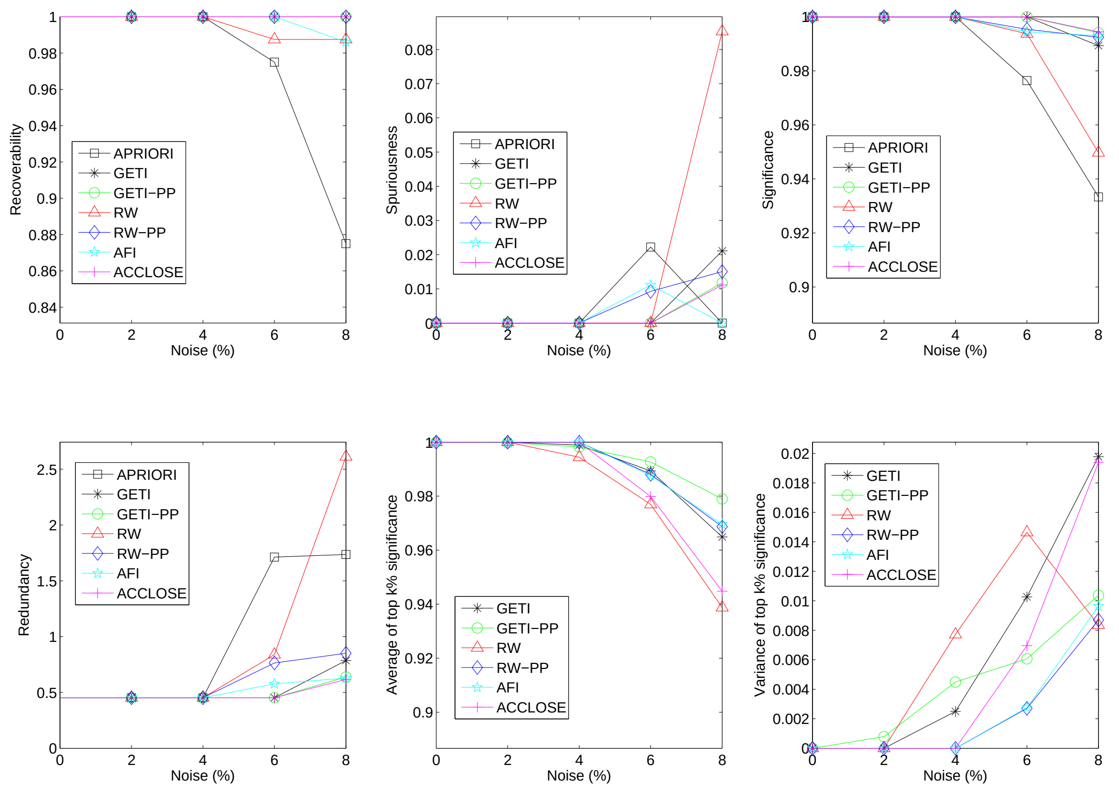

Figure 4: Results on synthetic data 8 - Comparing different algorithms in terms of performance and robustness.

Algorithms	Noise Levels (%)				
	0	2	4	6	8
APRIORI (sup)	(100)	(70) (80)	(80)	(80)	(80)
GETI (sup, ϵ_r)	(100,0)	(70,0) (100,0.05)	(80,0) (100,0.05)	(100,0.05)	(100,0.1)
GETI-PP $(sup, \epsilon_r, \epsilon_c)$	(100,0,0)	(70,0,0) (100,0.05,0.05)	(80,0,0) (100,0.05,0.05)	(100,0.05,0.05)	(100,0.05,0.1)
RW (sup, ϵ_r)	(100,0)	(70,0) (100,0.05)	(80,0) (100,0.1)	(100,0.05)	(100,0.1)
RW-PP $(sup, \epsilon_r, \epsilon_c)$	(100,0,0)	(70,0,0) (100,0.25,0.05)	(80,0,0) (100,0.25,0.1)	(90,0.25,0.05) (100,0.25,0.1)	(100,0.25,0.1)
AFI $(sup, \epsilon_r, \epsilon_c)$	(100,0,0)	(100,0.25,0.1)	(80,0,0)	(100,0.25,0,2)	(100,0.25,0,2)
ACCLOSE $(sup, \epsilon_r, \epsilon_c, \alpha)$	(100,0,0,1)	(100,0.25,0.1,0.8)	(80,0,0,1)	(100,0.25,0.2,0.7)	(100,0.25,0.2,0.6)

Table 3: Optimal parameters for different algorithms on synthetic data 8.

consider transactions to be animals and items to be different features that characterizes them. Finding frequent itemsets in this data provides the ability to predict the class of an animal. In general, approximate itemsets are more suited for this problem because not all instances of the same class have all the common features. For example, though most mammals produce milk, are covered in hair, and are toothed, platypus lack teeth and dolphin lack hair.

4.2.1 *Results on real dataset*

We only focused on three classes (mammals, birds and sea-creatures) as they have more than 10 instances each. As we saw from the results on synthetic datasets, ‘GETI-PP’ and ‘RW-PP’ usually outperforms ‘GETI’ and ‘RW’ respectively, we only show the results of ‘AFI’, ‘GETI-PP’ and ‘RW-PP’ (see table 4). Our results indicate that all the algorithms discussed in this paper (except of course ‘APRIORI’) can find the itemsets that defines the two classes, mammals and birds, almost perfectly. An itemset of size-7 is supported by $\frac{40}{41}$ instances of mammals, and interestingly no other instance (animal) from any other class supports it. Similarly, an itemset that is only supported by the instances of the bird’s class can be found. For example, ‘GETI’ finds an itemset of size-6 at $minsup = 18$ and $\epsilon_r = 0.09$, which is supported by all (and only) the instances from the bird’s class. The inconsistency of these results with those in [5, 3] is due to the difference in selection of optimal parameters.

The instances of the third class sea-creatures share 3 common features but the same 3 features are also shared by some instances from mammals and birds class. Hence, an itemset comprising of

Algorithms	Mammals	Birds
GETI-PP	(40, 0.07, 0.15)	(20, 0.1, 0.1)
RW-PP	(40, 0.15, 0.15)	(20, 0.1, 0.18)
AFI	(40, 0.2, 0.15)	(20, 0.2, 0.18)

Table 4: Parameters ($sup, \epsilon_r, \epsilon_c$) for different algorithms on zoo data

these 3 features alone cannot be used to predict the instances of the class sea-creatures. Truly speaking, sea-creatures distinguish themselves from other classes because they lack some features that instances from other classes have. Association pattern mining in general does not find patterns to capture such information. This requires generalizing the definition of patterns to not only include patterns like (A and B and C) but also like ((A or B) and ($notC$)) but this generalization is beyond the scope of this paper.

4.3 Efficiency and Scalability

4.3.1 Run-Time

In this section, we compare the efficiency of different algorithms for varying amount of noise in the dataset. Considering the fact that different parameters will result in different run-time for each algorithm, we run the algorithms on all the different parameter combinations and use the total run-time to compare them. All the algorithms are run on a linux machine with 8 Intel(R) Xeon(R) CPUs (E5310 @ 1.60GHz) (with 10 processes). Because 'GETI-PP' and 'RW-PP' are the variations of 'GETI' and 'RW' respectively, we only report results on 'GETI-PP' and 'RW-PP'. Also, the run-time of 'AC-CLOSE' algorithm is not included. In table 5, we report the run-times (in seconds) of the algorithms 'GETI-PP', 'RW-PP' and 'AFI' on synthetic data 8 with noise levels varied from 0% to 14% in increments of 2%. Note, this is the total time taken by the algorithm for 144 paramater combinations (4 different $minsup$ values and 6 different ϵ_r and ϵ_c values). It is interesting to see the differences among the algorithms in terms of the increase in run-time as the noise increases. Though, 'AFI' is computationally more efficient than 'GETI-PP' when noise in the dataset is low, it is very expensive when noise in the dataset is high. However, it is important to note that this is also due to high value of row and column error threshold. It is also interesting that run-time of 'RW-PP' only increase marginally when the noise is increased from 0% to 12% after which it also shows rapid increase.

Algorithms	Noise Level (%)										
	0	2	4	6	8	10	12	14	16	18	20
AFI	11	11	12	11	12	21	1453	-	-	-	-
GETI-PP	20	26	87	135	292	296	406	411	420	429	427
RW-PP	11	11	11	12	14	16	29	81	267	700	-

Table 5: Comparison of run-times of different algorithms for varying amount of random noise.

4.4 Effect of higher random noise

In the experiments shown so far, random noise added to the synthetic data was limited by the size (number of transaction) of the smallest base (or true) pattern. This was done (1) to make sure that the truth remains meaningful and (2) to make the parameter space search feasible for mining algorithm AFI, which is computationally expensive when the random noise in the data is high. However, in some real-life applications, small but true patterns are hidden in even larger amounts of random noise. In such cases, it is still desired that algorithms could recover as many significant and useful patterns from the dataset as possible. Hence, the trade off between recoverability and spuriousness becomes even more challenging. However, as the computational efficiency of the algorithms decreases as the noise increases, we designed the following two schemes to make the comparison among algorithms feasible:
• Scheme 1: For algorithms that could finish in time δ ($\delta = 1$ hour in our experiments), we search the whole parameter space.
• Scheme 2: For algorithms that could not finish certain parameter combinations in time δ, we search the whole parameter space in the order of complexity. Also, the row and the column thresholds were set equal. 'Stricter' parameter combinations (high support threshold, small error tolerance) take less time to finish and hence will be tested before the 'less strict' ones that have small support threshold and high error tolerance. Any combination of parameters, support threshold and error tolerance, for which the algorithm does not finish in time δ, is not considered while selecting the optimal parameters.

We compare the performance of different algorithms under more noise only on synthetic data 8 since it appears to be closest to a real dataset. In addition to the original noise levels of 0%, 2%, 4%, 6%, and 8% considered in the previous experiment, noise levels of 10%, 12%, and 14% were also tried. As the algorithm 'AFI' could not finish certain parameter combinations at noise level of 14% within time δ, we use the second scheme above to search the parameter space. Although other algorithms also took longer, they were able to finish all the runs in time δ. Figure 5 shows the results on synthetic data 8 including more noise levels 10%, 12%, and 14%. It is clear the performance of all the algorithms suffers due to more random noise in the data. However, performance of 'RW' seems to suffer more than the others as it is picking more spurious items. Also as 'AFI' could not finish certain runs at noise level 14% in time δ and hence had to choose the optimal parameters from a smaller parameter space, its performance falls at noise level 14%. Moreover, as we derive 'AC-CLOSE' patterns from the 'AFI' patterns using the core block constraint, performance of 'AC-CLOSE' falls as well. A more efficient implementation of 'AC-CLOSE' may not have this problem. Interestingly, in this case 'GETI' and 'GETI-PP' have better performance than 'RW-PP' but as we see from the variance of the top $k\%$ (where $k = 16.67$ ($\frac{1}{6}$ of the total runs)) significance values (figure 5), 'GETI' and 'GETI-PP' seems to be less robust as the noise in the data increases.

5. CONCLUSIONS

In this paper, we have proposed an evaluation framework and showed its applicability to compare different approximate pattern mining algorithms on both synthetic and real datasets. Following are the general conclusions of our evaluation study:
• Our results suggest that enforcing the column error tolerance ϵ_c (as introduced in AFI algorithm [5]) over the concepts of strong ETI or recursive weak ETI, makes them much more effective in terms of finding the true patterns with less spurious information.
• All the existing ('AFI' and 'AC-CLOSE') as well as our proposed variations ('GETI-PP' and 'RW-PP') of the algorithms, which use the column error tolerance ϵ_c perform similarly when the optimal parameters are selected. This is because adding an additional constraint helps filter out patterns with spurious items.
• The computational efficiency of the variations 'GETI-PP' and 'RW-PP' seems to be much better than 'AFI' specially at higher noise levels. Moreover, their performance in terms of the quality of the patterns is also comparable to those generated by 'AFI'.

These conclusions are in contrast to some of the previously published studies ([6, 5, 2]). [6, 5] compared only 'GETI' and 'AFI' and suggested 'AFI' significantly outperforms 'GETI'. We showed that although this is true, the difference is not that significant. More-

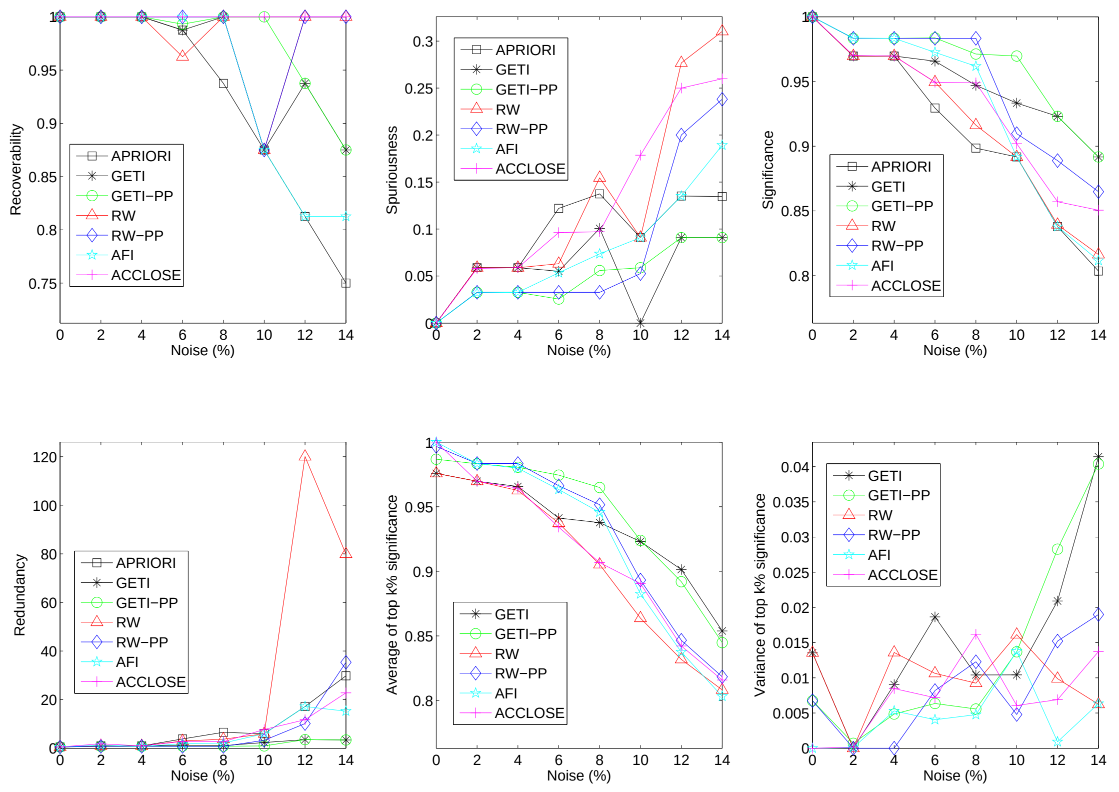

Figure 5: Results on Synthetic Data 8 with more random noise levels $10\%, 12\%, 14\%$

over 'GETI-PP', a simple variation of 'GETI' we proposed performs comparable to 'AFI'. [2] compared only 'AC-CLOSE' and 'AFI' and suggested 'AC-CLOSE' performs better than 'AFI'. However, we observed no significant differences in 'AFI' and 'AC-CLOSE'. We believe these differences are partly due to the fact that previous studies did not select the optimal parameters for each algorithm and partly because of the choice of the datasets.

This comparative study, though far more comprehensive than other previous studies, has several limitations. Most of the patterns considered in this paper are simple embedded patterns in the synthetic datasets and hence may not reflect various aspects of complex real datasets. Even the real zoo dataset is not very complex in terms of its size and availability of ground truth. Though, it would be better to apply the evaluation framework on a complex real dataset, lack of ground truth knowledge makes this much harder.

There are still many interesting problems that need to be studied. In the evaluation framework, it would be interesting to incorporate redundancy and other measures in the process of optimal parameter selection. On the algorithm side, extending approximate pattern mining algorithms to work with categorical and continuous valued data could prove to be very beneficial to many application domains.

6. ACKNOWLEDGEMENTS

This work was supported in part by NSF and by the University of Minnesota Rochester Biomedical Informatics and Computational Biology Program traineeship award and research seed grants. Access to computing facilities was provided by the Minnesota Supercomputing Institute.

7. REFERENCES

[1] C. Blake and C. Merz. Uci repository of machine learninig databases. University of California, Irvine, 2007.

[2] H. Cheng, P. S. Yu, and J. Han. Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In *ICDM*, pages 839–844, 2006.

[3] H. Cheng, P. S. Yu, and J. Han. Approximate frequent itemset mining in the presence of random noise. In *Soft Computing for Knowledge Discovery and Data Mining*, pages 363–389. Oded Maimon and Lior Rokach. Springer, 2008.

[4] H.-C. Kum, S. Paulsen, and W. Wang. Comparative study of sequential pattern mining frameworks — support framework vs. multiple alignment framework. In *ICDM Workshop*, 2002.

[5] J. Liu, S. Paulsen, X. Sun, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets in the presence of noise: algorithm and analysis. In *SDM*, pages 405–416, 2006.

[6] J. Liu, S. Paulsen, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets from noisy data. In *ICDM*, pages 721–724, 2005.

[7] J. Pei, A. K. H. Tung, and J. Han. Fault-tolerant frequent pattern mining: Problems and challenges. In *DMKD*, 2001.

[8] J. Seppanen and H. Mannila. Dense itemsets. In *KDD*, pages 683–688, 2004.

[9] M. Steinbach, P.-N. Tan, and V. Kumar. Support envelopes: A technique for exploring the structure of association patterns. In *KDD*, pages 296–305, New York, NY, USA, 2004. ACM Press.

[10] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Addison-Wesley, May 2005.

[11] C. Yang, U. Fayyad, and P. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *KDD*, pages 194–203, 2001.