Efficient Discovery of Error-Tolerant Frequent Itemsets in High Dimensions

Cheng Yang
Stanford University
yangc@cs.stanford.edu

Usama Fayyad
Microsoft Research
fayyad@microsft.com

Paul S. Bradley
Microsoft Research
bradley@microsoft.com

February, 2000
Technical Report
MSR-TR-2000-20

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Abstract

We present a generalization of frequent itemsets allowing the notion of errors in the itemset definition. We motivate the problem and present an efficient algorithm that identifies error-tolerant frequent clusters of items in transactional data (customer-purchase data, web browsing data, text, etc.). This efficient algorithm exploits sparsity of the underlying data to find large groups of items that are correlated over database records (rows). The notion of transaction coverage allows us to extend the algorithm and view it as a fast clustering algorithm for discovering segments of similar transactions in binary sparse data. We evaluate the new algorithm on three real-world applications: clustering high-dimensional data, query selectivity estimation and collaborative filtering. Results show that we consistently uncover structure in large sparse databases that other more traditional clustering algorithms in data mining fail to find.

26th International Conference on Very Large Databases, submitted.

1 Preliminaries and Motivation

Progress in database technology has provided the foundation that made massive stores of transactional data ubiquitous. Such stores are common in commerce (products purchased by customers), web logs (websites visited by users), text (words occurring in documents), etc. The *frequent itemset* problem is that of determining which items frequently occur together in a transaction. We consider relaxing the criteria commonly associated with frequent itemsets to a more flexible version that tolerates error and propose an algorithm for finding all such error-tolerant frequent itemsets. We then provide an efficient algorithm approximating the complete algorithm. The primary motivation for this generalization is to find frequent groups of transactions (groups of users, web sessions, etc.) instead of focusing primarily on just the items themselves, allowing for the discovery of groups of similar transactions that share most items. We believevthis to be a more general and more intuitive characterization of groups of transactions.

The frequent itemset generalization, based on relaxing the exact matching criteria in frequent item sets and allowing a transaction to violate some conditions, is motivated by the following example. Consider a set of customer purchase data over 5 products (P1,…,P5). Figure 1 shows a graphical snapshot of the data where items (columns) P1,…,P5 are listed first (the other products are not of interest for this example), and customers purchasing these 5 products are similarly listed first. The shaded regions depict sets of products bought by sets of customers. Table 1 gives the counts of customers depicted as blocks in Figure 1. Let the total number of customers (rows) in the database be 10,000. For simplicity, suppose that no other customers in the database purchased these 5 products. Notice that for any minimum support value $\kappa > 0$, the itemset {P1,…,P5} will not appear to be frequent. In fact, for a support level of 5%, none of the 5 items would appear in any frequent itemset enumeration. Note, however, that 5.7% of the transactions contain 4 of the 5 products. If products P1,…,P5 are different brands of soda then these 5.7% of the customers purchase a significant portion of these 5 brands. This pattern may be useful for the data analyst but would be undiscovered by traditional frequent itemset approaches due to the harsh definition of support. In fact, when reducing the support to 4%, traditional frequent itemsets would only find {P2, P4} as the longest itemset over {P1,…,P5}. By relaxing the definition of frequent itemsets to be error-tolerant, one could identify this cluster of customers who, purchase "most" of the products {P1,…,P5}.

The intuition is made specific with the following definition and problem statement. Adopting the notation of [AMSTV96], let $I = \{i_1, i_2, \ldots, i_d\}$ be the full set of items over a database D consisting of transactions T, where each transaction is a subset of the full itemset I. Each transaction T may be viewed as record with d attributes $\{i_1, i_2, \ldots, i_d\}$ taking values in {0,1}, where a 1 occurs over the attributes specifically listed in the transaction T (hence viewing the database as a table with one record for each transaction and d columns). This view of the data often results in a very sparse table (i.e. the majority of the table

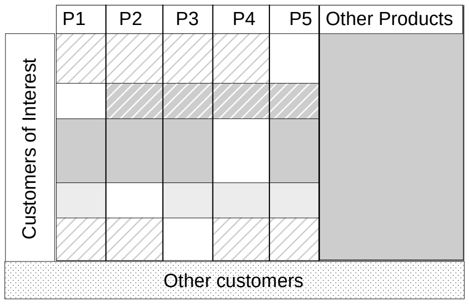


Figure 1: graphical depiction purchase regions in data sub-matrix

Count	P1	P2	P3	P4	P5
100	1	1	1	1	0
100	0	1	1	1	1
80	1	1	1	0	1
90	1	0	1	1	1
200	1	1	0	1	1

Table 1: Counts of customer-purchase data patterns for submatrix.

elements have value 0). The *support* of an itemset is the fraction of total transactions in the database containing the given itemset. The *frequent itemset* problem is that of finding all itemsets with support greater than a minimum threshold κ (called *minimum support* or *minsup*) [RG99]. Note that for a single transaction *T* to contribute to the support of a given itemset, it must contain the entire itemset. We relax this exact matching criterion to yield a more flexible definition of support and consequently of error-tolerant itemsets, eventually leading to an algorithm for clustering rows in sparse binomial databases.

We have found that our algorithm is capable of identifying the presence of structure (clusters) in large sparse databases that traditional clustering algorithm consistently fail to find. This leads to both a more effective (and faster) method of clustering, as well as an effective way of determining the number of clusters: an open problem for most classical clustering algorithms [DH73,CS96,BFR98].

Definition 1: Error-Tolerant Itemset [ETI] (general): An itemset $E \subseteq I$ is an *error-tolerant itemset* having error ε and support κ with respect to a database *D* having *n* transactions if there exists at least κ*n* transactions in which the probability of observing a 1 over the itemset is not less than 1-ε.

Problem Statement: Given a sparse binomial database *D* of *n* transactions (rows) and *d* items (columns), error tolerance ε > 0, and minimum support κ in [0,1], determine all error-tolerant itemsets (ETIs) over *D*. The fundamental difference between this problem and that of traditional frequent itemsets is a relaxation in support criteria. An error threshold ε = 0 collapses Definition 1 to the standard frequent itemset definition. For ε > 0, the problem is to efficiently determine itemsets for which support can be determined by a function requiring that (1-ε) of the *m* items in the ETI *E* be present. For example, the itemset *E* = {P1,P2,P3,P4,P5} from Table 1 is an error-tolerant itemset with support κ = 5.7% and ε = 0.2. Note that the support for this itemset can also be interpreted as those transactions containing 4 of 5 of the items in *E*. This definition is not confined to binary {0,1} data, but can be extended to find error-tolerant itemsets over transactional databases with categorical-valued attributes (more than 2 values). Continuous-valued attributes may be preprocessed with a discretization algorithm [FI93]. We discuss generalizations in Section 6, but we focus on the binary case in this paper.

We define maximal ETIs as those ETIs whose supersets are not ETIs. Sometimes both maximal and nonmaximal ETIs are of interest when finding clusters of similar transactions (clustering rows versus just columns). We illustrate this notion in the example shown in Figure 2. On the left we illustrate 3 groups of transactions: customers who bought 5 products P1-P5 (35% of the transactions), customers who bought only P1-P2 (20%) and customers who bought only P4 and P5 (45%). On the right, we show the 3 possible ETIs (note that ETI identifiers overlap). An algorithm that looks for coverage of the data would do the job with {P1, P2} (55%) and {P4, P5} (80%). Even worse, imagine a product, say P6, that most shoppers bought. That's an itemset of length 1 that has approximately 100% support, but hardly indicates structure in the data. However, intuitively the cluster of people who bought all five products should be identified and is indeed significantly different from the others. We will show how the ETIs can be used to efficiently uncover the structure on the left, identifying the three "natural" clusters in the example.

1.1 Are ETIs Random Artifacts?

Before studying properties of ETIs and their efficient extraction, we pause briefly to address the question of whether finding such item sets is of interest, and whether ETI discovery could

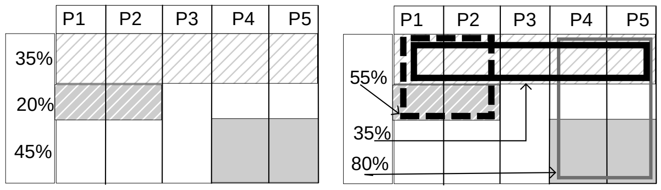


Figure 2: ETIs and data clusters

be a side effect of correlations that happen to appear in data by pure chance. It should be obvious from the definition of ETIs that as the error ε is increased, the expected number of items in ETIs should increase, which raises the question of the validity of such patterns to begin with. Essentially, **is one really finding structure in data, or simply fishing out random correlations in large data sets?**

Lemma: Assume a binary $N \times D$ sparse matrix over {0,1} is generated at random with the probability that an entry is 1 is p. Then the probability of a frequent error-tolerant item set with r items appearing in it with support κ and error ε is not greater than $\binom{D}{r}\binom{N}{\kappa N} p^{(1-\varepsilon)\kappa N r}(1-p)^{\varepsilon \kappa N r}$.

Proof: see Appendix A □

An application of this lemma to some realistic assumptions over market-basket type data quickly shows that this probability is vanishingly small. For example, for $p = 0.15$, $\varepsilon = 0.2$, $\kappa = 0.01$, $N = 1{,}000{,}000$, $D = 500$ and $r = 5$, using Stirling's approximation [GKP89] for the combinatorial terms one obtains that the probability of finding an ETI with 5 items by chance is approximately 10^{-9300} -- essentially zero (for r=10 items this probability drops to $10^{-43,000}$).

Further, assuming a Zipf distribution [Z49] over items yields even smaller probabilities. For a detailed description, see Appendix A. Using the original example and making the Zipf assumption, we have $\varepsilon = 0.2$, $\kappa = 0.01$, $N = 1{,}000{,}000$, $D = 500$, $r = 5$, on average p is (6/501) = 0.012 and probability of finding an ETI with 5 items by chance is less than $10^{-52,000}$ (drops quickly to less than $10^{-108,000}$ for r=10 items) – again essentially zero. Hence the identification of submatrices with high frequency of 1's in them is indeed interesting as such submatrices, especially when the number of columns involved is large, are extremely unlikely to occur by pure chance.

1.2 Applications of ETIs

While our primary aim is to introduce the generalization to frequent item sets, we also use some applications of ETIs to demonstrate their benefits. We show that ETIs provide great utility as a technique for initializing clustering algorithms like the EM (Expectation-Maximization) algorithm [DLR77, CS96]. We show that ETIs are very effective at leading EM to clusters it would not otherwise find over real and synthetic sparse transactional data. The cluster initialization problem over sparse transactional data in high dimensions is *effectively* addressed by error-tolerant itemsets. In fact in many cases ETIs find the solutions quickly and the clustering algorithm adds little improvements to it. We also use error-tolerant itemsets for query selectivity estimation over sparse binomial databases, and for a collaborative filtering prediction task [MRK97,R*97,RV97] predicting items likely to be included in a transaction based upon the presence of other items.

2 Finding Error-Tolerant Frequent Itemsets

In binomial {0,1} datasets, an error-tolerant frequent itemset (ETI) is represented as a set of dimensions (called *defining dimensions*) where "1" appears with high probability among a set of rows. Formally, we give the following two ETI definitions, a strong one and a weak one:

ETI Definition 1 (strong): A strong ETI consists of a set of items, called defining dimensions DD $\subseteq$ I, such that there exists a subset of transactions R $\subseteq$ T consisting of at least κn transactions and, for each of $r \in$ R, the fraction of items in DD which are present in r is at least 1-ε.

ETI Definition 2 (weak): A weak ETI consists of a set of items, called defining dimensions DD $\subseteq I$, such that there exists a subset of transactions R $\subseteq T$, $|R| \geq \kappa n$ transactions and,

$$\frac{\sum_{x \in R} \sum_{d \in DD} d(x)}{|R| \cdot |DD|} \geq (1 - \varepsilon).$$

Here $d(x)$ is 1 if item d occurs in x and 0 otherwise. The weak definition basically requires that the data sub-matrix defined by the records in R and the columns in DD be mostly "1"s, with the fraction of "0"s not greater than ε. It is clear that anything that satisfies the strong definition above also satisfies the weak definition, but not vice versa.

In both definitions above, we say that the set DD *defines* the ETI, and we call κ the *support threshold* and ε the *error threshold*. A set of defining dimensions DD is called *maximal* if and only if DD defines an ETI and no superset of DD defines an ETI. For example, in Table 2 with κ=25% and ε=20%, {1 2 3 4 5} defines a maximal strong ETI, {1 2 3 4} defines a non-maximal strong ETI, and {6 7 8 9} defines a maximal weak ETI (but not a strong ETI). Our ultimate goal is to find strong ETIs, but the notion of weak ETIs will also be needed in our algorithm.

1	2	3	4	5	6	7	8	9	10
1	1	1	1	1					1
1	1	1	1				1		
1	1	1	1	1					
1	1	1	1	1					
1	1	1	1	1					
1	1	1	1	1					
1	1	1	1	1					
1	1	1	1		1	1		1	
1	1	1	1	1	1		1	1	
					1	1	1	1	
					1	1	1	1	
					1	1	1	1	
					1	1		1	
									1
					1				
	1								
			1				1		
		1							
1									1

Table 2: Example binary data set

2.1 Properties

Lemma 1: If a set DD of m dimensions defines a weak ETI, then there exists a set DD' of m-1 dimensions such that DD' $\subset$ DD and DD' also defines a weak ETI. (In other words, it is possible to remove one defining dimension from DD and still maintain a weak ETI).

Proof: Let DD={$d_1, d_2, \ldots, d_m$}, and let DD_j = DD – {d_j}, for j=1,2,…,m. Since DD defines a weak ETI, then by definition there must exist a set of records R $\subseteq T$ such that $|R| \geq \kappa n$ and DD defines the weak ETI on the set R. Assume to the contrary that there does not exist a set DD' that satisfies the given properties. Then for all j, DD_j does not define a weak ETI with the set of records R.

Let z_j = number of "0"s over records R and dimensions DD_j,

δ_j = probability of "0"s over records R and dimensions DD_j, $\delta_j = z_j / [(m-1)\,|R|\,]$,

z = number of "0"s over records R and dimensions DD, and

δ= probability of "0"s over records R and dimensions DD, $\delta = z / [\,m\,|R|\,]$

Then the assumption implies that $\delta_j > \varepsilon$ for all j=1,2,…,m.

z = number of "0"s over records R and dimensions DD

= $(1/(m-1))$ [number of "0"s over records R and dimensions $DD_1+DD_2+\ldots+DD_m$]

= $(1/(m-1))\,(z_1 + z_2 + \ldots + z_m)$

$\delta = z / [m\,|R|\,] = (1/(m(m-1)|R|))\,(z_1 + z_2 + \ldots + z_m) = (1/m)\,(\delta_1 + \delta_2 + \ldots + \delta_m)$

$> (1/m)\, m\varepsilon = \varepsilon$

This contradicts with the precondition that DD defines a weak ETI with the set of records R. □

Corollary of Lemma 1: if a set DD = $\{d_1, d_2, ..., d_m\}$ defines a weak ETI, then there exists a permutation of its defining dimensions $\{d_{p_1}, d_{p_2}, ..., d_{p_m}\}$ such that for all *j*, $1 \le j \le m$, the set $\{d_{p_1}, ..., d_{p_j}\}$ defines a weak ETI.

Proof: Obvious by induction on Lemma 1. □

Lemma 2: Given a set of *m* dimensions, their eligibility to be defining dimensions for a weak ETI can be tested with one pass over the database.

Proof: The test can be done by the following algorithm, which makes one pass over the database:

While scanning the database once, we keep *m*+1 counters $C_0, C_1, \ldots, C_m$ where C_i keeps track of the number of data points (records) that have *i* "1"s out of *m* candidate dimensions, $i = 0,1,2,\ldots, m$. From these counters, we find the maximum *t* such that: $\sum_{i=t}^{m} C_i \ge \kappa \cdot n$, where *n* is the total number of records in the database. With this *t* value, let $C_t' = \sum_{i=t}^{m} C_i - \kappa \cdot n$. Define $\delta = \frac{\left(\sum_{i=t+1}^{m} (m-i)C_i\right) + (m-t)(C_t - C_t')}{\kappa \cdot n \cdot m}$.

Then, the *m* candidate dimensions are eligible to be defining dimensions for a weak ETI if and only if $\delta \le \varepsilon$, and δ is referred to as the *error rate* of this weak ETI. The algorithm works as follows: it ranks all rows based on the number of "1"s out of *m* candidate dimensions, picks the top κn rows, and checks the probability δ of "0"s occurring among the κn rows over *m* candidate dimensions. Such ranking ensures that, if we picked any other set of κn rows, the probability of "0" occurring among those rows over *m* candidate dimensions would have been at least δ. Hence, if $\delta > \varepsilon$, no weak ETI exists. If $\delta \le \varepsilon$, these κn rows and *m* dimensions form a weak ETI, so a weak ETI exists. □

2.2 The Exhaustive Algorithm

The lemmas and corollary in the previous section suggest the following algorithm to find maximal weak or strong ETIs, which parallels the *a-priori* algorithm [AIS93, AS94]:

Exhaustive Growing Algorithm:

1. Find all dimensions d_i where the global count of "1"s is at least $\kappa n(1-\varepsilon)$. Each of these dimensions forms a singleton set $\{d_i\}$ which defines a weak ETI. We call each of these singleton sets a "seed". Set $i = 1$.
2. For every seed that contains *i* dimensions, grow it by adding a dimension to it so that the new seed still defines a weak ETI (obtained with one pass over the database as in the proof of Lemma 2). If one or more such dimension can be found, keep all of the new seeds (each of which contains *i*+1 dimensions).
3. Increment *i* and repeat step 2 until no more growing is possible.
4. (to find maximal strong ETIs) Among all seeds, pick those satisfying the strong ETI definition (done in one pass over the database in a straightforward way). Output the maximal strong ETIs.

The Corollary of Lemma 1 ensures that this algorithm will find all weak or strong ETIs.

For example, with the database shown in Table 3, κ=40% and ε=25%, the exhaustive growing algorithm produces the hierarchy shown in Figure 3. The only maximal strong ETI is {1,2,3,4}. There are two other maximal weak ETIs {1,4,5} and {6}, and 16 other non-maximal ETIs shown in the hierarchy.

1	2	3	4	5	6
	1	1	1		
1	1	1			
1	1		1		
1		1	1		
1			1	1	
1				1	
1				1	1
					1
					1
1					

Table 3

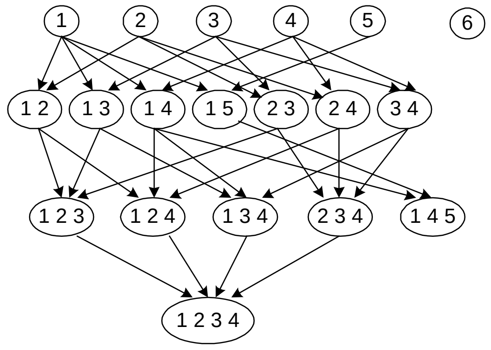


Figure 3: Hierarchy of ETIs

1	2	3	4	5	6
1	1	1	1	1	
1	1	1	1		
1	1	1	1		
1	1	1	1		1
					1
					1
					1
					1
					1

Table 4

2.2 Approximating Approach: Greedy Growing

The time complexity of the exhaustive growing algorithm is exponential in the maximum number of defining dimensions for ETIs. To reduce complexity, we modify it to become a greedy method which takes polynomial time and finds most of the ETIs. In practice, we believe that the chance of the approximate approach missing some of the ETIs is very low. We characterize these situations below.

The modifications to the algorithm will be done in three stages: in the first stage, we show how to reduce complexity, in the second stage we show how missed ETIs can be recovered by doing a few iterations. This second stage also sets us up for addressing the efficient identification of clusters of similar transactions. In the third stage, given in Section 3, we show that the overall scheme can be sped up dramatically using a sampling and validation framework.

2.2.1 Heuristics to Reduce Complexity

First, we make three changes to step 2 of the exhaustive growing algorithm:

i. When looking for a dimension to grow a seed, we only consider those dimensions that have been picked in step 1, i.e., dimensions that have enough “1”s to form singleton weak ETIs.
ii. When testing whether a dimension can be added to a seed, we require not only that the expanded seed still define a weak ETI, but also that the new dimension have at most ε probability of “0”s within the weak ETI.
iii. When two or more dimensions are found as possible candidate dimensions for seed growth, we only keep one. We throw away the old seed once it has been grown to a new seed.

The first two changes remove from consideration those dimensions that have too few “1”s globally or too few “1”s in a weak ETI. Even though those dimensions could potentially be in a weak ETI as dimension 5 in Table 4 (which could be part of a weak ETI {1 2 3 4 5}), they are not likely to make any interesting contributions to the result in real-world applications.

To implement the second change, we need to augment the algorithm given in the proof of Lemma 2. In addition to the counters C_0, C_1, …, C_m, we keep an extra set of $m+1$ counters Z_0, Z_1, …, Z_m, where Z_i keeps track of how many “0”s there are in the new candidate dimension d, over those data points that have i “1”s out of m existing candidate dimensions. These counters, together with the other counters C_0, C_1, …, C_m, can be updated in the same pass over the database. Then, the probability of “0”s along the new dimension d within the weak ETI is approximately the fraction:

$$\frac{(\sum_{i=t+1}^{m} Z_i) + \max(0, Z_t - C_t^{'})}{\kappa \cdot n}.$$

The third change ensures that the total number of seeds at any time would not exceed the total number of seeds we started with, reducing the exponential behavior to polynomial. When two or more candidate dimensions are found, we use the heuristics that picks one that causes the smallest error rate in the new weak ETI. Other heuristics are possible. Although the third change dramatically reduced the amount of time and memory required, it may cause some ETIs to be missed, such as {1,2,5,6} in Table 5 (with κ=30% and any ε). We address this issue by extending to an iterative scheme.

1	**2**	**3**	**4**	**5**	**6**
1	1	1	1		
1	1	1	1		
1	1	1	1		
1	1	1	1		
		1	1	1	1
		1	1	1	1
		1	1	1	1
1	1	1	1	1	1
1	1			1	1
1	1			1	1

Table 5

2.2.2 Iterative Scheme to Improve Approximation

We make two more changes here. First, after steps 1, 2 and 3 of the exhaustive growing algorithm (in which step 2 is modified as in section 2.2.1), we go through the entire database once and check if each row is covered by the ETIs we found. (A row *r* is covered by an ETI if the fraction of items in the ETI which are present in *r* is at least 1-ε.) For all rows that are not covered by any ETI, we put them together to form a smaller database, and perform steps 1, 2 and 3 again. We keep repeating this process until no more ETIs can be found. Each pass of steps 1, 2 and 3 is called a "round". Typically no more ETIs can be found after 2 or 3 rounds.

Secondly, starting with the second round, we replace the support threshold κ with a smaller value κ/λ, except at the very last step (corresponding to step 4 in the exhaustive growing algorithm), where we use the original κ value to pick out strong ETIs. Increasing λ would reduce the probability of missing ETIs, but at the same time increase running time and memory requirement. Typically, we use λ=2. Returning to our example of Table 5, this would enable the algorithm to discover the missing ETI {1,2,5,6}.

2.2.3 Summary of Greedy Growing Algorithm (GGA)

With all the changes above, we have converted the Exhaustive Growing Algorithm into the Greedy Growing Algorithm, which is summarized below.

1. Set of candidate dimensions = {all dimensions whose count of "1"s in the database is at least κ*n*(1-ε)/λ} (λ is initialized to 1 but will be set to 2 after the first round.)
2. If no candidate dimension exists, go to step 8.
3. Each candidate dimension forms a singleton seed.
4. Grow every seed by trying to add one best candidate dimension to it, while maintaining weak ETIs with support threshold κ/λ and error threshold ε.
5. Repeat step 4 until no seed can be grown further.
6. Add all fully-grown seeds to the set of potential ETIs.
7. Remove from the database all rows covered by any potential ETI. Set λ=2 and go to step 1.
8. Restore the original database, count the number of rows covered by every potential ETI, and remove those ETIs that are covered by fewer than κ*n* rows. Output all remaining ETIs.

2.2.4 Analysis

GGA is able to find all maximal strong ETIs in the database except:

i. ETIs consisting of a dimension whose global count of “1”s is smaller than $\kappa n(1\text{-}\varepsilon)/\lambda$;
ii. ETIs consisting of a dimension whose probability of “0”s within the ETI is greater than ε;
iii. ETIs consisting of fewer than $\kappa n(1\text{-}\varepsilon)/\lambda$ unique rows (a row is unique to an ETI if it does not belong to any other ETI), and no unique dimension (a dimension is unique to an ETI if it does not occur in any other ETI).

An example of each of the three cases is given in Table 6 (i) (ii) and (iii), with κ=40% and ε=35%. In Table 6 (i) and (ii), the GGA algorithm finds ETI {1,2,3,4}, but not ETI {1,2,3,4,5}. In Table 6 (iii), the GGA algorithm (with λ=2) finds ETIs {1,2,3,4} and {4,5,6,7}, but not ETI {3,4,5}, which consists only 1 unique row and no unique dimension. We believe that these three cases are not particularly important, especially when considering our primary application: identifying similar clusters of transactions.

1	**2**	**3**	**4**	**5**
1	1		1	1
1	1	1	1	
1	1	1	1	
1		1	1	
1	1	1	1	

(i)

1	**2**	**3**	**4**	**5**
1	1		1	1
1	1	1	1	
1	1	1	1	
1		1	1	
1	1	1	1	
				1
				1
				1
				1

(ii)

1	**2**	**3**	**4**	**5**	**6**	**7**
	1	1	1			
1	1	1	1			
1	1	1	1			
1	1	1	1	1		
1	1		1	1		
		1	1	1		
		1	1		1	1
			1	1	1	1
			1	1	1	1
			1	1	1	1

(iii)

Table 6

1	**2**	**3**	**4**	**5**
1	1	1	1	1
1		1	1	1
1	1	1	1	1
	1	1	1	1
1	1	1	1	1
1	1	1		
1	1	1		
1	1	1		
1	1			
1	1	1		

Table 7

Also, a side-effect of the iterative scheme of the algorithm is that it may find some non-maximal strong ETIs, as illustrated in Table 7, with κ=40% and ε=35%. In Table 7, both {1,2,3,4,5} and {1,2,3} will be identified as strong ETIs in successive iterations, although {1,2,3} is not strictly maximal (however, it covers many rows that do not overlap with the rows covered by ETI {1,2,3,4,5}). We can certainly add an additional step to remove such non-maximal ETIs, but we choose not to do so, because this side-effect turns out to be quite useful in real-world applications, as will be discussed in section 4.

Worst-case running time of the Greedy Growing Algorithm is $O(cdh^2)$, where

c = # of ETIs

d = average # of defining dimensions in ETIs

h = # of high-support dimensions (dimensions whose global count of “1”s is at least $\kappa n(1\text{-}\varepsilon)/\lambda$)

There are h seeds. Each seed has up to h possible candidate dimensions to grow to, at each of d growing steps. In the worst case, c iterations are needed to find all ETIs (one in each iteration). So $O(cdh^2)$ is the worst-case time complexity. However, in most cases, all ETIs can be found in 1 or 2 iterations, in which case the running time is only $O(dh^2)$.

Memory requirement is $O(hd+cd+D)$, where D is the total number of dimensions. We need $O(hd)$ space to store all seeds while they are being grown, $O(cd)$ space to store all ETIs found, and an additional $O(D)$ space as a buffer to count the number of “1”s in every dimension.

The database is scanned a total of $O(hd)$ times, one for each growing step of each seed. If database scanning becomes a bottleneck, we can reduce the number of scans to $O(d)$ by growing all seeds in parallel. This would increase the memory requirement to O(h^2d+cd+D) to store intermediate results needed to grow seeds.

When h is large and memory is tight, it is possible to limit to a constant the maximum number of seeds. In case there are more seeds than the maximum limit, we just throw away a random subset of seeds and grow the rest. This would bring down the time complexity to O(cdh); memory requirement is still O(hd+cd+D) but with a smaller constant factor. When doing so, the iterative scheme helps us recover most of the ETIs, but we have a higher risk of losing ETIs that have unique dimensions but fewer than $\kappa n(1\text{-}\varepsilon)/\lambda$ unique rows. These ETIs would have been found if we did not limit the number of seeds to grow. In our experience with real and synthetic data sets, we did not encounter cases where h >500. Hence we do not believe this is a concern.

3 Iterative Sampling and Validation

GGA makes multiple passes over the database and can be quite slow when the database is large. In this section, we present an extension to the algorithm by an iterative sampling and validation scheme which wraps around the original GGA algorithm. The sampling framework results in the algorithm SGGA.

3.1 Framework

Instead of running GGA on the entire database, we randomly sample two independent non-overlapping samples of the database, run GGA on one sample and validate the results on the other. Then, we return to the original database and check if any rows are still uncovered by the ETIs discovered so far. If so, we repeated the process on the remaining database until no more new ETIs can be discovered. This is illustrated in Figure 4.

3.2 Validation Schemes

We randomly sample the database into two independent non-overlapping samples *RS* and *VS*, and run GGA on the set *RS* to get a list of ETIs. Some of the ETIs may contain spurious dimensions which are purely random due to sampling. Also, some of the ETIs may be totally spurious.

Identifying Spurious Defining Dimensions of an ETI: An artifact of sampling to be avoided is adding a dimension to an ETI description if it happens to occur by chance with the other defining dimensions over the random sample *RS*. Suppose the first defining dimension of an ETI description was really random, but we observe that it occurs frequently along with the remaining defining dimensions in the ETI *S*. We identify this situation by first considering the ETI with the first defining dimension removed, then we count the number of points from the validation set *VS* satisfying the reduced ETI description. Determining whether or not the first defining dimension is

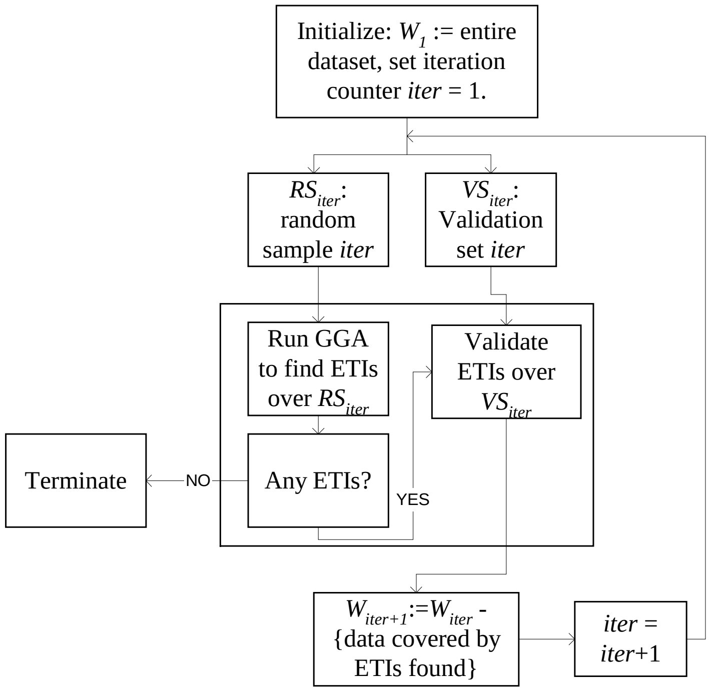

Initialize: W_1 := entire dataset, set iteration counter *iter* = 1.
RS_{iter}: random sample *iter*
VS_{iter}: Validation set *iter*
Run GGA to find ETIs over RS_{iter}
Validate ETIs over VS_{iter}
Any ETIs?
Terminate
NO
YES
W_{iter+1}:=W_{iter} - {data covered by ETIs found}
iter = *iter*+1

Figure 4: Flowchart of SGGA: sampling and validation scheme over greedy growing

spurious is done by comparing the probability of observing a "1" in the removed dimension over points satisfying the reduced ETI description with the probability of observing a "1" over the entire validation set. Let the removed defining dimension for ETI *S* be d_1. Let $p(d_1 \mid \{S - d_1\})$ be the probability of observing a "1" in dimension d_1 over the data in the validation set *VS* belonging to the reduced ETI consisting of the defining dimensions in *S* minus d_1. Let $p(d_1)$ be the probability of observing a "1" in dimension d_1 over the entire validation set *VS*.

If $p(d_1 \mid \{S - d_1\})$ is within 1 standard deviation from the value $p(d_1)$, then we consider dimension d_1 as a spurious defining dimension and remove it from the description of *S*. The standard deviation of $p(d_1)$ is given by: $s(d_1) = \sqrt{\frac{p(d_1)(1-p(d_1))}{|RS|}}$. Specifically, dimension d_1 is removed from the defining dimensions for ETI *S* if: $p(d_1) - s(d_1) \le p(d_1 \mid \{S - d_1\}) \le p(d_1) + s(d_1)$.

We do this for every defining dimension of every ETI, and prune out defining dimensions that appear spurious.

Identifying Spurious ETIs: To identify ETIs which are found by chance over a random sample from the database, *RS*, we consider the number of rows of *RS* which match the ETI as a random variable. Suppose the ETI has *NS* points in it and the random sample from *RS* consists of |*RS*| points. If the ETI truly exists in the database, then given another random sample *VS*, one would expect to observe approximately the same proportion of points belonging to the ETI in *VS*. If the proportion of points belonging to the ETI in *VS* is too low, then one may conclude that the ETI is spurious and needs to be adjusted. This is the motivation behind the mechanism for removing spurious ETIs found over random samples.

Let *NS*(*RS*) be the number of data points from *RS* which belong to the ETI. We view the proportion of points in a random sample belonging to *S* as a random variable *p*. We estimate the value of *p* over *RS* as: *p* = [*NS*(*RS*)]/|*RS*|. The standard deviation in this estimate is given by: $s = \sqrt{\frac{p(1-p)}{|RS|}}$.

We consider an ETI to be valid if the number of data points belonging to it over the validation set, *NS*(*VS*) is no less than a standard deviation of the expected number in *VS*: $\frac{NS(VS)}{|VS|} \ge \frac{NS(RS)}{|RS|} - \sqrt{\frac{p(1-p)}{|RS|}}$.

ETIs which do not satisfy the above criteria must be adjusted (if possible) so that they are valid. We do so by iteratively removing from an ETI the defining dimension with the least number of "1"s, until the ETI satisfies the criteria. In some cases, all dimensions may get removed (deleting entire ETI).

4 Applications and Evaluation Methodology

We have tested SGGA with sampling and validation schemes on both synthetic data and real-world data. For synthetic data, since we know the "true" ETIs in the database, we evaluate performance of our algorithm based on the number of true ETIs found and number of extra (false) ETIs found. For real data, we evaluate performance based on three applications: clustering, query selectivity estimation and a collaborative filtering prediction task. The clustering application consists of producing the ETIs, and then using them as initial models (clusters) for the EM statistical clustering algorithm [DLR77, CS96, BFR98]. Query selectivity estimation involves answering aggregate queries using a statistical model derived from the data initialized with the ETIs found by SGGA. Collaborative filtering or recommender systems use data about user preferences/behavior to predict additional topics, products, websites (items in general) that a user might like [MRK97,R*97,RV97,BHK98]. Collaborative filtering is applied in domains typically generating large stores of sparse {0,1} data (e.g. e-commerce), making ETIs particularly applicable.

4.1 Determining Initial Mixture Models from ETIs

ETIs are extremely effective in identifying structure in sparse {0,1} data. This structure can easily be exploited by using ETIs to form an initial statistical model of the underlying data. This initial model may then be further fit to data using an iterative optimization procedure such as the Expectation-Maximization (EM) algorithm [DLR77, CS96, BFR98]. EM estimates the probability density function of the data as a mixture of densities (binomial or multinomial distributions for discrete data). Each cluster corresponds to one component of the mixture model and specifies a distribution for its population. Within each cluster, for an observation (transaction) *x*, the probability of *x* being a member of cluster *C*, is proportional to $\Pr(x \mid C) \propto \prod_{i \in I} \Pr(x_i \mid C)$. Hence the model for each cluster is simply the probability of each attribute (item) being present, for all items in the database. Note that conditional independence within clusters is very different from assuming independence. The conditional independence assumption, typically applied when clustering discrete data in the statistics, learning, and pattern recognition literature [CS96, BFR98,DLR77], is a much more powerful model than the global independence model.

In the case of binary {0,1} data, we model each attribute with a binomial probability distribution within a cluster or component. In this case $\Pr(x_i \mid C)$ is simply $\Pr(x_i = 1 \mid C)$ if item *i* occurs in the record *x* or $(1-\Pr(x_i = 1 \mid C))$ if item *i* does not appear in *x*.

While EM provides many advantages for clustering data, including the fact that it produces a statistically meaningful model, the solution it produces is determined completely by the initial model. The state of the art of initializing EM over {0,1} data is via random restarts [MH98]. In high dimensions, EM falls prey to two problems in particular: clusters often go empty (have no data records assigned to them); and different clusters converge on the same distribution (the two clusters become identical). In either case, the algorithm effectively identifies fewer clusters in the data. Certainly this should happen if the user asked the algorithm to find more clusters than truly exist in the data. However, in practice these problems surface frequently when clustering high-dimensional sparse data when more structure actually exists. We demonstrate that ETIs consistently uncover more structure in the data than EM is capable of discovering with a multitude of repeated random initializations.

Each ETI is treated as a potential cluster. To construct a binomial model for each ETI (which will be the initial binomial distribution of that cluster) a pass is made over the data, collecting counts of items for all transactions that belong to the ETI. A transaction belongs to an ETI if the fraction of items in the ETI which are present in the transaction is at least $1-\varepsilon$. Since ETIs can overlap on items, some transactions could belong the multiple ETIs. A transaction *t* that belongs to a set of ETIs gets a fractional membership score in each ETI proportional to the number of defining dimensions in that ETI. Hence longer ETIs, when they match a transaction, claim "more" of the transaction than shorter ETIs. With a single pass over the database, degree of membership for each transaction to each ETI is computed (typically there is only one matching ETI) and the corresponding item counts for items appearing in the transaction are incremented by degree of membership. At the completion of the scan, the counts for each ETI are normalized to probabilities, and an initial cluster model for each ETI is produced.

This initial model is then refined via EM. If EM converges with stable clusters (few clusters have gone empty or converged to model the same set of records), it is an indicator that the ETIs indeed detected real structure in the data. Hence a quick way to compare a clustering solution based on ETIs with one obtained from a random starting model is to compare the number of non-empty clusters after EM converges. See results in Section 5.2.2.

4.2 Query Selectivity Estimation

Having derived a statistical model based on ETI initialization, one can evaluate the fit of the model to the data. An application measuring this directly is to estimate a `count(*)` query using the model and measure the difference between the model-based estimate and the true value of the query. A comparison is made between the statistical model estimated from ETI initialization and the statistical model estimated from random initialization.

Queries with *h* conjunctive items are generated with respect to a statistical cluster model as follows.

1. Cluster *C* is selected with probability given by the number of records associated with cluster *C* divided by the total number of database records.
2. Generate a distribution over each attribute $i \in I$ given by: $p_{i,C} = \dfrac{\Pr(x_i = 1 | C)}{\sum_{i \in I} \Pr(x_i | C)}$.
3. Select *h* items randomly according to the computed probabilities, yielding the `count(*)` query.

To use a probabilistic model to estimate an aggregate query, such as the number of shoppers who bought items 1, 5, and 16 together, we simply compute the probability of the event that items 1, 5, and 16 occur together for every cluster and adding them together weighted by the size of each cluster. Not only is this an extremely fast operation involving access only to the model and no access to the data, but it can also be a very accurate estimator for an underlying model that fits the data well.

4.3 Collaborative Filtering Prediction

Succinctly, the collaborative filtering prediction task is the following: given a set of items, predict other items that typically occur with the given set [MRK97,R*97,RV97]. For example, given that a set of web-pages have been browsed by a user, predict other web-pages are likely to be browsed by the user in this session. Or, given products in a shopper's basket, what other items will the shopper likely place in their basket also?

By using the given set of items (viewed as a partial record, since more items may be added), it is possible to associate the record with a cluster. Once the record has been associated with a cluster (possibly with fractional membership), it is possible to predict other items that would most likely appear with the given set based on the values of $\Pr(x_i \mid C)$. Intuitively, an item *i* with a large corresponding value of $\Pr(x_i \mid C)$ would be likely to occur along with the given itemset. See [BHK98] for a detailed description.

4.4 Comparison with Traditional Frequent Itemsets

Probabilistic clustering algorithms (such as EM) do not rely upon harsh cluster membership criteria, but assign a data point to all clusters with a different probability of membership. In the standard frequent itemset framework, a data point (row) must contain *all* of the items in the itemset to support it. This "all-or-nothing" behavior often results in poor initial models for EM-clustering and structure that exists in the data (capable of being located by a EM) is simply missed. For example, consider the binary data set shown in Table 8. Assume that each row in the table represents 100 rows in the actual data. If minimum support is such that 300 rows are needed for a frequent itemset, the only one existing in this example is

{A,B}. But allowing for 34% error in ETIs, we get: {A,B}, {D,E,F} and {G,H,I}. If we simply used the result of standard frequent itemsets and generated an initial cluster model, we would have one cluster corresponding to the itemset {A,B} and a single cluster corresponding to "not {A,B}". EM would not be capable of further segmenting the data based on the collections {D,E,F} and {G,H,I}. Obvious, using ETIs, this structure is identified and the final mixture model more accurately fits the given data.

A	B	C	D	E	F	G	H	I
1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1

Table 8: Frequent Itemset

5 Results

5.1 Results on Synthetic Data

We generated synthetic data with parameters and default values shown in Table 9. We generated more than 70 different datasets while perturbing some parameters away from default values, and studied the response of our algorithm in terms of running time and quality of ETIs found (i.e., number of true ETIs found and number of extra ETIs found). Unless otherwise noted, we ran the Greedy Growing Algorithm with Sampling and Validation, setting support threshold κ=1%, error threshold ε=20%, sample size of *RS* = 6000 and sample size of *VS* = 6000. With all data parameters set to default in Table 9, the algorithm finds all 50 true ETIs, and no extra ETIs, in about 540 seconds on a Pentium II Xeon processor.

Parameter	Default
Total number of ETIs	50
Approximate number of points in each ETI	10,000
Approximate support of each ETI	2%
Total number of dimensions	5,000
Number of defining dimensions in each ETI	5,6,7, or 8
Probability{defining dimension being "1"}	92.5%
Probability{non-defining dimension being "1"}	0.01%
Number of random dimensions	0
Probability{a random dimension being "1"}	0
Defining dimensions overlap in different ETIs	None

Table 9: Synthetic Data Parameter Settings

5.1.1 Adding Random dimensions

Figure 5 shows the quality of ETIs found when we add 10, 25, 50 and 100 random dimensions to the data. We vary the probability of a random dimension being "1" from 0.25% to 2%. There is no major change in running time. All 50 true ETIs are found in every case, but the number of extra ETIs varies. This is reasonable and expected behavior: as the random dimension probability increases beyond the support

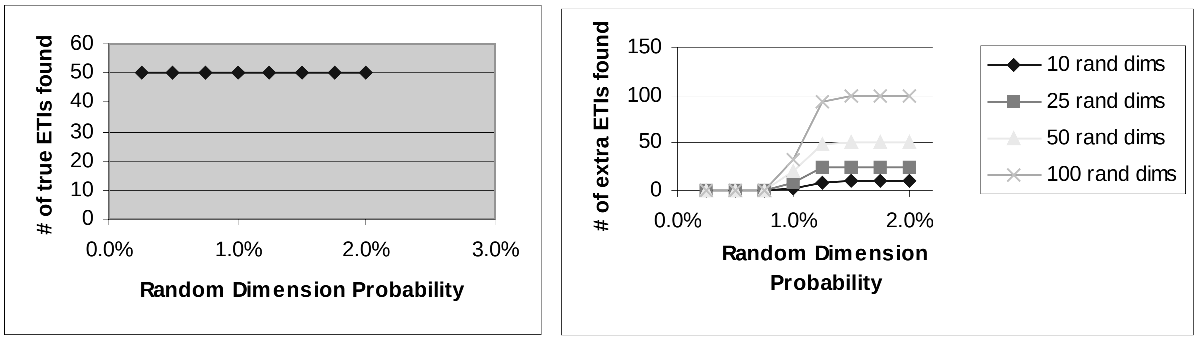


Figure 5: Adding Random Dimensions

threshold κ=1%, more and more random dimensions are detected as forming singleton ETIs themselves.

5.1.2 Varying Non-Defining Dimension Probability

As we vary the probability of non-defining dimensions from 0% to 2%, running time and ETI quality are shown in Figure 6. As the non-defining dimension probability is increased, number of high-support dimensions becomes larger, and data becomes denser. Both contribute to the increase in running time. As the non-defining dimension probability increases beyond the support threshold κ=1%, real ETIs become hard to identify. So the quality of ETIs found is deteriorated.

There is a "bump" in running time when iterative sampling and validation is used, because 2 iterations are made here, as opposed to 1 iteration elsewhere. If we disable validation and use only 1 iteration everywhere (but still use sampling), then the bump disappears and running time decreases slightly. However, this is at the cost of losing some true ETIs, as shown in the figure.

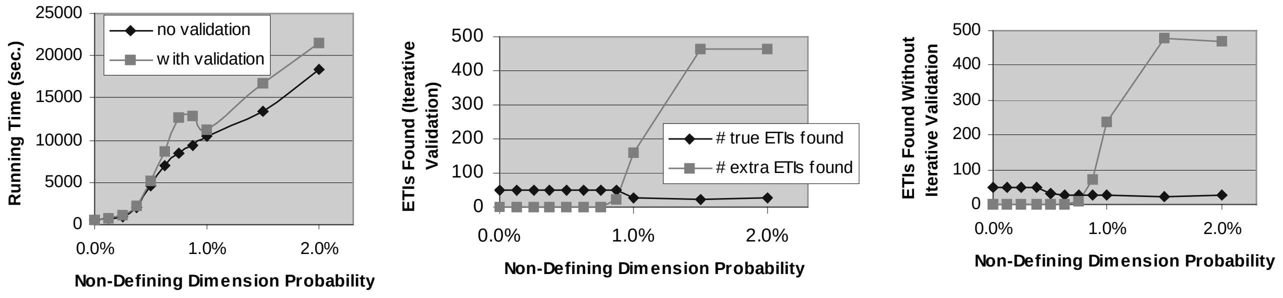


Figure 6: Effect of varying non-defining dimension Probability

5.1.3 Varying Degree of Overlap in Defining Dimensions

Real datasets could have ETIs which overlap in defining dimensions. To simulate this, we generated datasets with different degrees of overlap in defining dimensions. We define the degree of overlap by the sum of number of defining dimensions for every ETI divided by the total number of different dimensions used in one or more ETIs. For example, if every ETI has 6 defining dimensions and there are 50 ETIs, but only 100 different dimensions are used as defining dimensions, then the degree of overlap is $\frac{6\times 50}{100}=3$. A degree of 1 means no overlap, and a larger degree means more overlap. Figure 7 shows the quality of ETIs found on datasets of different degrees of overlap. It can be seen that, our algorithm performs reasonably well for overlap up to degree 10, then deteriorates gradually. This is acceptable for practical purposes, since from our experience degree of overlap in real data rarely goes this high.

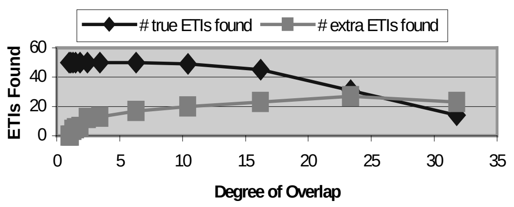


Figure 7: Varying Degree of Overlap

5.1.4 Scalability Experiments

Figure 8(left) shows the running time as we increase the average number of defining dimensions of each ETI. With no overlap, this also means that we increase the total number of high-support dimensions. As can be derived from Section 2, worst-case running time should grow cubically as we increase this

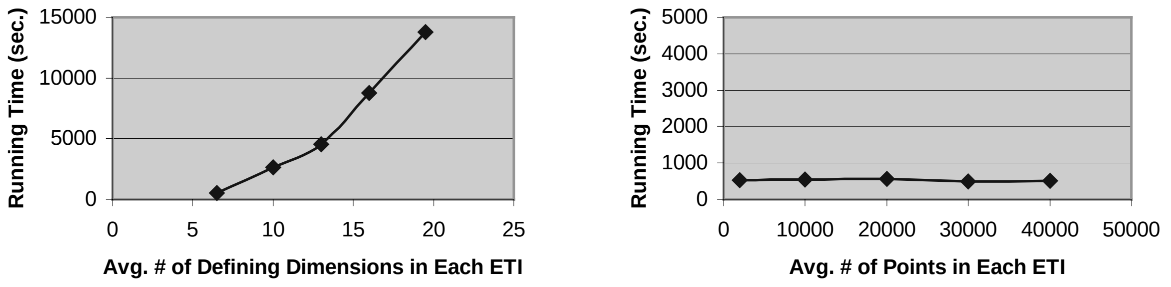


Figure 8: Scalability Results

number. The figure shows the actual result is much better than the worst-case scenario. Figure 8(right) shows the running time as we increase the total number of data points in each ETI, which also increases the database size. Since we use sampling, there is no major change in running time as the database size increases. In terms of qualitative performance over all these data sets, all 50 true ETIs and no extra ETIs are found for all of these experiments.

5.1.5 Parameter Sensitivity Experiments

We show that our method is fairly insensitive to parameter settings in Appendix B. The 6 charts provided in Figures B.1 and B.2 demonstrate this for the primary parameters.

5.2 Results on Real Data

Error Tolerant Itemsets are particularly suited to identifying structure for cluster initialization over sparse {0,1} high-dimensional data. In this respect, we evaluate ETIs in this application over 4 real sparse {0,1} databases. ETI-initialized cluster models are evaluated in comparison to randomly-initialized cluster models in 3 areas: 1) preservation of the number of clusters in the model (see Section 4.1); 2) ability of the model to approximate `count(*)` queries over the database (see Section 4.2); 3) a collaborative filtering prediction task (see Section 4.3).

5.2.1 Real Databases

Web-1: This database consists of the browsing behavior of 516,511 users over 218 web-page categories. This data was obtained from a major internet service provider/web portal. Viewed as a data table, this databases consists of 516,511 rows and 218 columns. An entry of 1 in row *h* and column *j* indicates that user *h* visited a web-page in category *j*. Since clustering is much more expensive that ETI extraction, when clustering this database (initialized with either ETIs or randomly), a random sample of 100,000 users was used.

Web-2: This database consists of the browsing behavior of 602,479 users over 9822 websites. This data set was obtained from one of the largest web-content providers on the internet. Prior to clustering this database, a random sample of 100,000 users was obtained. Clustering was done using the 5000 most frequent items out of the total 9822. After the cluster model was constructed, binomial distributions were estimated over the entire set of 9822 items for each cluster.

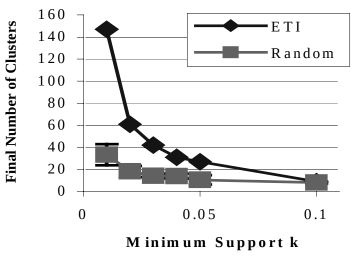
Product-Purchase-1: This database consists of the products purchased by 29989 users. The number of possible products is 297 and were obtained from a small e-commerce site that sells software products. Sampling was not employed in clustering this database.

Product-Purchase-2: This database consists of purchasing behavior of 703,510 customers purchasing a subset of 32,301 products. This data set is obtained from a major software/hardware retailer. Similar to Web-1 and Web-2, clustering was done over a random sample of 100,000 customers. In addition the cluster model was computed over the most frequently occurring 5,000 items. Once the cluster model was obtained, binomial distributions were estimated over the full set of 32,301 items for each cluster.

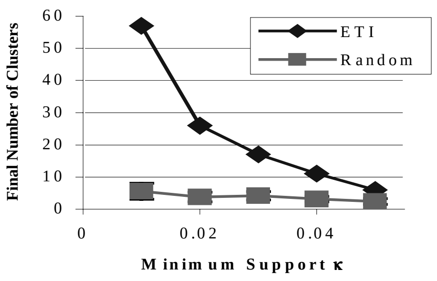
5.2.2 Number of Clusters

For a given minimum support value κ, SGGA was applied to the database. Upon termination, the ETIs constructed were used to initialize the binomial cluster model. EM [BFR98] was then applied to the database to refine the given initial model. EM was also applied to the database from 5 random initial binomial cluster models. The number of clusters in the random models was initially set to the number of ETIs returned by SGGA. We then compare the number of clusters at EM convergence from the ETI-initialized model with the average number of clusters from the 5 random initial models. Results are summarized for the 4 datasets in Figure 9. The random initial cluster models had a number of clusters equal to the corresponding ETI-value. *No clusters went empty when initialized via ETIs.* Random values are averages over 5 random initial models, error bars are 1 standard deviation.

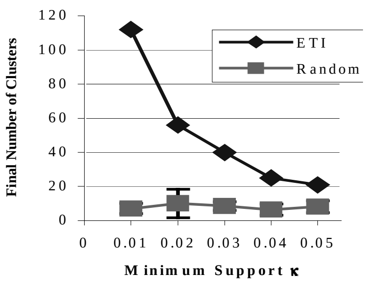
Note that when EM is initialized via ETIs, no empty clusters are observed when EM converges. In contrast, when initializing EM with a random cluster model with the same number of clusters as the corresponding ETI model, *EM converges with many empty clusters*. For minimum support κ = 0.01, ETI-initialized models uncovered as many as 16 times as many clusters as randomly initialized models on the Product-Purchase-1 database. On average over the 4 databases tested, for minimum support κ = 0.01, ETI-initialized models uncovered 9.6 times as many clusters as the randomly initialized models.



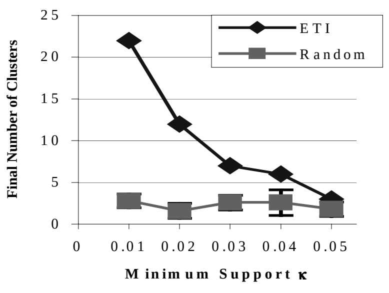
(a) Web-1



(b) Web-2



(c) Product-Purchase-1



(d) Product-Purchase-2

Figure 9: Final number of clusters after EM converges

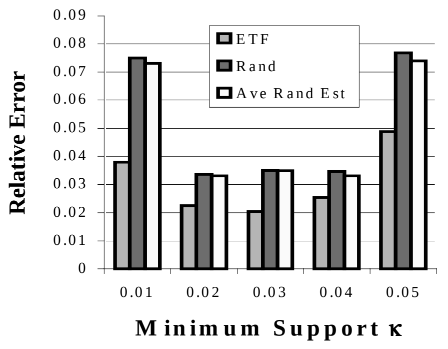
5.2.3 Query Selectivity Estimation

Similar to Section 5.2.2, for a given minimum support value κ, SGGA was applied to the database. The ETIs constructed were used to initialize the binomial cluster model as in the previous section. As above, we compare against initializing EM with 5 different random initial binomial cluster models. The number of clusters in the random models was initially set to the number of ETIs returned by SGGA. We then randomly generated 50 `count(*)` queries with 2 items in the "where" clause, based on the ETI-intialized model as described in Section 4.2. We did experiments with 2 and 3 conjunctions in the "where clause". Having more than 3 conjuncts usually results in most queries returning zero or a very small number for the count result. When a query has a small true result (e.g. less than 5), it is known that the estimate of these small queries with statistical models is poor [SFB99]. Over these high-dimensional, sparse databases, we have observed very small true result sizes for `count(*)` queries with more than 3 attributes. Estimates of `count(*)` queries with only 1 attribute in the "where" clause were deemed uninteresting as these queries can obviously be effectively estimated without modeling dependency among attributes. For very long queries, a constant zero is a good estimator, hence uninteresting. Suppose the "where" clause of the query of interest consists of items i and j. The probability that these two items appear in a given cluster C is $\Pr(x_i \mid C) \cdot \Pr(x_j \mid C)$. The number of records in cluster C containing both items i and j is the total number of records in C times $\Pr(x_i \mid C) \cdot \Pr(x_j \mid C)$. Let $N(C)$ be the number of records in cluster C, the query :

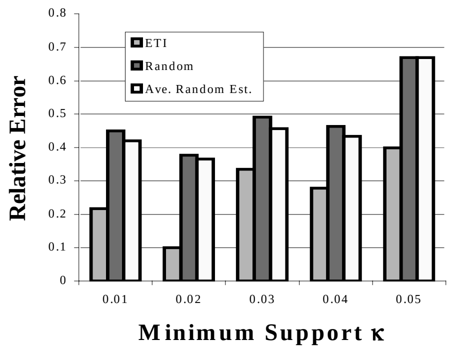
```
        "select count(*) from DB.Table where (i = 1) and (j = 1)"
```

is then simply approximated by the cluster model by: $\sum_C N(C) \cdot \Pr(x_i \mid C) \cdot \Pr(x_j \mid C)$.

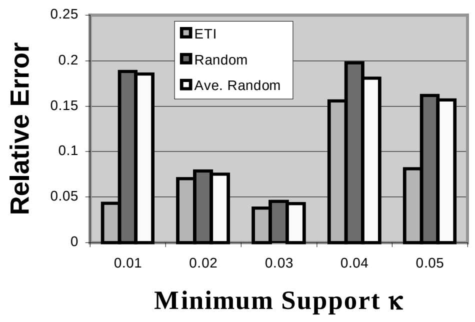
We tested the ability of cluster models initialized with ETIs versus 5 randomly initialized models. Each initial cluster model had the same number of clusters as discovered by algorithm SGGA. Average results over 50 two conjunct queries (see Section 4.2 for description of query generation) for the ETI-initialized model are given in Figure 10 noted as "ETI". Results over 3-conjunct queries yielded similar results. Average results over the same 50 queries and over the 5 randomly initialized models are noted in Figure 10 as "Random". The method noted as "Ave. Random Est." in Figure 10 refers to the following: for each individual query, we obtain the 5 approximations with the random models, then average these to



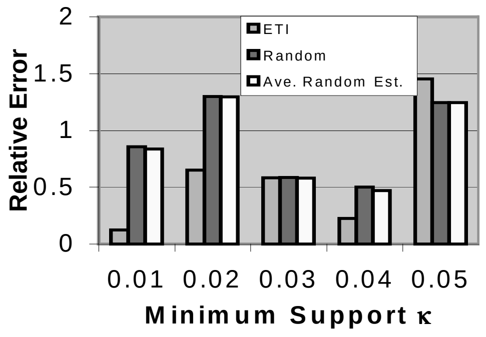
(a) Web-1



(b) Web-2



(c) Product-Purchase-1



(d) Product-Purchase-2

Figure 10: Average query selectivity relative error for 4 real databases.

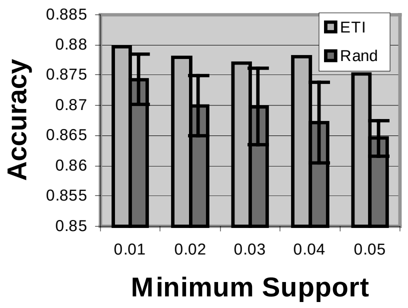
get an approximate result for the specific query in question. Relative error in Figure 10 is obtained by taking the absolute value of the difference between the true result and the approximation and dividing by the true result. Over all databases, we discarded queries where the true result was less than or equal to 5.

Over all 4 databases the ETI-initialized models were better query estimators than the randomly initialized models. Except in over the Product-Purchase-2 database with minimum support $\kappa = 0.05$. For this value of minimum support, the number of ETIs found was 3 and the average number of clusters from random initialization was 1.8. Hence these models are very similar. We note that data which does not match any ETI does not contribute to the initial cluster model. So in this case where there are only 3 ETIs we conjecture that there is a non-trivial portion of the data space which was not appropriately modeled by the ETI-initialized cluster model and it should be expected that multiple randomly initialized cluster models would capture some (if not most) of this space.

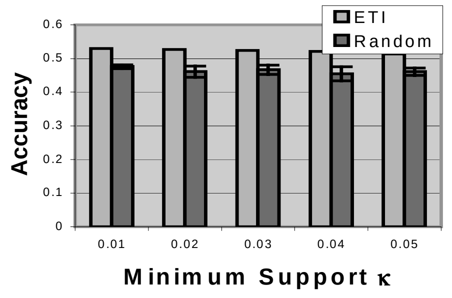
5.2.4 Collaborative Filtering Prediction

We tested the utility of cluster models initialized via ETIs in the collaborative filtering task of predicting other items that a record may contain based on a set of given items occurring in the record. We compared the predictive accuracy of ETI-initialized cluster models and randomly initialized models using a subset of data not used on the clustering process. Recall that for databases Web-1, Web-2 and Product-Purchase-2, clustering was performed over a random sample of 100,000 records. For these databases, an additional 10,000 records were held out and used to score the predictive accuracy of the models. For the Product-Purchase-1 database, having 29989 records, 10% of the records were set aside for scoring and the cluster models were built on the remaining 90%.

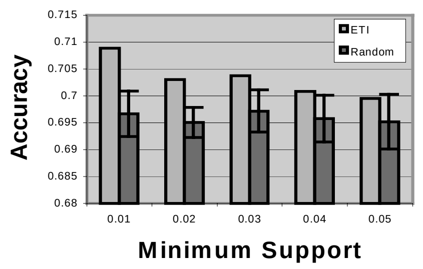
The scoring scheme works as follows. For each record in the hold-out set, we remove one of the items occurring in the record. Call this record with 1 item removed the *partial record*. The record is assigned to clusters (with fractional membership) based on the remaining items that appear in the record. Let $w(C)$ be the fractional membership assignment for cluster C. The $w(C)$-values satisfy: $\sum_C w(C) = 1$. For every missing item in the partial record (including the item removed), we ask for a prediction for that item. For missing item m, the prediction is given by:



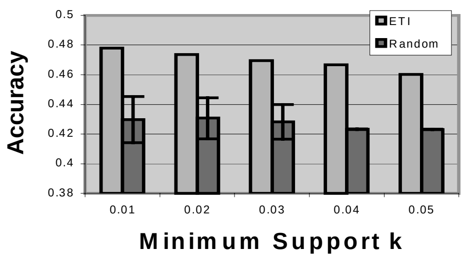
(a) Web-1



(b) Web-2



(c) Product-Purchase-1



(d)Product-Purchase-2

Figure 11: Collaborative filtering predictive accuracy

$$p(m) = \sum_{C} w(C) \cdot \Pr(x_m \mid C) .$$

The missing items are sorted in descending order by their *p*(*m*) values. If the item which was explicitly removed to form the partial record occurs in the top 10 of the sorted list, the score is incremented by 1. Accuracy is then the score divided by the number of items held out (to form partial records) over the entire hold-out set. Accuracy scores are summarized for the 4 databases in Figure 11 for ETI-initialized cluster models and average collaborative filtering predictive accuracy over 5 randomly initialized cluster models. Error bars associated with the random results are 1 standard deviation.

6 Generalization to Categorical Data

Note that the definition of our algorithm does not fundamentally rely on the assumption that the data is binary. It can be generalized to handle categorical data as well. For efficiency purposes, we assume that each attribute has one value that is somehow distinguished as the "default" value. In transaction data, that value is implicitly the "zero" value. Assuming that the "default" value for each attribute is pre-dominant (i.e. the data can be efficiently represented in sparse format), we can generalize our algorithm to count combinations of attribute values that do not involve the "default" value for each attribute. Note that the treatment is almost identical algorithmically as thinking of each non-default value of a multi-valued categorical attribute as a new binary attribute. As long as we have an efficient mechanism for counting frequent combinations of attribute values, the algorithm carries through and can be applied to finding clusters in categorical data. This is similar to the algorithm presented in [SA96].

7 Related Work

Frequent itemsets were first developed by Agrawal et al. in the *a-priori* algorithm for association rule mining [AIS93, AS94, AMSTV96]. The key optimization in finding frequent itemsets was based on the fact that, if an itemset of length *m* has enough support, then any of its subset of length *m-1* must also have enough support. This property enables building frequent itemsets in a bottom-up manner. As we introduce the notion of errors into the definition of frequent itemsets, this property no longer holds. A similar but weaker property exists as discussed earlier, but it was not suffieicient to ensure fast discovery of error-tolerant frequent itemsets, hence the additional optimization schemes developed in this paper.

One problem that arises in *a-priori* is that the algorithm scales exponentially with longest pattern length. Many variants have been proposed to address this issue. Zaki et al. [ZPOL97] developed the algorithms MaxEclat and MaxClique which "look ahead" during initialization so that long frequent itemsets are identified early. Bayardo [B98] presented an optimized search method called Max-Miner that prunes out all subsets of long patterns of frequent itemsets that are discovered early. Gunopulos et al. [GMS97] suggested iteratively extending a working pattern until failure, using a randomized algorithm, which is similar to the idea we used in our algorithm to grow itemsets in a greedy fashion.

Srikant and Agrawal [SA96] extended the *a-priori* algorithm to the domains of non-binary values, such as categorical or continuous values. They discussed ways to map intervals and categorical values to integers that effectively converted the non-binary databases into binary ones, where the original *a-priori* algorithm could be used. The same idea applies to our algorithm as well, so our algorithm can also be used on non-binary databases. We are not aware of problem formulations that introduce the notion of error-tolerance, and we believe that efficient algorithms presented in previous work rely on itemsets being exact.

Much work has been done in automatic clustering methods. Traditionally, clustering is done by finding a set of centroids in a high-dimensional space [CS96, DLR77, NH94, ZRL96], and cluster membership is determined by some distance function to the centroids. This leads to cluster shapes similar to spheres. Later work by Guha et al. [GRS98] was able to handle arbitrarily-shaped clusters by using several representative points to define a cluster. They also used a sampling scheme to reduce I/O costs. Recognizing that most clusters are defined on subspaces rather than the entire high-dimensional space, Agrawal et al. [AGGR98] presented a method to build subspace clusters in a bottom-up way, using the property that if a collection of points form a cluster in a *k*-dimensional space, then must also form a cluster in all of its *(k-1)*-dimensional subspaces (where a cluster is defined as a dense region in the subspace). As presented in this paper, our algorithm to find error-tolerant frequent itemsets may be used independently as an efficient subspace clustering algorithm, with a more general definition of clusters than that used in [AGGR98]. It can also be used as an effective initialization method for existing cluster refinement algorithms such as EM.

Discrete clustering algorithms, as opposed to generalized frequent itemsets, include CACTUS [GGR99], STIRR [GKR98], and ROCK [GRS99]. The first two require the computation of a similarity matrix between all attributes (items), which takes $O(d^2)$ time (d = number of attributes). CACTUS uses a more efficient refinement method on the computed similarities and hence is faster. ETIs are a generalization to frequent itemsets, and cluster initialization is just an application. In fact, ETIs could be used as a preprocessor to these algorithms, reducing the similarity matrix needed and hence alleviating the primary bottleneck. We plan to test this application in future work. ROCK requires a distance metric between transactions and is cubic in their number. In general, clustering is much more time consuming than extracting ETIs.

8 Concluding Remarks

We have presented a generalization to the standard frequent itemset problem and an efficient and scalable algorithm to find error-tolerant frequent itemsets (ETIs). ETIs describe simpler and more intuitive frequent structures in data. Starting with an exhaustive approach which guarantees that all such ETFs will be found, we developed an efficient approximation which runs in polynomial time and produces good results. We demonstrated that this method can be used as a fast initialization method for clustering algorithms such as EM, and generates far more stable models than existing techniques. Query selectivity estimation and collaborative filtering are two other useful applications of our algorithm.

One possible future direction is to study the extension of the algorithm to continuous-valued domains. Approaches suggested in [SA96] are applicable, but there may be other methods as well. We are currently exploring its use to identify which attribute similarities to focus on in algorithms driven by such similarity matrices [GGR99,GKR98,GRS99]. It would also be interesting to explore other sampling schemes to improve performance. Furthermore, much of the previous work that made use of traditional (error-free) frequent itemsets can now be reconsidered in this new framework.

Acknowledgements

We gratefully acknowledge Jeong Han Kim and Dimitris Achlioptas for discussions and assistance regarding the probability of error-tolerant itemsets occurring by chance. We thank Ilya Vinarsky for help with implementation and experiments.

9 References

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos and P. Raghavan, Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proc. of the ACM SIGMOD Conf.*, 1998.

[AIS93] R. Agrawal, T. Imielinski and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proc. of the ACM SIGMOD Conf.*, 1993, pp. 207-216.

[AMSTV96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo. Fast Discovery of Association Rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 307-328, AAAI Press, Menlo Park, CA, 1996.

[AS94] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases*, 1994.

[B98] R. J. Bayardo Jr. Efficiently Mining Long Patterns from Databases. In *Proc. of the 1998 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 85-93, 1998.

[BFR98] P. S. Bradley, U. M. Fayyad and C. Reina. Scaling EM (Expectation-Maximization) Clustering to Large Databases. Technical Report MSR-TR-98-35, Microsoft Research, 1998.

[BHK98] J. Breese, D. Heckerman and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Technical Report MSR-TR-98-12, Microsoft Research, 1998.

[CS96] P. Cheeseman and J. Stutz. Bayesian Classification (AutoClass): Theory and Results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurasamy (eds.) *Advances in Knowledge Discovery and Data Mining*, pages 153-180. AAAI Press, Menlo Park , CA, 1996.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society B*, 39:1-38, 1973.

[FI93] U.M. Fayyad and K.B. Irani. "Multi-interval Discretization of Continuous-valued Attributes for Classification Learning." *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence. IJCAI-93*: Chambery, France (1993).

[GGR99] V. Ganti, J. Gehrke, and R. Ramakrishnan. "CACTUS –Clustering Categorical Data Using Summaries". *Proc. of KDD-99,* ACM Press, 1999.

[GKP89] R. L. Graham, D. E. Knuth and O. Patashnik. *Concrete Mathematics*. Addison Wesley, Reading, MA, 1989.

[GKR98] D. Gibson, J. Kleinburg, and P. Raghavan. "Clustering categorical data: an approach based on dynamical systems". *Proc. VLDB-98,* pp. 311-323. 1998.

[GMS97] G. Gunopulos, H. Mannila and S. Saluja. Discovering All Most Specific Sentences by Randomized Algorithms. In *Proc. Of the 6th Int'l Conf. On Database Theory*, pp. 215-229, 1997.

[GRS98] S. Guha, R. Rastogi and K. Shim. CURE: An efficient algorithm for clustering large databases. In *Proceedings of the ACM SIGMOD conference*, 1998.

[GRS99] S. Guha, R. Rastogi, K. Shim. “A Robust Clustering Algorithm for Categorical Attributes”. *Proc. ICDE-99*, IEEE Press, 1999.

[MH98] M. Meila and D. Heckerman. An Experimental Comparison of Several Clustering and Initialization Methods. *Technical Report MSR-TR-98-06*, Microsoft Research, Redmond, WA, 98052, 1998.

[MRK97] B. Miller, J. Riedl and J. Konstan. Experiences with GroupLens: Making Usenet Useful Again. In *Proc. USENIX 1997 Tech. Conf.*, pp. 219-231, Anaheim, CA, 1997.

[NH94] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the International Conference on Very Large Databases*, 1994.

[RG99] R. Ramakrishnan and J. Gehrke. *Principles of Database Management (2nd Edition).* 1999.

[R*97] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnew. In *Proc. ACM 1994 Conf. Computer Supported Cooperative Work*, pp. 175-186, New York. ACM, 1997.

[RV97] P. Resnick and H. Varian. Recommender Systems. *Comm. of the ACM*, 40(3):56-58, 1997.

[SFB99] J. Shanmugusundaram, U. M. Fayyad and P. S. Bradley. Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions. In *Proc. 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pp. 223-232, 1999.

[SA96] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proceedings of the ACM SIGMOD Conference*, 1996.

[ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara and W. Li. New Algorithms for Fast Discovery of Association Rules. In *Proc. of the Third Int’l Conf. On Knowledge Discovery in Databases and Data Mining*, pp. 283-286, 1997.

[ZRL96] T. Zhang, R. Ramakrishnan and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference*, 1996, pp. 103-114.

[Z49] G.E. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press, Inc, 1949.

Appendix A:

On Chance Occurrences of Error-Tolerant Frequent Itemsets

Lemma: Assume a binary $N \times D$ sparse matrix over {0,1} is generated at random with the probability that an entry is 1 is *p*. Then the probability of a frequent error-tolerant item set with *r* items appearing in it with support κ and error ε is not greater than $\binom{D}{r}\binom{N}{\kappa N} p^{(1-\varepsilon)\kappa N r}(1-p)^{\varepsilon \kappa N r}$.

Proof: Let *A* be the event that there exists a submatrix with κ*N* rows and *r* columns with (1-ε)% of the entries having value 1. Let *B* be the random variable counting the number of such submatrices occurring over the $N \times D$ sparse matrix over {0,1}, then we obviously have:

$$P(A) \le E[B].$$

Here $E[B]$ is the expected value for *B* taken over all such $N \times D$ sparse matrices over {0,1}, where 1 is generated at random with the probability *p*. The right-hand-side of the above inequality is given by the number of $\kappa N \times r$ submatrices in an $N \times D$ matrix times the probability of observing the given structure. The number of ways to choose a submatrix with κN rows and r columns from the $N \times D$ matrix is

$$\binom{\kappa N}{N} \cdot \binom{r}{D}.$$

The probability that (1-ε)% of the entries have value 1 is given by:

$$p^{(1-\varepsilon)\kappa N r}(1-p)^{\varepsilon\kappa N r}.$$

□

We next discuss the case when the assumption that the column densities of the data matrix follow a Zipf distribution (i.e. the probability of observing a 1 in column j is proportional to (1/j)). Now, when we choose a sub-matrix of size $\kappa N \times r$ we can approximate (overestimate) the probability of observing a 1 over the sub-matrix by the probability of observing a 1 in the column with minimal index in the sub-matrix (this column has maximum probability). We consider the expected value of the minimal index over all such sub-matrices of size $\kappa N \times r$, where the probability of choosing a given column for the sub-matrix is uniform.

Let X be the random variable which is the value of the minimum index over the r columns chosen. What is the expected value of X over all possible ways of choosing r distinct columns from D? Once we know the expected value of X, we can use the Zipf assumption to obtain a value for p and apply the lemma above.

By definition, the expected value of X is:

$$\begin{aligned} E[X] &= \sum_{x=1}^{D} x \cdot \Pr(x) \\ &= \sum_{x=1}^{D} x \cdot \left[\Pr(\{0,\ldots,(x-1)\}\text{ are not chosen}) \cdot \Pr(x\text{ is chosen} \mid \{0,\ldots,(x-1)\}\text{ are not chosen})\right] \\ &= \sum_{x=1}^{D} x \prod_{f=0}^{x-2}\left(1-\frac{r}{D-f}\right)\cdot\left(\frac{r}{D-x+1}\right) \\ &= \frac{D+1}{r+1} \end{aligned}$$

Hence, on average, the minimum column index that we pick for the submatrix with κN rows and r columns is $(D + 1)/(r + 1)$. By Zipf, we can make the assumption that the probability of observing a 1 in this column is one over the column index $(r + 1)/(D + 1)$. We can then over-estimate the probability of observing a 1 in the entire submatrix by assuming this probability is constant and equal to $p = (r + 1)/(D + 1)$ and apply the result of the lemma.

Appendix B: Parameter Sensitivity Results

Figure B.1 shows the effects of changing the error threshold ε, while keeping the support threshold κ at 1.5%. There is no major change in running time, but the quality of ETI is very bad when the error threshold ε is set too low. Since the true ETIs have an average of 5 defining dimensions, setting ε<20% would mean that a row must contain a "1" in every defining dimension of an ETI in order to be considered "covered". This is too harsh and causes many ETIs to be missed.

Figure B.2 shows the effects of changing the support threshold κ, while keeping the error threshold ε at 20%. Quality of ETIs is best when κ is below but not far from the real support of ETIs. Smaller κ leads to longer running time, because the total number of high-support dimensions is larger. However, when iterative sampling and validation is used, there is a large bump in running time when κ coincides with the real support of 2%. This is because many iterations of sampling and validation are happening here, with each iteration generating a few more new ETIs based on different samples of the data. This is an instability which can be removed by terminating after one iteration without using validation.

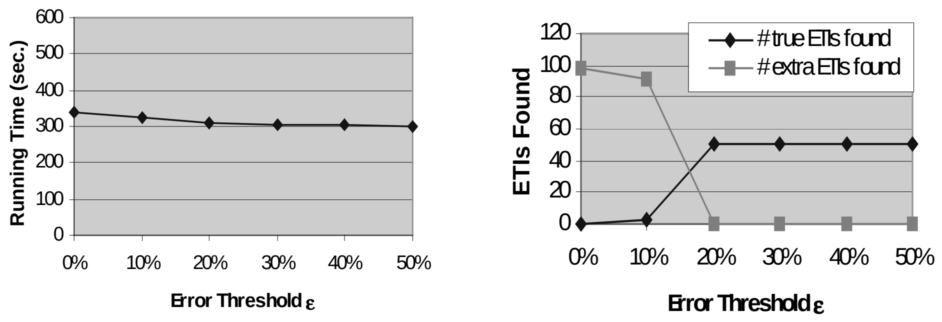


Figure B.1: Effects of Varying Error

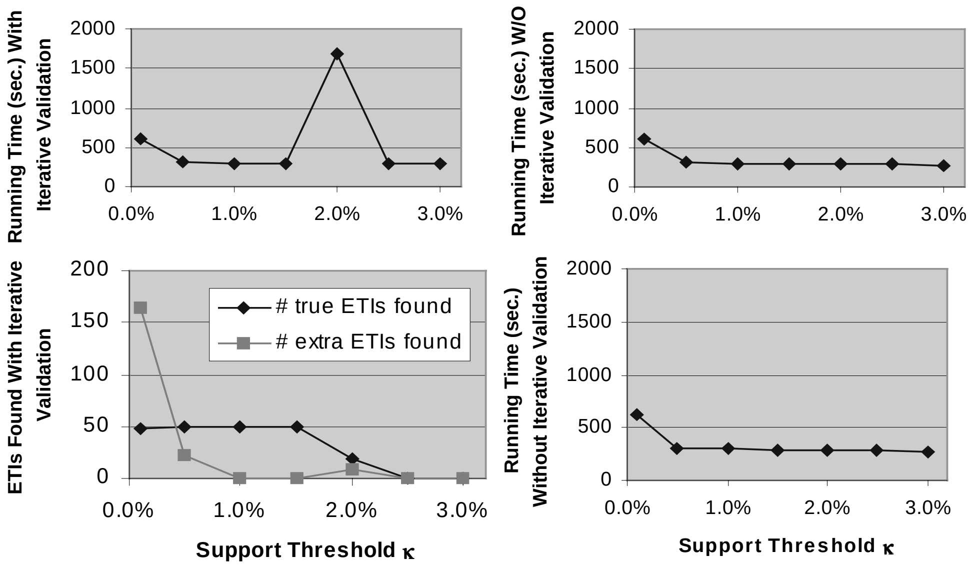


Figure B.2: Effect of Varying Support Threshold