Fault-Tolerant Frequent Pattern Mining: Problems and Challenges*

Jian Pei Anthony K.H. Tung Jiawei Han
School of Computing Science, Simon Fraser University
{peijian, khtung, han}@cs.sfu.ca

Abstract

Real-world data tends to be dirty. Discovering knowledge over large real-world data calls for fault-tolerant data mining, which is a fruitful direction for future data mining research. Fault-tolerant extensions of data mining techniques would gain useful insights into the data.

In this paper, we introduce the problem of fault-tolerant frequent pattern mining. With fault-tolerant frequent pattern mining, many kinds of novel, interesting and practically useful knowledge can be discovered. For example, one can discover the following fault-tolerant association rules: 85% of students doing well in three out of the four courses: "data structure", "algorithm", "artificial intelligence", and "database", will receive high grades in "data mining".

We extend *Apriori* and develop FT-*Apriori* for efficient fault-tolerant frequent pattern mining. Our experimental results show that FT-*Apriori* is a solid step towards fault-tolerant frequent pattern mining, however, it is still challenging to develop efficient fault-tolerant mining methods. The extensions and implications of related fault-tolerant data mining tasks are also discussed in the paper.

1 Introduction

Real-world data tends to be diverse and dirty. Usually, there are few non-trivial frequent patterns and rules with both high support and confidence in real datasets. Knowledge discovery from large real-world datasets calls for *fault-tolerant data mining*, as shown in the following example.

Example 1 To study students' performance in courses, one may find a rule as follows.

$$R_1 : good(x, data\ structure) \wedge \wedge good(x, algorithm)$$
$$good(x, AI) \wedge good(x, DBMS) \rightarrow good(x, data\ mining) \quad (1)$$

* The work was supported in part by Natural Sciences and Engineering Research Council of Canada, and the Networks of Centres of Excellence of Canada (NCE/IRIS-3). Also, we thank Joyce Lam and Helen Pinto for their comments on the early version of this paper.

The rule could have high prediction accuracy (i.e., high confidence), but it may cover only a very small subset of cases (i.e., low support), since there are not that many students who do well in all of the four courses on the left hand side of the rule.

On the other hand, one may find that another rule, R_2 : "*A student good in at least three out of four courses: data structure, algorithm, AI and DBMS, is also good at data mining*", also has high confidence. Rule R_2 is *fault-tolerant* in the sense that it requires the data to match only part of its left hand side.

Rule R_2 is more general than R_1 since every student satisfying R_1 also satisfies R_2, but not true vice versa. Thus, R_2 may have dramatically higher support than R_1 does. In the case that both rules have high confidence, we may expect R_2 is more interesting and more useful. □

Situations similar to that in Example 1 happen in many real applications. To discover more general, more interesting and more useful knowledge, we motivate the study of fault-tolerant data mining: *Instead of finding exact patterns in data, one may look for approximate and more general fault-tolerant patterns.* We argue that fault-tolerant patterns are more interesting and more applicable, since they cover more cases and the implication becomes stronger.

It has been well recognized, such as in [1, 2, 3, 4, 5, 6], that frequent pattern mining plays an essential role in discovering associations correlations, causality, sequential patterns, episodes, multi-dimensional patterns, max-patterns, partial periodicity, emerging patterns, and many other important data mining tasks. However, frequent pattern mining often generates a large number of frequent itemsets and rules, which reduces not only the efficiency but also the effectiveness of mining since users have to sift through a large number of mined rules to find useful ones.

To improve the effectiveness of frequent pattern mining, we focus this study to *fault-tolerant frequent pattern mining* and organize the paper as follows. The concept of *fault-tolerant frequent pattern mining* is introduced in Section 2. In Section 3, the *Apriori* method is extended to FT-*Apriori* for mining fault-tolerant frequent patterns. The performance study on FT-*Apriori* is reported in Section 4, which shows the effectiveness, efficiency, as well as the bottleneck of FT-*Apriori*. The extensions, implications and challenges of fault-tolerant frequent pattern mining as well as fault-tolerant data mining are discussed in Section 5.

2 Problem Definition

Let $I = \{i_1, \ldots, i_n\}$ be a set of **items**. An **itemset** X is a non-empty subset of I, i.e., $X \subseteq I$. For brevity, we write itemsets as $X = i_{j_1} \cdots i_{j_k}$, omitting set brackets. An itemset having k items is called a **k-itemset**. A **transaction** $T = (tid, X)$ is a 2-tuple, where tid is a transaction-id and X an itemset. A transaction $T = (tid, X)$ is said to **contain** itemset Y iff $Y \subseteq X$.

A **transaction database** TDB is a set of transactions. The number of transactions in transaction database TDB containing itemset X is called the **support** or **support count** of X, denoted as $sup(X)$. Given a transaction database TDB and a **support threshold** $min\_sup > 0$, an itemset X is a **frequent pattern** iff $sup(X) \geq min\_sup$[1]. The **problem of frequent pattern mining** is to *find the complete set of frequent patterns in a given transaction database with respect to a given support threshold.*

Example 2 (Frequent pattern mining) Table 1 shows the transaction database TDB as our running example. There are 5 transactions in the database TDB. Let the support threshold be 2, i.e., $min\_sup = 2$. Since itemset $X = \{b, e, f\}$, or $X = bef$ in short, is contained in transaction 10 and 50, respectively, i.e., $sup(X) = 2$, X is a frequent pattern. □

Transaction ID	Items
10	b, c, e, f
20	d, e, g
30	a, b, c, e
40	a, d, f
50	a, b, e, f

Table 1: A transaction database TDB.

[1]In this paper, we assume that all thresholds are positive integers. We adopt the notion of *absolute* threshold instead of *relative* (ratio) threshold as used in some studies.

For the transaction database TDB as shown above, if support threshold min_sup is set to 3, there exists no pattern with more than two items. That is, there are many *short* patterns with *low* support counts. To get generalized knowledge, people may like to find *longer* patterns with *higher* support count. Can we observe any longer "*approximate*" frequent patterns in the database with support 3 or more?

A close look at the transaction database would suggest that three transactions, transaction 10, 30, and 50, contain four out of five items: a, b, c, e and f. This frequent phenomenon is interesting in terms of that it captures features of the database concisely by slightly relaxing the notion of frequent pattern.

Can we generalize and develop systematic method to find such kind of knowledge? This motivates the notion of *fault-tolerant frequent patterns* as follows.

Definition 2.1 (Fault-tolerant frequent pattern) Given a **fault tolerance** δ ($\delta > 0$). Let P be an itemset such that $|P| > \delta$. A transaction $T = (tid, X)$ is said to **FT-contain** itemset P iff there exists $P' \subseteq P$ such that $P' \subseteq X$ and $|P'| \geq (|P| - \delta)$.[2] The number of transactions in a database FT-containing itemset P is called the **FT-support** of P, denoted as $\tilde{sup}(P)$.

Let $\tilde{B}(X)$ be the set of transactions FT-containing itemset X. It is called the **FT-body** of X. Given (1) a **frequent-item support threshold** min_sup^{item} and (2) an **FT-support threshold** min_sup^{FT}. An item set X is called a **fault-tolerant frequent pattern**, or **FT-pattern** in short, iff

1. $\tilde{sup}(X) \geq min\_sup^{FT}$; and
2. for each item $x \in X$, $sup_{\tilde{B}(X)}(x) \geq min\_sup^{item}$, where $sup_{\tilde{B}(X)}(x)$ is the number of transactions in $\tilde{B}(X)$ containing item x. □

The above definition has two support thresholds. The frequent-item support threshold is used to filter out infrequent item, since users may want to see patterns consisting of only items with statistic significance. On the other hand, FT-support threshold is used to capture frequent patterns in the sense of allowing at most δ (the fault tolerance) mismatches.

Example 3 (FT-patterns) Let us look at the transaction database TDB in Table 1 again. Suppose the frequent-item support threshold $min\_sup^{item} = 2$ and the FT-support threshold $min\_sup^{FT} = 3$.

Suppose one mismatch is allowed, i.e., fault tolerance $\delta = 1$. For itemset $X = abcef$, $\tilde{B}(X)$ includes

[2]An equivalent condition is $|P \cap X| \geq (|P| - \delta)$.

transaction 10, 30 and 50, since each of them FT-contains X. On the other hand, it is easy to check that each item in X, i.e., a, b, c, e and f, appears in at least two transactions in $\tilde{B}(X)$, respectively. Thus, itemset $abcef$ is an FT-frequent pattern. □

An item x is called a **global frequent item** iff $sup(x) \geq min\_sup^{item}$, i.e., item x appears in at least min_sup^{item} transactions. Clearly, an FT-pattern contains only global frequent items. To avoid triviality, an FT-pattern must have at least $(\delta + 1)$ items, where δ is the fault tolerance. About length-$(\delta + 1)$ FT-patterns, we have the following lemma.

Lemma 2.1 (Length-$(\delta + 1)$ FT-patterns) *Let X be an itemset containing $(\delta + 1)$ global frequent items. X is an FT-pattern iff $\tilde{sup}(X) \geq min\_sup^{FT}$.*

Proof. Let $X = x_{i_1} \cdots x_{i_{\delta+1}}$ be an itemset containing $(\delta + 1)$ global frequent items, i.e., $sup(x_{i_j}) \geq min\_sup^{item}$ $(1 \leq j \leq \delta + 1)$. $\tilde{B}(X)$ is the set of transactions FT-containing itemset X. A transaction t is in $\tilde{B}(X)$ iff t contains at least one item in X, since up to δ mismatches to X are allowed. So, we have $\tilde{B}(X) = \bigcup_{j=1}^{\delta+1} P(x_{i_j})$, where $P(x_k)$ is the set of transactions containing item x_k. Thus, for item $x_{i_j} \in X$, its support in $\tilde{B}(X)$ is exactly the same as that in the whole database TDB, i.e., $sup_{\tilde{B}(X)}(x_{i_j}) = sup(x_{i_j}) \geq min\_sup^{item}$. Therefore, provided that X passes the FT-support threshold min_sup^{FT}, X is an FT-pattern. □

Lemma 2.1 says that a $(\delta + 1)$-itemset consisting of global frequent items is an FT-pattern provided it passes FT-support threshold. In practice, there could be a large number of length-$(\delta+1)$ FT-patterns. For example, 100 global frequent items and a fault-tolerance of 2 may lead to up to $\begin{pmatrix} 100 \\ 3 \end{pmatrix} = 161,700$ length-3 FT-patterns. In sequel, there could be a large number of length-$(\delta+k)$ FT-patterns when k is small, e.g. $k = 1$ or 2.

Note that short FT-patterns, i.e., length-$(\delta + k)$ patterns with a small k, may not be interesting since too many mismatches usually lead to uninteresting results. Therefore, in general, fault tolerance δ is a small positive number. Furthermore, we apply a **length threshold** min_l $(min\_l > \delta)$ such that only FT-patterns having at least min_l items are output.

Given a transaction database, a fault tolerance, a frequent-item support threshold, an FT-support threshold, and a length threshold, **the problem of fault-tolerant frequent pattern mining** is to *find the complete set of FT-patterns passing the length threshold.*

To verify whether FT-patterns can capture more general (i.e., more frequent) features in databases, we have the following lemma.

Lemma 2.2 (Effect of FT-patterns) *For any itemset X, $\tilde{sup}(X) \geq sup(X)$.*

Proof. For each transaction containing itemset X, it also (trivially) FT-contains X and thus contributes 1 to $\tilde{sup}(X)$. So we have the lemma. □

Lemma 2.2 implies that fault-tolerant frequent pattern mining leads to longer and more frequent patterns.

3 FT-*Apriori*

All previously proposed efficient frequent pattern mining methods are directly or indirectly based on the *Apriori* heuristic [1], which is an anti-monotone property: *if a k-itemset is not frequent, any of its superset cannot be frequent.* Based on this heuristic, a candidate-generation-and-test approach, Apriori, iteratively generates the set of candidate patterns of length $(k + 1)$ from the set of frequent patterns of length k $(k \geq 1)$, and checks their corresponding occurrence frequencies in the database [1]. This method achieves good reduction on the size of candidate sets.

Can we extend the Apriori *heuristic to attack fault-tolerant frequent pattern mining problem*? Fortunately, we have the following theorem.

Theorem 3.1 (Fault-tolerant *Apriori*) *If X $(|X| > \delta)$ is not an FT-pattern, then none of its supersets is an FT-pattern, where δ is the fault tolerance.* □

Based on this heuristic, a candidate generation-and-test method can be developed for computing fault-tolerant frequent patterns, as shown in the following example.

Example 4 (Fault-tolerant *Apriori*) Let's mine FT-patterns in Table 1. Suppose the frequent-item support threshold $min\_sup^{item} = 2$, the FT-support threshold $min\_sup^{FT} = 3$, fault tolerance $\delta = 1$, and length threshold $min\_l = 4$. The complete set of FT-patterns can be mined as follows.

1. The first scan of the database derives the set of global frequent items, i.e., the items appearing in at least min_sup^{item} transactions: $\{a, b, c, d, e, f\}$.

2. The shortest FT-pattern must be of length $(\delta + 1) = 2$. According to Lemma 2.1, all length-2

combinations of global frequent items are length-2 candidates. There are 15 length-2 candidates: ab, ac, $\ldots$, af, bc, $\ldots$, ef. Scanning the database TDB the second time collects FT-support counts for them. Based on Lemma 2.1, there is no need to check against frequent-item support threshold. It happens that every candidate is an FT-pattern. However, since their length is only 2 and does not pass the length threshold, they are not output.

3. Based on fault-tolerant *Apriori*, length-3 candidates are generated from length-2 FT-patterns. The rule of candidate generation is that *a length-$(k+1)$ candidate is generated iff its every length-k subset is an FT-pattern.* For example, abc is qualified as a length-3 candidate since ab, ac, and bc are length-2 FP-patterns. In total, 20 length-3 candidates are generated: abc, abd, $\ldots$, abf, acd, $\ldots$, aef, bcd, $\ldots def$.

   The database TDB is then scanned the third time to check these candidates, which derives 12 length-3 FT-patterns: abc, abe, abf, ace, acf, ade, aef, bce, bcf, bef, cef and def.

   Note that although some candidates, such as abd, pass the FT-support threshold, some items within the candidate, such as item d in candidate abd, cannot pass the frequent-item threshold in the corresponding FT-body. Such candidates cannot be qualified as FT-patterns. We do not output any of length-3 FT-patterns since they still cannot pass the length threshold.

4. From length-3 FT-patterns, we generate 5 length-4 candidates: $abce$, $abcf$, $abef$, $acef$ and $bcef$. By scanning the database TDB the fourth time, the candidates are checked against the database, and are verified as FT-patterns. They pass the length threshold and thus are output.

5. From length-4 FT-patterns, we generate only one length-5 candidate: $abcef$. It is checked in the fifth database scan and is identified as an FT-pattern. It is also output.

6. Since no length-6 candidate can be generated, the mining process terminates.

The above mining process finds the complete set of 33 FT-patterns from the database, among which 6 FT-patterns are output. It scans the database 5 times. □

Based on the above example, we have the FT-*Apriori* algorithm as follows.

Algorithm 1 (FT-*Apriori*)

Input: Transaction database TDB, frequent-item support threshold min_sup^{item}, FT-support threshold min_sup^{FT}, fault tolerance δ and length threshold min_l.

Output: The complete set of FT-patterns.

Method:

```
1. Scan $TDB$ once, find the set $F_1$ of global frequent items. An item $x$ is global frequent iff $sup(x) \geq min\_sup^{item}$;
2. Let $C_{\delta+1}$ be the set of all length-$(\delta+1)$ subsets of $F_1$. Let $i = \delta + 1$.
3. Do {
   (a) Scan $TDB$, check candidate itemsets in $C_i$;
   (b) Let $F_i$ be the set of FT-patterns in $C_i$; If ($i \geq min\_l$) then output patterns in $F_i$;
   (c) If $F_i$ is not empty, generate $C_{i+1}$ from $F_i$. A length-$(i+1)$ itemset $X$ is in $C_{i+1}$ iff every length-$i$ subset of $X$ is in $F_i$;
   (d) $i = i + 1$;
   } until either $F_{i-1}$ or $C_i$ is empty;
```

Analysis. The initialization of candidate-generation (Step 1 and 2) is based on Lemma 2.1. The rest of the algorithm is based on Theorem 3.1. So, we show the algorithm is correct and complete. □

Based on the above analysis, we have the following theorem.

Theorem 3.2 *Algorithm 1 finds the complete set of FT-patterns without duplication.* □

In the next section, we will examine the efficiency as well as the effectiveness of fault-tolerant frequent pattern mining using FT-*Apriori*.

4 Performance Evaluation

In this section, we present a performance evaluation of FT-*Apriori* and discuss the efficiency and effectiveness of fault-tolerant frequent pattern mining.

We implemented the FT-*Apriori* algorithm as stated in Section 3, using Microsoft Visual C++6.0. All the experiments were conducted on a PC with an Intel Pentium III 500MHz CPU and 128M main memory, running Microsoft Windows/NT.

We designed a synthetic dataset generator. It generates transaction databases with many potential fault-tolerant frequent patterns. We conducted experiments on various datasets generated by our generator. The results are consistent in trend. Limited by space, we report only results on one such dataset D. In this dataset, a transaction contains 10 items on average, while the average length of potential fault-tolerant patterns is 6. There are 10,000 transaction in total.

Figure 1 shows the scalability of FT-*Apriori* with respect to item support threshold, where the FT support threshold is set to 3%, 5%, 8% and 10%, respectively. We set the fault tolerance $\delta = 1$.

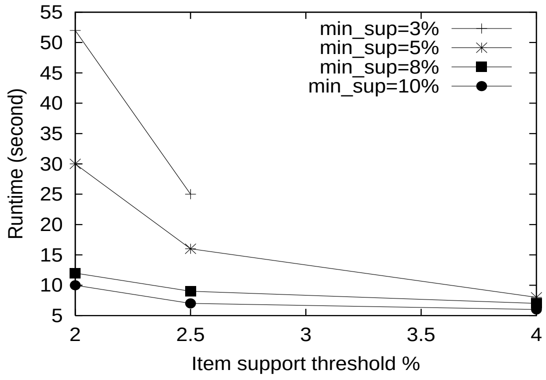

Figure 1: Scalability of FT-*Apriori* w.r.t. item support threshold.

As can be seen from Figure 1, the runtime increases as the frequent-item support threshold min_sup^{item} goes down. When min_sup^{item} is low, the number of patterns as well as candidates increase exponentially, and thus the cost would increase dramatically. In this sense, the trend of FT-*Apriori* is consistent with that of *Apriori*.

FT-*Apriori* scales better when FT-support threshold is high, since FT-support threshold prunes both the FT-patterns and candidates. Figure 2 shows the scalability of FT-*Apriori* with respect to FT-support threshold, where min_sup^{item} are set to 2%, 2.5% and 4%, respectively. Again, we set the fault tolerance $\delta = 1$.

We do not directly compare the performance of FT-*Apriori* and *Apriori*, since they are computing different things. However, with similar support threshold, *Apriori* can finish within 5 seconds on this dataset. In general, *Apriori* is 10 to 100 times faster than FT-*Apriori* with similar support threshold settings. FT-*Apriori* also generates much more candidates than *Apriori* does.

Interestingly, with similar support threshold settings, FT-*Apriori* can find much longer patterns.

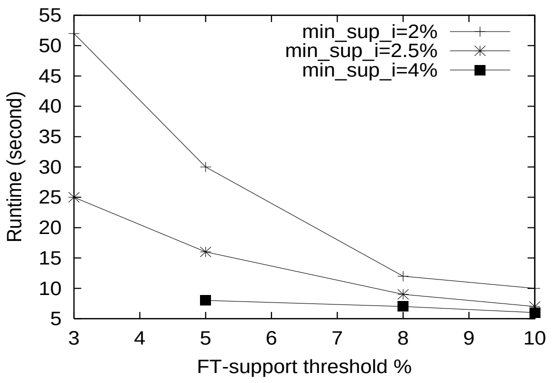

Figure 2: Scalability of FT-*Apriori* w.r.t. FT-support threshold.

In this dataset, the length of the longest pattern that FT-*Apriori* finds is up to twice that found by *Apriori* with a similar support threshold. That means FT-*Apriori* can find more general knowledge, which is the power of fault-tolerant frequent pattern mining.

We tried FT-*Apriori* for higher fault tolerance, such as $\delta = 2$. Unfortunately, FT-*Apriori* does not scale well. It is much slower (about 100 times or even more) than it does when $\delta = 1$.

From the experimental results, we have the following observations.

- *Fault-tolerant frequent pattern mining finds more general knowledge.* With similar setting, FT-*Apriori* finds longer patterns. However, FT-*Apriori* can find patterns with much higher support count. In other words, the patterns that FT-*Apriori* finds are more popular and general.

- *Fault-tolerant* Apriori *property reduces the number of candidates dramatically.* As shown in Example 4, FT-*Apriori* property helps reduce the number of length-4 candidates from $\binom{6}{4} = 15$ to only 5.

- *A major cost in* FT-Apriori *is that it might have to handle a huge number of candidates.* To discover a FT-pattern having 100 items, such as $a_1 \cdots a_{100}$, it must generate $2^{100} - 1 \approx 10^{30}$ candidates in total. Also, it is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

- *The first candidate set, $C_{\delta+1}$ in Step 2 in the algorithm, could be very large.* For example, if fault tolerance $\delta = 2$ and there are 1,000 frequent items, C_3 contains all length-3 itemsets assembled by frequent items and thus has $\begin{pmatrix} 1000 \\ 3 \end{pmatrix} = 166,167,000$ candidates! The cost of checking such a large set of candidates is non-trivial.

5 Discussion and Conclusions

We have introduced the fault-tolerant data mining problem and extended the *Apriori* method to FT-*Apriori* for mining fault-tolerant frequent patterns. It is worth noting the following findings from analysis and performance evaluation of fault-tolerance frequent pattern mining and FT-*Apriori*.

1. **Fault-tolerant frequent pattern mining finds more general and useful knowledge.** Fault-tolerant frequent pattern mining can find longer and more frequent patterns. Using the frequent item support threshold, fault-tolerant frequent pattern mining focuses on patterns consisting of only significant items. Thus, it avoids trivial data items and captures general trends in the data.

2. FT-*Apriori* **is an efficient and scalable fault-tolerant frequent pattern mining algorithm when support thresholds are not too low and fault-tolerance is low.** In many real applications, where the support thresholds are reasonably high and the fault-tolerance as 1 is enough, FT-*Apriori* is a feasible and even nice solution. However, when the support thresholds are set low and more faults are allowed, FT-*Apriori* faces significant challenges.

3. **Efficient fault-tolerant frequent pattern mining is very challenging.** As shown by our experimental results, a minor increase of fault tolerance may blow up the mining process. Also, decreasing the support thresholds may increase the runtime dramatically. Fault toleration usually introduces a huge number of candidates. Efficient fault-tolerant frequent pattern mining is still an open problem.

As to related work, Yang, Fayyad, and Bradley [6] proposed a method for efficient discovery of error-tolerant frequent itemsets in high dimensions. Their concept of error-tolerant frequent itemset is similar to our fault-tolerant frequent pattern, however, since we have introduced a few additional constraints, such as items being considered confined to global frequent items, patterns being considering confined by a length threshold, and so on, it is easy for us to develop more efficient methods for fault-tolerant frequent pattern mining. In their study, they focus on the development of random and approximate algorithms; whereas according to our study, it is still promising to develop reasonable mining algorithms to find complete patterns if the faults are small and the constraints are tight.

As an ongoing study, we are now exploring efficient fault-tolerant frequent pattern mining in pattern-growth methods. Motivated by the pattern-growth methods in frequent pattern mining, we are investigating how to generate less or no candidates by pattern-growth [3]. We are also considering other alternative approaches, such as random algorithms [6].

The idea of fault-tolerant data mining can be integrated into many other data mining tasks. We consider the following two as future work.

- **Fault-tolerant sequential pattern mining.** Though most of previous work on sequential pattern mining requires exact match between the pattern and the sequence data, many real applications require fault-tolerant capability. A typical example is DNA mining. When mining with DNA data, dealing with mismatches including insertion, deletion and mutation becomes essential.

- **Mining fault-tolerant fascicles.** A fascicle [4] is a subset of records sharing a set of compact attributes. Recent study shows that fascicles are useful in database compression and pattern extraction. It is interesting to see whether fault-tolerant fascicles can further improve the effectiveness of storage reduction and other tasks.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB'94*, pp. 487–499, Santiago, Chile, Sept. 1994.

[2] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD'98*, pp. 85–93, Seattle, WA, June 1998.

[3] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00*, pp. 1–12, Dallas, TX, May 2000.

[4] H. V. Jagadish, J. Madar, and R. Ng. Semantic compression and pattern extraction with fascicles. In *VLDB'99*, pp. 186–197, Edinburgh, UK, Sept. 1999.

[5] H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[6] C. Yang, U. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Technical Report MSR-TR-00-20*, Microsoft Research, Feb. 2000.