Inf Syst Front (2009) 11:461–469
DOI 10.1007/s10796-009-9158-z

Proportional fault-tolerant data mining with applications to bioinformatics

Guanling Lee · Sheng-Lung Peng · Yuh-Tzu Lin

Published online: 19 February 2009

Abstract The mining of frequent patterns in databases has been studied for several years, but few reports have discussed for fault-tolerant (FT) pattern mining. FT data mining is more suitable for extracting interesting information from real-world data that may be polluted by noise. In particular, the increasing amount of today's biological databases requires such a data mining technique to mine important data, e.g., motifs. In this paper, we propose the concept of proportional FT mining of frequent patterns. The number of tolerable faults in a proportional FT pattern is proportional to the length of the pattern. Two algorithms are designed for solving this problem. The first algorithm, named FT-BottomUp, applies an FT-Apriori heuristic and finds all FT patterns with any number of faults. The second algorithm, FT-LevelWise, divides all FT patterns into several groups according to the number of tolerable faults, and mines the content patterns of each group in turn. By applying our algorithm on real data, two reported epitopes of spike proteins of SARS-CoV can be found in our resulting itemset and the proportional FT data mining is better than the fixed FT data mining for this application.

Keywords Bioinformatics · FT support · Fault-tolerant frequent pattern · Data mining

G. Lee (✉) · S.-L. Peng · Y.-T. Lin
Department of Computer Science and Information Engineering,
National Dong Hwa University,
Hualien 974,
Taiwan, Republic of China
e-mail: guanling@mail.ndhu.edu.tw

1 Introduction

Valuable information is one of the most powerful weapons in the current knowledge age, and hence knowledge discovery has become a popular field of research. For example, in bioinformatics, biological data grow exponentially in size and complexity. It is not an easy work to extract useful information from it. Thus an important goal of data mining in bioinformatics is to extract valuable information from a large amount incomprehensible, biological data. Traditional algorithmical techniques use pattern matching algorithms to find valuable information for biological sequences. For example, linear scan (Knuth et al. 1977) and suffix tree (Ukkonen 1995) are two well-known approaches. For more references and applications on patter matching, we refer to (Gusfield 1997).

In contrast, data mining in bioinformatics deals with different techniques and algorithms to gain knowledge from data of biological sequences, structures, and microarrays (Chen 2005). Association-rule mining, which was first investigated in (Agrawal et al. 1993), explores the relationships among data items. It is a very popular technique in data mining. For example, in the analysis of association rules of a metabolic pathway, the rule "$Gene_1 \rightarrow Gene_2$, support=10%, confidence=90%" means that the support of 10% indicates that $Gene_1$ and $Gene_2$ are expressed together in 10% of all datasets, and the confidence of 90% indicates that 90% of the metabolic pathway which activated $Gene_1$ also correlated with $Gene_2$. Recently, Kotlyar and Jurisica used this technique to predict protein–protein interactions (Kotlyar and Jurisica 2006).

The approaches used to find association rules can be roughly classified into two categories. The first category is the Apriori-based algorithm (Agrawal and Srikant 1994). This influential algorithm generates candidate patterns

according to the non-monotonicity heuristic. The two main problems of Apriori are that (1) it needs to scan the database several times and (2) it generates too many candidate patterns. Various techniques have been used to overcome these problems. In Park et al. (1995), a hash table is used to store candidate patterns to increase the scanning efficiency; Agrawal and Srikant (1994), Han and Fu (1995), and Park et al. (1995) attempt to reduce the number of transactions scanned in future iterations; Savasere et al. (1995) and Zaki (2000) adopt the techniques of partitioning and sampling; and Brin et al. (1997) proposes a dynamic pattern-counting method in which candidate patterns are added at different points during a scan. And the negative association rules are considered in Antonie and Zaïane (2004); Thiruvady and Webb (2004); Zhang and Zhang (2004). Moreover, the problem of mining temporal indirect association patterns is considered in Chen et al. (2006).

The second category is the tree-based algorithm, which was proposed as the FP-tree (frequent-pattern tree) in Han et al. (2000). This algorithm scans the database to find all frequent items, and compresses the database by representing the frequent items in an FP-tree. Finally, all frequent patterns can be obtained by searching the tree. When the database is large, it is sometimes unrealistic to construct an FP-tree that resides in main memory. This leads to the proposed extension of the pattern-growth concept, namely, H-mine. H-mine designs a dynamic structure to adjust links dynamically, instead of requiring an FP-tree to be maintained or a physical database to be created. The motivation of this method is to preserve space, and initially involves loading transactions into memory. However, H-mine has to maintain a head table at each level of the tree, and modify the links to build a queue of the collection of transactions containing the same prefix before the pattern support is counted.

Expect for the two categories of approaches, some works study on other ways to extract association rules. Matrix Apriori (Pavon et al. 2006) utilizes simple structures such as matrices and vectors in the process of generating frequent patterns, and it also minimizes the number of candidate sets. In (Lee et al. 2006), the idea of compressions rules is proposed and a data mining structure is used to extract association rules from a database. Redundant data will then be replaced by means of compression rules. (Chu et al. 2005) uses a simple method to transform the transactions read from the database into their corresponding patterns and then accumulates the occurring times of these patterns. (Chen et al. 2002) refines sampling functions to a two-phase sampling based algorithm that attempts to reduce the errors caused by sampling functions. (Chen and Ho 2005) proposed a sampling-based method that contains three phases. The first phase draws a small sample of data to estimate the set of frequent patterns, denoted as *FS*. The second phase computes the actual supports of the patterns in *FS* as well as identifies a subset of patterns in *FS* that need to be further examined in the next phase. Finally, the third phase explores this set and finds all missing frequent patterns.

Traditional association-rule mining extracts patterns that match exactly. However, real-world databases contain noise that can make important information ambiguous, resulting in it not appearing in the mining result. Moreover, sometimes a decision maker will not be helped by the limited knowledge mined from a small database. Therefore, we need a method that copes with such variations in an association pattern (within predefined limits), which is called a *fault-tolerant* (*FT*) *pattern*.

In contrast to traditional frequent-pattern mining, the mining of FT patterns must tolerate a certain degree of inexactitude. For example, coughing, fever, a runny nose, a headache, and a sore throat are all signs of catching a cold. However, these symptoms are seldom present at the same time, and hence a doctor will not diagnose the disease exactly following the rule *R*1: {*coughing*, *fever*, *runny nose*, *headache*, *sore throat*} à{*catch a cold*}. Instead, a better rule corresponding to the real-world situation would be *R*2: *Patients who have at least two of the following symptoms* {*coughing*, *fever*, *runny nose*, *headache*, *sore throat*} *are catching a cold*. *R*2 requires matching just part of the data, which illustrates the sense of allowing for fault tolerance in data mining.

Yang et al. (2001) was the first to propose discovering FT frequent patterns in many dimensions. Their primary motivation was to find frequent groups of transactions (user groups, web sessions, and so on.) instead of focusing on just the items themselves, allowing for the discovery of groups of similar transactions that share most items. Unfortunately, the approach proposed in Yang et al. (2001) may generate *sparse patterns*, which may contain subpatterns that do not appear frequently.

Another milestone of FT pattern mining is the work described in Pei et al. (2001), in which extending Apriori and developing FT-Apriori for FT frequent-pattern mining allows a complete set of FT patterns to be mined out. However, the disadvantages of Apriori-based algorithms, including a huge number of candidate patterns and high database scanning frequency, also occurred in Pei et al. (2001). In response, Wang and Lee (2002) suggested the algorithm FTP-mine that finds FT patterns using the concept of pattern growth.

The main defect of Pei et al. (2001) and Wang and Lee (2002) is their definition of the number of tolerable faults in a pattern as a fixed number. Defining the number of tolerable faults in the patterns as a fixed number of items is not objective. The matters of "tolerant 1 item" in a pattern with length 4 and that in a pattern with length 10 give people entirely different sense. For example, the function of a protein is determined by its structure but not sequence. It is possible that two proteins of similar function have different sequence lengths, e.g., the family of heat shock proteins. In

this case, it is hard to mine them together using FT pattern mining with fixed number of tolerable faults. In this paper, we introduce the problem of mining proportional FT patterns; in these patterns, the number of tolerable faults in the pattern is proportional to the pattern length. Two approaches are proposed to solve this problem. The remainder of this paper is organized as follows: background knowledge and problem definitions are presented in Section 2, the principles underlying our approaches are presented in Section 3, Section 4 describes the experimental results, and the conclusions and future work are discussed in Section 5.

2 Problem definition

Let pattern $X=\{i_1,\ldots,i_n\}$ be a set of items, where the *length* of X is the cardinality of X, denoted as $|X|$. Moreover, X is called an $|X|$-*pattern* since it contains $|X|$ items. A transaction $T = (tid, X)$ is a 2-tuple record, where *tid* is the transaction-id and X is a pattern. Transaction $T=(tid, X)$ is said to contain pattern Y iff $Y\subseteq X$.

A transaction database TDB is a set of transactions. The number of transactions in TDB containing pattern X is called the *support* of X, denoted as $sup(X)$. Given a transaction database TDB and a user-defined support threshold min_sup>0, pattern X is a *frequent pattern* iff sup(X)≥min_sup. A frequent pattern with length k is denoted as a frequent-k pattern. In the process of frequent-pattern mining, possible patterns are generated as *candidate* patterns, and these are later tested to determine whether they are frequent. 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.

Extending the problem of mining frequent patterns, the FT frequent-pattern- mining problem relaxes the definition of containing to *FT-containing*. In addition to mining exact patterns that occur with high frequencies, we find those frequent patterns that contain some errors. In Pei et al. (2001) and Wang and Lee (2002), FT-containing is defined as mismatches in a fixed number of items in a pattern. However, as mentioned above, it is disadvantageous for the same number of faults to be tolerated in patterns with different lengths. Therefore, the problem of proportional FT frequent-pattern mining is proposed.

Definition 2.1 (Proportional FT frequent pattern)

Let P be a pattern. A transaction $T=(tid, X)$ *is said to FT-contain pattern P with respect to a given FT parameter* δ $(0<\delta\leq 1)$ *iff there exists* $P'\subseteq P$ *such that* $P'\subseteq X$ *and* $\frac{|P'|}{|P|}\geq\delta$. *The number of transactions in a database FT-containing pattern P is called the FT-support of P, denoted as* $\sup^{FT}(P)$. *Let* $B(P)$ *be the set of transactions FT-containing pattern P. Given a frequent item-support threshold* $\min\_sup^{item}$ *and an FT-support threshold* $\min\_sup^{FT}$, *a pattern P is called an FT frequent pattern iff*

1. $\sup^{FT}(P)\geq\min\_sup^{FT}$; *and*
2. *for each item* $p\in P$, $\sup^{item}_{B(P)}(p)\geq\min\_sup^{item}$, *where* $\sup^{item}_{B(P)}(p)$ *is the number of transactions in* $B(P)$ *containing item p.*

Definition 2.1 mostly extends (Pei et al. 2001), except the FT parameter and the definition of FT-containing. The item-support threshold avoids the problem of sparse patterns appearing which can occur in (Yang et al. 2001), by constraining the frequency of occurrence of each item in a pattern. Moreover, our definition of FT-containing releases the constraint that the number of fault items tolerable in a pattern is fixed—instead, the number of tolerable-fault items increases depending on the length of the pattern. Figure 1 shows the relation between the length of pattern X and #fault($|X|$), where #fault($|X|$) denotes the number of fault items tolerable in pattern X. According to Definition 2.1, we have the following equation: $\#\,\text{fault}(|X|) = \lfloor(1-\delta)\times|X|\rfloor$.

In the horizontal parts of the stair shown in Fig. 1, our problem can be simplified to previous works, i.e., FT-Apriori (Pei et al. 2001) can be extended as the following lemma to solve part of our problem.

Lemma 2.1 (Extended FT-Apriori) If X is not an FT pattern, then none of its superset that tolerates the same number of faults will be an FT pattern

However, we have a challenge where the gaps occur in Fig. 1, since these areas do not comply with the non-monotonicity property. That is, if a pattern is not a frequent FT pattern, its superset can still be a frequent FT pattern. Therefore, solutions from previous studies cannot solve our problem.

For example, please refer to Table 1. Let FT parameter δ= 0.6, $\min\_sup^{FT}$=5, and $\min\_sup^{item}$=2. Consider pattern X= {*abcd*}, where $\#\text{fault}(|X|) = \#\text{fault}(|X|) = \lfloor(1-0.6)\times 4\rfloor = 1$: X is an infrequent FT pattern since $\sup^{FT}(X)=|\{tid{:}040, tid{:}050\}| = 2<5$, yet X's superset $Y=\{abcde\}$, where

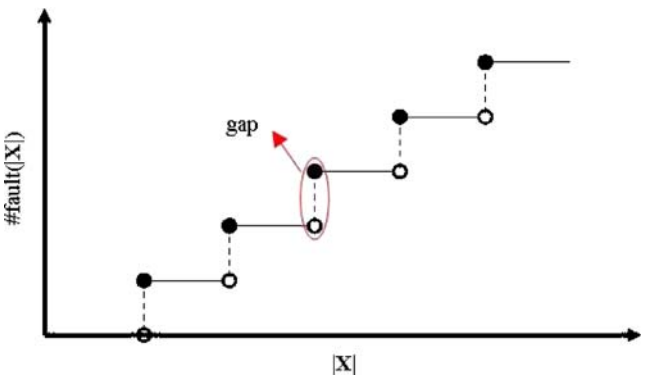

Fig. 1 Relation between $|X|$ and #fault($|X|$)

#fault(|Y|)=2, is a frequent FT pattern. Because $\sup^{FT}(Y)$=|{*tid*:010, *tid*:020, *tid*:030, *tid*:040, *tid*:050}|=5, $\sup^{item}_{B(Y)}(a)$=$\sup^{item}_{B(Y)}(b)$=$\sup^{item}_{B(Y)}(c)$=$\sup^{item}_{B(Y)}(d)$=$\sup^{item}_{B(Y)}(e)$=3>2, which proves that Y is a frequent FT pattern.

However, observation of the properties of patterns separated by the gap produces the following lemma:

Lemma 2.2 Let set($X_{subpattern}$) denote the set of X's subpatterns whose length is one less than the length of X. Moreover, let #fault(|X|)–1=#fault(|X|–1) (i.e., X and the considered subset are separated by the gap). If X is not a frequent FT pattern, then we have the following two conditions:

case 1. if $\sup^{FT}(X)<\min\_sup^{FT}$, then for all $P\in$set($X_{subpattern}$), P cannot be an FT pattern.

case 2. if $\sup^{item}_{B(X)}(x_j)<\min\_sup^{item}$ for an item $x_j\in X$, then none of the patterns in set($X_{subpattern}$) that contains item x_j can be an FT pattern.

Proof Let the set of subpatterns of X with length |X|–k be S, where k = #fault(|X|), and the set of subpatterns of $X_{subpattern}$ ($X_{subpattern}\in$set($X_{subpattern}$)) with length |X|–1–(k–1) be S'. Because $X_{subpattern}\in$set($X_{subpattern}$), we know that $S'\subset S$. Suppose X is not a frequent FT pattern:

case 1. $\sup^{FT}(X)<\min\_sup^{FT}$: if there exists a frequent FT pattern $X_{subpattern}$ ($X_{subpattern}\in$set($X_{subpattern}$)), then we have $\sup^{FT}(X_{subpattern})>\min\_sup^{FT}$. Because $\sup^{FT}(X_{subpattern})$ is included in $\sup^{FT}(X)$, $\sup^{FT}(X)>\min\_sup^{FT}$, which disagrees with the supposition. Therefore, no patterns in set($X_{subpattern}$) can be frequent patterns if $\sup^{FT}(X)<\min\_sup^{FT}$.

case 2. $\sup^{item}_{B(X)}(x_j)<\min\_sup^{item}$: if there exists a pattern $X_{subpattern}$, where $X_{subpattern}\in$set($X_{subpattern}$) and $X_{subpattern}$ contains x_j, then we have $\sup^{item}_{B(X_{subpattern})}(x_j)>\min\_sup^{item}$. Since the item support of $X_{subpattern}$ counts from S', that of X is from S, and $S'\subset S$, we obtain $\sup^{item}_{B(X)}(x_j)>\min\_sup^{item}$, which disagrees with the supposition. Therefore, none of patterns in set($X_{subpattern}$) that contains item x_j can be an FT pattern if $\sup^{item}_{B(X)}(x_j)<\min\_sup^{item}$.

Table 1 An example TDB

Tid	Items
010	*c d e*
020	*b d e*
030	*a d e*
040	*a b c*
050	*a b c*

Lemma 2.2 holds because the supports of patterns parted by the gap are from the same set of subpatterns.

Two approaches are developed using Lemmas 2.1 and 2.2. First, based on Lemma 2.1, we propose the FT-BottomUp algorithm as a basic solution, which is explained in detail in Section 3.1. Although this algorithm is closed to violent solution, it finds the complete set of proportional FT patterns.

To improve the efficiency, the second algorithm (named FT-LevelWise) is proposed. Several pruning properties are adopted in this algorithm to improve the performance. The algorithm FT-LevelWise is discussed in Section 3.2.

3 Mining proportional FT frequent patterns

3.1 FT-BottomUp algorithm

The principle underlying the basic algorithm, FT-BottomUp, is to find all FT patterns for which the number of faults is from $(1–\delta)\times|X|$ to MaxFault at level-|X|, where MaxFault is the maximum possible number of faults. Let DB' be the preprocessed database that is equivalent to the original database with the infrequent items removed. We estimate *MaxFault* by the maximum possible length of FT patterns, denoted as *MaxPattern*, using the following equations:

$$\text{MaxPattern} = \min\left\{\text{number of frequent} - 1 \text{ patterns}, \frac{\text{length of longest transaction in } DB'}{\delta}\right\}$$

$$\text{MaxFault} = \#\text{fault}(|\text{MaxPattern}|)$$

This conservative assumption obtains the lower bound for the number of frequent-1 patterns and the length of the longest transaction divided by the FT parameter δ, because the largest pattern either contains all frequent-1 items or FT-contains the longest transaction.

The length of the longest possible pattern can be used to calculate the maximum possible number of faults during the mining process. This upper bound tells us not only when the mining terminates but also what candidates should be generated.

Algorithm 1 (FT-BottomUp algorithm)

Input

Transaction database DB
Frequent item-support threshold $\min\_sup^{item}$
Frequent FT support threshold $\min\_sup^{FT}$
FT parameter δ

Output

FT patterns

/* $F_{i,j}$ is the set of FT patterns with length i and containing j faults */

Method:

1. Scan DB to find the set of frequent 1-patterns, denoted as $F_{1,0}$;

   Let $DB' = DB \cap F_{1,0}$;

2. MaxPattern=$\min\left\{\frac{\text{lengthoflongesttransactionin}DB'}{\delta}, |F_{1,0}|\right\}$;

   MaxFault = #fault(MaxPattern);

3. **for** (int i=2; i<MaxPattern; i++){

   j=#fault(i)
   generate $C_{i,j}$ by $F_{i-1,j}$;
   /* $C_{i,j}$ are the candidate patterns for $F_{i,j}$ */
   $F_{i,j}$ = FT_frequent($C_{i,j}$, min_sup^{FT}, min_sup^{item});
   /* FT_frequent($C_{i,j}$, min_sup^{FT}, min_sup^{item}) returns the set of patterns
   in $C_{i,j}$ that comply with the two support thresholds*/}
   output $F_{i,j}$;

Algorithm 1 shows that at the first $\lceil \frac{\delta}{1-\delta} \rceil$ levels we can adopt the basic Apriori method since patterns at those levels tolerate 0 items as faults, and FT patterns with 0 faults are only required to pass the item-support threshold to be frequent. When the number of tolerable faults is greater than 0, FT-Apriori is used repeatedly to generate candidate FT patterns with the tolerable faults in that level.

3.2 FT-LevelWise algorithm

The FT-LevelWise algorithm adopts the principle of partitioning the FT patterns into *MaxFault* groups at each step of the stair shown in Fig. 1.

Consider each group G_i (i=0 to *MaxFault*), where i is the label of the group and also presents the number of faults tolerated by the patterns in the group. The shortest FT patterns of group G_i are called the *head patterns* of G_i, denoted as $head_i$, while the longest ones are called *tail patterns* and are denoted as $tail_i$. The depth of group G_i, denoted as $depth_i$, is the difference between the lengths of $tail_i$ and $head_i$, and can be evaluated by FT parameter δ as $depth_i = \lceil \frac{1}{1-\delta} \rceil$ or $\lfloor \frac{1}{1-\delta} \rfloor$. After finding frequent-1 patterns, for each group G_i ($0 \le i \le$ MaxFault), we first generate candidate patterns for $head_i$ by frequent-1 items, and scan the database to check whether the candidates are frequent patterns. If $head_k$ contains no frequent patterns, group G_k can be deleted. Then, we can generate the candidates of $tail_i$ using $head_i$ since they tolerate the same number of fault items. Furthermore, according to Lemma 2.2, some candidates of $tail_i$ can be pruned by the information collected in $head_{i+1}$. That is, a candidate pattern of $tail_i$ will never be an FT pattern if its superpattern does not exist in $head_{i+1}$ because of the small FT-support, or because any item support is not large enough and the candidate pattern contains the item as well.

Moreover, in middle layers mid_{ij} (j is the difference between the lengths of mid_{ij} and $head_i$), we adopt Lemma 2.1 from both sides. That is, candidate patterns of mid_{ij} are generated by the existing longest subpatterns of mid_{ij}, and if there any FT patterns are superpatterns of this candidate, the candidate is already set to be frequent and does not need to be checked by scanning the database. The main principle underlying this approach is that, for each group, the head FT patterns are generated by frequent-1 patterns, while the tail FT patterns are generated by the head and pruned by the head of the next group. Then, FT patterns are generated by the longest subpattern and pruned by the shortest superpattern from both sides of the group.

Algorithm 2 (FT-LevelWise algorithm)

Input

Transaction database DB
Frequent item-support threshold min_sup^{item}
Frequent FT support threshold min_sup^{FT}
FT parameter δ

Output

FT patterns

Method:

1. Scan DB to find the set of frequent 1-patterns, denoted as F_1;

   Let $DB' = DB \cap$ frequent $1-$ patterns;

2. MaxPattern=$\min\left\{\frac{\text{lengthoflongesttransactionin}DB'}{\delta}, |F_1|\right\}$;

   MaxFault = #fault(MaxPattern);

3. Construct *MaxFault*+1 groups, G_i, i=0 to *MaxFault*;

   For each group G_i {
   $head_i$: generate candidate by F_1;
   Check whether candidates are frequent FT patterns;
   $tail_i$: generate candidate by $head_i$;
   Prune by the head of the next group ($head_{i+1}$);
   Check whether candidates are frequent FT patterns;
   Mid_{ij}: **For** j = 1 to $depth_i/2${
   Generate candidate C_{ij} and $C_{i(\text{depth}i-j)}$ by $mid_{i(j-1)}$;

Table 2 Parameters considered in the experiments

Parameter	Symbol	Range of values	Default value
FT parameter	δ	0.7~0.99	0.8
Minimum FT-support threshold	min_sup^{FT}	0.06~0.1	0.1
Minimum item-support threshold	min_sup^{item}	0.04~0.09	0.075

Prune{
If (candidate is the subset of $mid_{i(\text{depth}i-j+1)}$)
Set the candidate as frequent;}
Check other candidate frequent FT patterns;}
Output mid_{ik}, k=0 to depth_i
}

4 Experimental results

4.1 Algorithm efficiency

This subsection evaluates the performance of our approach. As mentioned in Section 2, the solutions used in previous studies on mining FT patterns cannot be used to solve the proportional FT-patterns mining problem. Therefore, our basic FT-BottomUP algorithm is used to show the improvement of the FT-LevelWise algorithm. The two algorithms were implemented in Java, and all experiments were performed on a 1.8-GHz Pentium 4 CPU with 384 MB of RAM running Windows XP. The experimental datasets were generated using an IBM synthetic-data generator. Each dataset contained 1000 different items and 10,000 transactions (i.e., an average of ten items in a transaction), and several potential frequent patterns with an average length of 8.

The parameters used in our simulation are listed in Table 2. The performances of FT-BottomUp and FT-LevelWise are compared on the basis of their execution times.

In the first simulation, the relation between the FT parameter and the number of FT patterns was considered. It is trivial that a larger δ results in the extraction of fewer FT patterns. This situation is especially obvious in the range of δ from 0.7 to 0.9, as shown in Fig. 2. When δ is over 0.9, the patterns mined out are close to traditional frequent patterns, i.e., exactly matched patterns without the FT property.

Figure 3 presents the total execution time of FT-BottomUp and FT-LevelWise, which clearly shows that the latter algorithm outperforms the former one.

In second set of simulations, δ was set to the default value and the reciprocal effect of the two support thresholds was investigated. Figure 4 shows the variation in the number of FT patterns when min_sup^{FT} and min_sup^{item} change.

The scalabilities of FT-BottomUp and FT-LevelWise with respect to the two support thresholds are presented in Fig. 5. The performance of FT-LevelWise is still universally better than that of FT-BottomUp. Moreover, with a constant min_sup^{item}, a smaller min_sup^{FT} results in a longer execution time.

4.2 Epitope prediction

An *epitope*, which is known as an *antigenic determinant*, is a small part of the molecular structure of an antigenic molecule that is recognized by the immune system, e.g., antibodies, B cells, and T cells (Murphy et al. 2008). A *linear epitope* is an epitope that is recognized by antibodies by its linear sequence of amino acids, or primary structure. In contrast, a *conformational epitope* is a sequence of discontinuous amino acids that come together in three dimensional conformation, or tertiary structure. Macromolecular antigens such as proteins usually have many

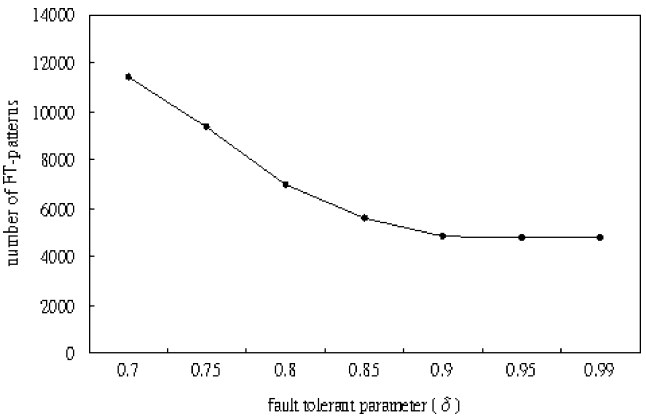

Fig. 2 Number of FT patterns vs. the FT parameters

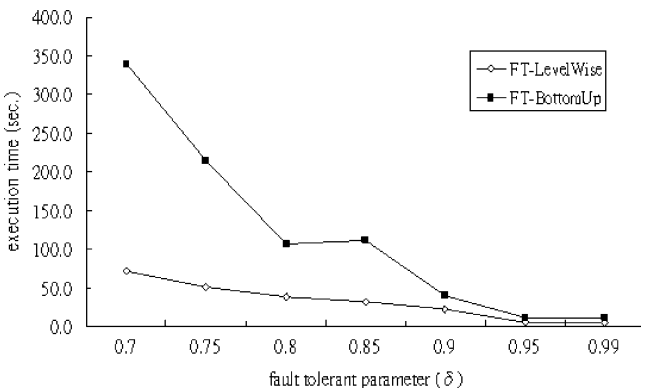

Fig. 3 Execution times of the two algorithms vs. the FT parameters

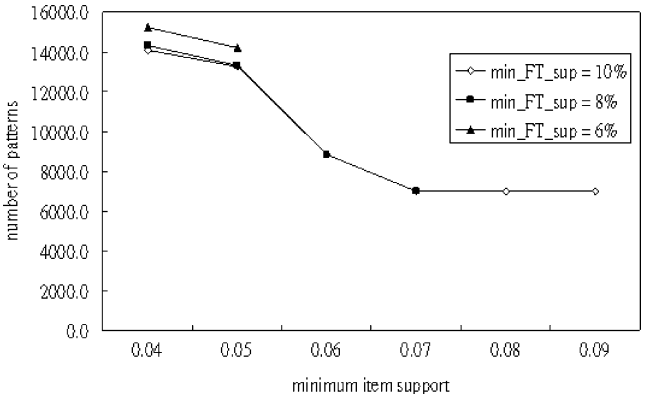

Fig. 4 Number of FT patterns

different epitopes. Moreover, the same segment of protein can be a part of more than one epitopes. In general, the length of an epitope can be from eight to 24 amino acids. Intensive research is currently taking place to design reliable tools to predict epitopes on proteins. For example, ELISA (Enzyme Linked Immunosorbent Assay) is one of well-known Immunochemical methods. The *epitope prediction problem* is to predict epitopes on proteins.

Many viruses have a spike protein that has similar function and location within the viral membrane. However, the spike protein of the Severe Acute Respiratory Syndrome-Associated coronavirus (SARS-CoV) has noted low amino acid homology with other viral spike proteins. The low similarity of amino acid sequence suggests that the spike protein of SARS-CoV may have additional functions other than the usual functions of coronavirus spike proteins (Rota et al. 2003). For SARS-CoV, the spike protein is about 1,255 amino acids long in general. The spike protein plays an important role in interactions with receptor and inducing neutralizing antibodies. Currently, only two epitopes are reported for SARS-CoV spike proteins, namely, "KLRPFERDISNV" and "PDPLKPTKRSF" (Saha et al. 2005). In this subsection, we consider the linear epitope prediction problem on spike proteins of SARS-CoV.

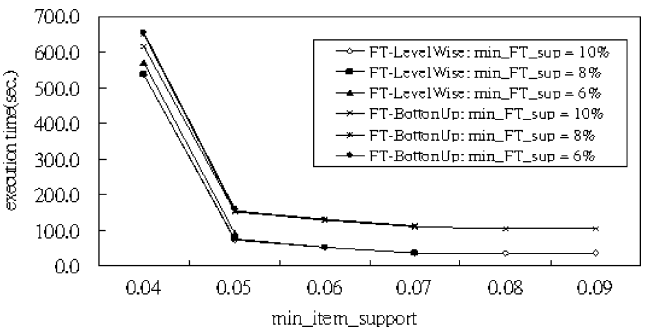

Fig. 5 Execution times of our algorithms

The inputs of most epitope prediction tools are a protein, e.g., the B-cell epitope prediction (Saha et al. 2005). In contrast, we consider a group of related proteins as our input. Our assumption is based on that most related proteins should have the same epitopes. It is like local view (a protein) versus global view (a group of related proteins). However, as mentioned above, the spike proteins of SARS-CoV are low similarity with the spike proteins of other coronaviruses. Thus, we only consider the spike proteins of SARS-CoV. Nevertheless, spike proteins among SARS-CoV are still not similar. That is, the idea of finding common segments in these proteins does not work. It happens that the concept of fault tolerance can be used in this problem. We transform the epitope prediction problem into the fault-tolerant data mining problem as follows.

We download 209 spike proteins of SARS-CoV from NCBI database. Each protein corresponds to a transaction. By using a sliding window, each segment obtained by sliding window corresponds to an item. Note that the same segment of protein can be a part of more than one epitopes. Thus, this transformation is reasonable. In this experiment, the window size is set to 15. Totally, we obtain an item set of size equal to 3,306. We expect that the output pattern is a set of epitope candidates since the pattern and each of its items obtain enough supports determined by min_sup^{FT} and $\text{min\_sup}^{\text{item}}$. The parameters used in our experiment are listed in Table 3.

By using our algorithm, the result we obtained is one maximal itemset with length 850 (respectively, 1,036) for $\text{min\_sup}^{\text{item}}$ equal to 0.65 (respectively, 0.6). By checking the obtained itemsets, the two reported epitopes (mentioned above) for SARS-CoV spike proteins are exactly contained in this result. It shows that our proposed approach is potential for predicting epitopes.

For fixed FT data mining, while the number of tolerance faults is over 3, FT-Apriori algorithm (Pei et al. 2001) runs out of memory. Therefore, we consider the case that the number of tolerance faults equal to 3 and $\text{min\_sup}^{\text{item}}$= 0.65. In the results, four maximal itemsets (with length 680, 768, 367, and 229, respectively) are obtained. The union of these four sets has similar effect to the set of size 850. It shows that the proportional FT data mining is better than the fixed FT data mining for this application.

Note that in this experiment we do not consider the property test of epitopes. For example, to be an epitope, we

Table 3 Parameters considered in the experiment

Parameter	Symbol	Value
FT parameter	Δ	0.9
Minimum FT-support threshold	min_sup^{FT}	0.7
Minimum item-support threshold	$\text{min\_sup}^{\text{item}}$	0.6/0.65

should consider some properties such as hydrophilicity, flexibility/mobility, accessibility, polarity, exposed surface, and turns. However, it is beyond the purpose of this paper.

5 Conclusion and future work

In this paper, we propose the concept of proportional FT mining of frequent patterns. In contrast to previous studies on FT data mining, the number of tolerable faults in FT patterns found by our approach is proportional to the length of the patterns. The maximum number of tolerable faults can be obtained by evaluating the maximum possible length of an FT pattern. Two algorithms that find an effective solution to the problem of mining proportional FT frequent patterns are designed. The FT-BottomUp algorithm generates all candidate patterns with the number of fault items at each level. The FT-LevelWise algorithm divides all FT patterns into several groups, generates candidate patterns from both sides of each group, and applies some criteria to prune candidate patterns. Both algorithms extract the complete set of FT patterns. The experiments demonstrate the scalability of FT-LevelWise, which clearly performs much better than FT-BottomUp since it does not need to generate as many candidate patterns, with the database scanning time being half that of FT-BottomUp. By applying our algorithm on real data as shown in Section 4.2, two reported epitopes of spike proteins of SARS-CoV can be found in our resulting itemset and the proportional FT data mining is better than the fixed FT data mining for this application.

Based upon our results, there is an ongoing challenge to find more efficient algorithms to improve the scalability of mining proportional FT frequent patterns. Another direction is to apply proportional FT data mining to real-world databases such as medical data or biological data though our proposed concept is general. As mentioned, biological data grow very fast. It is unavoidable that its data may contain noise or contaminated data. Therefore, the technique of proportional FT data mining will become an important tool in bioinformatics or biomedicine.

Reference

Agrawal, R., & Srikant, R. (1994). "Fast Algorithm for Mining Association Rules." *In Proceedings of Int. Conf. Very Large Data Bases* (VLDB'94), pp. 487–499, Santiago, Chile.

Agrawal, R., Imielinski, T., & Swami, A. (1993). "Mining Association Rules between Sets of Items in Large Databases." *In Proceedings of ACM-SIGMOD International Conference Management of Data* (SIGMOD'93), pp. 207–216, Washington, DC.

Antonie, M., Zaïane, O. R. (2004). "Mining Positive and Negative Association Rules: An Approach for Confined Rules," *European Conference on Principles of Data Mining and Knowledge Discovery* (PKDD).

Brin, S., Motwani, R., Ullman, J. D., & Tsur, S. (1997). "Dynamic Itemset Counting and Implication Rules for Market Basket Analysis," *In Proceedings of ACM-SIGMOD International Conference Management of Data* (SIGMOD'97), pp. 255–264, Tucson, AZ.

Chen, Y. P. P. (2005). *Bioinformatics Technologies*. Berlin: Springer.

Chen, Y. - L., & Ho, C. - Y. (2005). A Sampling-Based Method for Mining Frequent Patterns from Databases. *In FSKD 2005*, Changsha, China, pp 536–545.

Chen, B., Haas, P., & Scheuermann, P. (2002). "A New Two-Phase Sampling Based Algorithm for Discovering Association Rules." *In Proceedings of the 8th ACM SIGKDD International Conference Knowledge Discovery and Data Mining* (SIGKDD'02), Alberta, Canada.

Chen, L., Bhowmick, S. S., & Li, J. (2006). Mining Temporal Indirect Associations." *In Proceedings of 10th International Conference Pacific-Asia Conference* (PAKDD 2006), Singapore, pp. 425–434.

Chu, T. - P., Wu, F., & Chiang, S. - W. (2005). "Mining Frequent Pattern Using Item-Transformation Method." *In Proceedings of 4th Annual ACIS International Conference on Computer and Information Science* (ICIS 2005), South Korea, pp. 698–706.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge: Cambridge University Press.

Han, J., & Fu, Y. (1995). "Discovery of Multiple-level Association Rules from Large Databases." *In Proceedings of International Conference Very Large Data Bases* (VLDB'95), pp. 420–431, Zurich, Switzerland.

Han, J., Pei, J., & Yin, Y. (2000). "Mining Frequent Patterns Without Candidate Generation.". *In Proceedings of 2000 ACM SIGMOD International Conference on Management of data*. ACM SIGMOD Record.

Knuth, D. E., Morris, J. H., & Pratt, V. B. (1977). Fast Pattern Matching in Strings. *SIAM Journal on Computing*, *6*, 323–350.

Kotlyar, M., & Jurisica, I. (2006). Predicting protein–protein interactions by association mining. *Information Systems Frontiers*, *8*, 37–47.

Lee, C. F., Changchien, S. W., Wang, W. T., & Shen, J. J. (2006). A data mining approach to database compression. *Information Systems Frontiers*, *8*, 147–161.

Murphy, K., Travers, P., & Walport, M. (2008). *Janeway's immuno biology* (7th ed.). London: Garland Science.

Park, J. S., Chen, M. S., & Yu, P. S. (1995). An Efficient Hash-based Algorithm for Mining Association Rules." *In Proceedings of ACM-SIGMOD International Conference Management of Data* (SIGMOD'95), pp. 175–186, San Jose, CA.

Pavon, J., Viana, S., & Gomez, S. (2006). *Matrix Apriori: Speeding Up the Search for Frequent Patterns* pp. 75–82. Austria: Databases and Applications.

Pei, J., Tung, A. K. H., & Han, J. (2001). Fault-Tolerant Frequent Pattern Mining: Problems and Challenges. *DMKD'01*, Santa Barbara, CA.

Rota, P. A., et al. (2003). Characterization of a novel Coronavirus associated with Severe Acute Respiratory Syndrome. *Science*, *300*, 1394–1399.

Saha, S., Bhasin, M., & Raghava, G. P. S. (2005). Bcipep:A database of B-cell epitopes. *BMC Genomics*, *6*(1), 79.

Savasere, A., Omiecinski, E., & Navathe, S. (1995). An Efficient Algorithm for Mining Association Rules in Large Databases. *In Proceedings of International Conference Very Large Data Bases* (VLDB'95), pp. 432–443, Zurich, Switzerland.

Thiruvady, D. R., & Webb, G. I. (2004). Mining Negative Rules using GRD. *In Proceedings of PAKDD*.

Ukkonen, E. (1995). On-line Construction of Suffix-trees. *Algorithmica*, *14*, 249–260.

Wang, S. - S., & Lee, S. - Y. (2002). Mining Fault-Tolerant Frequent Patterns in Large Database. *Proceedings of International Computer Symposium*.

Yang, C., Fayyad, U., & Bradley, P. S. (2001). Efficient discovery of error-tolerant frequent itemsets in high dimensions." *In Proceedings of the seventh ACM SIGKDD International Conference on Knowledge discovery and data mining*.

Zaki, M. J. (2000). Scalable Algorithms for Association Mining. *IEEE Transaction on Knowledge and Information Engineering*, *12*(3).

Zhang, C., & Zhang, S. (2004). Efficient Mining of Both Positive and Negative Association Rules: generate both positive and negative association rules. *ACM Transactions on Information Systems*.

Guanling Lee received the B.S. M.S, and PHD degrees, all in computer science, from National Tsing Hua University, Taiwan, Republic of China, in 1995, 1997, and 2001, respectively. She joined National Dong Hwa University. Taiwan, as an assistant professor in the Department of Computer Science and Information Engineering in August 2001, and became an associate professor in 2005. Her research interests include resource management in the mobile environment, data scheduling on wireless channels, search in the P2P network and data mining.

Sheng-Lung Peng received the BS degree in Mathematics from National Tsing Hua University in 1988, and the MS and PhD degrees in Computer Science from National Chung Cheng University and National Tsing Hua University in 1992 and 1999, respectively. He joined National Dong Hwa University as an assistant professor in the Department of Computer Science and Information Engineering in August 1999, and became an associate professor in 2008. His research interests include graph theory, algorithms design, and Bioinformatics.

Yuh-Tzu Lin received the M.S. degrees in Computer Science and Information Engineering from National Dong Hwa University, Taiwan, Republic of China, in 2004. She joined Wistron Inc., as an Engineer in August 2004.