A Depth-First Search Approach for Mining Proportional Fault-Tolerant Frequent Patterns Efficiently in Large Database

Chih-Chieh Tseng, Guanling Lee*
Department of Computer Science and Information Engineering,National Dong Hwa University, Taiwan, R.O.C.

* Corresponding author. Tel.: +886-3-8634040; email: guanling@mail.ndhu.edu.tw
Manuscript submitted November 10, 2014; accepted July 1, 2015.
doi: 10.17706/jcp.10.6.388-395

Abstract: Mining of frequent patterns in databases has been studied for several years. However, real-world databases contain noise and frequent pattern mining which extracts patterns that are absolutely matched is not enough. Therefore, a research field called fault-tolerant frequent pattern (FT-pattern) mining is proposed to deal with this problem. In this paper, we consider the problem of mining proportional FT-patterns. That is, the number of faults tolerable in a pattern is proportional to the length of the pattern. To reduce the disk I/O times, a depth-first mining approach is proposed to mine proportional FT-patterns efficiently in large database. Moreover, a set of experiments is performed to show the advantage of the approach. Experimental results indicate that the proposed algorithm outperforms the other existing approach when the database size is large.

Key words: Data mining, association patterns, fault-tolerant patterns.

1. Introduction

Data mining has become a progressively important technology in recent year, especially association-rule mining which was first investigated in [1]. It explores the relationships among data items. For example, when analyzing market-basket database by association-rule mining, one may find such a rule: "Milk → Bread, support = 8%, confidence = 80%"; where the support of 8% means that milk and bread are purchased together in 8% of all transactions, and the confidence of 80% means that 80% of the customers who purchased milk also bought bread. The approaches used to find association rules can be roughly categorized into two categories. The first category is the Apriori-based algorithms. The most influential algorithm generates candidate patterns according to the anti-monotonicity is proposed in [2]. And thereafter, there are various techniques have been proposed to overcome these problems [3]-[7]. The second category is the tree-based algorithms. [8] was the first to propose the FP-tree (frequent-pattern tree) structure to mine frequent patterns. 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 data-base 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, H-mine [9]. 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. Since each of the above approaches exhibits specific advantages and limitations, [10] suggests opportunistically choosing between two structures, array-based or tree-based, to represent projected transaction subsets, and heuristically decides to build an unfiltered pseudo projection or to make a filtered copy according to features of the subsets. Basically, the algorithm grows the FP-tree by a depth-first search, whereas a breadth-first search is used to build the upper portion of the tree if necessary.

Traditional association-rule mining can only extracts patterns that are absolutely matched. However, real-world databases contain noise that can make important information ambiguous; resulting in it will not appear in the mining result. Therefore, we need a method that copes with such variations in an association pattern, which is called a *fault-tolerant pattern* (*FT-pattern*). Discovering frequent FT-patterns in muti-dimensional database is first proposed in [11]. However, this approach may generate *sparse patterns*, which may contain subpatterns that do not appear frequently. Another milestone of FT-pattern mining is [12], in which extending Apriori and developing FT-Apriori for frequent FT-pattern mining allows the mining out of a complete set of FT-patterns. However, [12] still has the drawbacks of Apriori-based algorithms. In response, [13] developed FTP-mine, which finds FT-patterns by the concept of pattern growth. [14] uses bit vector representation to represent data and developed a vector-based mining algorithm. [15] generalized the problem of mining fixed-value FT-patterns into relaxation criteria and constraints and proposed to mine only statistical information of the corresponding FT-patterns. [16] considers a constraint-based mining approach for relevant fault-tolerant formal concept mining. The main defect of previous approaches is their definition of the number of faults tolerable in a pattern as a fixed number. Defining the number of faults tolerable in the patterns as a fixed number of items is not objective. 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.

Instead of mining patterns of different lengths while tolerating a fixed number of faults, [17] addressed the problem of mining proportional FT-patterns in which the number of faults tolerable in the pattern is proportional to the pattern length. Moreover, the concept and proposed methods are demonstrated in [18] for predicting epitopes of spike proteins of SARS-CoV and concludes that the patterns are more concise than that of fixed FT-patterns mining for this application. However, the proposed algorithms are too violent. Therefore, [19] presents a framework named *PFM* to mine proportional FT-patterns efficiently. The PFM algorithm for mining proportional FT-patterns can be regarded as a breadth-first mining strategy, and is efficient if the entire bitmap of the database can be loaded into memory. However, because *PFM* generates candidates level by level, the bitvectors associated with each candidate should be loaded into memory for further checking. Memory swapping is performed when the bitvectors cannot fit into memory, and it makes the cost of I/O operations high. Therefore, to reduce the number of disk I/O operations, a depth-first mining strategy is proposed in this paper. The remainder of this paper is organized as follows. Section 2 introduces the problem definition and preliminaries. Section 3 describes the proposed algorithms in detail. Section 4 discusses the experimental results and analysis. Conclusions are finally drawn in Section 5.

2. Preliminaries

Let pattern $X = \{i_1, \ldots, i_n\}$ be a set of items and $|X|$ denotes the length of X. X is called a $|X|$-*itemset* since it contains $|X|$ items. A transaction $T = (tid, X)$ is a 2-tuple, where *tid* indicates 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. Extending the problem of mining frequent patterns, the frequent FT-pattern mining problem

relaxes the definition of containing to FT-containing. The definition of proportional frequent FT-patterns is given as follows.

Definition 2.1 (Proportional frequent FT-pattern) [8]

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 ($0.5 < \delta < 1$) *iff* there exists $P' \subseteq P$ such that $P' \subseteq X$ and $|P'| \geq \lceil \delta \times |P| \rceil$. 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 a frequent FT-pattern *iff*

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

For example, Table 1 shows a transaction database *TDB*. Suppose that the minimal item-support threshold $\text{min\_sup}^{item} = 3$, and the FT-support threshold $\text{min\_sup}^{FT} = 5$. Let the FT parameter $\delta = 0.6$, i.e., the maximal number of allowable mismatches of a pattern P is $\lfloor (1-0.6) \times |P| \rfloor$. For pattern $P = abcde$, $B(P)$ includes transaction 10, 20, 40, 50, 70, 90 since they all FT-contain P, and we have $\sup^{FT}(P) = 6$ which is no smaller than min_sup^{FT}. Moreover, each item of P appears in at least three transactions of $B(P)$. Therefore, pattern P is a proportional frequent FT-pattern.

Table 1. Example Database

	a	b	c	d	e
10	1	1	1	0	0
20	1	0	1	1	0
30	0	0	0	1	1
40	1	1	0	1	1
50	0	0	1	1	1
60	1	0	0	0	1
70	0	1	1	1	1
80	0	0	0	1	0
90	1	1	1	0	1
100	0	0	1	0	1

As discussed in [19], when the value of FT parameter δ is smaller than or equal to 0.5, i.e., $0 < \delta \leq 0.5$, will lead to generate a large number of candidates that are not interesting. For example, as shown in Table 2, let $\delta = 0.5$, min_sup^{item} =40%, min_sup^{FT} = 50%, and the pattern P = {hammer, nail, pencil, eraser} can be a legal FT-pattern. However, it is easily observed that there does not exist any interesting relation between the two subpatterns {hammer, nail} and {pencil, eraser}, and P is not an informative pattern. Therefore, following the concept proposed in [19], we also focus on the range, $0.5 < \delta < 1$, to promote the mining efficiency.

Table 2. Transactional Database

	Hammer	Nail	Pencil	Eraser
10	1	1	0	0
20	1	1	0	0
30	1	1	0	0
40	1	1	0	0
50	1	1	0	0
60	0	0	1	1
70	0	0	1	1
80	0	0	1	1
90	0	0	1	1
100	0	0	1	1

In [19], a graph structure called *FT-association graph* is proposed to present the original database. With

this structure, *PFM* algorithm is proposed to mine proportional FT-pattern. In the following, we show the three important lemmas proposed in [19] which was used to prune candidates once the FT-association graph is constructed.

Lemma 2.1 If an item *y* is away from *x* for the distance greater than 2 in the FT-association graph, then a pattern P containing both *x* and *y* cannot be a frequent FT-pattern.

Lemma 2.2. If *P* is a frequent FT-pattern, for each item *x* of *P*, there must exist $\lceil \delta \times |P| \rceil$-1 items in *P* which are neighbors of *x* in the FT-association graph.

Lemma 2.3. Given *a* pattern *P,* the upper bound of $\sup^{FT}(P)$, denoted as $\max\_\sup^{FT}(P)$, is equal to $\sum_{\forall P' \subseteq P, |P'| = \lceil \delta \ |P| \rceil} \max\_\sup(P')$. Additionally, if $\max\_\sup^{FT}(P) < \min\_\sup^{FT}$, then *P* cannot be a frequent FT-pattern.

The whole process of *PFM* can be decomposed into three parts. The first part is data preprocessing. In this step, the original database is transformed into a bitmap and FT-association graph is constructed. The second part is candidate generation and pruning. Instead of scanning the entire database, *PFM* only loads part of the bitmap according to the items of each candidate in the candidates checking phase. That is, when generating candidates containing item *x*, by Lemma 2.1, only item *y* such that $d(x, y) \le 2$ is considered to compose to a candidate with *x*. The last part is to check the loaded bitmap to determine whether the candidates generated in the second part are frequent FT-patterns.

3. Depth-First Mining Approach

PFM algorithm can mine proportional FT-patterns efficiently in the case of bitmap can fit into memory. However, in large database mining problem, the bitmap of the database are often too big to fit into memory. Because PFM generates candidates level by level, the bitvectors associated with each candidate should be loaded into memory again and again for further checking. Memory swapping is performed when the bitvectors cannot fit into memory, and it makes the cost of I/O operations high. Therefore, in this paper, a Depth-First Proportional FT-pattern Mining algorithm (*DFPFM*) is proposed to reduce the number of disk I/O operations. While generating candidates for item *x*, only the bitvectors of *x* and all *y* where $d(x, y) \le 2$ (Lemma 2.1) are needed to be loaded into memory. The complete set of candidates of *x* is generated, rather than generating candidates with same length first. For each candidate *P*, extract bitmap (*P*) from the loaded bitmap. Check *P* by using the bitmap (*P*).

The detail algorithm is listed in Fig. 1. $C_{i,x}$ denotes the set of candidates generated for *x* and with length-*I*, and $F_{i,x}$ denotes the set of frequent FT-patterns checked from the candidates in $C_{i,x}$. Lines 1~4 are precessing steps. The minimum length of mined out proportional FT-patterns, denoted as *MinPattern*, depends on δ. Since the mined patterns must be able to tolerate at least 1 mismatch, the inequality $\lfloor (1-\delta) \times |P| \rfloor \ge 1$ holds, and can be recast as $|P| \ge \left\lceil \frac{1}{1-\delta} \right\rceil$. Hence, the lengths of all mined patterns have to be greater than or equal to $\left\lceil \frac{1}{1-\delta} \right\rceil$, and candidates are generated from the length $\left\lceil \frac{1}{1-\delta} \right\rceil$. Then for each frequent 1 itemset *x* (line 5), we load the bitvectors from the bitmap of the database for those items which may form a FT-pattern with *x* (line 6). After that, we generate all possible FT-pattern candiates associated with *x* (lines 7-10), and check the associated bitmap to determine whether it is a frequent proposional FT-pattern (lines 11-26).

4. Experimental Results

The test data was generated by an IBM synthetic-data generator. The simulations were implemented in JAVA and all the experiments were performed by an Intel Pentium 3.0 GHz computer. Table 3 shows the

parameter settings for comparing efficiencies of *DFPFM* and *PFM*. Here the maximum bitmap size that can be stored in memory is assumed to be 30000×1000. Bitmaps that exceed this size can only be stored in disk.

Algorithm DFPFM

Input: Transaction database *TDB*
Frequent item-support threshold: min_sup^{item}
Frequent FT support threshold: min_sup^{FT}
FT parameter: δ

Output: frequent FT-patterns

Method:

1. Scan *TDB* to a. transform *TDB* to bitmap;
   b. construct the FT-association graph;
   c. Find F_1;
2. Remove items not in F_1 from the bitmap and FT-association graph;
3. $MinPattern = \lceil 1/(1-\delta) \rceil$;
4. $MaxPattern = min(\lfloor L/\delta \rfloor, |F_1|)$;
5. for each node *x* of FT-association graph {
6. Load columns which belong to *x* and all *y* where $d(x, y) \le 2$ from the bitmap;
7. for (*i* = MinPattern; *i* ≤ MaxPattern; *i*++) {
8. if (#*fault*(*i*) = #*fault*(*i*−1))
   Generate $C_{i,x}$ from $F_{i-1,x}$;
9. else
10. $C_{i,x} = C_{i,x} \cup \{ P \mid x \in P,\ |P| = i,\ \forall y \in P,\ d(x,y) \le 2,$
    $n(x), n(y) \ge \lceil \delta \times |P| \rceil - 1,\ max\_sup^{FT}(P) \ge min\_sup^{FT} \}$;
11. for each candidate *P* of $C_{i,x}$ {
12. Extract *bitmap*(*P*);
13. for each transaction *T* of *bitmap*(*P*) {
14. if there are $\lceil \delta \times |P| \rceil$ 1s in *T*
15. $sup^{FT}(P) = sup^{FT}(P) + 1$;
16. else remove *T*;
17. }
18. if $sup^{FT}(P) \ge min\_sup^{FT}$
19. for each column of *x* of *bitmap*(*P*) {
20. $sup^{item}{}_{B(P)}(x)$=the number of 1s in the column of *x*;
21. if $sup^{item}{}_{B(P)}(x) < min\_sup^{item}$ then discard *P*;
22. }
23. if *P* is not discarded, add *P* to $F_{i,x}$;
24. }
25. }
26. }

Fig. 1. DFPFM algorithm.

As shown in Fig. 2, since the size of the transformed bitmap is exactly 30000×1000 so that the bitmap can be stored in memory. As a result, PFM performs much better than DFPFM for it have not to perform disk I/O operations and generates less candidates than that of DFPFM during the mining process. Fig. 3 indicates that PFM still outperforms DFPFM when the bitmap size is less than 30000×1000. Nevertheless, PFM must perform one or more bitmap swap to check each candidate when the bitmap size is greater than 30000 ×1000. DFPFM outperforms PFM in this situation. Fig. 3 also shows that a denser dataset (with fewer distinct items) leads to poorer performance in both DFPFM and PFM, since fewer distinct items results in

more potential candidates.

Table 3. Parameter Settings for FT-Pattern Mining

Notation	Meaning	Default	Range
TLEN	Average length of a transaction	15	
I	Number of distinct items	1000	500~1500
N	Total number of transactions	30000	10000~100000
δ	FT parameter	0.8	0.6~0.9
min_sup^{item}	Minimum item support threshold	0.04	
min_sup^{FT}	Minimum FT support threshold	0.06	

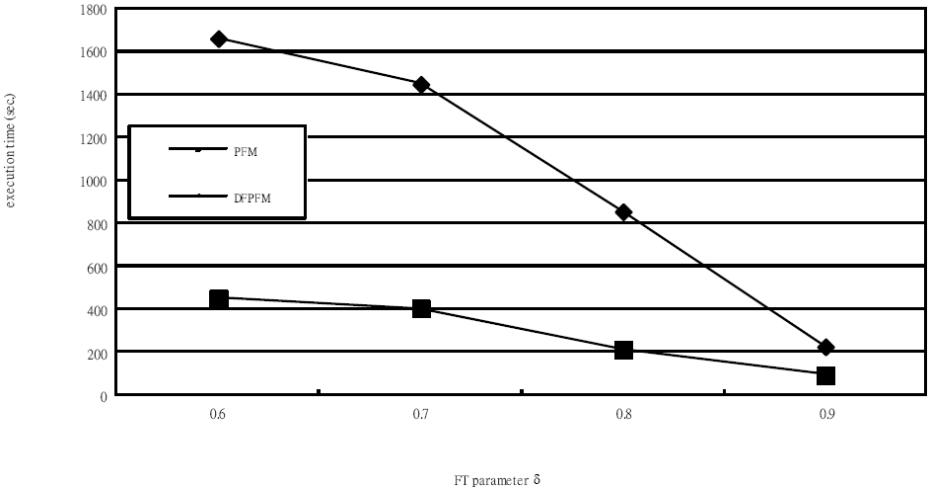

Fig. 2. Execution times of PFM and DFPFM vs. δ.

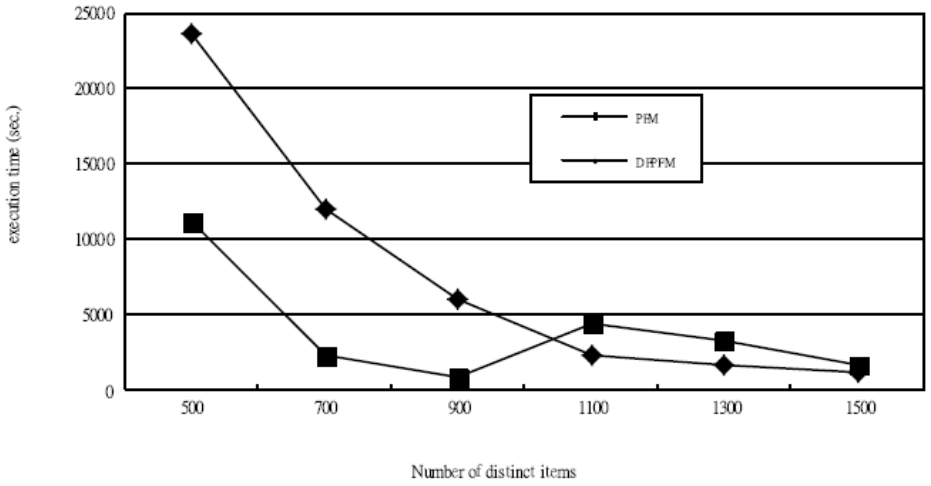

Fig. 3. Execution times of PFM and DFPFM vs. number of distinct items.

Finally, Fig. 4 shows the scalabilities of PFM and DFPFM. Due to the limitation of memory size, more data leads to more swap process handled by PFM for checking each candidate. Therefore, as shown in Fig. 4, PFM scales badly when the dataset is too big to fit into memory. DFPFM, which adopts a depth-first mining strategy, successfully reduces a large number of disk I/O operations. Hence, DFPFM is more scalable than PFM.

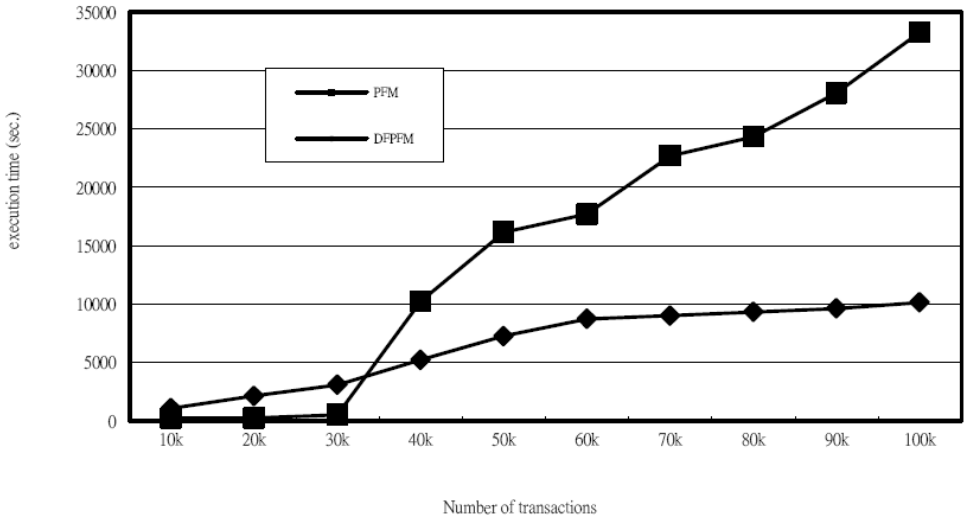

Fig. 4. Execution times of PFM and DFPFM vs. number of transactions.

5. Conclusions

In this paper, we consider the problem of mining proportional FT-patterns. Moreover, we proposed a depth-first mining approach to mine proportional FT-patterns efficiently in large database. Unlike *PFM* algorithm which generates candidates level by level, we propose a strategy of generating candidates in a depth first manner to reduce the disk I/O times when the bitmap of the database can not fit into the main memory. Moreover, a set of experiments is performed to show the advantage of the approach. Experimental results indicate that the *DFPFM* algorithm, by adopting a depth first mining strategy, can substantially reduce the number of disk I/O process when the bitmap is too big to fit into memory and is more scalable than *PFM*.

References

[1] Agrawal, R., Imielinski, T., & Swami, A. (May 1993). Mining association rules between sets of items in large databases. *Proceedings of 1993 ACM-SIGMOD Int. Conf. Management of Data* (pp. 207-216). Washington, DC.

[2] Agrawal, R., & Srikant, R., (Sept. 1994). Fast algorithm for mining association rules. *Proceedings of 1994 Int. Conf. Very Large Data Bases* (pp. 487-499). Santiago, Chile.

[3] Park, J. S., Chen M. S., & Yu, P. S. (May 1995). An efficient hash-based algorithm for mining association rules. *Proceedings of 1995 ACM-SIGMOD Int. Conf. Management of Data* (pp. 175-186). San Jose, Ca.

[4] Han, J., & Fu, Y. (Sept. 1995). Discovery of multiple-level association rules from large databases. *Proceedings of 1995 Int. Conf. Very Large Data Bases* (pp. 420-431). Zurich, Switzerland.

[5] Jouni, K. S., & Heikki, M. (2004). Dense itemsets. *Proceedings of Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 683-688). Seattle, WA, USA.

[6] Savasere, A., Omiecinski, E., & Navathe, S. (Sept. 1995). An efficient algorithm for mining association rules in large databases. *Proceedings of 1995 Int. Conf. Very Large Data Bases* (pp. 432-443). Zurich, Switzerland.

[7] Brin, S., Motwani, R., Ullman, J. D., & Tsur, S. (May 1997). Dynamic itemset counting and implication rules for market basket data. *Proceedings of ACM SIGMOD International Conference on the Management of Data* (pp. 255-264).

[8] Han, J.-W., Pei J., & Yin, Y.-W. (May 2000). Mining frequent patterns without candidate generation. *Proceedings of 2000 ACM SIGMOD International Conference on Management of Data*.

[9] Pei, J., Han, J., & Lakshmanan, L. V. S. (April 2001). Mining frequent itemsets with convertible constraints. *Proceedings of 2001 Int. Conf. Data Engineering* (pp. 332, 433). Heidelberg, Germany.

[10] Liu, J., Pan, Y., Wang, K., & Han, J. (July 2002). Mining frequent item sets by opportunistic pro-jection. *Proceedings of the 8th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining* (pp. 229-238). Alberta, Canada.

[11] Yang, C., Fayyad, U., Bradley, P. S. (2001). Efficient discovery of error-tolerant frequent itemsets in high dimensions. *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[12] Pei, J., Tung, A. K. H., & Han, J. (May 2001). Fault-tolerant frequent pattern mining: problems and challenges. *Proceedings of the DMKD'01*.

[13] Wang, S. S., & Lee, S. Y. (2002). Mining fault-tolerant frequent patterns in large database.

[14] Koh, J. L., & Yo, P. W. (2005). An efficient approach for mining fault-tolerant frequent patterns based on bit vector representations. *Proceedings of Int. Conf. Database Systems for Advanced Applications*.

[15] Poernomo, A. K., & Gopalkrishnan, V. (2007). Mining statistical infor-mation of frequent fault-tolerant patterns in transactional da-tabases. *Proceedings of IEEE International Conference on Data Mining*.

[16] Besson, J., Pensa, R., Robardet, C., & Boulicaut, J.-F. (2006). Constraint-based mining of fault-tolerant patterns from Boolean data. *Knowledge Discovery in Inductive Databases*, 55-71.

[17] Lee, G., & Lin, Y. T., (Nov. 2006). A study on proportional fault-tolerant data mining. *Proceedings of 2006 Int. Conf. Innovations in Information Technology* (pp. 1-5). Dubai.

[18] Lee, G., Peng, S. L., & Lin, Y. T. (2009). Proportional fault-tolerant data mining with applications to bioinformatics. *Journal of Information Systems Frontiers*.

[19] Zeng, J. J., Lee, G., & Lee, C. C. (2008). Mining fault-tolerant frequent patterns efficiently with powerful pruning. *Proceedings of Int. Conf. ACM Symposium on Applied Computing*. Fortaleza, Brazil.

Chih-Chieh Tseng received the M.S. degree in computer science and information engineering, from National Dong Hwa University, Taiwan, Republic of China, in 2008. His research interests include data mining and database.

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.