Mining statistical information of frequent fault-tolerant patterns in transactional databases

Ardian Kristanto Poernomo
Nanyang Technological University
Singapore, 639798
ardi0002@ntu.edu.sg

Vivekanand Gopalkrishnan
Nanyang Technological University
Singapore, 639798
asvivek@ntu.edu.sg

Abstract

Constraints applied on classic frequent patterns are too strict and may cause interesting patterns to be missed. Hence, researchers have proposed to mine a more relaxed version of frequent patterns, where transactions are allowed to miss some items in the itemset they support. Patterns exhibiting such “faults” are called frequent fault-tolerant patterns (FFT-patterns) if they are significant in number. In this paper, the term “pattern” is distinguished from “itemset” as referring to a pair (tidset × itemset).

Unlike classical frequent patterns, the number of FFT-patterns grows exponentially not only with the number of items, but also with the number of transactions. Since the latter may reach millions, mining FFT-patterns by enumerating them becomes infeasible. Hence, the challenge is to represent FFT-patterns concisely without losing any useful information. To address this, we draw on the observation that, in transactional databases, the transactions themselves are not important from the data mining point-of-view; i.e. researchers are interested in finding itemsets contained in lots of transactions, rather than in the transactions per se. Therefore, we propose to mine only the frequent itemsets along with the statistical information of the supporting transaction sets, rather than enumerate entire FFT-patterns. Then we present our approach - the BIAS framework, consisting of Backtracking algorithm, Integer Linear Programming (ILP) constraints, and aggregation statistics to solve this problem. Algorithms under this framework not only increase the efficiency of the FFT-patterns mining process by more than an order of magnitude, but also provide a more comprehensive analysis of FFT-Patterns.

1 Introduction

The frequent patterns mining problem [2] has been widely studied in data mining. Some researchers use the terms pattern and itemset interchangeably, but in this paper, we distinguish the term pattern as pair (tidset × itemset) instead of itemset only.

<table>
<thead>
<tr>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c, d</td>
</tr>
<tr>
<td>2</td>
<td>b, c</td>
</tr>
<tr>
<td>3</td>
<td>a, b, d</td>
</tr>
<tr>
<td>4</td>
<td>a, b</td>
</tr>
</tbody>
</table>

Figure 1. Transactional database: table and matrix form

Yang et al [14] argued that analysts would find it useful to mine groups of similar transactions that share most of the items, but these cannot be discovered by traditional approaches due to their harsh definition of support. Hence, they proposed to mine a more relaxed version of frequent patterns, where transactions are allowed to miss some items in the itemset they support.

Patterns significantly supported in such a manner are called frequent fault-tolerant patterns (FFT-patterns for short). Let us consider the transactional database as shown in Figure 1, which will be used as a running example in the rest of this paper. The database consists of a set of transactions (henceforth called tidset for brevity) over an itemset, and is illustrated in table and matrix forms. Here, we highlight some examples of FFT-patterns. We can see that when supporting transactions are allowed to miss one item in the itemset they support, three transactions, viz. 2, 3, 4 support itemset \{a, b\}; when supported items are allowed to be missed by one transaction in their supporting set, itemset \{a, b, c, d\} is supported by tidsets \{1, 3\}, \{1, 4\} and \{2, 3\}; and when three such transaction-item misses are allowed in a pair (tidset × itemset), \{(2, 3, 4) × \{a, b, c\}\}, \{(1, 2, 3) × \{b, c, d\}\}, \{(2, 3) × \{a, b, c, d\}\} satisfy the crite-
ria. These examples while not comprehensive, illustrate the variety of FFT-patterns.

Consequently, several definitions of FFT-patterns have been proposed [9, 10, 12, 3, 6, 7, 8, 11] to accommodate this diversity. However, a general problem with FFT-patterns is that they not only grow exponentially with the number of items, but also with the number of transactions. Since the number of transactions may reach millions, this implies that mining FFT-patterns is infeasible. Several approaches have been proposed to tackle this problem: [9, 14] relax the constraints, while [3, 10] add more constraints, and [14, 7, 8] perform approximation. Nevertheless, all of these approaches either still produce too many results, or might lose interesting patterns.

In transactional databases, the transactions themselves are not important from the data mining point-of-view, i.e. the interest is in finding itemsets which are contained in a lot of transactions, instead of finding the transactions themselves. Therefore, we propose to mine all frequent itemsets, along with the statistical information of the supporting-transactions sets (formally defined in section 3.1). We choose this approach because (1) this approach doesn’t lose any interesting itemsets (no false negatives), and (2) we find that the statistical information can concisely describe the characteristics of huge data.

In this paper, we formulate the general definitions of FFT-patterns in terms of basic relaxation criteria, and show how new and existing variants of such patterns can be formed from combinations of those criteria. Then we present our BIAS framework1 consisting of Backtracking algorithm, Integer Linear Programming (ILP) constraints, and aggregation statistics to solve this problem, and demonstrate its application for all problem variants. Note that ILP is used here to define constraints of a combinatoric problem, and not to optimize an objective function as in usual LP. This framework can be used to mine statistical information of the FFT-patterns efficiently without generating the patterns themselves.

The rest of this paper is organized as follows. Section 2 discusses previous works related to the problem of FFT-patterns mining, and links them to the variants described in our paper. Section 3 presents the general problem definition, and shows how variants of FFT-patterns can be derived from constraints. Section 4 describes the BIAS framework to mine frequent itemsets as well as the statistics of their supporting-tidsets, and algorithms under this framework are explained in detail for each variant in section 5. These approaches are compared with related techniques, and section 6 shows the experimental results of these comparisons. Finally, section 7 concludes with some directions for future work.

2 Related Work

Pioneering work in FFT-patterns mining was by Yang et al [14] in 2001, where the amount of relaxation is proportional to the size of the pattern. An FFT-pattern, also called strong-ETI, is defined as a sub-matrix of itemset and transaction-set, where each transaction contains a certain proportion of the itemset. Besides [14], other papers mining FFT-patterns using proportion are [10, 7, 8]. In [10] and [14], the objective is to find interesting patterns, while [7] and [8] aim to approximate the pure frequent itemsets in a noisy data.

FFT-patterns mining with proportional relaxation, though natural, is intractable [3], because this problem lacks the anti-monotonicity property2 of itemsets. With anti-monotonicity property, we can safely stop extending an itemset once it is not frequent, so the mining complexity is proportional to the number of frequent itemsets. This is not the case when mining using proportional relaxation. Moreover, techniques using fixed-value relaxation [9, 12, 3, 11] demonstrate that interesting itemsets can be mined in real-world data. Since our paper focuses on fixed-value fault-tolerance, we skip details on these other variants.

The first piece of work using fixed-value as relaxation on FFT-patterns mining was authored by Pei et al [9]. The definition of FFT-pattern here is similar to the one given in [14], but they employ fixed-value relaxation instead of proportional relaxation, i.e. every transaction is allowed to miss a fixed number of items. This definition is equivalent to that of FFT- pattern in our paper (c.f. section 3.2). To mine such patterns, the authors used an apriori-based method. Koh and Yo [6] tackled the same problem using a backtracking approach, and empirically showed that their approach improves the efficiency of the previous approach by an order of magnitude.

A more complicated problem was tackled by Sim et al [11], where the objective is to mine maximal quasi-bicliques in bipartite graphs. A quasi-biclique is maximal if it is not a proper subset of any other quasi-biclique. In [11], the relaxation criteria is applied to both sides of the bipartite graph, i.e. each vertex is allowed to disconnect to at most ε vertices in the other partite-set. To solve this problem, they proposed algorithm CompleteQB which is based on the backtracking approach. The constraints for this problem are exactly the same as for our FFT T -pattern (c.f. section 3.2).

Besson et al [3] relate transactional databases with formal concept analysis in lattice theory. They performed comparisons on several definitions of FFT-patterns (DRBS, CBS, and FBS) with the objective of finding interesting patterns. However, all those definitions incorporate a con-

1BIAS is an acronym for Backtracking, Integer linear programming, and Aggregation Statistics

2Anti-monotonicity property means if f(X) is true for a function f and a set X, then f(X1 ⊆ X) is also true.
sistency constraint, which restricts all transactions (resp: items) outside the patterns to contain lesser items (resp: transactions) than those inside the pattern. The implications of this constraint are illustrated in Example 1.

Example 1. Consider the database as depicted in Figure 1. Let the minimum support be 3, and the allowed number of misses in each transaction and item be 1. No FFT-pattern with itemset \( \{b,c\} \) can include transactions 3 and 4 together, since they both miss item c (no. of misses for item c becomes 2). Conversely, neither of them can be excluded from any pattern that contains transaction 2, as it would violate the consistency constraint. As a result, \( \{b,c\} \) is not considered interesting (frequent). On the other hand, if the dataset had contained only transactions 1, 2 and 3 (or 1, 2 and 4), we find patterns that support itemset \( \{b,c\} \), which is now considered interesting. Therefore we conclude that such constraints are not appropriate for mining itemsets with fixed tolerance in transactional databases.

3 Problem Definition

3.1 Basic Definition and Notation

In this paper, we deal with transactional databases. A transactional database \( \mathcal{D} \) consists of transactions \( T \) over a set of items \( I \), and can be represented by a \( (T \times I) \) matrix of transactions and items.

\( T(x \in I) \) (resp: \( T(X \subseteq I) \)) represents a tidset where each transaction contains item \( x \) (resp: itemset \( X \)). Similarly \( I(y \in T) \) (resp: \( I(Y \subseteq T) \)) represents an itemset where each item is contained in transaction \( y \) (resp: tidset \( Y \)).

Definition 1. [PATTERN] A pattern is defined as a submatrix \( (Y \subseteq T) \times (X \subseteq I) \) of \( \mathcal{D} \). A pattern \( Y \times X \) is said to be associated with itemset \( X \).

In classic frequent itemset mining, a pattern \( \mathcal{P} \) is interesting if all its transactions contains all its items, i.e. all \( (tid, item) \) pairs in the pattern are present in the database. In fault-tolerant mining, we relax this interesting constraint by allowing \( \mathcal{P} \) to contain a constant number of \( (tid, item) \) pairs which are actually missing in the database. This relaxation is formally defined by the following relaxation criteria, where we distinguish between the allowed number of misses in each transaction, each item, and the entire pattern.

Transaction relaxation In a pattern \( Y \times X \), the total number of misses in each transaction are at most \( \epsilon_t \), i.e. \( \forall y \in Y, |I(y) \cap X| \geq |X| - \epsilon_t \). This relaxation allows more transactions to be included in the pattern than would be allowed by the classic definition.

Item relaxation In a pattern \( Y \times X \), the total number of misses in each item are at most \( \epsilon_i \), i.e. \( \forall x \in X, |T(x) \cap Y| \geq |Y| - \epsilon_i \). This relaxation allows more items to be included in the pattern than would be allowed by the classic definition.

Pattern relaxation In a pattern \( Y \times X \), the total number of misses in the pattern are at most \( \epsilon_p \), i.e. \( \sum_{y \in Y} |I(y) \cap X| \geq |Y| - |X| - \epsilon_p \). This relaxation deals with the associativity of transactions and items in the pattern.

The set of all patterns satisfying the relaxation criteria may be very large and undesirable. In order to reduce this, we consider the utility of the patterns in terms of the support for their itemsets and maximality of their tidsets, where the support of a pattern is calculated as the size of its tidset. The set of such patterns can be limited by the following constraints.

Support constraint A pattern \( Y \times X \) is frequent, i.e. \( |Y| \geq \xi \), where \( \xi \) is the minimum support threshold desired by the mining process. This constraint ensures that any interesting behavior captured in the pattern is observed by a substantial number of transactions.

Maximality constraint A pattern \( Y \times X \) relaxed by any combination of the relaxation criteria is maximal. This implies that \( S(Y' \times X), Y \subseteq Y' \subseteq T \) satisfying the given combination of relaxation criteria.

Support constraint and maximality constraint are denoted as utility constraints. Note that these constraints only restrict support and maximality in the tidsets, and not in the entire pattern as in usual terminology.

Definition 2. [FFT-PATTERN] An FFT-pattern is a pattern satisfying all utility constraints and any given combination of relaxation criteria.

Definition 3. [SUPPORT-TIDSET] Tidset \( Y \) supports itemset \( X \) if \( (Y \times X) \) is an FFT-pattern. The set of all support-tidsets of \( X \) is denoted as \( S(X) \).

Definition 4. [FREQUENT ITEMSET] An itemset \( X \) is frequent if \( S(X) \neq \emptyset \).

As mentioned earlier, instead of mining all FFT-patterns, we propose to mine all frequent itemsets along with the statistics of their support-tidsets. In this work, we calculate the maximum support, the average support, and the number of FFT-patterns associated with an itemset as the statistical information, which we consider useful for itemset analysis. Other statistical information such as median and mode can also be mined without adding much complexity.

In a market basket database, while \( |I| \) may reach thousands, the maximum size of any practical itemset is around 10 [13]. We denote the maximum size of an FFT-itemset as \( X_{max} \) and use it to provide a tight upper bound for some operations (c.f. section 5), since \( X_{max} \ll |I| \) in practice.
3.2 Variants of the FFT-pattern

In this paper, combinations of above relaxation criteria have been used to form three variants of the FFT-pattern, viz. FFT\(_T\)-pattern (using transaction relaxation), FFT\(_{TT}\)-pattern (using transaction relaxation and item relaxation), and FFT\(_P\)-pattern (using pattern relaxation). Variables subscripted with \(T\), \(TT\), or \(P\) belong to the corresponding specific variants of FFT-pattern.

The aim of introducing three different variants is mainly to show how the BIAS framework addresses each relaxation criterion. Depending on the dataset, some relaxation criterion by itself might not be pertinent, as the resulting pattern may be sparse. However, combining the criteria can produce good variants, like FFT\(_T\)-pattern [9] and FFT\(_{TT}\)-pattern [11] which have already been successfully applied to real world problems. Differences among the variants in term of the patterns mined are shown in Example 2.

Example 2. Consider the database as illustrated in Figure 1. Let \(\epsilon_t = \epsilon_i = \epsilon_p = 1\), and \(\xi = 1\). Itemset \(X = \{b, c\}\) is associated with (a) one unique FFT\(_T\)-pattern, \(\{(1, 2, 3, 4) \times X\}\), and its corresponding \(S_T(X)\) is a singleton \(\{\{1, 2, 3, 4\}\}\); (b) two FFT\(_{TT}\)-patterns, \(\{(1, 2, 3) \times X\}\) and \(\{(1, 2, 4) \times X\}\), with \(S_{TT}(X) = \{\{1, 2, 3\}, \{1, 2, 4\}\}\); and (c) three FFT\(_P\)-patterns, \(\{(1, 2) \times X\}, \{(2, 3) \times X\}\), and \(\{(2, 4) \times X\}\), with \(S_P(X) = \{\{1, 2\}, \{2, 3\}, \{2, 4\}\}\).

Despite these differences, all the FFT-pattern variants share the anti-monotonicity property.

Theorem 1. FFT-pattern is anti-monotone.

Proof. Since every relaxation criterion on the FFT-patterns is based on fixed-value, the allowed number of misses is a constant. The number of misses in an FFT-pattern is thus invariant to its size, and addition of items and/or transactions will never reduce the amount of misses.

This property is very important for designing algorithms to mine FFT-itemsets. If an itemset is not frequent (regardless of the definition), then none of its supersets are frequent. Based on this property, we can enumerate all results using the pattern growth approach [2], and safely prune when the current itemset is not frequent.

4 BIAS framework

The BIAS framework, consisting of Backtracking algorithm, Integer linear programming (ILP), and aggregation statistic, is designed as follows. First, the backtracking algorithm iterates all necessary and relevant itemsets, while concurrently maintaining the necessary aggregation statistic of the transactions. For each itemset \(X\), the statistical information of \(S(X)\) is determined by ILP incorporating the corresponding aggregation statistic.

4.1 Backtracking Algorithm

The backtracking algorithm (Algorithm 1) enumerates the itemsets by inserting items one by one. Once it finishes enumerating all itemsets containing an item, it backtracks, then inserts another item. The complete process can be seen as an enumeration tree [4]. As FFT-itemset is anti-monotone, once an itemset is not frequent, we can safely prune it by not traversing its subtree.

Since we enumerate all FFT-itemsets, this backtracking algorithm combined with pruning, traverses all necessary and relevant result states. This means that we cannot expect any algorithm that iterates the itemsets to be asymptotically more efficient than this algorithm.

Algorithm 1: Backtracking algorithm

```plaintext
1: call Backtrack(∅);
2:
3: Function Backtrack(X) :: void
4: for each item i extension of X
5: extend itemset X with item i while maintaining aggregation statistics;
6: Form ILP based on the aggregation statistics;
7: if (X) is frequent
8: output the statistical information of S(X);
9: call Backtrack(X);
10: exclude item i from X while maintaining aggregation statistics;
```

4.2 ILP Constraints

Given an itemset \(X\) generated by the backtracking step, the aim of ILP is to find the statistical information of associated FFT-patterns. We first present the algorithm to mine all \(S(X)\), and then show how to calculate the statistics without enumerating all tidsets.

The problem of enumerating \(S(X)\) can be translated to a system of binary linear inequalities where the variables represent the transactions. We denote this system of integer linear equalities as an integer linear program (ILP). This ILP consists of |\(T|\) variables, denoted by integer array \(A = (a_1, a_2, ..., a_{|T|})\). Each variable \(a_i, 1 \leq i \leq |T|\), represents one transaction and is constrained to be 0 or 1 (binary). A value 1 of variable \(a_i\) means transaction \(i\) is present in the tidset.

The relaxation criteria and utility constraint on the pattern (c.f. section 3.1) presented in terms of the variables are as follows.

Transaction Relaxation (ILP\(_T\))

\[
\forall i \in \{1..|T|\}, \ a_i \cdot (|X \setminus I(t_i)|) \leq \epsilon_t
\]
Item Relaxation ($ILP_2$)
\[ \forall x \in X, \sum_{j,t_j \in T(x)} a_{j} \leq \epsilon \] (2)

Pattern Relaxation ($ILP_p$)
\[ \sum_{i=1}^{\vert T \vert} a_i \cdot (\vert X \setminus I(t_i) \vert) \leq \epsilon_p \] (3)

Maximality Constraint ($ILP_{\text{max}}$)
Let $A'_k = (a'_1, a'_2, ..., a'_{\vert T \vert})$
\[ a'_i = \begin{cases} a_i & \text{if } i \neq k; \\ a_i + 1 & \text{if } i = k. \end{cases} \] 
\[ \forall k, A'_k \text{ satisfies relaxation criteria} \] (4)

Support Constraint ($ILP_{\text{sup}}$)
\[ \sum_{i=1}^{\vert T \vert} a_i \geq \xi \] (5)

Note that we haven’t changed the problem statement, instead we have only represented it in a mathematical form. Analyzing the variables, we note that this problem is intractable since the number of transactions can be potentially large. Especially, it is because integer programming in general is NP-Hard.

This problem can be resolved by reducing the number of variables. Equivalent variables are substituted by one combined-variable, representing the equivalence classes. In any definition of relaxation criteria, two transactions $t_a$ and $t_b$ are equivalent if they contain the same itemset in $X$, i.e. $I(t_a) \cap X = I(t_b) \cap X$. This equivalent condition is even stronger in FFTP-pattern and FFTp-pattern, where two transactions are equivalent if they contain the same number of itemsets in $X$, i.e. $|I(t_a) \cap X| = |I(t_b) \cap X|$. By this reduction, the number of variables is equal to the number of equivalence classes. Let us call $ILP$ with reduced variable as $ILP^{\text{red}}$, with variables $B = (b_0, b_1, ..., b_{\vert B \vert})$ where $\vert B \vert$ is the number of equivalence classes. A value $k$ of variable $b_i$ means there are $k$ transactions from equivalence class $i$ in the tidset, hence it is constrained to be at most the number of variables it substitutes. Variable reduction is illustrated in example 3.

Example 3. Consider mining FFTP-pattern from the database in Figure 1. Let the itemset $X = \{a, b\}$. As described previously, transactions containing same number of items in $X$ are considered equivalent. Hence, transaction 3 (resp: variable $a_3$) is equivalent to transaction 4 since they both contain 2 items, while transactions 1 and 2 are different since they contain 0 and 1 item respectively. Hence, there are 3 equivalence classes, $b_1$ representing tidset $\{1\}$, $b_2$ representing tidset $\{2\}$, and $b_3$ representing tidset $\{3, 4\}$. These variables are constrained to be at most 1, 1, and 2 respectively.

Using this technique, this problem becomes tractable since the number of variables is generally small, as is shown in Section 5. Having a solution $B$ for $ILP^{\text{red}}$, the number of support-tidsets satisfying $B$, denoted as $|S^B(X)|$ can be computed using equation (6), where $n_i$ is the number of transactions in equivalence class $i$.

\[ |S^B(X)| = \prod_{i=1}^{\vert B \vert} \binom{n_i}{b_i} \] (6)

Example 4. Continuing Example 3, we have $n_1 = 1$, $n_2 = 1$, and $n_3 = 2$. If we consider one transaction of class 1 ($b_1 = 1$), none of class 2 ($b_2 = 0$), and one of class 3 ($b_3 = 1$), $B = (1, 0, 1)$ representing $\binom{1}{1} \times \binom{0}{0} \times \binom{2}{1} = 2$ tidsets, viz. $\{1\}$ and $\{1, 3\}$.

Total size of all transactions in $S^B_X$ can be calculated as $|S^B(X)| \times (\sum_{i=1}^{\vert B \vert} b_i)$. Summing these sizes for all $B$ solutions of $ILP^{\text{red}}$ gives the total size of all support transactions, from which the average transaction size can be computed.

Since there can be multiple solutions $A$ and $B$ for $ILP$ and $ILP^{\text{red}}$, the set of all distinct solutions $A$ and $B$ is denoted as $A$ and $B$ respectively. To summarize, the objective of $ILP$ is to enumerate $B$, in order to find the statistical information of $A$.

In order to obtain the equivalence classes of transactions, we need to maintain an aggregation statistic which concisely represents the connectivity in the database.

4.3 Aggregation Statistic

The choice of the aggregation statistic needed depends on the variant of FFT-patterns being considered. Example 5 gives a broad view of what aggregation statistics are maintained and how to calculate them.

Example 5. Consider the problem of mining $S_T(X)$. In this problem, all transactions containing the same number of items are equivalent. Hence, we maintain the number of misses of every transaction. To speed up the variable reduction process we can also maintain the number of transactions containing the same number of items as another aggregation statistic. The speed up occurs because instead of enumerating all transactions, we only need to enumerate the equivalence classes.

Let us consider current itemset $X = \{a, b\}$ in the database in Figure 1. We maintain the statistic that transaction 1 contains zero item, transaction 2 contains one item,
and transactions 3 and 4 contain both items. To speed up the reduction process, we can also maintain the statistic that there are two transactions (regardless of which transactions they are) that contain both items, one transaction that contains one, and one transaction that contains no items in \( X \).

These aggregation statistics can be maintained efficiently. Since we are using a backtracking algorithm, at most one item is added or removed at one time. Thus, in the worst case this operation needs \( O(|T|) \) iterations. Details of the specific aggregation statistic used in each variant are explained in section 5.

5 Algorithms

In this section, we present algorithms based on the BIAS framework for the three FFT-pattern variants described in section 3.2.

5.1 FFT\(_T\)-pattern

Recalling the definition, a pattern \((Y \times X)\) is an FFT\(_T\)-pattern if it satisfies the criterion of transaction relaxation as stated in section 3.2 (ILP\(_T\) in section 4.2). Since there is no relaxation on the number of misses on each item, each transaction is independent of each other. Hence the tidset \( Y \) supporting \( X \) is unique, i.e. the set of all transactions containing at least \( |X| - \epsilon \) items in \( X \).

In this problem, the information about the number of appearances or number of misses of each transaction is sufficient to determine \( S_{\gamma}(X) \). These two variables can be used interchangeably since the number of appearances equals the size of the itemset less the number of misses. For the sake of efficiency, we use the number of appearances as the aggregation statistic.

5.1.1 Number of appearances as aggregation statistic

Given itemset \( X \), one way to calculate its frequency is by maintaining the number of appearances of each transaction, denoted as \( count^a \). If \( x \) is the last added item in the backtracking step, \( count^a \) can be maintained by increasing the count of all transactions containing \( x \). Hence, every update needs \( |T(x)| \) iterations. The complexity is \( O(|T|) \) in the worst case but \( O(\gamma . |T|) \) in the expected case where \( \gamma \) is the density of database. This approach has complexity proportional to the database’s density, which is desirable since the database is generally sparse.

To deduce the supporting tidsets, we need to maintain another aggregation statistic, which stores the number of transactions for every distinct value in \( count^a \). This is denoted as \( count^a_x \), i.e. \( count^a_x = |\{ t | count^a_t = x \} | \) (see Example 5). This statistic can be maintained concurrently with the former, therefore the additional cost needed is constant.

The support of an itemset can be calculated by summing \( count^c \) from index \( |X| - \epsilon \) to \( |X| \), which needs \( \epsilon + 1 \) iterations. This technique requires \( |T| \) integers to store \( count^a \), and \( X_{\max} \) integers to store \( count^c \).

5.1.2 Bitmap representation of aggregation statistic

Another way to represent the aggregation statistic is by using \( X_{\max} \) bitmaps, where bit \( t \) in bitmap \( c \) is set if and only if transaction \( t \) misses \( c \) items. The insertion process is shown in following pseudo-code. A similar algorithm can be derived for the removal process. As before, \( x \) corresponds to the last added item, and \( miss_x \) is the bitmap corresponding to transactions not containing item \( x \). This operation needs \( O(\alpha . X_{\max} \cdot |T|) \) iterations for each update and \( O(\alpha . X_{\max} \cdot |T|) \) space. The symbol \( \alpha \) represents the ratio between bit-wise and integer operations which is roughly 1/32 in a 32 bits processor, where 32 operations can be performed in one clock cycle.

\[
\begin{align*}
count_{X_{\max}} & := count_{X_{\max}} \land \neg miss_x \\
\text{for } i & := X_{\max} - 1 \text{ down to } 0 \\
\text{count}_{i+1} & := count_{i+1} \lor (count_i \land miss_x) \\
\text{count}_i & := count_i \land \neg miss_x
\end{align*}
\]

By sacrificing space, we can reduce \( X_{\max} \) to \( \epsilon \) by maintaining \( \epsilon + 1 \) bitmaps in all recurrence levels. In this case, the maximum space allocated is \( O(\alpha . X_{\max} . \epsilon \cdot |T|) \), but each update only needs \( O(\alpha . (\epsilon + 1) \cdot |T|) \) iterations. This approach even though asymptotically inefficient, is very fast in practice because of the architecture of current processors.

5.1.3 Comparisons with other techniques

A similar problem has been tackled by Pei et al in [9] and Koh and Yo in [6]. Both their algorithms, as ours, iterate the itemsets and then find their supporting transactions. Pei et al proposed the FT-Apriori algorithm [9] that enumerates itemsets using an \( apriori \)-like approach. FT-Apriori first finds all frequent itemsets of size \( \epsilon \) + 1. Then for each \( k \), it generates candidates of size \( k + 1 \) whose \( k \)-subsets are frequent. Once these candidates are accumulated, they are all checked using one pass of the database.

To generate one candidate \( X \), the algorithm checks whether all its subsets are frequent in \( O(|X|^2) \) time, and then verifies whether \( X \) is frequent in another pass of the database taking \( O(|T| \cdot |X|) \) time.

The number of candidates generated in this approach is relatively small compared to that in backtracking. However, generation of the candidates itself is an expensive process, consuming most of the mining time. Another disadvantage
of this approach is that it needs another database pass while processing itemsets of each size.

Koh and Yo presented VB-FT-Mine [6] which uses a backtracking technique similar to ours. However instead of using an aggregation statistic, they analyzed the recurrence relation of the support-tidsets. Let \( P(X, d) \) be the set of transactions which support itemset \( X \) when \( \epsilon_t = d \). They show that \( P(X \cup \{x\}, d) = P(X, d-1) \cup (P(X, d) \cap T(x)) \), hence this problem can be solved using dynamic programming [4]. The time and space complexity of VB-FT-Mine is exactly the same as ours using bitmap. This algorithm is fast because the recurrence relation happens to be simple, and using bitmaps is efficient.

Both approaches limit each item \( x \in X \) to be contained in at least \( \delta \) supporting transactions. Our technique can easily be adjusted to this requirement by iterating each item \( x \in X \), as is done in [9] and [6]. However, this constraint is neglected for the sake of consistency among all variants discussed in this paper.

A summary comparison among these three approaches is presented in Table 1. Our approaches representing the aggregation statistic as appearances and bitmaps are characterized by BIAS(appear) and BIAS(bit) respectively. \( N \) denotes the total number of frequent itemsets, and the constant \( \alpha \) represents the ratio of time taken for bit operations to integer operations, which in this case is roughly 1/32.

Table 1 shows that no approach is totally superior compared to others. Given a sparse database and large tolerance \( \epsilon_t \), ILP \(_{FFT}\)(appear) seems promising, while ILP \(_{FFT}\)(bit) and VB-FT-Mine [6] might be faster in a dense database and with smaller \( \epsilon_t \). These results must be viewed in light of the main focus of this paper which is to mine the statistical information of support-tidsets. This variant is incorporated to illustrate how the BIAS framework handles the criterion of transaction relaxation.

### 5.2 \( FFT_{TT} \)-pattern

\( FFT_{TT} \)-pattern is defined as a pattern that satisfies the criteria of item relaxation and transaction relaxation as defined in section 3 (ILP \(_T\) and ILP \(_T^{red}\) in section 4.2). This problem is seemingly more complicated compared to the FFT\(_T\)-pattern problem since each itemset might have multiple support-tidsets.

Since the approach used in FFT\(_T\)-pattern can be applied to handle the transaction relaxation, the pending problem is to generate FFT-patterns that satisfy the criterion of item relaxation.

#### 5.2.1 Finding the maximum sized \( S_{TT} \)

A pattern is frequent if the size of its largest support-tidset is greater than or equal to \( \xi \). Therefore finding the largest sized support-tidset can verify whether the pattern is frequent. This problem can be expressed as the ILP \(_I\) objective function to maximize \( \sum_{i=1}^{\xi} a_i \). Clearly, finding the maximum sized \( S_{TT} \) is at most as difficult as generating \( S_{TT} \). While it may seem easier, we show that finding the maximum result in ILP \(_I\) is an NP-Complete problem.

**Lemma 2.** Finding maximum \( \sum_{i=1}^{\xi} a_i \) in ILP \(_I\) is NP-Complete.

**Proof.** To prove NP-Hardness we take the problem of finding Maximum Independent Set (MIS) in a graph which is a well-known NP-complete problem, and prove that MIS can be reduced into this problem. Given a graph \( G \), the objective of MIS is to find a maximum set of vertices, such that there is no edge between any pair of them. The reduction is as follows. For every edge \( e(u, v) \), add inequality \( u + v \leq 1 \) in ILP \(_T\). This reduction is complete since the result of ILP \(_T\) will be the maximum set of variables (vertices) which are not connected. Since this case is a subset of ILP \(_I\), the ILP \(_I\) problem is at least as hard as MIS, implying that it is NP-Hard. Proving the completeness of its decision-problem \( \sum_{i=1}^{K} a_i \geq K \) is straight-forward.

Since this problem is NP-Complete, we cannot expect to have a practically efficient solution unless \( P = NP \). In our implementation, we check if itemsets are frequent by calculating the statistical information of the FFT-pattern.

#### 5.2.2 Finding statistical information of \( S_{TT} \)

As in section 4.2, we reduce the number of variables by finding the equivalence classes. In this variant, two transactions are equivalent if they contain exactly the same itemset in \( X \). It is easy to see that the maximum number of equivalence classes is bounded by \( \min(k, 2^{\lfloor |X| \rfloor}) \). Since \( |X| \) is less than 10 in many real-world datasets [13], the number of variables is at most 1024, hence enumerating all solutions is feasible.

To make this process more efficient, we introduce some heuristics, i.e., we iterate from the most constrained variable and from the highest possible value (most constraining), to reduce the search space as much as possible. Our experimental results show that all solutions can be found fast.

Once a solution \( B \) for ILP \(_I^{red}\) is obtained (section 4.2), we can calculate the desired statistic using equation (6).
5.3 FFT\textsubscript{p}-pattern

A pattern \((Y \times X)\) is an FFT\textsubscript{p}-pattern if it satisfies the criterion of pattern relaxation as given in section 3 (ILP\textsubscript{p} in section 4.2). In this problem, transactions are equivalent if they miss the same number of items in \(X\). Let variable \(b_i\) represent the equivalence class for transactions missing \(i\) items in \(X\). Note that only transactions that miss one to \(\epsilon_p\) items are important. Therefore, the number of equivalence classes, which is also the number of variables in \(ILP\textsubscript{p}^{red}\), is equal to \(\epsilon_p\). \(ILP\textsubscript{p}^{red}\) itself is formulated as follows:

\[
\sum_{i=1}^{\epsilon_p} b_i \times i \leq \epsilon_p
\]  

(7)

5.3.1 Finding the maximum sized \(S_p\)

Let \(Y_{max}\) be the largest tidset in \(S_p\). The problem of finding \(Y_{max}\) can be optimally solved using a greedy algorithm which continuously takes transactions with the lowest number of misses, until the total number of misses exceeds \(\epsilon_p\). This algorithm takes \(O(|T|)\) iterations. By maintaining the number of transactions that miss the same number of items, the complexity can be reduced to \(O(\epsilon_p)\).

5.3.2 Finding statistical information of \(S_p\)

As in FFT\textsubscript{t\textasteriskcentered} pattern, we can find the statistical information by enumerating \(B\) in \(ILP\textsubscript{p}^{red}\) (c.f. (7) and section 4.2). To show the bound of the number of solutions, we first introduce the term partition number [1].

**Definition 5. [Partition number]** Partition number of an integer \(n\) means the number of distinct multi-sets of positive integers that add-up to \(n\), denoted as \(part(n)\). For example, \(part(3) = 3\), since \(3 = 3\), \(2+1\), and \(1+1+1\). \(part(4) = 5\), since \(4 = 4\), \(3+1\), \(2+2\), \(2+1+1\), and \(1+1+1+1\), and so on. Each multi-set that adds-up to \(n\) is called a partition of \(n\).

Let the number of transactions missing \(i\) items be denoted as \(n_i\). We show that \(|B|\) is bounded by \(part(\epsilon_p)\).

**Lemma 3.** \(|B|\) is bounded by \(part(\epsilon_p)\).

**Proof.** It should be clear that if the number of transactions that miss \(i\) items, \(n_i = \infty\) for all \(i\), then \(|B|\) is equal to \(part(\epsilon_p)\). Now if the total number of misses in \(B\) equals \(\epsilon_p\), then it is a partition of \(\epsilon_p\). Otherwise, \(b_1 = n_1\), i.e. all transactions missing one item are already considered in \(B\). This is true, because if \(b_1 < n_1\), we can always increase \(b_1\) (i.e. we can add transactions missing one item to \(B\)), implying that \(B\) was not maximal. Assuming we have \(n_1 = \infty\), we can always increase \(b_1\) until the number of misses in \(B\) equals \(\epsilon_p\), so that it becomes a partition of \(\epsilon_p\). This partition is a unique extension for any maximal \(B\). Hence, all \(B\) are subsets of distinct partitions of \(\epsilon_p\), implying that \(|B|\) is at most \(part(\epsilon_p)\). \(\square\)

Lemma 3 shows that the number of maximal \(B\) is bounded by \(part(\epsilon_p)\), which is quite small for small \(\epsilon_p\). For example, \(part(10) = 42\) and \(part(20) = 627\). In a practical situation, we don’t need \(\epsilon_p\) to be larger than 20, and hence enumerating all solutions is feasible.

The algorithm to generate \(B\) is same as the one in section 5.2.2. We iterate the variable from the most constrained (having most number of misses), and most constraining (choosing the variable with highest value, so that we start with the largest number of transactions possible). The number of iterations is bounded by \(O(\epsilon_p \times part(\epsilon_p))\). It can be shown by assuming we have enough \(n_1\) that this algorithm will generate subsets of partition of \(\epsilon_p\) (similar to the proof of Lemma 3).

6 Experimental Results

All approaches are implemented in C++ and evaluated on an AMD 2.2 GHz machine with 2GB main memory. In all experiments the items are sorted in a non-decreasing order of appearances in the database. This performs best compared to other ordering techniques in our empirical studies.

For the experiments we used synthetic datasets generated using the IBM market-basket data generator. We used two different datasets, a small database (400 items, 10,000 transactions) to show the effect of parameters and comparison between approaches, and a practical-sized database (1,000 items, 100,000 transactions) to show the scalability of our approach in real world applications. The density of the databases is varied from 0.5% to 4% as in real-world market basket data.

6.1 FFT\textsubscript{t\textasteriskcentered}-pattern

We demonstrate this problem using the small database. The result of FT\textsubscript{apriori} algorithm isn’t plotted since it takes too long. Figure 2(a) shows the scalability of the algorithms while varying minimum support. We observe that in this case, BIAS(appear) performs best, followed closely by VB-FT-Mine then BIAS(bit). This is due to the operation’s complexity, while VB-FT-Mine only needs one “and” and one “or” operations, BIAS(bit) needs 2 “and”s, 2 “or”s, and 1 “not” operations (refer to section 5.1).

However, VB-FT-Mine only finds the number of transactions which miss less than or equal to \(\epsilon_i\) items, while both BIAS(appear) and BIAS(bit) maintain the number of transactions that miss all 0 to \(\epsilon_i\) items. In the case this statistic is useful - as in other variants, VB-FT-Mine cannot be used (i.e. it needs post processing).
Figure 2(b) shows that BIAS(appear) is more affected by density compared to BIAS(bit). From this experiment, we can conclude that if we have a sparse database, we should use BIAS(appear) to maintain the aggregation statistic, and otherwise use BIAS(bit). All of these behaviors are as expected as they follow the characteristics given in Table 1.

As we’ve already mentioned, this problem tries to reduce the number of FT-patterns by relaxing the constraint (it doesn’t consider item relaxation). In practical-sized database and $\epsilon_t > 1$, this problem is intractable. To conclude, this problem is not scalable in terms of database size and $\epsilon_t$ as has also been discussed in [9].

### 6.2 FFT$_{\text{T-I}}$-pattern

For this problem, we use the practical-sized dataset. Figure 2(c) shows the scalability of our algorithm. Even though CompleteQB [11] doesn’t tackle exactly the same problem, it is used as competitor because to the best of our knowledge, there are no other relevant references. In our work, the objective is to mine all frequent itemsets along with the statistical information of associated patterns, while that of [11] is to mine all maximal quasi-bicliques.

It can be seen that the number of frequent itemsets increases as the density increases and as the minimum support decreases. Using the BIAS framework, the time taken is almost linear to the number of frequent itemsets. CompleteQB [11] algorithm takes longer than one hour (the operation was terminated), because the number of FFT-patterns is much larger than the number of frequent itemsets as shown in Figure 2(d). The results also show that enumerating all results in market-basket database is meaningless since we might get as many as $10^{72}$(!) FFT-patterns, while there are only 1,600 frequent itemsets.

Experimenting with the value of $\epsilon_i$, it is interesting to observe that the value of $\epsilon_i$ doesn’t increase the number of
frequent itemsets much, even though the number of FFT-patterns grows exponentially (Figure 2(e)). This fact cannot be discovered if we enumerate all patterns as in CompleteQB.

### 6.3 FFT\textsubscript{p}-pattern

Since this variant hasn’t been mentioned previously, we can’t compare the efficiency of BIAS with any other approach. However, the experimental results in a practical-sized dataset indicate that this algorithm is efficient and scalable. Figure 2(f) shows that the time needed is approximately linear to the number of frequent itemsets. The number of frequent itemsets itself scales similar to classical frequent-itemsets, which grows exponentially as the density increases and/or as the minimum support decreases.

In this problem, the number of frequent itemsets is also much smaller compared to FFT\textsubscript{T}-patterns, even though the ratio is not as dramatic as in FFT\textsubscript{T}-pattern. For example in database with 4% density and 0.5% minimum support, with \( \epsilon_p = 20 \), the proportion is 46K frequent itemsets to 637K FFT-patterns, which is around 7%.

As seen in Figure 2(g), the ratio becomes smaller as \( \epsilon_p \) increases and minimum support decreases. Similar to the previous problem, the value of \( \epsilon_p \) doesn’t affect the number of FFT-itemsets much (Figure 2(h)).

### 7 Conclusion

We have generalized the problem of mining fixed-value FFT-patterns into relaxation criteria and constraints, and have shown that these criteria and constraints can be used to derive several available variants of FFT-patterns, as well as many new definitions. For FFT-patterns mining in transactional database, we proposed to mine only statistical information of the corresponding FFT-patterns, and we have shown with several experiments that such statistics are sufficient to describe the characteristic of an itemset.

We have developed a new framework called BIAS framework for mining FFT-patterns based on backtracking algorithm, ILP constraints, and aggregation statistics. We have also shown that this framework can be used to efficiently mine several variants of FFT-patterns.

For future work, we plan to analyze the utility of statistical information in real world data. We are also interested in analyzing the impact of each relaxation criteria, and in researching the problem using proportional relaxation. This technique may also be applied to other problems such as sequential mining, stream mining, and graph mining.

### References


