Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 79 (2006) 79–92

The Journal of Systems and Software

www.elsevier.com/locate/jss

Mining association rules with multi-dimensional constraints

Anthony J.T. Lee *, Wan-chuen Lin, Chun-sheng Wang

Department of Information Management, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei 106, Taiwan, ROC

Received 22 October 2003; received in revised form 14 March 2005; accepted 14 March 2005
Available online 18 April 2005

Abstract

To improve the effectiveness and efficiency of mining tasks, constraint-based mining enables users to concentrate on mining their interested association rules instead of the complete set of association rules. Previously proposed methods are mainly contributed to handling a single constraint and only consider the constraints which are characterized by a single attribute value. In this paper, we propose an approach to mine association rules with multiple constraints constructed by multi-dimensional attribute values. Our proposed approach basically consists of three phases. First, we collect the frequent items and prune infrequent items according to the Apriori property. Second, we exploit the properties of the given constraints to prune search space or save constraint checking in the conditional databases. Third, for each itemset possible to satisfy the constraint, we generate its conditional database and perform the three phases in the conditional database recursively. Our proposed algorithms can exploit the properties of constraints to prune search space or save constraint checking. Therefore, our proposed algorithm is more efficient than the revised FP-growth and FIC algorithms.

Keywords: Data mining; Association rule; Constraint-based mining

1. Introduction

Discovering association rules between items in a large transactional database plays a considerable role in the data mining research areas. An association rule is of the form $X \Rightarrow Y$, where X and Y are both frequent itemsets in the given database and the intersection of X and Y is an empty set, i.e., $X \cap Y = \phi$. The support of the rule $X \Rightarrow Y$ is the percentage of transactions in the given database that contain both X and Y. The confidence of the rule $X \Rightarrow Y$ is the percentage of transactions in the given database containing X that also contain Y. Association rule mining is to find all association rules among itemsets in a given database, where the supports and confidences of these association rules must satisfy the user-specified minimum support and minimum confidence. It has been shown that the overall performance of mining association rules is dominated by frequent itemset discovery (Agrawal et al., 1993).

Frequent itemset mining is one of the fundamental data mining problems in numerous studies. These studies cover a broad spectrum of data mining topics such as association rules (Agrawal et al., 1993; Agrawal and Srikant, 1994; Han et al., 2000), sequential patterns (Agrawal and Srikant, 1995), partial periodicity (Han et al., 1999), calling path patterns (Lee and Wang, 2003), etc. Typically, the goal of frequent itemset mining is to discover all the itemsets whose supports in the database exceed a user-specified threshold.

However, frequent itemset mining algorithms often generate a large number of frequent itemsets and rules, which reduce not only the efficiency but also the

* Corresponding author. Tel.: +886233661188; fax: +886233661199.
E-mail addresses: jtlee@ntu.edu.tw, smal@ms85.url.com.tw (A.J.T. Lee), d91001@im.ntu.edu.tw (C.-s. Wang).

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.03.005

effectiveness of the mining algorithms since only the subset of the complete frequent itemsets and association rules is of interest to users, and users need additional post-processing to filter through a large number of mined rules to find the useful ones. Constraint-based mining enables users to provide restraints to search for those useful ones. It concentrates on itemsets that are interesting from a user's point of view, rather than wasting time on discovering itemsets the user has not asked for.

Many constraint-based frequent itemset discovery techniques have been proposed recently for various constraint models. Item constraints in frequent itemset mining were first discussed in Srikant et al. (1997), which considered the problem of integrating constraints that are Boolean expressions over the presence or absence of items into the association rule discovery algorithms. Ng et al. (1998) introduced two interesting classes of itemset constraints: anti-monotonicity and succinctness, and proposed a mining algorithm for handling constraints belonging to these classes within the Apriori framework, called CAP. Pei and Han (2000) developed a new constraint-based frequent itemset discovery method, called CFG, which pushed a constraint into the FP-growth method. In Grahne et al. (2000), the third class of constraint, monotonicity, was introduced in the context of mining correlated sets. In Pei et al. (2001), more classes of constraints were introduced: convertible constraints, and methods which enable these classes of constraints to be pushed deep inside the FP-growth algorithm for frequent itemset mining, were developed. Bucila et al. (2002) proposed the DualMiner algorithm and used both monotone and anti-monotone constraints to prune the search space. Kifer et al. (2003) presented a method to mine itemsets with restrictions on their variance. Gade et al. (2004) introduced a new class of block constraints that determined the significance of an itemset pattern.

In most of previous constraint-based frequent itemset mining methods, an item in a transaction is identified by its item ID and characterized only by a single attribute value (such as price). In some circumstances, users may want to keep records of items with more than one attribute. For example, in addition to item prices, merchants may want to keep track of the brand of the item sold in order to see what items are most popular given the brand and price range; they may also want to keep records of the cost of items as well to see what items are often bought together, where the total profit (i.e., the difference between the price and the cost) of those items is greater than a certain desired level. In order to meet the above requirement, we enhance the item representation by using a number of attributes to describe the item's properties. We call them dimensional attributes (dimensions for short), because these attributes in fact form a multi-dimensional data space. Items with multiple dimensions are called multi-dimensional items.

With one-dimensional items, previously proposed methods are mainly contributed to handling a single constraint on one-dimensional attribute. However, with multi-dimensional items, constraints can be placed on multiple dimensional attributes. Furthermore, users are more likely to impose multiple constraints. To differentiate constraints developed in this paper from those in the previously proposed approaches, we call constraints against multiple dimensional attributes multi-dimensional constraints.

In comparison with the Apriori algorithm (Agrawal and Srikant, 1994), the FP-growth algorithm (Han et al., 2000) is more efficient to mine frequent itemsets. Furthermore, FP-growth-based methods also achieve performance gain in constraint-based mining by the employment of conditional databases. More properties of constraints can be pushed into FP-growth-based methods than Apriori-like methods. Therefore, in this paper, we choose the FP-growth method as the basic approach in our model and develop the algorithms to mine frequent itemsets with multi-dimensional constraints.

Our proposed algorithms basically consist of three phases. First, we collect frequent items and prune infrequent items. According to the Apriori property, if a set is not frequent, all of its supersets will be infrequent as well. Thus, if a single item b is not frequent, any itemsets containing b is impossible to be frequent. Second, we exploit the properties of the given constraints to prune search space or save constraint checking in the conditional databases. Third, for each itemset possible to satisfy the constraint, we generate its conditional database, construct its corresponding FP-tree and perform the three phases in the FP-tree recursively. Our proposed algorithms can exploit the properties of constraints to prune search space or save constraint checking. The experimental results show that our proposed algorithms are more efficient and scalable than the revised FP-growth (FP-growth+) and FIC (FIC+) algorithms, where the FP-growth+ algorithm first finds all frequent itemsets by the FP-growth method and then selects from them the itemsets satisfying the constraint, and the FIC+ algorithm is revised from the FIC algorithm (Pei et al., 2001) to mine the multi-dimensional constrained frequent patterns.

The remaining of this paper is organized as follows. In Section 2, we define multi-dimensional items and discuss the characteristics of multi-dimensional constraints. In Section 3, we describe the main idea of our proposed algorithms. Next, we proposed the algorithms to mine the frequent itemsets with multi-dimensional constraints in Section 4. In Section 5, we conduct a series of experiments to show the efficiency and scalability of our proposed algorithms. Conclusions and future work are presented in Section 6.

2. Multi-dimensional items and constraints

Let $I = \{i_1, i_2, \ldots, i_n\}$ is a set of data items, where an item is an object with m dimensional attributes such as weight, brand, price, cost, etc. We call items with multiple dimensions multi-dimensional items. Each item has a unique item identifier, itemID. In other words, items with the same itemID have the same dimensional values. When counting the number of dimensions, we do not regard itemID as a dimension.

A *transaction* $t = <tid, I_t>$ is a tuple, where *tid* is the identifier of the transaction, and $I_t \subseteq I$, recorded only by itemIDs. A *transaction database* T contains a set of transactions. An *itemset* is a sub-set of I. A *k-itemset* is an itemset of size k.

An itemset S is *contained* by a transaction $t = <tid, I_t>$ if and only if $S \subseteq I_t$. The *support* $sup(S)$ of an itemset S in the transaction database T is the percentage of transactions in T containing S. Given a *minimum support threshold* ξ $(0 \leqslant \xi \leqslant 1)$, an itemset S is *frequent* if and only if $sup(S) \geqslant \xi$.

A *constraint* Φ is a predicate on the powerset of the set of items I, i.e., $\Phi{:}2^I \to \{\text{true}, \text{false}\}$. An itemset S satisfies a constraint Φ if and only if $\Phi(S)$ is true. The complete set of itemsets satisfying a constraint Φ is $SAT_\Phi(I) = \{S | S \subseteq I \wedge \Phi(S) = \text{true}\}$.

In most previously proposed constraint-based mining methods (Srikant et al., 1997; Pei and Han, 2000; Pei et al., 2001; Boulicaut and Jeudy, 2000), constraints are placed over one-dimensional itemsets where items are characterized by a single attribute value. Ng et al. (1998) presented 1-variable and 2-variable constraints. In this paper, we extend constraints to place over multi-dimensional itemsets.

Definition 1. A multi-dimensional constraint is a constraint against multiple dimensional attributes.

Note that one-dimensional constraints can be viewed as a special case of multi-dimensional constraints, where the number of dimensional attributes involved by the constraints is equal to one.

Multi-dimensional constraints can be categorized into anti-monotone, monotone, convertible, and inconvertible like one-dimensional constraints as well. What is different is that one-dimensional constraints are against a single dimensional attribute, but multi-dimensional constraints are against multi-dimensional attributes. We briefly recall the definitions below (Ng et al., 1998; Grahne et al., 2000; Pei et al., 2001).

Definition 2. A constraint Φ is *anti-monotone* if and only if whenever an itemset S violates Φ, so does any superset of S.

Definition 3. A constraint Φ is *monotone* if and only if whenever an itemset S satisfies Φ, so does any superset of S.

Definition 4. A constraint Φ is *convertible anti-monotone* provided there is an order on items such that whenever an itemset S satisfies Φ, so does any prefix of S. A constraint Φ' is *convertible monotone* provided there is an order on items such that whenever an itemset S violates Φ', so does any prefix of S.

Note that any anti-monotone (or monotone) constraint is also convertible anti-monotone (or convertible monotone) (Pei et al., 2001).

Definition 5. A constraint Φ is *inconvertible* if it belongs to none of the three categories above.

Lemma 1. *A conjunction of anti-monotone constraints is also anti-monotone.*

Proof. Let the constraint Φ is a conjunction of anti-monotone sub-constraints $\Phi_1, \Phi_2, \ldots, \Phi_p$, i.e., $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_p$. Assume Φ is not anti-monotone, i.e., if an itemset S violates Φ, there exists some superset S' of S, where S' satisfies Φ. On one hand, given S violates Φ, where $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_p$, it means that S violates at least one sub-constraint Φ_i, $1 \leqslant i \leqslant p$. On the other hand, given S' satisfies Φ, it means that S' satisfies all sub-constraints $\Phi_1, \Phi_2, \ldots, \Phi_p$. Because all sub-constraints are anti-monotone constraints, by the definition of anti-monotone constraints (Definition 2), whenever S' satisfies $\Phi_i (1 \leqslant i \leqslant p)$, it implies that S satisfies Φ_i as well. So, S satisfies all sub-constraints $\Phi_1, \Phi_2, \ldots, \Phi_p$. That is, S violates no sub-constraint. Hence, S satisfies Φ. It is a contradiction. Therefore, Lemma 1 is proved. □

Lemma 2. *A disjunction of anti-monotone constraints is anti-monotone.*

Theorem 1. *A conjunction (or disjunction) of anti-monotone constraints is also anti-monotone.*

Corollary 1. *Any combination of conjunctions and disjunctions of anti-monotone constraints is also anti-monotone.*

The case of a conjunction (or disjunction) of anti-monotone constraints has been proved in Theorem 1. In the case of a mixture of some conjunctions and some disjunctions of anti-monotone constraints, taking an example of $\Phi_1 \wedge \Phi_2 \vee \Phi_3$, it can be treated as $(\Phi_1 \wedge \Phi_2) \vee \Phi_3$. Since $(\Phi_1 \wedge \Phi_2)$ is anti-monotone (Lemma 1), the case of $(\Phi_1 \wedge \Phi_2) \vee \Phi_3$ can be rewritten as $\Phi' \vee \Phi_3$, where $\Phi' = \Phi_1 \wedge \Phi_2$. $\Phi' \vee \Phi_3$ is anti-monotone (Lemma 2). That is, $\Phi_1 \wedge \Phi_2 \vee \Phi_3$ is anti-monotone. It is similar for the case of a combination of all monotone constraints.

In the case of a conjunction (or disjunction) of convertible constraints, it is convertible only if there exists an order on items to push all constraints deep inside the mining process. Because multi-dimensional items have multiple dimensional attributes, it is unusual to have a proper order to satisfy all the order requirements of the convertible constraints. Therefore, most probably a conjunction/disjunction of convertible constraints is inconvertible.

According to the results of (Pei et al., 2001), monotone constraints may be helpful in reducing the number of constraint checking, but it does not improve the efficiency too much, because usually constraint checking is cheap in comparison with the enumeration of frequent itemsets. Therefore, in this paper, we shall focus on mining the anti-monotone constrained frequent itemsets.

3. Main idea

In this section, we describe the main idea of the algorithms for mining frequent itemsets with multi-dimensional constraints. We first classify multi-dimensional constraints into two cases according to the number of sub-constraints included: <1> a single constraint against multiple dimensions, such as $max(S.cost) \leqslant min(S.price)$, where S is an itemset and each item in S contains two attributes *cost* and *price*, $max(S.cost)$ denotes the maximum cost of all items in S and $min(S.price)$ denotes the minimum price of all items in S; <2> a conjunction and/or disjunction of multiple sub-constraints, such as $(S.cost \leqslant v_1) \wedge (S.price \leqslant v_2)$, where v_1 and v_2 are constant values, and $S.cost \leqslant v_1$ denotes every cost of the items in S is less than or equal to v_1.

We will first describe the main idea of the algorithms for both cases of constraints briefly. Then, we will explain and illustrate the algorithm for each case, and then extend the similar concepts to more generic methods. Our algorithm is based on the FP-growth approach (Han et al., 2000) using conditional databases. Before explaining the algorithm, we must define some terms we will use in the following description.

Definition 6. Given a database T and a project condition pc,

1. $pc(s_1, s_2) =$ true if the relationship between itemsets s_1 and s_2 is correct. For example, let $s_1 = ab, s_2 = abcd$, and $pc =$ prefix relationship, $pc(s_1, s_2) =$ true because s_1 is the prefix of s_2 (By default, $pc =$ prefix relationship).
2. Itemset β is called the max-α projection of a transaction <tid, I_t>, w.r.t pc, if and only if <i>$\alpha \subseteq I_t$ and $\beta \subseteq I_t$; <ii>$pc(\alpha, \beta) =$ true; <iii>there exists no proper superset γ of β such that $\gamma \subseteq I_t$ and $pc(\alpha, \gamma) =$ true, too.
3. The α-*conditional database* is the collection of max-α projections of transactions containing α w.r.t. pc.

Note that because α appears in every transaction in the α-conditional database, α is omitted in further usage. Because we use an FP-tree as the structure of databases, when we say α-conditional database, what we actually mean is the FP-tree of α-conditional database, denoted as $T|_\alpha$ w.r.t. pc ($T|_\alpha$ for short if pc is the default).

Definition 7. Let α be a frequent itemset and λ be the set of frequent items in α's conditional database. $\alpha \cup \lambda$ forms the *potential largest frequent itemset* in the α's conditional database.

Let $\Phi_{\text{freq}}(S)$ (Φ_{freq} for short) be the constraint where the support of itemset S is not less than the minimum support threshold. In the following discussion, although we do not mention Φ_{freq} specially, in fact, we include Φ_{freq} into the constraints of the problem automatically. The main idea of our algorithm is described in detail in Fig. 1.

As shown in Fig. 1, the overall mining process can be partitioned into three phases: <1> frequency checking phase, <2> constraint checking phase, and <3> conditional FP-tree construction phase. The main concept of each phase is explained in the following sections.

3.1. Phase 1: Frequency checking

Because our problem is to mine frequent itemsets with multi-dimensional constraints, only itemsets which have satisfied the constraint of frequency (Φ_{freq}) are worth doing further steps of constraint checking. According to the Apriori property, if an itemset is not frequent, all of its supersets will be infrequent as well. Thus if a single item a is not frequent, any itemsets containing a is impossible to be frequent. So the first phase of our algorithm is to collect the frequent items and to prune infrequent items.

3.2. Phase 2: Constraint checking

In this phase, we exploit the properties of constraints to prune search space or save constraint checking in further conditional databases. Phase 2 can be divided into two sub-phases: phase 2.1 checks the potential largest frequent itemset, and phase 2.2 checks each individual

```
Input: Given a transaction database T, a minimum support threshold ξ, and a
       multi-dimensional anti-monotone constraint Φ over a set of items I.
Output: All frequent itemsets satisfying Φ.

T|α denotes α-conditional database, which contains itemsets co-occurring with α.
procedure ValidItemsets(T|α)
begin
   Phase 1: frequency checking
     Collect the set of frequent items L in T|α.
   Phase 2: constraint checking
     Phase 2.1: Check the potential largest frequent itemset against Φ.
        Let β be L.
        β∪α forms the potential largest frequent itemset in T|α.
        Check if Φ(β∪α) is true.
        If it is, skip phases 2.2 and 3.
     Phase 2.2: Check each individual item in L against Φ.
        for each item a∈L do begin
          Check if Φ({a}∪α) is true.
        end
   Phase 3: conditional FP-tree construction
     for each item a∈L do begin
       if Φ({a}∪α) is true then
         Generate the {a}∪α-conditional database and construct its FP-tree T|{a}∪α.
         Recursively call the procedure ValidItemsets(T|{a}∪α).
       endif
     end
end
```

Fig. 1. The main idea of our proposed algorithm.

frequent item. If the potential largest frequent itemset satisfies Φ, the steps in phase 2.2 and phase 3 can be skipped.

3.3. Phase 3: Conditional FP-tree construction

After constraint checking in phase 2, for each individual frequent item a, we will know if the itemsets co-occuring with $\{a\} \cup \alpha$ are possible to satisfy the constraint. If yes, we generate the $\{a\} \cup \alpha$-conditional database, construct its FP-tree $T|_{\{a\}\cup\alpha}$, and recursively perform the three phases in $T|_{\{a\}\cup\alpha}$.

4. Mining algorithms

Since multi-dimensional constraints can be classified into two cases according to the number of sub-constraints included, we will start from the first case "a single constraint against multiple dimensions" to introduce our proposed algorithm, and then extend the concepts used in the first case to the second case "a conjunction and/or disjunction of multiple sub-constraints".

4.1. Algorithm E-CFG for the first case

In this section, let us discuss the mining algorithms for the first case. We propose the E-CFG algorithm to mine the anti-monotone constraint-based frequent itemsets.

In phase 1 (frequency checking), because we use an FP-tree data structure to implement the conditional databases, the frequent items can be collected from the header table of the FP-tree.

In phase 2 (constraint checking), following the similar concept of strategy 1.3 in Pei and Han (2000), we can save constraint checking in further conditional databases based on the anti-monotone property by checking the potential largest frequent itemset. That is, if an itemset S satisfies an anti-monotone constraint Φ, so does any sub-set of S (Lemma 5). Checking each individual frequent item is used to prune search space based on the anti-monotone property. That is, if an itemset S violates Φ, so does any superset of S (Lemma 6).

Lemma 5. *Let β be the set of frequent items in $T|_\alpha$ and γ be a sub-set of the set of frequent items in $T|_{\{a\}\cup\alpha}$. If we have confirmed $\Phi(\alpha \cup \beta)$ = true in $T|_\alpha$ where Φ is an* *anti-monotone constraint, we do not need to check Φ in $T|_{\{a\}\cup\alpha}$ for each $a \in \beta$ because $(\{a\} \cup \alpha \cup \gamma) \subseteq (\alpha \cup \beta)$ and thus $\Phi(\{a\} \cup \alpha \cup \gamma)$ is certainly true.*

Lemma 6. *Let γ be a sub-set of the set of frequent items in $T|_{\{a\}\cup\alpha}$. If we have confirmed $\Phi(\{a\} \cup \alpha) = false$ where Φ is an anti-monotone constraint and a is an individual frequent item in $T|_\alpha$, we do not need to generate $T|_{\{a\}\cup\alpha}$ because $\{a\} \cup \alpha \cup \gamma$ contains $\{a\} \cup \alpha$ and thus $\Phi(\alpha \cup \{a\} \cup \gamma)$ is certainly false.*

According to Lemma 5, although we know the itemsets containing α certainly satisfy an anti-monotone constraint Φ given $\Phi(\alpha \cup \beta) = \text{true}$, we are still not sure which of those itemsets are frequent. So we need to check the supports of such itemsets. Naively, we can enumerate all such itemsets, where the number of such itemsets is $2^{|\beta|}$. But if $|\beta|$ is large, the number of enumerated itemsets can be very large. For this reason, we find all frequent itemsets in α-conditional database by using the FP-growth method (Han et al., 2000) instead of enumerating.

According to Lemma 6, for each individual frequent item b in $T|_\alpha$ if $\Phi(\{b\} \cup \alpha) = \text{false}$, b is removed from the set of frequent items L in $T|_\alpha$. After removing each individual frequent item b in L such that $\Phi(\{b\} \cup \alpha) = \text{false}$, let the set of remaining items be L'. For each item a in L', the corresponding FP-tree of $T|_{\{a\}\cup\alpha}$ is constructed in step 3 in the *e-cfg function*. The mining process for $T|_{\{a\}\cup\alpha}$ is performed recursively. The algorithm is described in detail in Fig. 2.

Now, let us demonstrate how the E-CFG algorithm works. Consider the database T in Table 1. Items in database T are 2-dimensional items, which are listed in Table 2. Assume that the minimum support threshold is 20%. In Fig. 3, the frequent intemsets found by the FP-growth method are surrounded by a dashed rectangle.

Let $\Phi \equiv max(S.cost) \leqslant min(S.price)$ which is an anti-monotone constraint. Fig. 3 illustrates how the E-CFG

Table 1
An example database T

Database T	
TID	Items
1	A, B, C, E
2	A, B, E
3	A, D
4	$A\ B, D$
5	B, D, E
6	A, B, C, D, E
7	B, C, E
8	B, C, D, E
9	C, D, E
10	B, C, D

Algorithm: E-CFG

Input: A transaction database T, a minimum support threshold ξ, and an anti-monotone multi-dimensional constraint Φ over a set of items I.

Output: All frequent itemsets satisfying Φ.

The transaction database T must be converted to the FP-tree data structure first. The items in the FP-tree are sorted in descending order of supports.

Method: Call *e-cfg* (ϕ, T)

Function: *e-cfg* $(\alpha, T|_\alpha)$

1. Collect the set of frequent items L (and their supports) from the FP-tree header table of $T|_\alpha$.
2. Let $\beta = L$.
   If $\Phi(\beta\cup\alpha)$=true **then**
   // for each $\beta'\cup\alpha$, where β' is any non-empty subset of β, we only need to check if its support is not less than ξ.
   Call **FP-growth (α, $T|_\alpha$)** to find all frequent itemsets in $T|_\alpha$.
   Only output the frequent itemsets. Return. (Lemma 5)
   endif
3. Check each individual item, b, in L against Φ and remove b from L if $\Phi(\{b\}\cup\alpha)$=false. Let the set of remaining items be L'. For each item a in L', output $\{a\}\cup\alpha$, generate the $\{a\}\cup\alpha$-conditional database and construct its FP-tree $T|_{\{a\}\cup\alpha}$, where transactions in $T|_{\{a\}\cup\alpha}$ contain only items in L'. (Lemma 6)
4. For each item a in L', call *e-cfg* $(\{a\}\cup\alpha, T|_{\{a\}\cup\alpha})$.

Fig. 2. E-CFG algorithm.

Table 2
Items in database T

ItemID	A_1: *cost*	A_2: *price*
A	40	50
B	30	35
C	55	60
D	25	40
E	20	45

algorithm finds all frequent itemsets satisfying the constraint Φ in T.

First of all, we collect the set of frequent items L in T where L contains frequent items B, D, E, C, A in descending order of supports. Then, check if the potential largest frequent itemset $\{BDECA\}$ satisfies Φ. $\Phi(\{BDECA\}) = \text{false}$, so continue to check each individual item in L against Φ. All of them satisfy Φ. Therefore, we generate four conditional databases. Note that it is not necessary to generate the conditional database of the item with the largest support, i.e, $T|_B$ in this case, it will be empty. Continue to perform mining tasks in these conditional databases.

Take $T|_A$ for example. We first collect the set of frequent items L in $T|_A$ where L contains B, D, E, C in descending order of supports. Then, check if the potential largest frequent itemset $\{BDECA\}$ satisfies Φ. $\Phi(\{BDECA\}) = \text{false}$, so continue to check each individual item in L against Φ. For each item a in L, check if $\Phi(\{a\} \cup \{A\}) = \text{true}$. Among them, only $\{DA\}$ and $\{EA\}$ satisfy Φ. Since Φ is an anti-monotone constraint, we only need to generate DA-conditional and EA-conditional databases. Note that because $\Phi(\{BA\}) = \text{false}$, $T|_{DA}$ and $T|_{EA}$ should not contain item B. So $T|_{DA}$ is empty. Continue to perform further mining task in $T|_{EA}$.

Take $T|_E$ for illustration of another situation. We first collect the set of frequent items L in $T|_E$ where L contains B, D in descending order of supports. Then, check whether the potential largest frequent itemset $\{BDE\}$ satisfies Φ. Φ $(\{BDE\}) = \text{true}$, so call the FP-growth method to find all frequent itemsets in the E-conditional database $T|_E$, i.e., $\{BE\}$, $\{DE\}$, and $\{BDE\}$. Output them. The mining task in $T|_E$ is finished.

The mining processes for other conditional databases are performed in the similar manner. As shown in Fig. 3,

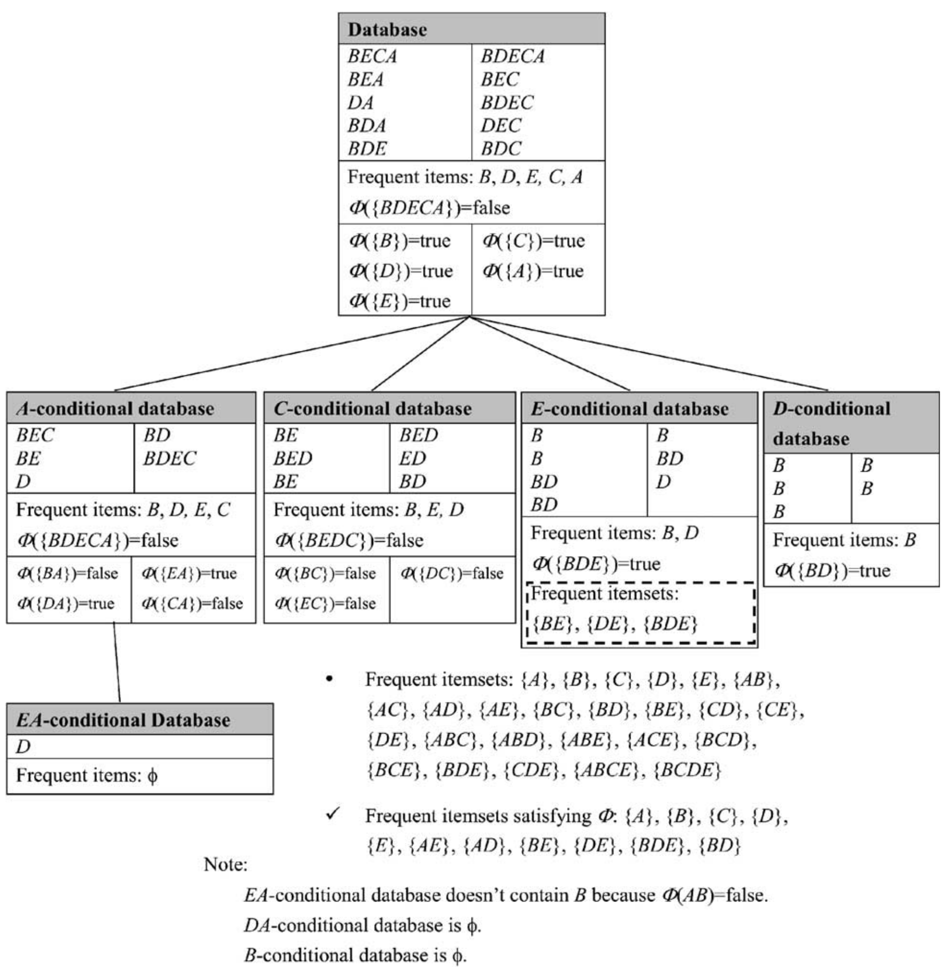

Fig. 3. Mining frequent itemsets with constraint Φ by the E-CFG algorithm.

there are 25 frequent itemsets in total. Among them, there are 11 frequent itemsets satisfying the constraint Φ.

4.2. Algorithm GE-CFG for the second case

After discussing the case of a single constraint against multiple dimensions, in this section we shall discuss the algorithms for the second case "a conjunction and/or disjunction of multiple sub-constraints". Let Φ be $\Phi_1\rho\Phi_2\rho\ldots\rho\Phi_p$, where $\rho\in\{\wedge,\vee\}$. Although Φ is the combination of anti-monotone sub-constraints, Φ itself can be treated as a single anti-monotone constraint. Thus, it can be handled by applying the algorithm in case 1, i.e., the E-CFG algorithm, to mining the frequent itemsets with anti-monotone multi-dimensional constraints.

Let us consider an example, $\Phi=\Phi_1\wedge\Phi_2$, where Φ_1 and Φ_2 are both anti-monotone constraints. In the E-CFG algorithm, by Lemma 5, we check if $\Phi(\alpha\cup\beta)=$ true in $T|_\alpha$ where β be the set of frequent items in $T|_\alpha$. If $\Phi(\alpha\cup\beta)=$ true, then we do not need to check Φ in $T|_{\{a\}\cup\alpha}$ for each $a\in\beta$. But if $\Phi(\alpha\cup\beta)=$ false, then we still need to check Φ in $T|_{\{a\}\cup\alpha}$ for each $a\in\beta$. However, there probably exist unnecessary checks of Φ_1 or Φ_2. If $\Phi_1(\alpha\cup b)=$ false for some $b\in\beta$, it implies $\Phi(\alpha\cup b)=$ false. Hence, we do not need to check Φ_2 in $T|_{\{b\}\cup\alpha}$. Similarly, there exist unnecessary checks for $\Phi=\Phi_1\vee\Phi_2$. Therefore, we modify the E-CFG algorithm to the GE-CFG algorithm, which treats Φ as the combination of anti-monotone sub-constraints rather than a single anti-monotone constraint. The GE-CFG algorithm is the generic version of the E-CFG algorithm. The GE-CFG algorithm employs p *check_flags*, where p is the number of sub-constraints included in Φ. There are three possible values of a *check_flag*: unknown (0), true (1) and false (-1). They are all unknown initially.

Since the processes of phase 1 and phase 3 in the GE-CFG algorithm are similar to those in the E-CFG algorithm, we will concentrate on phase 2 (constraint checking). By Lemma 5, when the *check_flag*$_q$, $1\leqslant q\leqslant p$, is unknown, we check if $\Phi_q(\alpha\cup\beta)=$ true in $T|_\alpha$, where β is the set of frequent items in $T|_\alpha$. If $\Phi_q(\alpha\cup\beta)=$ true, then we do not need to check Φ_q in $T|_{\{a\}\cup\alpha}$ for each $a\in\beta$, and we can set the *check_flag*$_q$ to be true. If $\Phi_q(\alpha\cup\beta)=$ false, the *check_flag*$_q$ remains unknown. According to the flags from *check_flag*$_1$ to *check_flag*$_p$, we can determine if $\Phi(\alpha\cup\beta)$ is true. Just as in the E-CFG algorithm, although we know the itemsets containing α certainly satisfy the constraint Φ given $\Phi(\alpha\cup\beta)=$ true, we are still not sure which of those itemsets are frequent. So we need to check the supports of such itemsets. Naively, we can enumerate all such itemsets, where the number of such itemsets is $2^{|\beta|}$. But if $|\beta|$ is large, the number of itemsets we need to check the supports can be huge. Hence, we find all frequent itemsets in α-conditional database by using the FP-growth method.

By Lemma 6, when the *check_flag*$_q$ is unknown, we check if $\Phi_q(\{a\}\cup\alpha)$ is false in $T|_\alpha$, for each sub-constraint Φ_q, $1\leqslant q\leqslant p$, where a is a frequent item in $T|_\alpha$. If $\Phi_q(\{a\}\cup\alpha)=$ false, we set the *check_flag*$_q$ to be false inside the a's *CheckItem*$_a$ *function*, *CheckItem*$_a$ $(\alpha, a, check\_flag_1, check\_flag_2\ldots check\_flag_p)$. If $\Phi_q(\{a\}\cup\alpha)=$ true, the *check_flag*$_q$ inside the a's *CheckItem*$_a$ *function* remains unknown. According to the flags from *check_flag*$_1$ to *check_flag*$_p$ returned by the a's *CheckItem*$_a$ *function*, we can determine if $\Phi(\{a\}\cup\alpha)$ is true in step 4 in the *ge-cfg function*, where if *check_flag*$_q$, $1\leqslant q\leqslant p$, is still unknown, it is treated as $\Phi_q(\{a\}\cup\alpha)=$ true. Just as in the E-CFG algorithm, only when $\Phi(\{a\}\cup\alpha)=$ true, we need to generate $T|_{\{a\}\cup\alpha}$. As in step 5 in the *ge-cfg function*, each individual item b in L such that $\Phi(\{b\}\cup\alpha)=$ false is removed from each transaction in $T|_{\{a\}\cup\alpha}$.

The GE-CFG algorithm is described in detail in Fig. 4. The *CheckItem*$_a$ *function* is shown in Fig. 5.

Now, let us demonstrate how the GE-CFG algorithm works. Consider the database shown in Tables 1 and 2. Assume that the minimum support threshold is 20%. In Fig. 6, the frequent intemsets found by the FP-growth method are surrounded by a dashed rectangle, where the FP-growth method is used instead of the *ge-cfg* method to find the frequent itemsets.

Consider an anti-monotone constraint $\Phi=\Phi_1\vee\Phi_2$, where $\Phi_1\equiv\max(S.cost)\leqslant\min(S.price)$ and $\Phi_2\equiv S.cost\leqslant 50$. Fig. 6 illustrates how the GE-CFG algorithm finds all frequent itemsets satisfying the constraint Φ in T.

In Fig. 6, the numbers inside the parentheses behind the database are the *check_flag*$_1$ and *check_flag*$_2$, respectively. We first collect the set of frequent items L in T, which contains B, D, E, C, A in descending order of supports. Because the *check_flag*$_1$ and *check_flag*$_2$ are both unknown (0), we continue to check the potential largest frequent itemset $\{BDECA\}$ against both Φ_1 and Φ_2. Both Φ_1 $(\{BDECA\})$ and $\Phi_2(\{BDECA\})$ are false, so both *check_flag*$_1$ and *check_flag*$_2$ remain unknown (0). We consider $\Phi_i(\{BDECA\})=$ false, $i=1, 2$, if *check_flag*$_i$ is unknown (0), so $\Phi(\{BDECA\})=$ false. Then, continue to check each individual item in L against both Φ_1 and Φ_2 because the *check_flag*$_1$ and *check_flag*$_2$ are both unknown (0). Among them, only Φ_2 $(C)=$ false so only the *check_flag*$_2$ in the C-conditional database is set to be false (-1), and the other *check_flags* remain unknown (0). Here, we consider $\Phi_i(\{a\}\cup\alpha)=$ true for $a\in L$, $i=1$, 2, if *check_flag*$_i$ is unknown (0), so all $\Phi(\{a\}\cup\alpha)=\Phi_1$ $(\{a\}\cup\alpha)\vee\Phi_2(\{a\}\cup\alpha)=$ true. Therefore, we generate five conditional databases.

Note that it is not necessary to generate the conditional database of the item with the largest support, i.e, $T|_B$ in this case, it will be empty. Continue to per-

Algorithm: GE-CFG

Input: A transaction database T, a minimum support threshold ξ, and an anti-monotone multi-dimensional constraint Φ over a set of items I, where Φ is a combination of anti-monotone sub-constraints, i.e., $\Phi = \Phi_1 \rho \Phi_2 \rho \ldots \rho \Phi_p$, where $\rho \in \{\wedge, \vee\}$ and each sub-constraint Φ_q, $1 \le q \le p$, is anti-monotone.

Output: All frequent itemsets satisfying Φ.

The transaction database T must be converted to the FP-tree structure first. The items in the FP-tree are sorted in descending order of supports. Initially, all *check_flag*s are unknown.

Method: Call *ge-cfg*(ϕ, T, 0, 0, …, 0)

Function: *ge-cfg*(α, $T|_\alpha$, *check_flag*$_1$, *check_flag*$_2$, …, *check_flag*$_p$)

1. Collect the set of frequent items L (and their supports) from the FP-tree header table of $T|_\alpha$.
2. Let $\beta = L$. For each sub-constraint Φ_q, $1 \le q \le p$, if *check_flag*$_q$ = unknown, check if $\Phi_q(\beta \cup \alpha)$ is true. If it is, let *check_flag*$_q$ be true. (Lemma 5)
3. Determine if $\Phi(\beta \cup \alpha)$ is true according to the *check_flag* of each sub-constraint. If *check_flag*$_q$= unknown, $1 \le q \le p$, we consider $\Phi_q(\beta \cup \alpha)$=false.

   If $\Phi(\beta \cup \alpha)$=true **then**

   // for each $\beta' \cup \alpha$, where β' is any non-empty subset of β, we only need to check the support.

   Call **FP-growth (α, $T|_\alpha$)** to find all frequent itemsets in $T|_\alpha$.

   Output the frequent itemsets. Return. (Lemma 5)

   endif
4. For each individual item a in L, perform the following steps: (1) call *CheckItem*$_a$(α, a, *check_flag*$_1$, *check_flag*$_2$, …,*check_flag*$_p$); (2) set *check_flag*$^a{}_q$ = *check_flag*$_q$, $1 \le q \le p$; (3) determine if $\Phi(\{a\} \cup \alpha)$ is true according to the values of *check_flag*$_1$, *check_flag*$_2$, …, *check_flag*$_p$. If *check_flag*$_q$ = unknown, we consider $\Phi_q(\{a\} \cup \alpha)$=true.
5. Remove individual item b from L such that $\Phi(\{b\} \cup \alpha)$=false. Let L' be the set of remaining items. For each item a in L', perform the following steps: (1) output $\{a\} \cup \alpha$; (2) generate the $\{a\} \cup \alpha$-conditional database; (3) construct its FP-tree $T|_{\{a\} \cup \alpha}$; (4) call *ge-cfg*($\{a\} \cup \alpha$, $T|_{\{a\} \cup \alpha}$, *check_flag*$^a{}_1$, *check_flag*$^a{}_2$,…, *check_flag*$^a{}_p$), where the transactions in $T|_{\{a\} \cup \alpha}$ contain only items in L'. (Lemma 6)

Fig. 4. GE-CFG algorithm.

Function: CheckItem$_a$(α, a, *check_flag*$_1$, *check_flag*$_2$,…, *check_flag*$_p$)

1. For each sub-constraint Φ_q, $1 \le q \le p$, if *check_flag*$_q$ = unknown, check if $\Phi_q(\{a\} \cup \alpha)$ is true. If it is not, let *check_flag*$_q$ be false. (Lemma 6)
2. Return *check_flag*$_q$, $1 \le q \le p$.

Fig. 5. Function *CheckItem*$_a$.

form mining tasks in these conditional databases with the *check_flag*$_1$ and *check_flag*$_2$ for each conditional database. In this example, all pairs of (*check_flag*$_1$, *check_flag*$_2$) are $(0,0)$, except $(0,-1)$ for the C-conditional database, T_C.

Take $T|_E$ for example. Collect the set of frequent items L in $T|_E$, which contains B and D. Because the *check_flag*$_1$ and *check_flag*$_2$ are both unknown (0), we continue to check the potential largest frequent itemset $\{BDE\}$ against both Φ_1 and Φ_2. Both $\Phi_1(\{BDE\})$ and $\Phi_2(\{BDE\})$ are true, so both *check_flag*$_1$ and *check_flag*$_2$ are set to be true (1). Since $\Phi = \Phi_1 \vee \Phi_2$, $\Phi(\{BDE\}) =$ true. Because $\Phi(\{BDE\}) =$ true, call the FP-growth method to find all frequent itemsets in the E-conditional database $T|_E$, i.e., $\{BE\}$, $\{DE\}$, and $\{BDE\}$. Output them. The mining task in $T|_E$ is finished.

The mining processes for other conditional databases are performed in the similar manner. As shown in Fig. 6, there are 25 frequent itemsets in total. Among them, there are 11 frequent itemsets satisfying the constraint

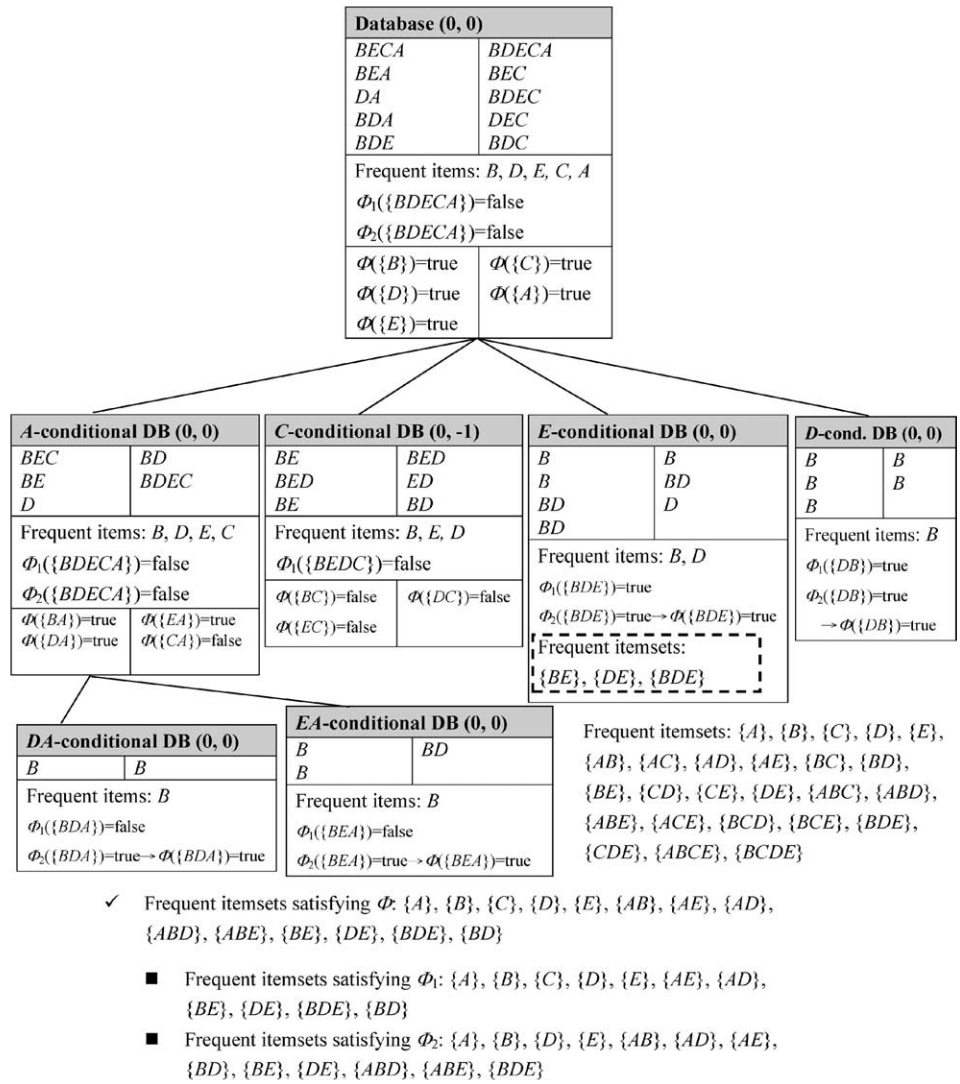

Fig. 6. Mining frequent itemsets with constraint Φ by the GE-CFG algorithm.

Φ_1 and 13 frequent itemsets satisfying the constraint Φ_2. However, there are 10 frequent itemsets in common. So, there are 14 frequent itemsets satisfying the constraint Φ.

4.3. Mining non-monotonic nor anti-monotonic constraints

To tackle a non-monotonic nor anti-monotonic constraint, we first rewrite the constraint into a conjunction of several sub-constraints. Next, we classify them into two groups since we focus on mining the anti-monotone constrained frequent itemsets in this paper. The first group contains all anti-monotonic sub-constraints. The second group contains the other sub-constraints. For the first group, we can push the sub-constraints into the mining algorithm to find the constrained frequent patterns. Then, we can check if the frequent itemsets founded by the first group of sub-constraints satisfy the sub-constraints in the second group in the mining algorithm. Thus, we can still use the first group of sub-constraints to reduce the search space. The NA-CFG algorithm is described in detail in Fig. 7, where a non-monotonic nor anti-monotonic constraint Φ is rewritten into a conjunction of several sub-constraints. Let $\Phi = \Phi_a \wedge \Phi_{na}$, where Φ_a contains all anti-monotonic sub-constraints and Φ_{na} contains the other sub-constraints in Φ.

5. Experiments and performance evaluation

All the experiments were performed on a Pentium IV 2.4G personal computer with 512MB main memory, running Windows XP. The program is written in C++ and compiled with Microsoft Visual C++ 6.0.

Algorithm: **NA-CFG**

Input: A transaction database T, a minimum support threshold ξ, and a non-monotonic nor anti-monotonic constraint Φ $(=\Phi_a \wedge \Phi_{na})$ over a set of items I.

Output: All frequent itemsets satisfying Φ.

The transaction database T must be converted to the FP-tree data structure first. The items in the FP-tree are sorted in descending order of supports.

Method:

If Φ_a contains no constraints **then**

Call **FP-growth** $(\alpha, T|_\alpha)$ to find all frequent itemsets in $T|_\alpha$.

For each frequent itemset found γ, if $\Phi_{na}(\gamma)$ =true, output the frequent itemsets. Return.

else

Call *na-cfg* (ϕ, T).

endif

Function: *na-cfg* $(\alpha, T|_\alpha)$

1. Collect the set of frequent items L (and their supports) from the FP-tree header table of $T|_\alpha$.
2. Let $\beta = L$.

   If $\Phi_a(\beta \cup \alpha)$=true **then**

   // for each $\beta' \cup \alpha$, where β' is any non-empty subset of β, we need to check if it satisfies Φ_{na} and its support is not less than ξ.

   Call **FP-growth** $(\alpha, T|_\alpha)$ to find all frequent itemsets in $T|_\alpha$.

   For each frequent itemset found γ, if $\Phi_{na}(\gamma)$ =true, output the frequent itemsets. Return.

   endif
3. Check each individual item, b, in L against Φ_a and remove b from L if $\Phi_a(\{b\} \cup \alpha)$=false. Let the set of remaining items be L'. For each item a in L', (1) if $\Phi_{na}(\{a\} \cup \alpha)$=true, output $\{a\} \cup \alpha$. (2) generate the $\{a\} \cup \alpha$-conditional database and construct its FP-tree $T|_{\{a\} \cup \alpha}$, where transactions in $T|_{\{a\} \cup \alpha}$ contain only items in L'.
4. For each item a in L', call *na-cfg* $(\{a\} \cup \alpha, T|_{\{a\} \cup \alpha})$.

Fig. 7. NA_CFG algorithm.

5.1. Synthetic data generation and experiment parameters

The method used to generate datasets is similar to the one used in (Agrawal and Srikant, 1994), with some modifications. Besides the standard procedure, we need an additional step to generate the dimensional values for each item. For the sake of simplicity, we assume each dimension of an item is independent. We determine the value for every dimension separately and independently.

To evaluate the effect of a constraint on mining frequent itemsets, the *selectivity* δ of a constraint Φ is defined as the percentage of frequent itemsets in T not satisfying Φ (Pei et al., 2001), i.e.,

$$\delta = \frac{\text{Number of frequent itemsets NOT satisfying } \Phi}{\text{Number of frequent itemsets}}$$

Therefore, a constraint with 0% selectivity means every frequent itemset satisfies the constraint, while with 100% selectivity means every frequent itemset violates the constraint. Table 3 summarizes the parameters used and their default settings. We use ***V25F20T50KN1KL100*** to represent the default setting where ***V25*** denotes that the average size of the transactions is 25, ***F25*** denotes that the average size of the maximal potentially frequent itemsets is 20, ***T50K*** denotes that the number of transactions is 50K, ***N1K*** denotes that the number of items is 1K, and ***L100*** denotes that the number of maximal potentially frequent itemsets is 100.

Table 3
Parameters and settings

Parameter	Meaning	Setting
N	Number of items	1000
M	Number of item dimensional attributes	5
$\|T\|$	Number of transactions	20K–100K
V	Average size of the transactions	25
F	Average size of the maximal potentially frequent itemsets	20
L	Number of maximal potentially frequent itemsets	100
ξ	Minimum support threshold	0.25–2%
δ	Selectivity of constraints	0–100%

5.2. Experiments of E-CFG and GE-CFG

To evaluate the efficiency of our algorithms, we compare them with the FP-growth+ and FIC+ algorithms since Pei et al. (2001) claimed that the FIC algorithm is more efficient than the CFG algorithm. The FP-growth + algorithm first finds all frequent itemsets by the FP-growth method and then selects from them the itemsets satisfying the constraint. The FIC+ algorithm is revised from the FIC algorithm (Pei et al., 2001) to mine the multi-dimensional constrained frequent patterns. The experiments in this section are performed over the dataset against an anti-monotone constraint $\Phi = \Phi_1 \wedge \Phi_2$, where $\Phi_1(S) \equiv S.A_2 \leqslant v_1$ denotes that the second attributes of the items in itemset S are all not greater than v_1, and $\Phi_2(S) \equiv Max(S.A_1) - Min(S.A_2) \leqslant v_2$ denotes that the maximum value of the first attributes of the items in itemset S minus the minimum value of the second attributes of the items in S is not greater than v_2.

5.2.1. Effect of the constraint selectivity

The first experiment is conducted by using the default settings and the minimum support threshold $\xi = 0.5\%$. Fig. 8 shows how the E-CFG and GE-CFG algorithms behave as the constraint selectivity increases from 0% to 100%.

In Fig. 8, the execution time of the E-CFG/GE-CFG algorithms decreases as the constraint selectivity increases. The execution time of the FP-growth+ algorithm is constant w.r.t. the constraint selectivity. The E-CFG/GE-CFG algorithms are 15% ~ 1600 times faster than the FP-growth+ algorithm and 10 ~ 20% faster than the FIC+ algorithm. As the constraint selectivity increases, i.e., more frequent itemsets violate the constraint, the E-CFG/GE-CFG algorithms can prune more search space. The main difference between the E-CFG and GE-CFG algorithms is that the GE-CFG algorithm can save some unnecessary constraint checks.

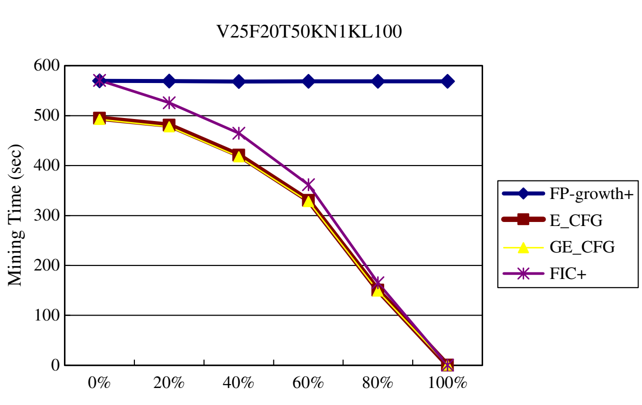

Fig. 8. Scalability with constraint selectivity (runtime).

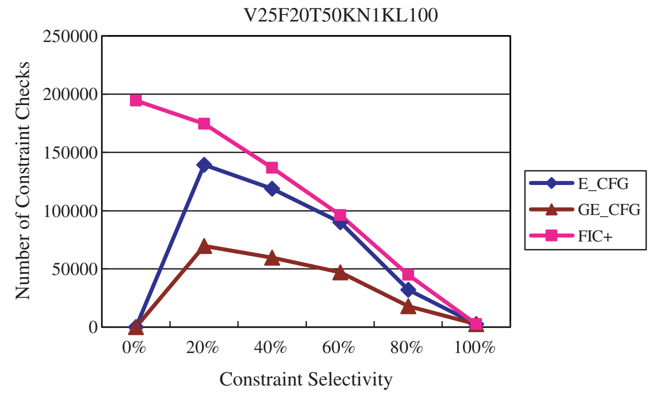

Fig. 9. Scalability with constraint selectivity (number of constraint checks).

However, the cost of checking constraints is CPU-bounded, and the cost of mining the whole frequent itemsets is I/O-bounded. The effect of saving unnecessary constraint checks is hard to be observed from the runtime reduction. So we show how the GE-CFG algorithm can save constraint checks in Fig. 9.

In Fig. 9, the curves of the E-CFG and GE-CFG algorithms exhibit a spike shape. The higher the constraint selectivity, the smaller number of constraint checks is in most of cases resulting from pruning search space (Lemma 6). However, when the constraint selectivity is quite low, it is more likely that the potential largest frequent itemset will satisfy the constraint Φ so that we can save constraint checks (Lemma 5). When the selectivity increases from 20% to 80%, the number of constraint checks of the FIC+ algorithm is 2–2.5 times of that of the GE-CFG algorithm, and 1.1–1.4 times of that of the FIC+ algorithm. The FIC+ algorithm checks more constraints than the E-CFG and GE-CFG algorithms because they exploit both sub-set and super-set properties of the constraint to prune the search space and save constraint checking.

5.2.2. Effect of the number of transactions

The experiments in this section are performed over the dataset by using the default settings, the minimum support threshold $\xi = 0.5\%$ and the constraint selectivity $\delta = 20\%$ and $\delta = 80\%$, respectively. Fig. 10 shows how the E-CFG and GE-CFG algorithms behave as the number of transactions increases.

In Fig. 10, the execution time of all algorithms increases as the number of transactions increases because they spend more time to construct FP-trees with more transactions in a database. As shown, the mining time for each case increases linearly. We can also observe that the runtime of the E-CFG /GE-CFG algorithms with the constraint selectivity $\delta = 20\%$ is more than that with the constraint selectivity $\delta = 80\%$.

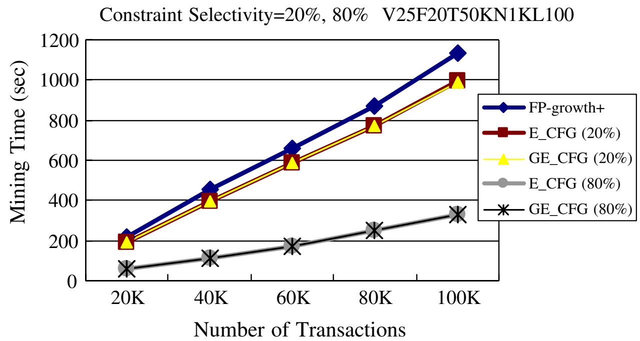

Fig. 10. Scalability with number of transactions.

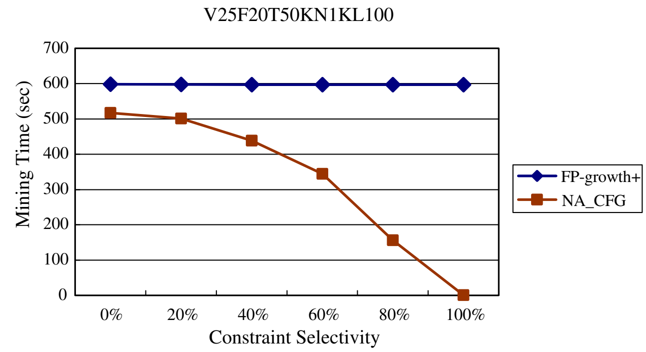

Fig. 12. NA_CFG vs. FP_growth.

5.2.3. Effect of the minimum support threshold

The experiments in this section are performed over the dataset with the default settings and the constraint selectivity $\delta = 20\%$ and $\delta = 80\%$ respectively. Fig. 11 shows how the E-CFG and GE-CFG algorithms behave as the minimum support threshold increases from 0.25% to 2%.

In Fig. 11, as the minimum support threshold increases, the execution time of all algorithms decreases because of the reduction in the number of frequent itemsets. Here we can see again that the runtime of the E-CFG/GE-CFG algorithms with the constraint selectivity $\delta = 80\%$ is less than that with constraint selectivity $\delta = 20\%$.

5.2.4. Non-monotonic nor anti-monotonic constraint

The experiment in this section is performed over the dataset against a non-monotonic nor anti-monotonic constraint $\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$, where $\Phi_1(S) \equiv S.A_2 \leqslant v_1, \Phi_2(S) \equiv Max(S.A_1) - in(S.A_2) \leqslant v_2$ and $\Phi_3(S) \equiv \mathrm{sum}(S.A_3) \leqslant v_3$, where $\mathrm{sum}(S.A_3) \leqslant v_3$ denotes the summation of the third attributes of items in itemset S is less than or equal to v_3. We have known that $\Phi_1 \wedge \Phi_2$ is an anti-monotonic constraint in Section 5.2. However, $\Phi_3(S)$ is a non-monotonic nor anti-monotonic constraint since each value of A_3 in S could be of any real values. Since the FIC+ algorithm can not tackle a non-monotonic nor anti-monotonic constraint, we compare the NA-CFG algorithm with FP-growth algorithm.

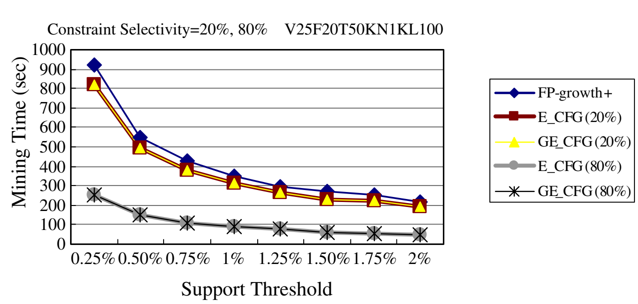

Fig. 11. Scalability with minimum support threshold.

In Fig. 12, the execution time of the NA-CFG algorithm decreases as the constraint selectivity increases since it can use $\Phi_1 \wedge \Phi_2$ to reduce the search space. The execution time of the FP-growth+ algorithm is constant w.r.t. the constraint selectivity. The NA-CFG algorithm is 15% ~ 1620 times faster than the FP-growth + algorithm.

In summary, it has been shown that our proposed algorithms are efficient and scalable for mining association rules with multi-dimensional constraints. Both E-CFG and GE-CFG algorithms outperform the FP-growth+ and FIC+ algorithms for all the cases because our algorithms exploit both sub-set and super-set properties of the constraint. The GE-CFG algorithm can save more constraint checks than the E-CFG algorithm in general because the GE-CFG algorithm can take advantage of the property of each sub-constraint, respectively.

6. Conclusions and future work

In this paper, we have enhanced the item representation by associating items with a number of attributes, so-called multi-dimensional items, and discussed the characteristics of multi-dimensional constraints. Then, we propose algorithms to mine association rules with multiple constraints constructed by multi-dimensional attribute values. Our proposed approach basically consists of three phases. First, we collect the frequent items and prune infrequent items according to the Apriori property. Second, we exploit the properties of the given constraints to prune search space or save constraint checking in the conditional databases. Third, for each itemset possible to satisfy the constraint, we generate its conditional database, construct its corresponding FP-tree and perform the three phases in the FP-tree recursively.

Our proposed algorithms can handle the case of a conjunction and/or disjunction of anti-monotone sub-constraints. That is, the conjunction and/or disjunction of sub-constraints still holds the anti-monotonic properties. Moreover, our proposed algorithms can exploit the properties of constraints to prune search space or save constraint checking. Therefore, our proposed algorithms are more efficient than the FP-growth+ and FIC+ algorithms.

Acknowledgement

The authors are grateful to the anonymous referees for valuable comments and suggestions, which significantly improve the presentation of this article.

References

Agrawal, R., Imielinski, T., Swami, A., 1993. Mining association rules between sets of items in large databases. In: Proceedings of ACM-SIGMOD, pp. 207–216.

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of International Conference on Very Large Data Bases, pp. 487–499.

Agrawal, R., Srikant, R., 1995. Mining sequential patterns, In: Proceedings of IEEE International Conference on Data Engineering, pp. 3–14.

Boulicaut, J.-F., Jeudy, B., 2000. Using constraint for set mining: should we prune or not? In: Proceedings of Bases de Donn'ees Avanc'ees, pp. 221–237.

Bucila, C., Gehrke, J., Kifer, D., White, W., 2002. DualMiner: a dual-pruning algorithm for itemsets with constraints. In: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining, pp. 42–51.

Gade, K., Wang, J., Karypis, G., 2004. Efficient closed pattern mining in the presence of tough block constraints. In: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining, pp. 138–147.

Grahne, G., Lakshmanan, L.V.S., Wang, X., 2000. Efficient mining of constrained correlated sets. In: Proceedings of IEEE International Conference on Data Engineering, pp. 512–521.

Han, J., Dong, G., Yin, Y., 1999. Efficient mining of partial periodic patterns in time series database. In: Proceedings of IEEE International Conference Data Engineering, pp. 106–115.

Han, J., Pei, J., Yin, Y., 2000. Mining frequent patterns without candidate generation. In: Proceedings of ACM-SIGMOD, pp. 1–12.

Kifer, D., Gehrke, J., Bucila, C., White, W., 2003. How to quickly find a witness. In: Proceedings of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 272–283.

Lee, A.J.T., Wang, Y.T., 2003. Efficient data mining for calling path patterns in GSM networks. Information Systems 28 (8), 929–948.

Ng, R., Lakshmanan, L.V.S., Han, J., Pang, A., 1998. Exploratory mining and pruning optimizations of constrained association rules. In: Proceedings of ACM-SIGMOD, pp. 13–24.

Pei, J., Han, J., 2000. Can we push more constraints into frequent pattern mining? In: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining, 350–354.

Pei, J., Han, J., Lakshmanan, L.V.S., 2001. Mining frequent itemsets with convertible constraints. In: Proceedings of IEEE International Conference on Data Engineering, pp. 433–442.

Srikant, R., Vu, Q., Agrawal, R., 1997. Mining association rules with item constraints. In: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining, pp. 67–73.