Mining Frequent Itemsets with Dualistic Constraints

Anh Tran[1], Hai Duong[1], Tin Truong[1], and Bac Le[2]

[1] Department of Mathematics and Computer Science, University of Dalat, Dalat, Vietnam
{anhtn,haidc,tintc}@dlu.edu.vn
[2] University of Natural Science Ho Chi Minh, Ho Chi Minh, Vietnam
lhbac@fit.hcmus.edu.vn

Abstract. Mining frequent itemsets can often generate a large number of frequent itemsets. Recent studies proposed mining itemset with the different types of constraint. The paper is to mine *frequent itemsets*, where a one: *does not contain any item of* C_0 *or contains at least one item of* C_0. The set of all those ones is partitioned into equivalence classes. Without loss of generality, we only investigate each class independently. One class is represented by a frequent closed set L and splits into two disjoint sub-classes. The first contains frequent itemsets that do not contain any item of C_0. It is generated from the corresponding generators. The second includes in two subsets of the frequent itemsets coming from the generators containing in C_0, and the ones obtained by connecting each non-empty subset of $L \cap C_0$ with each element of the first.

Keywords: Closed itemsets, frequent itemsets, dualistic, constraints, generators.

1 Introduction

First introduced and researched by Agrawal et al. [1] in 1993, mining frequent itemsets has been become one of the important problems in data mining. As usual, users are only interested in frequent itemsets that satisfy given constraints. The problem has been receiving attentions of many researchers [3, 4, 5, 7]. Let us consider searching documents in the Internet. The databases for obtaining them are usually saved into the tables. Each row in a table contains keywords appeared in a document. Assume that, a new user U wants to touch in the document D. Thus, U needs to know some keywords according to D. It is difficult to U, in the meaning that, U can get the set C of all keywords related to the subject that D belongs to, but he can not to know the keywords containing in D. Hence, the important task is to determine keyword sets containing in C. Those sets help the users to touch in documents quickly. For a transaction database T included in the set A of all items, let A^F be the set of all frequent ones. The paper focuses on the following problem: *Given a constraint* C ($C \subseteq A^F$)*, mine frequent itemsets (keyword sets) whose items are in* C (*Cons1*) *?*". They do not contain any item of the complement set C_0 of C ($C_0 := A \setminus C$). On the extension, we consider its dualistic problem: *Generate frequent itemsets L' such that L' contains at least one item of* C_0 (*Cons2*). For example, users need to know frequent keyword sets based on a few given keywords. They can lead users quickly to the desired documents.

P. Anthony, M. Ishizuka, and D. Lukose (Eds.): PRICAI 2012, LNAI 7458, pp. 807–813, 2012.

Solving these two problems by the algorithms of mining directly frequent itemsets such as Eclat [10], Apriori [1], etc is not suitable because minimum support and constraint often change (see [2] for details). Recently, in [2], we proposed the suitable model for mining frequent itemsets restricted on constraint. It can be applied to mine frequent itemsets with above dualistic constraints. Let us consider the class represented by frequent closed itemset L. For mining itemsets with *Cons2*, we split it into two parts. The first contains the ones generated from the generators that each of them contains at least one item of C_0. The other includes in the ones created by connecting each frequent itemset that does not contain any item of C_0 and each non-empty subset of $L \cap C_0$. The paper is organized as follows. Section 2 recalls some concepts of frequent itemset mining. Section 3 proposes the ways to generate non-repeatedly all frequent itemsets with dualistic constraints. Experimental results and the conclusion are shown in Sections 4 and 5.

2 Preliminaries

Given non-empty set $\mathcal{O}$ containing transactions. Let $\mathcal{A}$ be the set of items that are in transactions and $\mathcal{R}$ a binary relation on $\mathcal{O}x\mathcal{A}$. Consider two functions: $\lambda: 2^{\mathcal{O}} \to 2^{\mathcal{A}}$, $\rho: 2^{\mathcal{A}} \to 2^{\mathcal{O}}$ defined as follows: $\forall A \subseteq \mathcal{A}$, $O \subseteq \mathcal{O}$: $\lambda(O) = \{a \in \mathcal{A} \mid (o, a) \in \mathcal{R}, \forall o \in O\}$, $\rho(A) = \{o \in \mathcal{O} \mid (o, a) \in R, \forall a \in A\}$. Assign that $h = \lambda \circ \rho$, h(A) are called the closure of A. A is called a closed set [12] if $h(A) = A$. A set of items containing at least one transaction is called an *itemset*. For itemset S, $supp(S) := |\rho(S)| / |\mathcal{O}|$ is called the support of S. Let $s_0 \in (0; 1]$ be minimum support, S is frequent iff $supp(S) \geq s_0$ [1]. Let $\mathcal{FS}$ and $\mathcal{FCS}$ be respectively the classes of all frequent itemsets and all frequent closed itemsets. For G, A: $\emptyset \neq G \subseteq A \subseteq \mathcal{A}$, G is called a generator [6] of A if $h(G)=h(A)$ *and* $(\forall G': \emptyset \neq G' \subset G \Rightarrow h(G') \subset h(G))$. Since the cardinality of the class of all generators of A is finite, they can be numbered as follows: $\mathcal{G}(A) = \{A_i, i \in \{1, 2, \ldots, |\mathcal{G}(A)|\}\}$.

3 Mining Frequent Itemsets with Dualistic Constraints

Theorem 1 [8] (A partition of $\mathcal{FS}$). $\mathcal{FS} = \sum_{L \in \mathcal{FCS}} [L]$.

Each class contains frequent itemsets of the same closure L. Without loss of generality, *it only needs to exploit independently mining frequent itemsets with the dualistic constraints in each class*. Afterwards, we write $L \in \mathcal{FCS}$ simply L.

Definition 1. The set of the elements in [L] "*containing in* C" (*Cons1*) and the set of the ones "*involved with* C_0" (*Cons2*) are defined as follows:

$$\mathcal{FS}_C(L) = \{L' \in [L] \mid L' \cap C_0 = \emptyset\}, \quad \mathcal{FS}_{\cap C0}(L) = \{L' \in [L] \mid L' \cap C_0 \neq \emptyset\}.$$

Theorem 2 (Structure of Each Equivalence Class). Let us denote "+" as the union of two disjoint sets, we have:

$$[L] = \mathcal{FS}_C(L) + \mathcal{FS}_{\neg C0}(L).$$

Definition 2. The set of generators L_i containing in C and its complement on $\mathcal{G}(L)$ containing the generators involved with C_0 are defined in the following:

$$\mathcal{G}_C(L) = \{G \in \mathcal{G}(L) \mid G \cap C_0 = \varnothing\}, \quad \mathcal{G}_{\neg C}(L) = \{G \in \mathcal{G}(L) \mid G \cap C_0 \neq \varnothing\}.$$

Let N be the cardinality of $\mathcal{G}(L)$. All n generators of L in $\mathcal{G}_C(L)$ are numbered as L_1, L_2, .., L_n. The ones in $\mathcal{G}_{\neg C}(L)$ are L_{n+1}, L_{n+2}, .., L_N.

3.1 Generating Non-repeatedly Frequent Itemsets Containing in C

Using GEN-ITEMSETS [2], we derive the class [L]. For each L'$\in$[L], we test "L'$\cap C_0 = \varnothing$?". The ones passed are in $\mathcal{FS}_C(L)$. This way is simple (MFS-CC-SIMPLE is the corresponding algorithm). However, when the cardinality of [L] is big, and the one of $\mathcal{FS}_C(L)$ is small, it runs slowly. How to generate directly the elements of $\mathcal{FS}_C(L)$? Based on propositions 2 and 3 in [2], we can do it quickly.

MFS-CC (*L*, $\mathcal{G}(L)$):
1. NewGL = CLASSIFY-GENERATORS (L, $\mathcal{G}(L)$, out n, out N)
2. **return** Sub-MFS-CC (L, NewGL, n) // $\mathcal{FS}_C(L)$

Sub-MFS-CC (*L, NewGL, n*):
3. $\mathcal{FS}_C(L) = \varnothing$ and $X_U = \bigcup_{L_i \in \mathcal{G}_C(L)} L_i$ and $X\_ = (L \cap C) \backslash X_U$
4. **for** (i=1; i <= n; i++) **do**
5. $X_{U,i} = X_U \backslash L_i$; // $L_i \in \mathcal{G}_c(L)$
6. **for all** $X'_i \subseteq X_{U,i}$ **do**
7. IsDuplicate = false
8. **for** (k=1; k<i; k++) **do**
9. **if** $X_k \subset X_i + X'_i$ **then** IsDuplicate = true and *break*
10. **if** *not(IsDuplicate)* **then**
11. **for all** $X^{\sim} \subseteq X\_$ **do** $\mathcal{FS}_c(L)$.add($X_i + X'_i + X^{\sim}$)
12. **return** $\mathcal{FS}_C(L)$

Fig. 1. The algorithm MFS-CC for mining frequent itemsets containing in C

Table 1. Database T

Trans	Items	Trans	Items
1	aceg	5	aceg
2	acfh	6	bceg
3	adfh	7	acfh
4	bceg		

Example 1. Consider database T in Table 1. Fix now s_0 = 2/7, using *Charm-L* [11] and *MinimalGenerators* [9], we have the frequent closed itemset L=aceg together $\mathcal{G}$(L) = {ae, ag}. With $\mathcal{C}_0$ = cfh. Then, $\mathcal{C} = \mathcal{A}^F \setminus \mathcal{C}_0$ = abcefgh \ cfh = abeg. From definition 2, $\mathcal{G}_C$(L) = {$G_{C,1}$=ae, $G_{C,2}$=ag}. Using MFS-CC, X_U = aeg, $X_{U,1}$ = g, $X_{U,2}$ = e, $X_{\_}$ = (L∩abeg)\ X_U = ∅. For $G_{C,1}$, we have: ae+∅, ae+g ∈ $\mathcal{FS}_C$(aceg). For $G_{C,2}$: ag+∅ ∈ $\mathcal{FS}_C$(aceg) (ag+e does not appear again). Thus, $\mathcal{FS}_C$(aceg) = {ae, aeg, ag}.

3.2 Mining Frequent Itemsets Involved with $\mathcal{C}_0$

Based on directly definition 1, we can obtain the algorithm MFS-IC-SIMPLE (by replacing "L'∩$\mathcal{C}_0$ = ∅" in MFS-CC-SIMPLE by "L'∩$\mathcal{C}_0$ ≠ ∅") for mining the class of frequent itemsets involved with $\mathcal{C}_0$. However, it works slowly. So, how to generate quickly elements of $\mathcal{FS}_{\cap C0}$(L)? We could assume that L∩$\mathcal{C}_0$ ≠ ∅ (conversely, $\mathcal{FS}_{\cap C0}$(L) = ∅). We split $\mathcal{FS}_{\cap C0}$(L) into two parts. The first one $\mathcal{FS}^+_C$(L) contains frequent itemsets created by adding each non-empty subset of L∩$\mathcal{C}_0$ into each frequent itemset containing in $\mathcal{C}$. We have:

$$\mathcal{FS}^+_C(L) = \mathcal{FS}_C(L) \oplus (2^{L\cap C_0}\setminus\{\varnothing\}),$$

where: the "sum" operator ⊕ of X and Y (X, Y ⊆ $2^A\setminus\{\varnothing\}$) is defined: $X \oplus Y = \{A+B: \varnothing \neq A \in X, \varnothing \neq B \in Y\}$. Using the generators involved with $\mathcal{C}_0$, we generate the frequent itemsets involved with it. For L ∈ $\mathcal{FCS}$, $L_U = \bigcup_{L_i \in \mathcal{G}(L)} L_i$, $L_k \in \mathcal{G}_{\neg C}(L)$, $L_{U,k}$ = $L_U \setminus L_k$, $L_{\_}$ = L \ L_U, let assign

$$\mathcal{FS}_{\neg C}(L) = \{L_k + L'_k + L\sim: L_k \in \mathcal{G}_{\neg C}(L), L'_k \subseteq L_{U,k}, L\sim \subseteq L_{\_} \text{ and } (L_j \not\subset L_k + L'_k, L_j \in \mathcal{G}(L), \forall j: 1 \leq j < k)\}.$$

Theorem 3 (Structure of the Set of Frequent Itemsets Involved with $\mathcal{C}_0$). For L ∈

$$\mathcal{FCS}: \mathcal{FS}_{\cap C0}(L) = \mathcal{FS}^+_C(L) + \mathcal{FS}_{\neg C}(L).$$

Proof: It is easy to see that $\mathcal{FS}^+_C$ (L) and $\mathcal{FS}_{\neg C}$(L) are two disjoint subsets of $\mathcal{FS}_{\cap C0}$(L). Then, let us prove that: $\mathcal{FS}_{\cap C0}$(L) ⊆ $\mathcal{FS}^+_C$ (L) + $\mathcal{FS}_{\neg C}$(L). Denoted that L_U = $\bigcup_{L_i \in \mathcal{G}_C(L)} L_i$, $L_{U,C,i} := L_{U,C} \setminus L_i$, $L_{\_,C} := (L\cap C)\setminus L_{U,C}$. ∀L'= L_k+ L'_k+ L~ ∈ $\mathcal{FS}_{\cap Co}$(L), $L_k \in \mathcal{G}$(L), $L'_k \subseteq L_{U,k}$, L~⊆$L_{\_}$ and L'∩$\mathcal{C}_0 \neq \varnothing$, consider two cases: [*Case 1*] If $L_k \in \mathcal{G}_C$(L), then (L'_k+L~)∩$\mathcal{C}_0 \neq \varnothing$. Let us call $L''_k = L'_k \cap L_U$, T ⊆ $L_{U,C,k}$, L'''_k = ($L'_k \setminus L_{U,C}$)∩T ⊆ $L_{\_,C}$, L''''_k = ($L'_k \setminus L_{U,C}$)\T ⊆ L\T, $L\sim_T$=L~∩T ⊆ $L_{\_,C}$, $L\sim_{\neg T}$=L~\T ⊆ L\T, so ($L'''k$+$L\sim_T$) ⊆ $L_{\_,C}$ and ∅≠(L''''_k+$L\sim_{\neg T}$)⊆L\T. Indeed, if (L''''_k+$L\sim_{\neg T}$)=∅, L''''_k = $L\sim_{\neg T}$ = ∅. Then, L~⊆$\mathcal{C}_0$, $L'_k \setminus L_{U,C} \subseteq \mathcal{C}_0$ and ∅≠(L'_k+ L~)∩$\mathcal{C}_0$= L'_k+L~ = L''''_k+$L\sim_{\neg T}$ = ∅: contradiction! Hence, L'=[L_k+L''_k+(L'''_k+$L\sim_T$)]+(L''''_k+$L\sim_{\neg T}$) ∈ $\mathcal{FS}^+_C$(L). [*Case 2*] If $L_k \in \mathcal{G}_{\neg C}$(L), L'∈ $\mathcal{FS}_{\neg C}$(L).

Example 2. Consider L=aceg, $\mathcal{G}$(L)={ae, ag} and C_0=e. So C=abcfgh. Thus, $\mathcal{G}_C$(L)={ag}. Since $\mathcal{FS}_C$(aceg)={ag, agc}, L$\cap C_0$=e, so $\mathcal{FS}^+_C$(L)={ag+e, agc+e}. Moreover, $\mathcal{G}_{\neg C}$(L) = {ae}, L_U = aeg, $L_{U,1}$ = g and $L_{\_}$ = c. Thus, $2^{L_{\_}}$ = {0, c}. Then $\mathcal{FS}_{\neg C}$(L) = {ae, ae+c}. Hence, $\mathcal{FS}_{\cap C0}$(L) = $\mathcal{FS}^+_C$(L) + $\mathcal{FS}_{\neg C}$(L) = {ag+e, agc+e, ae, ae+c}.

MFS-IC (L, $\mathcal{G}(L)$):
1. NewGL = CLASSIFY-GENERATORS (L, $\mathcal{G}$(L), out n, out N)
2. $\mathcal{FS}^+_C$(L) = ∅ and LC = L$\cap C_0$
3. **if** LC ≠ ∅ **then**
4. $\mathcal{FS}_C$(L) = Sub-MFS-CC (L, $\mathcal{G}$(L)) and LC_Class = $2^{LC}\backslash\{\emptyset\}$
6. **for all** L' ∈ $\mathcal{FS}_C$(L) **do**
7. **for all** L''' ∈ $\mathcal{FS}_C$(L) **do** $\mathcal{FS}^+_C$(L).add (L'+L''')
8. $\mathcal{FS}_{\neg C}$(L) = ∅ and $L_U = \bigcup_{L_i \in \mathcal{G}(L)} L_i$ and $L_{\_} = L\backslash L_U$;
9. **for** (k=n+1; k <= N; k++) **do**
10. $L_{U,k} = L_U \backslash L_k$ // $L_k \in \mathcal{G}_{\neg C}$(L)
11. **for all** $L'_k \subseteq L_{U,k}$ **do**
12. IsDuplicate = false
13. **for** (j=1; j<k; j++) **do**
14. **if** $L_j \subset L_k + L'_k$ **then** *IsDuplicate* = true and *break*
15. **if** *not(IsDuplicate)* **then**
16. **for all** L˜ ⊆ $L_{\_}$ **do** $\mathcal{FS}_{\neg C}$(L).add(L_k+L'_k+L˜)
17. **return** $\mathcal{FS}^+_C$(L) + $\mathcal{FS}_{\neg C}$(L)

Fig. 2. The algorithm MFS-IC to mine frequent itemsets involved with C_0

4 Experimental Results

The following experiments were performed on i5-2400 CPU, 3.10 GHz @ 3.09 GHz, 3.16 GB RAM, running Windows. The algorithms were coded in C#. Four databases in FIMDR (http://fimi.cs.helsinki.fi/data/) are used during these experiments: Pumsb contains 49046 transactions, 7117 items (P, 49046, 7117); Mushroom (M), 8124, 119; Connect (C), 67557, 129; and Pumsb* (P*), 49046, 7117.

We compare the running times of MFS-CC-SIMPLE with MFS-CC when mining frequent itemsets containing in C. For each pair of database (DB) and minimum support (MS), we consider the lengths of C ranging from 20% to 70% of $|\mathcal{A}^F|$ (step 2%). For each one, 15 constraints are considered. We test 234 cases for each algorithm. Experiments showed that in almost cases, MINE-CC runs quickly than MFS-CC-SIMPLE. We can save the amounts of the running time ranging from *88.7% to 95.5%*.

Next, we compare the running time of MFS-IC-SIMPLE (T_ICS) with the one of MFS-IC (T_IC) when mining frequent itemset involved with C_0,. For each (DB, MS), the lengths of C_0 are ranged from 1 to at most 20% of $|A^I|$ with step 2. Table 2 shows the comparison, where: NCons is the number of the selected constraints, NLess is the number of constraints such that T_IC < T_ICS. The percent ratio of NLess to NCons and the average number of the percent ratios of T_IC to T_ICS are shown in columns *RNLess* and R_IC (%). In almost cases, MFS-IC runs more quickly than MFS-IC-SIMPLE. The time can be reduced into a value of *62.3% to 19.1%*.

Table 2. The reductions in the time for mining frequent itemsets involved with C_0

(DB, MS)	NCons	RNLess (%)	R_IC (%)	(DB, MS)	NCons	RNLess (%)	R_IC (%)
M, 0.15	135	*100.0*	27.6	P, 0.75	135	99.3	*62.3*
M, 0.1	150	100.0	22.4	P, 0.7	135	98.5	49.0
M, 0.05	150	100.0	*19.1*	P, 0.65	135	100.0	43.3
Co, 0.65	135	100.0	26.3	P*, 0.35	150	99.3	36.9
Co, 0.6	135	100.0	23.2	P*, 0.3	150	100.0	34.3
Co, 0.55	135	100.0	26.2	P*, 0.25	150	100.0	30.2

5 Conclusions

We presented the efficient algorithms MFS-CC and MFS-IC for mining frequent itemsets with the dualistic constraints. Those algorithms are built based on the explicit structure of frequent itemset class. The class is split into two sub-classes. Each sub-class is found by applied the efficient representation of itemsets to the suitable generators. The tests on four benchmark databases showed the efficiency of our approach.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceeding of the 20th International Conference on Very Large Data Bases, pp. 478–499 (1994)
2. Anh, T., Hai, D., Tin, T., Bac, L.: Efficient Algorithms for Mining Frequent Itemsets with Constraint. In: Proceedings of the Third International Conference on Knowledge and Systems Engineering, pp. 19–25 (2011)
3. Bayardo, R.J., Agrawal, R., Gunopulos, D.: Constraint-Based Rule Mining in Large, Dense Databases. Data Mining and Knowledge Discovery 4(2/3), 217–240 (2000)
4. Cong, G., Liu, B.: Speed-up Iterative Frequent Itemset Mining with Constraint Changes. In: ICDM, pp. 107–114 (2002)
5. Nguyen, R.T., Lakshmanan, V.S., Han, J., Pang, A.: Exploratory Mining and Pruning Optimizations of Constrained Association Rules. In: Proceedings of the 1998 ACM-SIGMOD Int'l Conf. on the Management of Data, pp. 13–24 (1998)

6. Pasquier, N., Taouil, R., Bastide, Y., Stumme, G., Lakhal, L.: Generating a condensed representation for association rules. J. of Intelligent Information Systems 24(1), 29–60 (2005)
7. Srikant, R., Vu, Q., Agrawal, R.: Mining association rules with item constraints. In: Proceeding KDD 1997, pp. 67–73 (1997)
8. Truong, T.C., Tran, A.N.: Structure of Set of Association Rules Based on Concept Lattice. In: Nguyen, N.T., Katarzyniak, R., Chen, S.-M. (eds.) Advances in Intelligent Information and Database Systems. SCI, vol. 283, pp. 217–227. Springer, Heidelberg (2010)
9. Zaki, M.J.: Mining non-redundant association rules. Data Mining and Knowledge Discovery (9), 223–248 (2004)
10. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD 1997), pp. 283–296 (1997)
11. Zaki, M.J., Hsiao, C.J.: Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Trans. Knowledge and Data Engineering 17(4), 462–478 (2005)
12. Wille, R.: Concept lattices and conceptual knowledge systems. Computers and Math. with App. 23, 493–515 (1992)