Fast and Memory Efficient Mining of Frequent Closed Itemsets

Claudio Lucchese
Salvatore Orlando
Raffaele Perego

From TKDE06

Outline

- **Introduction**
- **Memory-Efficient Duplicate Detection and Pruning**
- **DCI_Closed Algorithm**
- **Performance Analysis**
- **Conclusion**

Introduction

- **Dense data:**

  Contain strongly correlated items and long frequent patterns

- Such data sets are, in fact, very hard to mine, while the number of frequent itemsets grows up very quickly as the minimum support threshold is decreased.

Introduction(cont.)

- **Closed Itemsets**

  Given an itemset T ⊆ *D*, and I⊆ *I* and we define

  $f(T) = \{ i \in I \mid \forall t \in T , i \in t \}$

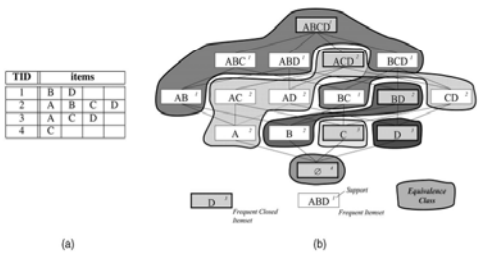
  $g(S) = \{ t \in D \mid \forall i \in I , i \in t \}$

- **An itemset I is said to be closed if and only if**

  $c(I)=f(g(I))=f \circ g(I)=I$

Introduction(cont.)

- min_supp = 1,



Introduction(cont.)

- Browsing the search space

  Lemma 1. Given two itemsets X and Y ,if X⊂Y and supp(X) = supp(Y), then c(X) = c(Y)

- Therefore, given a generator X, if we find an already mined closed itemsets Y that set-includes X, where the supports of Y and X are identical, we can conclude that c(X)=c(Y). In this case, we also say that Y subsumes X.If this holds, we can safely prune the generator X without computing its closure. Otherwise, we have to compute c(X) in order to obtain a new closed itemset.

Introduction(cont.)

- We could in fact mine all the closed itemsets by computing the closure of just this single representative itemset for each equivalence class, without generating any duplicate. Let us call representative itemsets *closure generators*.
- Other algorithms use a different technique, which we call *closure climbing.*
- For example, the closed itemset {A,B,C,D} of the figure could be mined twice since it can be obtained as the closure of two minima elements of its equivalence class, namely, {A,B} and {B,C}.

Introduction(cont.)

- Given an itemset X and an item $i \in I$, $g(X) \subseteq g(i) \Leftrightarrow i \in c(X)$
- From the above lemma, we have that if $g(X) \subseteq g(i)$, then $i \in c(X)$. Therefore, by performing this inclusion check for all the items in I not included in X, we can incrementally compute c(X).

Memory-Efficient Duplicate Detection and Pruning(cont.)

- For example, the closed itemsets {A,C,D} has four such generators, namely, {A}, {A,C}, {A,D}, and {C,D}.
- Denote with symbol < the usual lexicographic total order between two ordered itemsets, in turn, defined on the basis of R.

Memory-Efficient Duplicate Detection and Pruning(cont.)

- A generator of the form $X=Y \cup i$, where Y is a closed itemset and $i \notin Y$, is said to be order-preserving iff either $c(X) = X$ or $i < (c(X) \setminus X)$.
- Example of Figure, we have that $\{A\}=\psi \cup \{A\}$ is an order-preserving generator of the closed itemset {A,C,D}, while $\{C,D\}=\{C\} \cup \{D\}$ is not an order-preserving generator for the same closed itemset.

Memory-Efficient Duplicate Detection and Pruning(cont.)

- In order to mine all the closed itemsets by avoiding redundances, we compute the closure of order-preserving generators only and prune the others.

- **Theorem 1.**

  For each closed itemset $\bar{Y} \neq c(\psi)$, there exists a sequence of n items $i_0 < i_1 < \ldots < i_{n-1}$, $n \geq 1$, such that $\langle gen_0, gen_1, \ldots, gen_{n-1} \rangle = \langle Y_0 \cup i_0, Y_1 \cup i_1, \ldots, Y_{n-1} \cup i_{n-1} \rangle$, where the various geni are order-preserving generators, with $Y_0 = c(\psi)$, $j \in [0,n-1]$, $Y_{j+1} = c(Y_j \cup i_j)$, and $Y_n = \bar{Y}$.

Memory-Efficient Duplicate Detection and Pruning(cont.)

- **Corollary 1.**

  For each closed itemset $\bar{Y} \neq c(\psi)$, the sequence of order-preserving generators of Theorem 1 is unique.

- **Example:**

  For the closed itemset $\bar{Y}$={A,B,C,D}, we have $Y_0 = c(\psi)=\psi$, $gen_0=\psi \cup \{A\}$, $Y_1=c(gen_0)=\{A,C,D\}$, $gen_1= \{A,C,D\} \cup \{B\}$, and,finally, $\bar{Y} = c(gen_1)$.

Memory-Efficient Duplicate Detection and Pruning(cont.)

- **Detecting Order-Preserving Generator**
- **Definition 3.**

Given a generator gen = Y∪i, where Y is a closed itemset and i ∉Y, we define pre-set(gen) as follows:

pre-set(gen) = { j | j∈*I*, j ∉gen, and j < i}.

- **Lemma 3.**

Let gen = Y∪i, i be a generator where Y is a closed itemset and i ∉ Y. If ∃j∈pre-set{gen} such that g{gen} ⊂g{j}, then gen is not order-preserving.

DCI_CLOSED Alogrithm

- DCI_CLOSED starts by scanning the input data set D to determine the frequent single items F1⊆I and builds the bitwise vertical data set VD containing the various tidlists g(i).
- After this first step, DCI_CLOSED decides whether *VD* corresponds to either a dense or a sparse data set. Since *VD* is bitwise, if the percentage of 1s is large, the data set is soon classified as dense.

DCI_CLOSED Alogrithm(cont.)

Algorithm 1 DCI_CLOSED pseudocode

```
1: procedure DCI_CLOSED_d (CLOSED_SET, PRE_SET, POST_SET)
2:   while POST_SET ≠ ∅ do
3:     i ← min_≺(POST_SET)
4:     POST_SET ← POST_SET \ i
5:     new_gen ← CLOSED_SET ∪ i                    ▷ Build a new generator
6:     if supp(new_gen) ≥ min_supp and
       ¬is_dup(new_gen, PRE_SET) then
                                  ▷ if new_gen is both frequent and order preserving
7:       CLOSED_SET_New ← new_gen
8:       POST_SET_New ← ∅
9:       for all j ∈ POST_SET do                   ▷ Compute closure of new_gen
10:        if g(new_gen) ⊆ g(j) then
11:          CLOSED_SET_New ← CLOSED_SET_New ∪ j
12:        else
13:          POST_SET_New ← POST_SET_New ∪ j
14:        end if
15:      end for
16:      Write Out CLOSED_SET_New and its support
17:      DCI_CLOSED_d(CLOSED_SET_New, PRE_SET, POST_SET_New)
18:      PRE_SET ← PRE_SET ∪ i
19:    end if
20:  end while
21: end procedure
```

DCI_CLOSED Alogrithm(cont.)

```
22: function is_dup(new_gen, PRE_SET)              ▷ Duplicate check
23:   for all j ∈PRE_SET do
24:     if g(new_gen) ⊆ g(j) then
25:       return TRUE                    ▷ new_gen is not order preserving
26:     end if
27:   end for
28:   return FALSE
29: end function
```

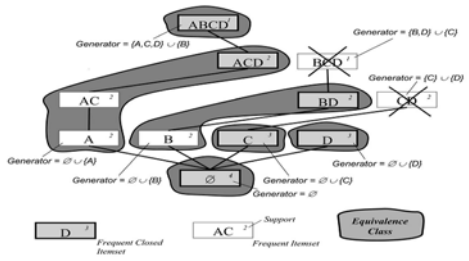
DCI_CLOSED Alogrithm(cont.)

- Once c(ψ)=ψ, is found, four generators can be constructed by adding a single item to c(ψ), namely, {A}, {B}, {C}, and {D}. Suppose we first compute the closure of gen=ψ∪ {A}={A}. Note that, since no items precede A in the lexicographic order, then its PRE_SET is empty and, thus, we can conclude that gen is order-preserving. DCI_CLOSEDd() checks if g(A) is set-included in g(j), ∀j∈POST_SET (i.e., g(B), g(C), and g(D)), and discovers that c(A)={A,C,D}.
- DCI_CLOSEDd() is then recursively called, with parameters CLOSED_SET = {A,C,D}, POST_SET = {B}, while PRE_SET is still empty. CLOSED_SET = {A,C,D} is thus extended with B (its POST_SET), so obtaining a new generator gen ={A,C,D} ∪{B}={A,B,C,D}. Since PRE_SET is empty, this generator is order-preserving by definition, but is also closed because POST_SET is now empty.

DCI_CLOSED Alogrithm(cont.)

- After this first recursive exploration, DCI_CLOSEDd() starts solving another independent subproblemby exploring generator gen=ψ∪{B}={B}, where PRE_ SET = {A} and POST_SET = {C,D}.
- Finally, DCI_CLOSEDd() starts exploring the last generator gen =ψ∪{D}={D}, where PRE_SET = {A,B,C} and POST_SET = ψ Since gen is order-preserving (this is checked by comparing g(D) with g(A), g(B), and g(C), i.e., with its PRE_SET), it is not pruned. But, we also can conclude that {D} is also closed since POST_SET = ψ.

DCI_CLOSED Alogrithm(cont.)



DCI_CLOSED Alogrithm(cont.)

- **Optimization Saving Bitwise Operations**

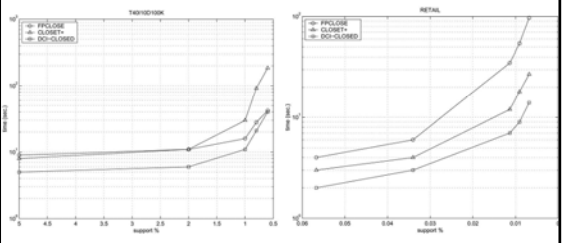
1.Data sets with highly correlated items

如果我們要mine的目標是x，那我們在每個columns與x無關的拿掉，但是這樣又會很花時間去check每一個column，那我們就限制條件，限制只有跟Φ相關的的itemset才可以執行。

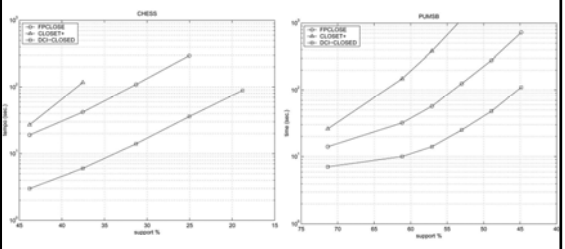
2.Data sets with highly correlated items

此方發使用在mine dense data上，則是利用A Multi-Strategy Algorithm for Mining Frequent Sets以及Adaptive and Resource-Aware Mining of Frequent Sets這兩篇paper的方式去作處理。

Performance Analysis



Performance Analysis(cont.)



Conclusion

- In this paper, we have investigated the problem of efficiency in mining closed frequent itemsets from transactional data sets.
- Finally, it showed that allows dense data sets to also be effectively mined with the lowest possible support threshold