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 \( T \subseteq D \), and \( I \subseteq I \) and we define
  \[
  f(T) = \{ i \in I | \forall t \in T, i \in t \}
  \]
  \[
  g(S) = \{ t \in D | \forall I \in S, i \in t \}
  \]
- 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,

Browsing the search space

Lemma 1. Given two itemsets \( X \) and \( Y \), if \( X \subseteq Y \) and \( \text{supp}(X) = \text{supp}(Y) \), 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.
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\}.

Given an itemset \(X\) and an item \(i\), \(g(X) \subseteq g(i)\)
\(\Leftrightarrow \exists \in \in c(X)\)

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 \(c(X)\).

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 \(R\).

A generator of the form \(X=Y_i\), 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\} = \(\psi\) \(\in\) \{A\} is an order-preserving generator of the closed itemset \{A,C,D\}, while \{C,D\} \(\in\) \{C\} \(\in\) \{D\} is not an order-preserving generator for the same closed itemset.

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

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

**Theorem 1.**
For each closed itemset \(\neq c(\psi)\), there exists a sequence of \(n\) items \(i_0 < i_1 < \ldots < i_n\), such that \(<g_{i_0}, g_{i_1}, \ldots, g_{i_n}> = \{Y_0 \cup U_0, Y_1 \cup U_1, \ldots, Y_n \cup U_n\}\), where the various \(g_{i}\) are order-preserving generators, with \(Y \neq c(\psi)\), \(j \in [0,n-1]\), \(Y_j := c(Y_j \cup U_j)\), and \(Y_n := \psi\).

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

**Example:**
For the closed itemset \(\neq c(\psi)\), we have \(Y = c(\psi) = \psi\), \(g_{\psi} = \psi \cup \{A\}\), \(Y_1 = c(g_{\psi}) = \{A,C,D\}\), \(g_{\psi} = \{A,C,D\} \cup \{B\}\), and, finally, \(Y_1 = c(g_{\psi})\).
Memory-Efficient Duplicate Detection and Pruning (cont.)

- Detecting Order-Preserving Generator
  - **Definition 3.**
    Given a generator gen = \( Y \cup \{ i \} \), where \( Y \) is a closed itemset and \( i \not\in Y \), we define pre-set(gen) as follows:
    \[
    \text{pre-set}(\text{gen}) = \{ j | j \not\in Y, j \in \text{gen}, \text{and} j < i \}.
    \]
  - **Lemma 3.**
    Let gen = \( Y \cup \{ i \} \), be a generator where \( Y \) is a closed itemset and \( i \not\in Y \). If \( \exists j \in \text{pre-set}(\text{gen}) \) such that \( g_j \cup \text{gen} \not\subset g_i \), then gen is not order-preserving.

DCI_CLOSED Algorithm

- **DCI_CLOSED** starts by scanning the input data set D to determine the frequent single items \( F_1 \) and builds the bitwise vertical data set \( VD \) containing the various tidlists \( g_j \).
  - After this first step, \( \text{DCI\_CLOSED} \) 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 Algorithm (cont.)

- **Algorithm 1 DCI\_CLOSED pseudocode**
  1. sort \( F_1 \) by \( \text{support} \);
  2. \( \text{POST\_SET} = \emptyset \);
  3. \( \text{PRE\_SET} = \emptyset \);
  4. \( \text{CLOSED\_SET} = \emptyset \);
  5. **while** \( \text{POST\_SET} \neq \emptyset \) **do**
    1. \( \text{PRE\_SET} = \emptyset \);
    2. \( \text{BUILD\_NEW\_GEN}() \);
    3. \( \text{SORT\_GENS}() \);
    4. \( \text{STORE\_GENS}() \);
  6. **end while**

- **DCI\_CLOSED\_d()**
  - Once \( c(\psi) = \psi \), is found, four generators can be constructed by adding a single item to \( c(\psi) \), namely, \( \{ A \} \), \( \{ B \} \), \( \{ C \} \), and \( \{ D \} \). Suppose we first compute the closure of \( c(\psi) \cup \{ A \} = \{ A \} \). 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\_CLOSED\_d()** checks if \( g(A) \) is set-included in \( g_j \).
  - \( \forall j \in \text{POST\_SET} \) (i.e., \( g(B) \), \( g(C) \), and \( g(D) \)), and discovers that \( c(\{ A \}) = \{ A, C, D \} \).

- **DCI\_CLOSED\_d()** is then recursively called, with parameters \( \text{CLOSED\_SET} = \{ A, C, D \} \), \( \text{POST\_SET} = \{ B \} \), while \( \text{PRE\_SET} \) is still empty. \( \text{CLOSED\_SET} = \{ A, C, D \} \) is thus extended with \( B \) (its \( \text{POST\_SET} \)), so obtaining a new generator \( \{ A, C, D \} \cup \{ B \} = \{ A, B, C, D \} \). Since \( \text{PRE\_SET} \) is empty, this generator is order-preserving by definition, but is also closed because \( \text{POST\_SET} \) is now empty.

- After this first recursive exploration, **DCI\_CLOSED\_d()** starts solving another independent subproblem by exploring generator \( \{ B \} \cup \{ D \} \), where \( \text{PRE\_SET} = \{ A \} \) and \( \text{POST\_SET} = \{ C, D \} \).

- Finally, **DCI\_CLOSED\_d()** starts exploring the last generator \( \{ C \} \cup \{ D \} \), where \( \text{PRE\_SET} = \{ A, B, C \} \) and \( \text{POST\_SET} = \emptyset \). Since gen is order-preserving (this is checked by comparing \( g(D) \)), \( g(A) \), \( g(B) \), and \( g(C) \), i.e., with its \( \text{PRE\_SET} \), it is not pruned. But, we also can conclude that \( \{ D \} \) is also closed since \( \text{POST\_SET} = \emptyset \).
Optimization Saving Bitwise Operations

1. Data sets with highly correlated items

If we are to mine our goal is x, we manipulate columns with x unrelated, but it might be difficult to check every column. If we restrict the conditions, we can only execute itemsets related to x.

2. Data sets with highly correlated items

This is utilized in dense data. We use the Multi-Strategy Algorithm for Mining Frequent Sets as well as Adaptive and Resource-Aware Mining of Frequent Sets, respectively.

Performance Analysis

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.