Introduction

- Frequent itemset mining
  - A set of items is referred to as itemset
  - \( X \) is an item (or itemset), \( \text{Support}(X) = \frac{\#X}{n} \)
  - Support is bounded by a threshold \( r \)
  - A frequent itemset is an itemset with a support larger than the minimum support
  - Given a database, find all the frequent itemsets

Problems with frequent itemset mining algorithms

- The computation may be intractable for a user-given frequency threshold: the number of frequent itemsets may explode
- Lack of focus leads to huge output of frequent itemsets
Introduction

- Two issues to tackle these problems
  - Constraint-based extraction of frequent itemsets: only a subset of the collection of frequent itemsets is interesting.
  - Condensed representation of frequent itemsets: extract a subset of the frequent patterns and regenerate the whole collection when necessary.

- Constraint-based extraction of frequent itemsets
  - Syntactic constraints
    - an item must not appear in the itemsets
  - Constraints related to objective measures of interestingness
    - the itemsets must be frequent

- Push constraint checking into algorithms
  - Anti-monotone constraints
  - Monotone constraints

- Condensed representation of frequent itemsets
  - Extract a particular subset of the frequent itemset collection
  - The condensed subset is much smaller than the original collection
  - Can be extracted efficiently
  - The whole frequent itemsets can be regenerated

- Main idea of the paper
  - Combine the above two approaches into one algorithm
  - This algorithm is based on the structure of Apriori
Introduction

Constrained itemset mining
  - Apriori revisit
  - Anti-monotone constrains
  - Monotone constrains
  - Generic algorithm

Frequent closed itemset mining
  - CLOSE algorithm
  - Incorporating constraints into Apriori

Conclusion

Summary of paper

Definition of constraints
  - \( T \) : transactional database
  - \( 2^{\text{items}} \) : set of all itemsets
  - \( C \) : constraint
  - \( S \) : itemset, \( S \in 2^{\text{items}} \)
  - \( I \) : subset of \( 2^{\text{items}} \)
  - \( S \) satisfies \( C \) in \( T \) if and only if \( (S, T) = \text{true} \)
  - \( SAT_C(I) = \{ S \in I, S \text{ satisfies } C \} \)
  - \( SAT_C \) denotes \( SAT_C(2^{\text{items}}) \)

Summary of paper

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Itemset</th>
<th>Support</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABCD</td>
<td>A</td>
<td>1,2,3,4,6</td>
<td>0.83</td>
</tr>
<tr>
<td>2</td>
<td>AC</td>
<td>B</td>
<td>1,4,5,6</td>
<td>0.67</td>
</tr>
<tr>
<td>3</td>
<td>AC</td>
<td>AB</td>
<td>1,4,5</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>ABCD</td>
<td>AC</td>
<td>1,2,3,4,6</td>
<td>0.83</td>
</tr>
<tr>
<td>5</td>
<td>BC</td>
<td>CD</td>
<td>1,4</td>
<td>0.33</td>
</tr>
<tr>
<td>6</td>
<td>ABC</td>
<td>ACD</td>
<td>1,4</td>
<td>0.33</td>
</tr>
</tbody>
</table>

\( C_{\text{freq}}(S) = F(S) \geq r \) : an itemset must be at least \( r \) frequent.

\( r = 0.6 \quad SAT_{c_{\text{freq}}} = \{ A, B, C, AC, BC \} \)

\( C_{\text{size}}(S) \leq 2 \) and \( C_{\text{miss}}(S) \equiv B \not\subseteq S \), then

\( SAT_{c_{\text{size}},c_{\text{miss}}} = \{ A, C, D, AC, AD, CD \} \quad SAT_{c_{\text{size}},c_{\text{miss}},c_{\text{freq}}} = \{ A, C, AC \} \)

Constrained itemset mining
  - \( T \) : transactional database
  - \( C \) : constraint
  - Computation of the collection of itemsets that satisfy \( C \) together with their frequencies
    - \( R_C = \{ (S, F(S)), S \in SAT_C \} \)
  - Use Apriori for constrained itemset mining
    where \( C \) is \( C_{\text{freq}} \)
### Summary of paper - Apriori

**Apriori Algorithm**

1. \( C_1^g := \{\} \)
2. \( k := 1 \)
3. **while** \( C_k^g \neq \emptyset \) **do**
4. **safe-pruning-on**
5. **generate** \( L_k := SAT_{C_{prior}}(C_k^g) \)
6. \( C_{k+1}^g := generate_{apriori}(L_k) \)
7. \( k := k + 1 \)
8. \( U_{j=0}^{k-1} L_j \)

**Phase 1 – Candidate safe pruning**
Eliminate candidates for which a subset of length \( k \) is not frequent.

**Phase 2 – frequency constraint**
(database scan)

**Phase 3 – candidate generation for level \( k+1 \)**
Fuse two elements that share the same \( k-1 \) first items.

### Anti-monotone constraints

- **Definition:** an anti-monotone constraint is a constraint \( C \) such that for all itemsets \( S, S' \):
  \( (S' \subseteq SAS \text{ satisfy } C) \Rightarrow S' \text{ satisfy } C \)
- If \( S \) does not satisfy \( C_{an} \), every superset of \( S \) does not satisfy \( C_{an} \).
- Example: \( \sum(S, \text{ price}) \leq v \)
- A disjunction or conjunction of anti-monotone constraints is an anti-monotone constraint.

### Monotone Constraints

- **Definition**
  \( S \in \text{Items}, C_n(S) \text{ is true } \Rightarrow \forall S', S, C_n(S') \text{ is true} \)
- Example
  \( \sum(S, \text{ price}) \geq v \)
- Given a monotone constraint \( C_n \), simply replacing Step 5 in Apriori with \( L_k := SAT_{C_n}(C_k) \) leads to the loss of the completeness of Apriori.

### Anti-monotone constraints

- **Apriori can be changed:**
  - Let \( C_{an} \) be an anti-monotone constraint. Step 5 of Apriori is replaced by \( L_k := SAT_{C_{an}}(C_k) \).
    - it is still correct and complete.
  - Apriori can be used to mine constrained itemsets when the given constraint is anti-monotone.

- **What about monotone constraints?**
Monotone Constraints

- Example
  - Assume \( C(S) \equiv C \in S \). Itemset ABC generated by \( \text{generate}_{\text{apriori}} \) from AB and AC but since \( C(AB) = \text{false} \), ACB is not generated whereas \( C(ABC) = \text{true} \).
  - Assume \( C(S) \equiv A \in S \). Itemset ABC is correctly generated by \( \text{generate}_{\text{apriori}} \) from AB and AC but since \( C(AB) = \text{false} \), ACB is incorrectly pruned whereas \( C(ABC) = \text{true} \).

The generation step and pruning step need to be modified in order to include monotone constraints.

Monotone Constraints

- Some definition in modified generation procedure
  - Negative border: If \( C_{am} \) denotes an anti-monotone constraint, \( B_{dm} \) is the collection of the minimal itemsets that verify \( C_{am} \).
  - \( C_{m} \) denotes the negation of \( C_{am} \).

Monotone Constraints

- Generation procedure
  - \( \text{generate}_{k}(L_k) = \{A\}B \) where \( A \in L_k \) and \( B \) is a \( k \)-itemset.
  - \( \text{generate}_{k}(L_k) = \{A\}B \) where \( A, B \in L_k \).
  - Assume \( C = C_{m}A \). \( C_{am} \) and \( m_{am} = \text{Max}_{s \in S} |S| \).
  - \( \text{generate}_{k}(L_k) = B_{dm} \) for items.

  \[
  \text{For } k \geq 1, \\
  \begin{align*}
  &\quad \text{if } k < m_{am}, \text{generate}_{k}(L_k) = \text{generate}_{k-1}(L_k). \\
  &\quad \text{if } k = m_{am}, \text{generate}_{k}(L_k) = \text{generate}_{k}(L_k). \\
  &\quad \text{if } k > m_{am}, \text{generate}_{k}(L_k) = \text{generate}_{k}(L_k).
  \end{align*}
  \]

  This generation procedure is complete and ensures that every candidate itemset verifies \( C_{am} \).

  We do not need to verify the monotone constraint after this generation procedure.

Monotone Constraints

- Pruning procedure \( \text{prune}_{m} \)
  - For all \( s \in C_{k+1}^e \) and for all \( S' \subseteq S \) such that \( |S'| = k \), do if \( S' \notin L_k \) and \( C_{am}(S') = \text{true} \), then delete \( S \) from \( C_{k+1}^e \).

  \( \text{prune}_{m} \) is correct and complete.

  The algorithm is correct because it does not prune any itemset that verify \( C = C_{am}A \). Its completeness means that if an itemset is not pruned then every proper subset of that itemset verify \( C_{am} \).
**Generic Algorithm**

- For a constraint \( c = C_m \neq C_m \), the generic algorithm uses the structure of Apriori and the procedures \( \text{generate}_c \) and \( \text{pruning}_c \):
  1. \( C_i^l = \text{Items}_{L_i} \cup \text{Items}_{L_m} \) \( \{0\} \)
  2. \( k := 1 \)
  3. while \( C_i^l \neq \emptyset \) do
  4. Phase 1 – candidate safe pruning
     \( C_i^l = \text{pruning}_c(C_i^l, L_i) \)
  5. Phase 2 – anti-monotone constraint checking
     \( L_i = S{\text{safe}}(C_i^l) \)
  6. Phase 3 – candidate generation for level \( k+1 \)
     \( C_i^{l+1} = \text{generate}_c(L_i) \)
  7. \( k := k+1 \)
  8. Output \( \cup_{i=1}^{k} L_i \)

**Apriori Algorithm**

- Constraints:
  - \( C_{am} \equiv A \in S \), \( C_{am} \equiv B \in S \)
- \( \text{Bd}_{\text{am}} = \{B, AB\} \)
- \( ms = \max_{x \in \text{Bd}_{\text{am}}} |S| = 2 \)

**CLOSE algorithm**

- The closure of an itemset \( S(\text{closure}(S)) \) is the maximal superset of \( S \) which has the same support as \( S \).
- A closed itemset is an itemset that is equal to its closure.
- The set of closed itemsets is a lattice called the closed itemset lattice.

---

**Content**

- Introduction
- Constrained itemset mining
  - Apriori revisit
  - Anti-monotone constrains
  - Monotone constrains
  - Generic algorithm
- Frequent closed itemset mining
  - CLOSE algorithm
  - Incorporating constraints into Apriori
- Conclusion
CLOSE algorithm

- We can consider CLOSE as an exploration of the classical itemset lattice with a new constraint.
- A constraint for CLOSE: 
  \[ C'_{\text{Free}}(S) \equiv S' \subseteq S \Rightarrow S \notin \text{closure}(S') \]
- Free itemsets: itemsets that are not included in any closure of their proper sub-set. Equivalently, free itemsets are itemsets that verify \( C'_{\text{Free}} \).

Example

- \( \text{Closure}(AB) \): items \( A \) and \( B \) are simultaneously in transactions \( 1, 4, 6 \). Item \( C \) is the only other item that is also present in these three transactions, thus \( \text{closure}(AB) = ABC \).
- \( \text{Closure}(A) = AC \), \( \text{Closure}(B) = BC \), and \( AB \notin \text{closure}(A) \) and \( AB \notin \text{closure}(B) \). Therefore \( C'_{\text{Free}}(AB) \) is true.
- If frequency threshold \( r = \frac{1}{2} \), \( \text{SAT}_{C_{\text{Free}}}(AC) = \{ {\emptyset, A, B, D^c, AB} \} \) where \( AB^c \) means that \( C'_{\text{Free}}(AB) = \text{true}, \text{closure}(AB) = ABC \).

CLOSE algorithm

- The \( C'_{\text{Free}} \) constraint is anti-monotone, it needs a database pass to be checked.
- Checking this constraint seems expensive if the closure of every subset of \( S \) has to be computed.

Incorporating constraints into Apriori

- Directly using \( C = C_{\text{Free}} \land C_{\text{am}} \land C_{\text{m}} \) causes two problems:
  - The closures of some candidates of level \( k \) are not computed \( \Rightarrow \) impossible to check \( C_{\text{Free}} \) at level \( k+1 \)
  - \( \text{SAT}_{C_{\text{Free}} \land C_{\text{am}} \land C_{\text{m}}} \) will no longer enables to compute \( \text{SAT}_{C_{\text{am}} \land C_{\text{m}}} \).
Incorporating constraints

- Assume we replace
  - $C'_{\text{Free}}$ with $C'_{\text{Free}AC_w}(S) = (S' \subset S \land C_p(S')) \Rightarrow S \subseteq \text{closure}(S')$
  - $C_{\text{Free}}$ with $C_{\text{Free}AC_w}(S) = (S' \subset S \land |S'| = |S| - |AC_w(S')|) \Rightarrow S \subseteq \text{closure}(S')$
- Then: the constraints $C'_{\text{Free}AC_w}$ and $C'_{\text{Free}AC_m}$ are equivalent and anti-monotone. The set $\text{SAT}_{\text{Free}AC_w}$ can be efficiently computed using the same method as in CLOSE using $\text{SAT}_{\text{Free}AC_m\text{Free}AC_w}$ i.e., the output of the generic algorithm with the constraint $C = C_{\text{Free}AC_w} \land AC_m \land AC_w$

Now we can find free-itemsets that verify conjunctions of anti-monotone and monotone constraints 😊

Conclusion

- Frequent itemset mining can be intractable for a given support threshold and a particular database
- Two issues to address this problem: constraint-based itemset mining and condensed representation of frequent itemsets
- The generic algorithm can be used to achieve constrained free-set mining when $C = C_{\text{Free}AC_m}$

Content

- Introduction
- Constrained itemset mining
  - Apriori revisit
  - Anti-monotone constrains
  - Monotone constrains
  - Generic algorithm
- Frequent closed itemset mining
  - CLOSE algorithm
  - Incorporating constraints into Apriori
- Conclusion