Mining Sequential Patterns

Authors: Rakesh Agrawal and Ramakrishnan Srikant

Presenter: Yunping Wang

Outline

- Background
- Introduction
- Problem Decomposition and Solution
- Algorithm
- Performance
- Discussions and Conclusion
- Correlation Literature

Background

- Sequential Pattern Mining was first introduced in 1995
- Sequential Pattern are ordered list of itemsets
- Sequential Pattern Mining Applications: shopping history, weblog mining, DNA sequence modeling, disease treatment, natural disasters, etc.

Introduction

- What is Sequential Pattern Mining?
- Key components of Sequential Pattern Mining
- Sequential Pattern Example
- Sequence Database Example

Introduction

- What is Sequential Pattern Mining?

Definition:

Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min support threshold, sequential pattern mining is to find all of the frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min support.

Introduction---Definition

- Itemset i, $(i_1\ i_2 \ldots i_m)$ where i_j is an item.
- Sequence s, $\langle s_1 s_2 \ldots s_n \rangle$ where s_j is an itemset.
- Sequence $\langle a_1 a_2 \ldots a_n \rangle$ contained in $\langle b_1 b_2 \ldots b_n \rangle$ if there exist integers $i_1 < i_2 \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$,…, $a_n \subseteq b_{in}$.
- A sequence s is maximal if it is not contained in any other sequence.

Introduction---Definition

- Support of a sequence - % of customers who support the sequence.
  - For mining association rules, support was % of transactions.
- Sequences that have support above minsup are *large* sequences.

Introduction

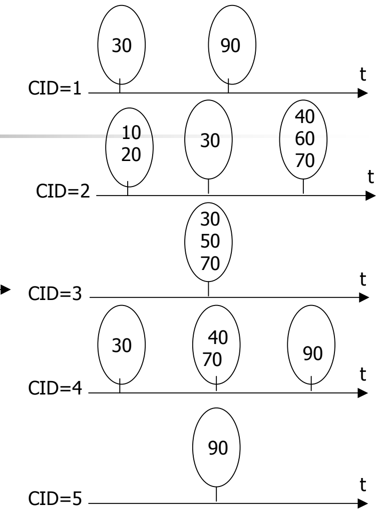
- Key Components of Sequential pattern mining:
  - Frequent time-ordered sequential patterns in the database.
  - Two conditions: Min Support and Maximal Sequence
  - Association rule --- intra-transaction;
  - Sequential rule --- inter-transaction

Sequence Pattern Examples

- Examples 1
  - 60% of customers typically rent "star wars", then "Empire strikes back", and then "Return of Jedi".
  - Note: these rentals need not to be consecutive.
- Example 2
  - 60% of customers buy "Fitted Sheet and flat sheet and pillow", followed by "comforter", followed by "drapes and ruffles"
  - Note: elements of a sequential pattern need not to be simple items.

Sequence Database

Customer ID	TransactionTime	Items
1	1	30
1	2	90
2	1	10,20
2	2	30
2	3	40,60,70
3	1	30,50,70
4	1	30
4	2	40,70
4	3	90
5	1	90



MinSupport =40%, i.e. 2 customers

Answer: (<30><90>) (CID1,4) (<30><40,70>) (CID2,4)

Not Answer: <30> <40><70><90>(<30><40>)(<30><70>)(<40 70>) Why?

Solution--- Sort Phases(1)

- Customer ID – Major key
- Transaction-time – Minor key

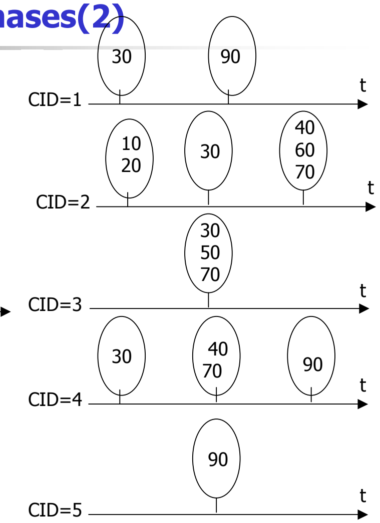
Converts the original transaction database into a database of customer sequences.

Solution--- Sort Phases(2)

- **Sort Phases**

  CID: major key, TID: secondary key

Customer ID	TransactionTime	Items
1	1	30
1	2	90
2	1	10,20
2	2	30
2	3	40,60,70
3	1	30,50,70
4	1	30
4	2	40,70
4	3	90
5	1	90



Solution--- Litemset Phase(1)

Litemset Phase:

- Find all large itemsets.

Why?

- Because each itemset in a large sequence has to be a large itemset.

Solution--- Litemset Phase(2)

- To get all large itemsets we can use the Apriori algorithms.
- Need to modify support counting.
  - For sequential patterns, support is measured by fraction of customers.

Solution--- Litemset Phase(3)

- **Litemset Phase:**
  - Example: find large itemset

Customer ID	TransactionTime	Items
1	1	30
1	2	90
2	1	10,20
2	2	30
2	3	40,60,70
3	1	30,50,70
4	1	30
4	2	40,70
4	3	90
5	1	90

Litemset Result:

{30} {40} {70} {40 70}{90}

Difference from Apriori:

- the support count should be incremented only once per customer

Solution --- Transform Phase(1)

- Each large itemset is then mapped to a set of contiguous integers.

Why?

Used to compare two large itemsets in constant time.

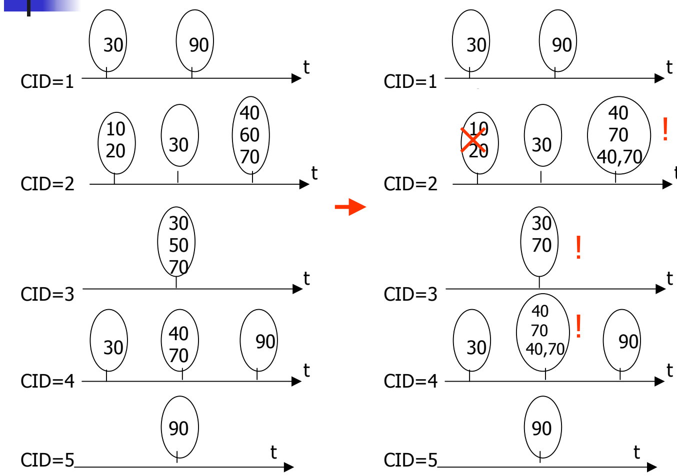
itemset	Map
{30}	1
{40}	2
{70}	3
{40 70}	4
{90}	5

Litemsets

Solution --- Transform Phase(2)

- Need to repeatedly determine which of a given set of large sequences are contained in a customer sequence.
- Represent transactions as sets of large itemsets.
- Customer sequence now becomes a list of sets of itemsets.

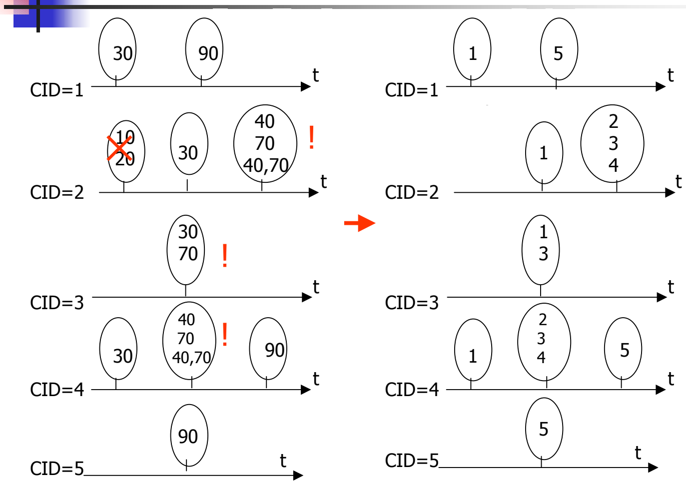
Solution --- Transform Phase(3)



itemset	Map
{30}	1
{40}	2
{70}	3
{40 70}	4
{90}	5

Litemsets

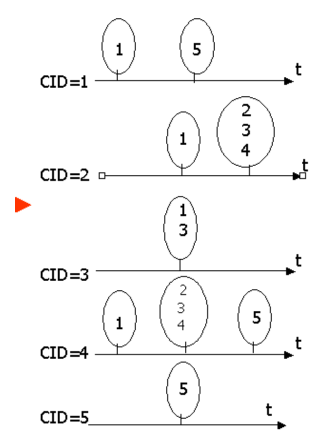
Solution --- Transform Phase (4)



itemset	Map
{30}	1
{40}	2
{70}	3
{40 70}	4
{90}	5

Litemsets

Solution --- Transform Phase (5)



- Transform Database :

<{1} {5}>

<{1}{2 3 4}>

<{1 3}>

<{1} {2 3 4} {5}>

<{5}>

Solution--- Sequence Phase (1)

- Use set of large itemsets to find the desired sequences.
- Similar structure to Apriori algorithms used to find large itemsets.
  - Use seed set to generate candidate sequences.
  - Count support for each candidate.
  - Eliminate candidate sequences which are not large.

Solution--- Sequence Phase (2)

Two types of algorithms:

- Count-all: counts all large sequences, including non-maximal sequences.
  - AprioriAll
- Count-some: try to avoid counting non-maximal sequences by counting longer sequences first.
  - AprioriSome
  - DynamicSome

Solution -- Maximal phase (1)

- Find the maximal sequences among the set of large sequences

  delete all sub-sequences in larger Sequence

```
for (k=n; k>1; k--) do
  for each k-sequence Sk do
    Delete from  all subsequences of Sk
```


Solution -- Maximal phase(2)

- **Maximal phase example:**

  The large sequence is <1 2 3 4>, the sub-sequence <1 2 3><1 2 4> <1 3 4> <1 3 5><2 3 4> need to be deleted from final result.

Large 3-Sequence	Candidate 4-sequences (after join)	Candiate 4-Sequence (after Pruning)
<1 2 3>	<1 2 3 4>	<1 2 3 4>
<1 2 4>	<1 2 4 3>	
<1 3 4>	<1 3 4 5>	
<1 3 5>	<1 3 5 4>	
<2 3 4>		

Algorithm

- AprioriAll
- AprioriSome
- DynamicSome

Algorithm ---AprioriAll Algorithm(1)

- **AprioriAll Algorithm**

  C_k: Candidate sequence of size k
  L_k : frequent or large sequence of size k

L_1 = {large 1-sequence}; //result of litemset phase
for (k = 2; L_{k-1} !=∅; k++) **do begin**
  C_k = candidates generated from L_{k-1};
  for each customer-sequence c in database do
    Increment the count of all candidates in C_k
    that are contained in c
  L_k =Candidates in C_k with minimum support
end
Answer=Maximal sequences in $\cup_k L_k$;

Algorithm ---AprioriAll Algorithm(2)

Highlight:

- Candidate generation similar to candidate generation in finding large itemsets.
- The order matters !

Algorithm ---AprioriAll Algorithm(3)

- **Candidate Generation --Join Step**:

  C_k is generated by joining L_{k-1} with itself

Insert into C_k,
Select p.litemset$_1$, …, p.litemset$_{k-1}$, q.litemset$_{k-1}$
From L_{k-1} p, L_{k-1} q
Where p.litemset$_1$= q.litemset$_1$,…,
  p.litemset$_{k-2}$= q.litemset$_{k-2}$

For example: {1,2,3} X {1,2,4} = {1,2,3,4} and {1,2,4,3}

Algorithm ---AprioriAll Algorithm(4)

- **Candidate Generation –Prune Step:**

  Any (k-1)-subsequences of s (length k) that is not frequent cannot be a subsequence of a frequent k-sequence.

Algorithm ---AprioriAll Algorithm(5)

- **Candidate Generation example:**

Large 3-Sequence	Candidate 4-sequences (after join)	Candiate 4-Sequence (after Pruning)
<1 2 3> <1 2 4> <1 3 4> <1 3 5> <2 3 4>	<1 2 3 4> <1 2 4 3> <1 3 4 5> <1 3 5 4>	<1 2 3 4>

Algorithm ---AprioriAll Algorithm(6)

- **AprioriAll Algorithm example:**

Customer Sequence:

<{1 5} {2} {3} {4}>
<{1} {3} {4} {3 5}>
<{1} {2} {3} {4}>
<{1} {3} {5}>
<{4} {5}>

The maximal large sequences with minSupp=40% is:
<1 2 3 4> <1 3 5> <4 5>

Algorithm

- Count-some Algorithms
- AprioriSome, DynamicSome
- Try to avoid counting non-maximal sequences by counting longer sequences first.
- 2 phases:
  - Forward Phase – find all large sequences of certain lengths.
  - Backward Phase – find all remaining large sequences.

Algorithm---AprioriSome(1)

- Determines which lengths to count using next() function.
- next() takes in as a parameter the length of the sequence counted in the last pass.
- next(k) = k + 1 - Same as AprioriAll
- Balances tradeoff between:
  - Counting non-maximal sequences
  - Counting extensions of small candidate sequences

Algorithm---AprioriSome(2)

```
function next(k: integer)
begin
    if (hit_k < 0.666)    return k + 1;
    elsif (hit_k < 0.75) return k + 2;
    elsif (hit_k < 0.80) return k + 3;
    elsif (hit_k < 0.85) return k + 4;
    else                 return k + 5;
end
```

- $\text{hit}_k = |L_k| / |C_k|$
- Intuition: As hit_k increases the time wasted by counting extensions of small candidates

Algorithm---AprioriSome(3)

Backward Phase:

- For all lengths which we skipped:
  - Delete sequences in candidate set which are contained in some large sequence.
  - Count remaining candidates and find all sequences with min. support.
- Also delete large sequences found in forward phase which are non-maximal.

Algorithm---AprioriSome(4)

C_k	L_k	
C_1	L_1	f(k)=2k
C_2	L_2	
C_3	L_3	
C_4	L_4	
C_5	---	

Algorithm---DynamicSome(4)

- Divided into 4 phase: initialization, forward, intermediate & backward phase.
- Use the variable *step* to decide how to jump.
- Use otf-generate function to generate candidate sequence.

Performance

Testing Setting:

- Performed experiments on a IBM RS/6000 530H workstation with CPU clock rate of 33MHZ, 64MB of main memory, and running AIX 3.2

- Number of maximal potentially large Sequence: 5000
- Number of maximal potentially large Itemsets: 25,000
- Number of Itemsets :10,000

Performance

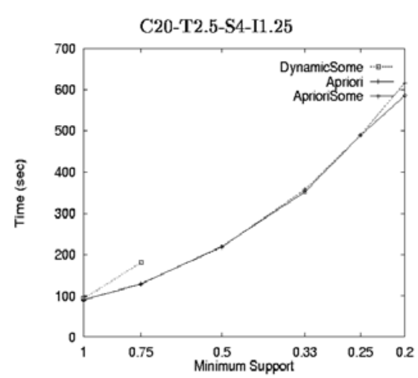
Testing Setting:

- |C|: Average number of transactions per customer
- |T|: Average number of items per Transaction
- |S|: Average length of maximumal potentially large Sequence
- |I|: Average size of Itemsets in maximal potentiallly large sequences

Name	$\|C\|$	$\|T\|$	$\|S\|$	$\|I\|$	Size (MB)
C10-T5-S4-I1.25	10	5	4	1.25	5.8
C10-T5-S4-I2.5	10	5	4	2.5	6.0
C20-T2.5-S4-I1.25	20	2.5	4	1.25	6.9
C20-T2.5-S8-I1.25	20	2.5	8	1.25	7.8

Parameter settings (Synthetic datasets)

Performance



- DynamicSome generates too many candidates.
- AprioriSome does a little better than AprioriAll.
  - It avoids counting many non-maximal sequences.

Performance

Advantage of AprioriSome is reduced for 2 reasons:

- DynamicSome generates more candidates.
- Candidates remain memory resident even if skipped over.
  - Cannot always follow heuristic.

Conclusion

- Pos:

- Described a new problem Sequential Pattern Mining
- Provided a solution --decomposed the problem into 5 steps to solve it
- In Sequence phase, three algorithm were introduced. *AprioriAll*, *AprioriSome*, and *DynamicSome*
- *AprioriALL* is the basis of many efficient algorithm developed later

Conclusion

- Cons:
- Algorithm limitation:

The solution is not memory efficient, it need to create transform database which need more disk space.

Correlation Literature

- R. Agrwal & R. Srikant, "*Mining Sequential Patterns:Generalizations and Performance Improvements* "1996

- The limitations of AprioriAll:
  - Absence of time constraints
  - Rigid definition of a transaction

Thank You !

Question?