Mining Sequential Patterns

Work by Rakesh Agrawal and Ramakrishnan Srikant
Proc. of the Int'l Conference on Data Engineering (ICDE),1995

Problem Introduction

- Bar code data allows us to store massive amounts of sales data (basket data)
- We would like to extract sequential buying patterns.
  - E.g. Video rental: Star Wars; Empire Strikes Back; Return of the Jedi
- One element of the sequence can include multiple items.
  - E.g. Sheets and pillow cases; comforter; drapes and ruffles.

Road Map

1. Basic Concepts
2. Finding Sequential Patterns
3. Summary

Terminology

- Association Rules refer to what items are bought together (at the same time)
  - Intra-transaction patterns
- Sequential Patterns refer to what items are bought at different times
  - Inter-transaction patterns

Problem Description

- Given a database of transactions:
- Each transaction:
  - *Customer ID*
  - *Transaction Time*
  - *Set of items purchased*

* No two transactions have same customer ID and transaction time

* Do not consider quantity of items purchased. Item was either purchased or not purchased

Problem Statement

- *Itemset*: non-empty set of items. Each itemset is mapped to an integer
- *Sequence*: ordered list of itemsets
- *Customer Sequence*: list of customer transactions ordered by increasing transaction time
  - A customer supports a sequence if the sequence is contained in the customer-sequence
- *Support for a Sequence*: fraction of total customers that support a sequence
- *Maximal Sequence*: a sequence that is not contained in any other sequence
- *Large Sequence*: sequence that meets minisup

Example

Customer ID	Transaction Time	Items Bought
1	June 25 ’17	30
1	June 30 ’17	90
2	June 10 ’17	10,20
2	June 15 ’17	30
2	June 20 ’17	40,60,70
3	June 25 ’17	30,50,70
4	June 25 ’17	30
4	June 30 ’17	40,70
4	July 25 ’17	90
5	June 12 ’17	90

Customer ID	Customer Sequence
1	< (30) (90) >
2	< (10 20) (30) (40 60 70) >
3	< (30 50 70) >
4	< (30) (40 70) (90) >
5	< (90) >

Maximal seq with support > 40%
< (30) (90) >
< (30) (40 70) >

Note: Use Minisup of 40%, no less than two customers must support the sequence
< (10 20) (30) > Does not have enough support (Only by Customer #2)
< (30) >, < (70) >, < (30) (40) > … are not maximal.

Road Map

1. Basic Concepts

2. Finding Sequential Patterns

3. Summary

The Algorithm

- Sort Phase
- Litemset Phase
- Transformation Phase
- Sequence Phase
- Maximal Phase

Sort Phase

- Sort the database:
  - *Customer ID* as the major key
  - *Transaction-Time* as the minor key
- Convert the original transaction DB into a customer sequence DB

Customer ID	Transaction Time	Items Bought
1	June 25 '17	30
1	June 30 '17	90
2	June 10 '17	10,20
2	June 15 '17	30
2	June 20 '17	40,60,70
3	June 25 '17	30,50,70
4	June 25 '17	30
4	June 30 '17	40,70
4	July 25 '17	90
5	June 12 '17	90

Litemset Phase

- Litemset (Large Itemset):
  - Supported by *fraction of customers* greater than minisup

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

- Reasons of mapping (treating litemsets as single entities)?
  - Compare two litemsets in constant time
  - Reduce the time to check if a sequence is contained in a customer sequence

Customer ID	Transaction Time	Items Bought
1	June 25 '17	30
1	June 30 '17	90
2	June 10 '17	10,20
2	June 15 '17	30
2	June 20 '17	40,60,70
3	June 25 '17	30,50,70
4	June 25 '17	30
4	June 30 '17	40,70
4	July 25 '17	90
5	June 12 '17	90

Transformation Phase

- Replace each transaction with all litemsets contained in the transaction.
- Transactions with no litemsets are dropped. (But empty customer sequences still contribute to the total count)

Cust ID	Original Cust Sequence	Transformed Customer Sequence	After Mapping
1	< (30) (90) >	<{(30)} {(90)}>	<{1} {5}>
2	< (10 20) (30) (40 60 70) >	<{(30)} {(40),(70),(40 70)}>	<{1} {2,3,4}>
3	< (30) (50) (70) >	<{(30),(70)}>	<{1,3}>
4	< (30) (40 70) (90) >	<{(30)} {(40),(70),(40 70)} {(90)}>	<{1} {2,3,4} {5}>
5	< (90) >	<{(90)}>	<{5}>

- **Note**: (10 20) dropped because of lack of support.
  (40 60 70) replaced with set of litemsets {(40),(70),(40 70)}
  (60) does not have minisup

Maximal Phase

- Find maximal sequences among large sequences
  - *k-sequence*: sequence of length k
  - *S*: the set of all large sequences

```
for (k=n; k>1; k--) do

    for each k-sequence s_k do

        Delete from S all subsequences of s_k
```

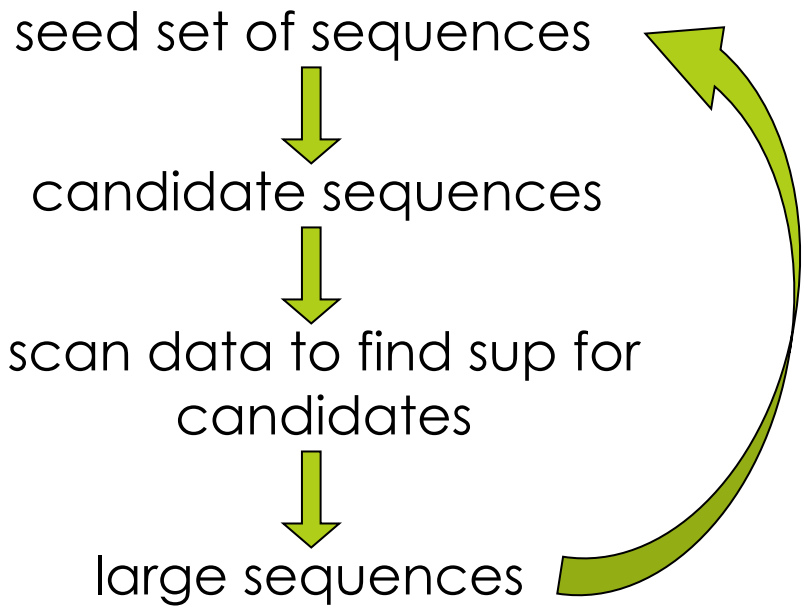
- Authors claim data-structures and an algorithm exist to do this efficiently. (*hash trees*)

Sequence Phase

- Finds Large Sequences
  - AprioriAll
  - AprioriSome
  - DynamicSome

Sequence Phase Overview

- Sequence phase finds the large sequences



AprioriAll

- Based on the normal Apriori algorithm
- Counts all the large sequences
- Prunes non-maximal sequences in the "Maximal phase"

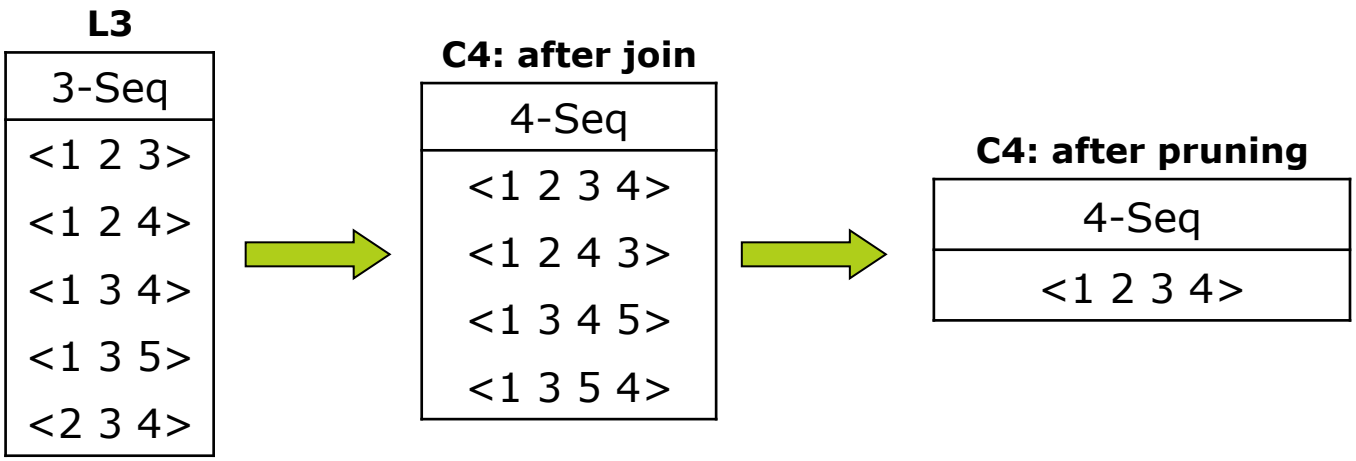
AprioriAll

L_1 = {large 1-sequences}
for (k = 2; $L_{k-1} \neq \{\}$; k++) **do**
    begin
    C_k = New candidates generated from L_{k-1}
    foreach customer-sequence c in the 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 L_k

Notation:
L_k: Set of all large k-sequences
C_k: Set of candidate k-sequences

AprioriAll(Example)

- Candidate generation
  - Join L_{k-1} with itself to form C_k
  - Delete c in C_k such that some (k-1)-sub sequence of c is not in L_{k-1}



AprioriAll(Example)

Minisup = 40%

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

L1

1-Seq	Sup
<1>	4
<2>	2
<3>	4
<4>	4
<5>	4

L2

2-Seq	Sup
<1 2>	2
<1 3>	4
<1 4>	3
<1 5>	3
<2 3>	2
<2 4>	2
<3 4>	3
<3 5>	2
<4 5>	2
~~<2 5>~~	~~0~~

L3

3-Seq	Sup
<1 2 3>	2
<1 2 4>	2
<1 3 4>	3
<1 3 5>	2
<2 3 4>	2
~~<3 4 5>~~	~~1~~

L4

4-Seq	Sup
<1 2 3 4>	2

Answer: <1 2 3 4>, <1 3 5>, <4 5>

AprioriSome

- Avoid counting sequences that are contained in longer sequences by counting the longer ones first. Also avoid having to count many subsequences because their super-sequences are not large
  - Forward phase: find all large sequences of certain lengths
  - Backward phase: find all remaining large sequences

AprioriSome

- Function "***next***" determines the next sequence length which is counted: this is based on the assumption that if, e.g, almost all sequences of length k are large (frequent), then many of the sequences of length k+1 are also large (frequent)
  - Most of the sequences are large (>85%) => next round is k+5
  - ...
  - Not many of the sequences are large (<67%) => next round is k+1 (AprioriAll)

AprioriSome(ForwardPhase)

```
L1 = {large 1-sequences}
C1 = L1
last = 1
for (k = 2; Ck-1 ≠ {} and Llast ≠ {}; k++) do
   begin
   if (Lk-1 known) then
      Ck = New candidates generated from Lk-1
   else
      Ck = New candidates generated from Ck-1
   if (k==next(last)) then begin // (next k to count?)
      foreach customer-sequence c in the database do
         Increment the count of all candidates in Ck
            that are contained in c.
      Lk = Candidates in Ck with minimum support.
      last = k;
      end
   end
```

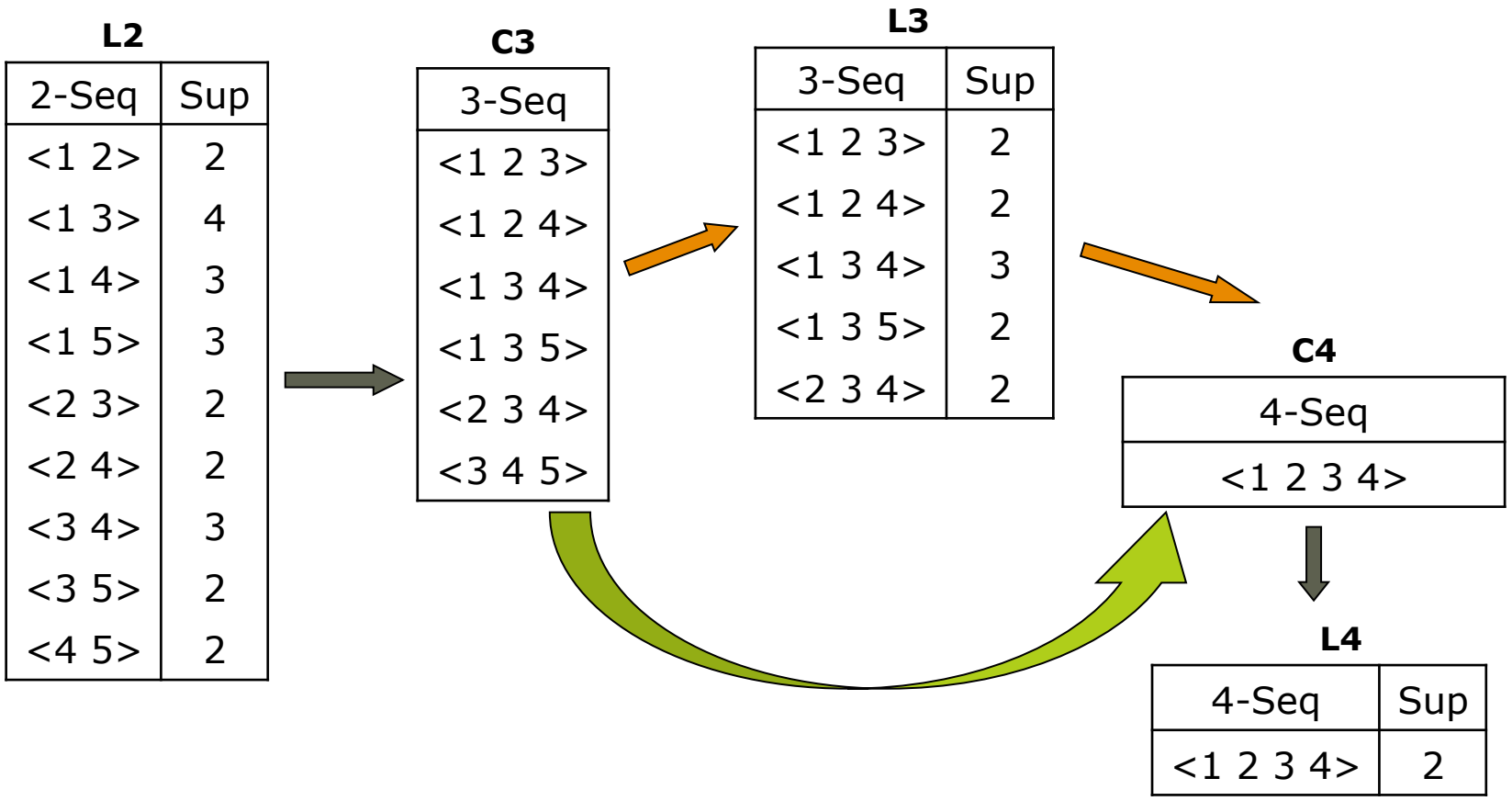
AprioriSome(BackwardPhase)

```
for (k=n; k>=1; k--) do
   if (L_k not found in forward phase) then begin
      Delete all sequences in C_k contained in
         some L_i i>k;
      foreach customer-sequence c in D_T 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
   else // l_k already known
      Delete all sequences in L_k contained in
         some L_i i>k;

Answer = L_k //(Maximal Phase not Needed)
```

*Notation: D_T; Transformed database

AprioriSome (Example)



DynamicSome

- Similar to AprioriSome
- AprioriSome generates C_k from C_{k-1}
- DynamicSome generates C_k "*on the fly*"
  - based on large sequences found from the previous passes and the customer sequences read from the database

DynamicSome

- In the **initialization phase**, count only sequences up to and including ***step*** variable length
  - If *step* is 3, count sequences of length 1, 2 and 3
- In the **forward phase**, generate sequences of length step, 2 × *step*, 3 × *step*, 4 × *step*, etc. on-the-fly based on previous passes and customer sequences in the database
  - While generating sequences of length 9 with a step size 3: While passing the data, if sequences $s_6 \in L_6$ and $s_3 \in L_3$ are both contained in the customer sequence c in hand, and they do not overlap in c, then $\langle s_3 . s_6 \rangle$ is a candidate (3+6)-sequence

DynamicSome

- In the **intermediate phase**, generate the candidate sequences for the skipped lengths
  - If we have counted L_6 and L_3 , and L_9 turns out to be empty: generate C_7 and C_8 , count C_8 followed by C_7 after deleting non-maximal sequences, and repeat the process for C_4 and C_5
- The **backward phase** is identical to AprioriSome

Road Map

1. Basic Concepts

2. Finding Sequential Patterns

3. Summary

Question1

- Compare and contrast association rules and sequential patterns. How do they relate to each other in the context of the Apriori algorithms?

- ✓ Association rules refer to intra-transaction patterns, while sequential patterns refer to inter-transaction patterns. Both of these are used in the Apriori algorithms studied here, because the algorithms are looking for different sequential patterns made up of association rules.

Question2

- There were two types of algorithms used in this paper to mine sequential patterns, AprioriSome and AprioriAll. What is the major difference between the two algorithms?

✓ *AprioriAll* is careful with respect to minimum support, and careless with respect to maximality. (The minimum support is checked for each sequence on each run, but maximal sequences must be checked later.)

AprioriSome is careful with respect to maximality, but careless with respect to minimum support. (Non-maximal sequences are pruned out during runtime, but the minimum support is not tested at all values of k.)

Question3

- Why is the Transformation stage of these pattern mining algorithms so important to their speed?

✓ The transformation allows each record to be looked up in constant time, reducing the run time.