Data Mining -- Sequential Pattern

Instructor: Jen-Wei Huang

Office: 92528 in the EE building
jwhuang@mail.ncku

Example

SID	Sequences
100	<(1,5) (2) (3) (4)>
200	<(1) (3) (4) (3,5)>
300	<(1) (2) (3) (4)>
400	<(1) (3) (5)>
500	<(4) (5)>

min_support = 2

- Some frequent sequential patterns
  - {1,2,3,4}, {1,3,5}, {4,5}

Sequential Pattern

- "Mining of frequently occurring patterns related to time or other sequences."
  - J. Han, *Data Mining – Concepts and Techniques*
- "Given a set of sequences, find the complete set of frequent subsequences"
  - J. Pei, *PrefixSpan*
- Ex) What items one will buy if he/she has bought some certain items

Time-related data

- Customers' buying behavior
- Stock price changes
- Natural phenomena
- Sensor network data
- Web access patterns
- DNA sequence applications

Definition

- Let $I = \{x_1, x_2, ..., x_n\}$ be a set of different items.
- An element e, denoted by $(x_i\ x_j\ ...)$, is a subset of items $\subseteq I$ of which items appear in a sequence at the same time.
- A sequence s, denoted by $< e_1, e_2, ..., e_m >$, is an ordered list of elements.
- A sequence database Db contains a set of sequences and $|Db|$ represents the number of sequences in Db.

Definition

- A sequence $\alpha = < a_1, a_2, ..., a_n >$ is a subsequence of another sequence $\beta = < b_1, b_2, ..., b_m >$ if there exists a set of integers, $1 \leq i_1 < i_2 < ... < i_n \leq m$, such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, ..., and $a_n \subseteq b_{in}$.
- The sequential pattern mining can be defined as "Given a sequence database, Db, and a user-defined minimum support, min_sup, find the complete set of subsequences whose occurrence frequencies $\geq min\_sup * |Db|$."

How?

- Apriori-like algorithms
  - AprioriAll – by Agrawal *et al*
  - GSP – by Srikant *et al*
- Vertical format algorithms
  - SPADE – by Zaki *et al*
  - SPAM – by Ayres *et al*
- Partition-based algorithms
  - FreeSpan – by Han *et al*
  - PrefixSpan – by Pei *et al*

Apriori-like Algorithms

- 1. Sort phase
  - Sort the database
  - Customer id as the primary key and time as the second key
- 2. Litemset phase
  - Count the frequency of the itemset
  - The fraction of customers who bought the itemset

Apriori-like Algorithms

- 3. Transformation phase
  - Transform each tx to all litemsets in the form of

    C01: <(1,5) (2) (3) (4)>
    C02: <(1) (3) (4) (3,5)>
    C03: <(1) (2) (3) (4)>
    C04: <(1) (3) (5)>
    C05: <(4) (5)>

CID	Items
2	10 20
5	90
2	30
2	40 60 70
4	30
3	30 50 70
1	30
1	90
4	40 70
4	90
3	10
5	10
1	40 70
5	20
2	90
3	20

CID	Items
1	30 90 {40 70}
2	{10 20} 30 {40 60 70} 90
3	{30 50 70} 10 20
4	30 {40 70} 90
5	90 10 20

Itemset	#
10	3
20	3
30	4
40	3
50	1
60	1
70	4
90	4
{10 20}	1
{40 60}	1
{40 70}	3
{60 70}	1
{40 60 70}	1
{30 50}	1
{30 70}	1
{50 70}	1
{30 50 70}	1

Itemset	#	New
10	3	1
20	3	2
30	4	3
40	3	4
70	4	5
90	4	6
{40 70}	3	7

CID	Items
1	30 90 {40 70}
2	{10 20} 30 {40 60 70} 90
3	{30 50 70} 10 20
4	30 {40 70} 90
5	90 10 20

CID	Items
1	3 6 {4, 5, 7}
2	{1, 2} 3 {4, 5, 7} 6
3	{3, 5} 1 2
4	3 {4, 5, 7} 6
5	6 1 2

Apriori-like Algorithms

- 4. Mining phase
  - Apriori-like algorithm
- 5. Maximal phase
  - Find the maximum patterns

CID	Items
1	3 6 {4, 5, 7}
2	{1, 2} 3 {4, 5, 7} 6
3	{3, 5} 1 2
4	3 {4, 5, 7} 6
5	6 1 2

Itemset	#
1 2	2
1 3	1
1 4	1
1 5	1
1 6	1
1 7	1
2 1	0
2 3	1
2 4	1
2 5	1
2 6	1
2 7	1
3 1	1
3 2	1

Itemset	#
3 4	3
3 5	3
3 6	3
3 7	3
4 1	0
4 2	0
4 3	0
4 5	0
4 6	2
4 7	0
5 1	1
5 2	1
5 3	0
5 4	0

Itemset	#
5 6	2
5 7	0
6 1	1
6 2	1
6 3	0
6 4	1
6 5	1
6 7	1
7 1	0
7 2	0
7 3	0
7 4	0
7 5	0
7 6	2

CID	Items
1	3 6 {4, 5, 7}
2	{1, 2} 3 {4, 5, 7} 6
3	{3, 5} 1 2
4	3 {4, 5, 7} 6
5	6 1 2

Itemset	#
3 4 6	2
3 5 6	2
3 7 6	2

Itemset	#	
10	3	1
20	3	2
30	4	3
40	3	4
70	4	5
90	4	6
{40 70}	3	7

Therefore, frequent sequential patterns are:
<1 2> <3 4> <3 5> <3 6> <3 7> <4 6> <5 6> <7 6>
<3 4 6> <3 5 6> <3 7 6>

According to mappings, original frequent sequential patterns are:
<10 20> <30 40> <30 70> <30 90> <30 {40 70}>
<40 90> <70 90> <{40 70} 90> <30 40 90> <30 70 90>
<30 {40 70} 90>

Maximal Sequential Patterns

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

Customer Sequences

Sequence	Support
<1>	4
<2>	2
<3>	4
<4>	4
<5>	4

Large 1-Sequences

Sequence	Support
<1 2>	2
<1 3>	4
<1 4>	3
<1 5>	2
<2 3>	2
<2 4>	2
<3 4>	3
<3 5>	2
<4 5>	2

Large 2-Sequences

Sequence	Support
<1 2 3>	2
<1 2 4>	2
<1 3 4>	3
<1 3 5>	2
<2 3 4>	2

Sequence	Support
<1 2 3 4>	2

Large 4-Sequences

Sequence	Support
<1 2 3 4>	2
<1 3 5>	2
<4 5>	2

Maximal Large Sequences

Drawbacks

- A huge set of candidate sequences generated.
  - Especially 2-item candidate sequence.
- Multiple Scans of database needed.
  - The length of each candidate grows by one at each database scan.
- Inefficient for mining long sequential patterns.
  - A long pattern grow up from short patterns
  - The number of short patterns is exponential to the length of mined patterns.

Vertical Format Algorithms

- Transform the database into vertical format
- Use vertical list format, for each sequential pattern
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

	<A>	<B>	<(AB)>	<AB>
C01,T01	1	1	1	0
C01,T02	1	0	0	0
C01,T03	0	1	0	1
C02,T01	1	1	1	0
C02,T02	0	1	0	1
C02,T03	1	0	0	0

SID	EID	Items
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b
4	1	e
4	2	g
4	3	af
4	4	c
4	5	b
4	6	c

a		b		...
SID	EID	SID	EID	...
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1	3	5	
2	4	4	5	
3	2			
4	3			

ab			ba			...
SID	EID (a)	EID(b)	SID	EID (b)	EID(a)	...
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				
4	3	5				

aba				...
SID	EID (a)	EID(b)	EID(a)	...
1	1	2	3	
2	1	3	4	

Drawbacks

- Still a huge set of candidate sequences to be examined.
- Inefficient for mining long sequential patterns.
- Large working space is needed.

Partition-based Algorithms

- A sequence $P=\{p_1,p_2,\ldots\ldots,p_k\}$ is a prefix of another sequence $S=\{s_1,s_2,\ldots\ldots,s_n\}$ if for all $i=1\sim k$, $p_i \subseteq s_i$,and $k \leqq n$
- <a>, <aa>, <a(ab)> and <a(abc)> are all prefixes of sequence <a(abc)(ac)d(cf)>

Partition-based Algorithms

- Given a sequence <a(abc)(ac)d(cf)>

Prefix	Projection
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<ab>	<(_c)(ac)d(cf)>

Partition-based Algorithms

- Step 1: find length-1 sequential patterns
  - <a>, <b>, <c>, <d>, <e>, <f>
- Step 2: divide the original database into the projections of each 1-sequential patterns
  - 6 sub-databases:
  - The ones having prefix <a>, <b>, ……,<f>

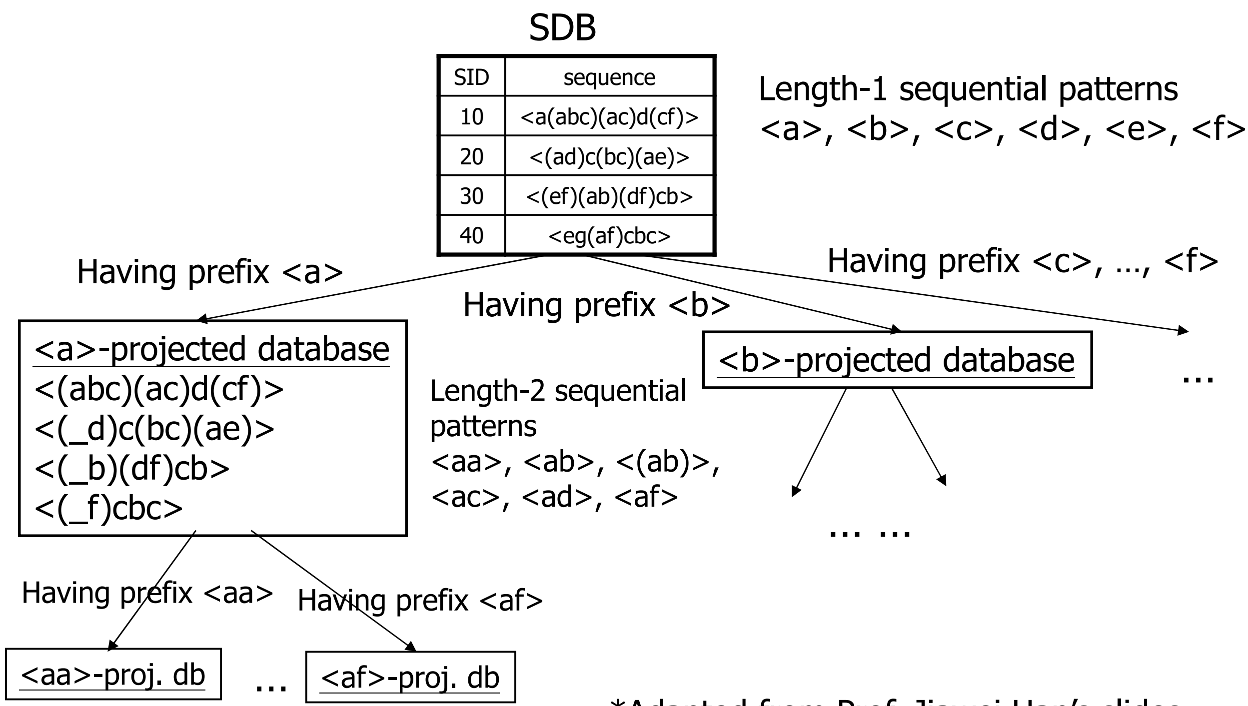
SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Partition-based Algorithms

- For <a>-projected database:
- Find all length-2 seq. pat. having prefix <a>
  - <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
- Then, partition into sub-sub-databases
  - Having prefix
  - <aa>, <ab>,
  - <(ab)>, <ac>,
  - <ad>, <af>

<u><a>-projected database</u>
<(abc)(ac)d(cf)>
<(_d)c(bc)(ae)>
<(_b)(df)cb>
<(_f)cbc>

Partition-based Algorithms



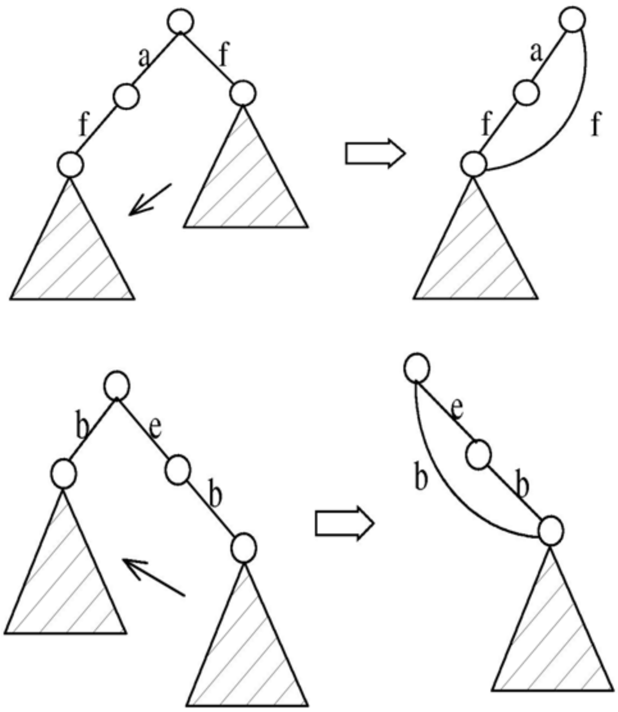
*Adapted from Prof. Jiawei Han's slides

Drawbacks

- Construction of projected databases.
- Large working space is needed.

Closed Sequential Patterns

- A closed sequential pattern *s*: there exists no superpattern *s'* such that $s' \supset s$, and *s'* and *s* have the same support
- Motivation: reduces the number of (redundant) patterns but attains the same expressive power
- Using Backward Subpattern and Backward Superpattern pruning to prune redundant search space



References

- [1] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. EDBT'96.
- [2] M. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning, 2001.
- [3] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. ICDE'01 (TKDE'04).
- [4] X. Yan, J. Han, and R. Afshar. CloSpan: Mining Closed Sequential Patterns in Large Datasets. SDM'03

References

- Slides from Prof. J.-W. Han, UIUC
- Slides from Prof. M.-S. Chen, NTU
- Slides from Prof. W.-Z. Peng, NCTU