

Principles of Knowledge Discovery in Data

Winter 2007

Chapter 3: Sequential Pattern Analysis

Dr. Osmar R. Zaiane



University of Alberta

Course Content



- Introduction to Data Mining
- Association Analysis
- **Sequential Pattern Analysis**
- Classification and prediction
- Contrast Sets
- Data Clustering
- Outlier Detection
- Web Mining
- Other topics if time permits (spatial data, biomedical data, etc.)



Chapter 3 Objectives

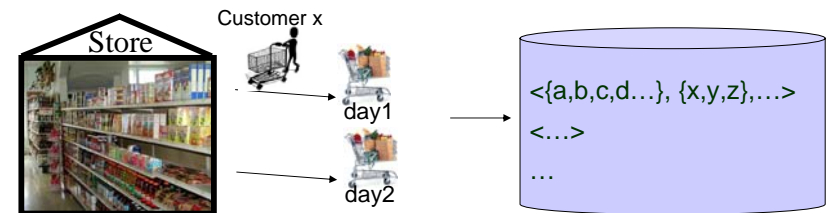
Understand Sequential Pattern Analysis in the context of transactional data and get a brief introduction to the different algorithms for sequential pattern discovery

Sequence of Transactions

- Association rule mining searches for relationships between items in a dataset where time is irrelevant.



- Sequential Pattern Analysis considers time (or order of transactions).



Data: sequences of evidences in time order
Target: sub-sequences that happened frequently

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.
 - <SW>, ..., <ESB>, ..., <RJ>
- 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.
 - <FittedSheet, FaltSheet, Pillow>, ..., <Comforter>, ..., <Drapes, Ruffles>

Lecture Outline

Part I: Concepts (30 minutes)

- Basic concepts

Part II: Apriori-based Approaches (45 minutes)

- Apriori-all
- GSP

Part III: Pattern-Growth-based Approaches (45 minutes)

- Free-Span
- Prefix-Span

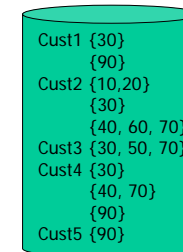
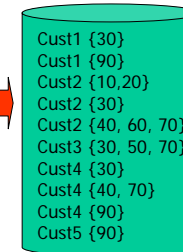
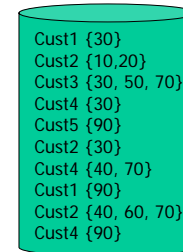
Why sequential pattern mining?

- Time or order in which actions appear or happen can be relevant in decision making.
- Many applications of sequential pattern mining
 - Customer shopping sequences (idem for book/video rental):
 - First buy computer, then CD-ROM, and then digital camera, within 3 months.
 - Medical treatment (e.g., symptoms and diseases)
 - Serial crime solving
 - Natural disasters (e.g., earthquakes),
 - Science & engineering processes,
 - Stocks and markets,
 - Telephone calling patterns,
 - Web access log click streams,
 - DNA sequences and gene structures, etc.

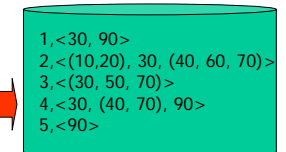
Sequence Database

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

Transactional database



Sequence database



Sort transactions:

• Customer ID = Major key

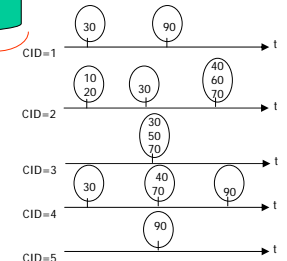
• Transaction Time = Minor key

<(10,20)(30)(40,60,70)>

< a1, a2, a3 >

<(20)(30)(40)> contained in <(10,20)(30)(40,60,70)>

<(20)(30,40)> **not** contained in <(10,20)(30)(40,60,70)>



What Is Sequential Pattern Mining?

- Given a set of sequences, find the complete set of *frequent* subsequences

A *sequence*: $\langle (ef) (ab) (df) c b \rangle$

A *sequence database*

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

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

$\langle a(bc)dc \rangle$ is a *subsequence* of $\langle a(abc)(ac)d(cf) \rangle$

Given *support threshold* $min_sup = 2$, $\langle (ab)c \rangle$ is a *sequential pattern* (cf. SID 10 & 30)

Sequential Patterns

- Given
 - a set of *sequences*, where each sequence consists of a list of *elements* and each element consists of set of *items*
 - user-specified *min_support* threshold



id	Sequence
10	$\langle a(abc)(ac)d(cf) \rangle$
20	$\langle (ad)c(bc)(ae) \rangle$
30	$\langle (ef)(ab)(df)cb \rangle$
40	$\langle eg(af)cbc \rangle$

$\langle a(abc)(ac)d(cf) \rangle$ - 5 elements, 9 items

$\langle a(abc)(ac)d(cf) \rangle$ - 9-sequence

$\langle a(abc)(ac)d(cf) \rangle = \langle a(cba)(ac)d(cf) \rangle$
 $\langle a(abc)(ac)d(cf) \rangle \neq \langle a(ac)(abc)d(cf) \rangle$

Order doesn't Matter (list)

Order matters (sequence)

Sequential Pattern Mining

- Find all the *frequent subsequences*, i.e. the subsequences whose occurrence frequency in the set of sequences is no less than *min_support*

Solution – 53 frequent subsequences

$\langle a \rangle$ $\langle aa \rangle$ $\langle ab \rangle$ $\langle a(bc) \rangle$ $\langle a(bc)a \rangle$ $\langle aba \rangle$ $\langle abc \rangle$
 $\langle (ab) \rangle$ $\langle (ab)c \rangle$ $\langle (ab)d \rangle$ $\langle (ab)f \rangle$ $\langle (ab)dc \rangle$ $\langle ac \rangle$
 $\langle aca \rangle$ $\langle acb \rangle$ $\langle acc \rangle$ $\langle ad \rangle$ $\langle adc \rangle$ $\langle af \rangle$
 $\langle b \rangle$ $\langle ba \rangle$ $\langle bc \rangle$ $\langle (bc) \rangle$ $\langle (bc)a \rangle$ $\langle bd \rangle$ $\langle bdc \rangle$ $\langle bf \rangle$
 $\langle c \rangle$ $\langle ca \rangle$ $\langle cb \rangle$ $\langle cc \rangle$
 $\langle d \rangle$ $\langle db \rangle$ $\langle dc \rangle$ $\langle dcb \rangle$
 $\langle e \rangle$ $\langle ea \rangle$ $\langle eab \rangle$ $\langle eac \rangle$ $\langle eacb \rangle$ $\langle eb \rangle$ $\langle ebc \rangle$ $\langle ec \rangle$
 $\langle ecb \rangle$ $\langle ef \rangle$ $\langle efb \rangle$ $\langle efc \rangle$ $\langle efcb \rangle$
 $\langle f \rangle$ $\langle fb \rangle$ $\langle fbc \rangle$ $\langle fc \rangle$ $\langle fcb \rangle$

id	Sequence
10	$\langle a(abc)(ac)d(cf) \rangle$
20	$\langle (ad)c(bc)(ae) \rangle$
30	$\langle (ef)(ab)(df)cb \rangle$
40	$\langle eg(af)cbc \rangle$

$min_support = 2$

Subsequence vs. super sequence

- Given two sequences $\alpha = \langle a_1 a_2 \dots a_n \rangle$ and $\beta = \langle b_1 b_2 \dots b_m \rangle$
- α is called a *subsequence* of β , denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$
- β is a *super sequence* of α
- A sequence s is maximal if it is not contained in any other sequence.



$\beta = \langle a(abc)(ac)d(cf) \rangle$

$\alpha_1 = \langle aa(ac)d(c) \rangle$

$\alpha_2 = \langle (ac)(ac)d(cf) \rangle$

$\alpha_3 = \langle ac \rangle$

~~$\beta = \langle a(abc)(ac)d(cf) \rangle$~~

~~$\alpha_4 = \langle df(cf) \rangle$~~

~~$\alpha_5 = \langle (cf)d \rangle$~~

~~$\alpha_6 = \langle (abc)dcf \rangle$~~

Sequence Support Count

- A **sequence database** is a set of tuples $\langle \text{sid}, s \rangle$
- A tuple $\langle \text{sid}, s \rangle$ is said to **contain** a sequence α , if α is a subsequence of s , i.e., $\alpha \subseteq s$
- The **support of a sequence** α is the number of tuples containing α

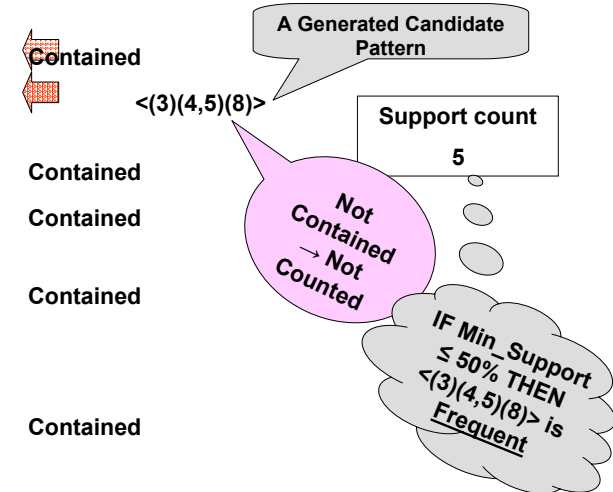


id	Sequence
10	$\langle a(abc)(ac)d(cf) \rangle$
20	$\langle (ad)c(bc)(ae) \rangle$
30	$\langle (ef)(ab)(df)cb \rangle$
40	$\langle eg(af)cbc \rangle$

$\alpha_1 = \langle a \rangle$ support(α_1) = 4
 $\alpha_2 = \langle ac \rangle$ support(α_2) = 4
 $\alpha_3 = \langle (ab)c \rangle$ support(α_3) = 2

Counting Sequences (An example)

$\langle (7)(3,8)(9)(4,5,6)(8) \rangle$	Contained
$\langle (8)(3,8)(9)(4,5)(6)(7) \rangle$	Contained
$\langle (3)(5,4) \rangle$	↑
$\langle (3)(5,4)(6)(9,8) \rangle$	Contained
$\langle (10)(3)(5,4)(9,8) \rangle$	Contained
$\langle (5,4)(6,6)(9,8) \rangle$	
$\langle (3)(5,4)(6)(9,8) \rangle$	Contained
$\langle (3)(5)(4)(6)(9,8) \rangle$	
$\langle (3)(5,2,1)(6)(9,8) \rangle$	
$\langle (3)(5,4)(6)(9,8) \rangle$	Contained



Challenges on Sequential Pattern Mining

- A **huge** number of possible sequential patterns are hidden in databases
- A mining algorithm should
 - find the **complete set of patterns**, when possible, satisfying the minimum support (frequency) threshold
 - be highly **efficient, scalable**, involving only a small number of database scans
 - be able to incorporate various kinds of **user-specific constraints**
- Comparison of association rules and sequence mining
 - Mining for association rules
 - Purpose: Discovery of frequent unordered itemsets.
 - Complexity: With n items there are $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ k -itemsets (sets with k items).
 - Mining for Sequential Patterns
 - Purpose: Discovery of frequent *sequences of* (unordered) itemsets.
 - Complexity: With n items there are n^k k -sequences (sequences with k items).

Association mining algorithms discover *isolated* item sets (*intra*-event patterns).
 Sequence mining algorithms discover *series of* item sets (*inter*-event patterns).

Studies on Sequential Pattern Mining

- Concept introduction and an initial Apriori-like algorithm
 - R. Agrawal & R. Srikant. "Mining sequential patterns," ICDE'95
- GSP—An Apriori-based, influential mining method (developed at IBM Almaden)
 - R. Srikant & R. Agrawal. "Mining sequential patterns: Generalizations and performance improvements," EDBT'96
- From sequential patterns to episodes (Apriori-like + constraints)
 - H. Mannila, H. Toivonen & A.I. Verkamo. "Discovery of frequent episodes in event sequences," Data Mining and Knowledge Discovery, 1997
- Data Projection-based approaches
 - FreeSpan (Han et al. "frequent pattern-projected sequential pattern mining" SIGKDD 2000)
 - PrefixSpan (Pei et al. "Prefix-projected pattern growth" ICDE 2001)
- Mining sequential patterns with constraints
 - M.N. Garofalakis, R. Rastogi, K. Shim: SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. VLDB 1999

Lecture Outline

Part I: Concepts (30 minutes)

- Basic concepts

Part II: Apriori-based Approaches (45 minutes)

- Apriori-All
- GSP

Part III: Pattern-Growth-based Approaches (45 minutes)

- Free-Span
- Prefix-Span

AprioriAll: The idea

- Basic method to mine sequential patterns
 - Based on the Apriori algorithm
 - Count all the large sequences, including non-maximal sequences
 - Use Apriori-generate function to generate candidate sequences: get candidates for a pass using only the large sequences found in the previous pass and make a scan over the data to find their support

AprioriAll: The big picture

Five-phase algorithm

1. Sort phase:
 - Create the sequence database from transactions.
2. Large itemset phase
 - Find all frequent itemsets using Apriori
3. Transformation phase:
 - Do integer mapping for large itemsets
4. Sequence phase:
 - Find all frequent sequential patterns using Apriori.
5. Maximal phase:
 - Eliminate non maximal sequences.

AprioriAll Algorithm(1)

• AprioriAll Algorithm

C_k : Candidate sequence of size k

L_k : frequent or large sequence of size k

$L_1 = \{\text{large 1-sequence}\}$; //result of itemset phase

for ($k = 2$; $L_{k-1} \neq \emptyset$; $k++$) **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 $\cup_k L_k$;

• Candidate Generation --Join Step:

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

Insert into C_k ,

Select $p.\text{itemset}_1, \dots, p.\text{itemset}_{k-1}, q.\text{itemset}_{k-1}$

From $L_{k-1} p, L_{k-1} q$

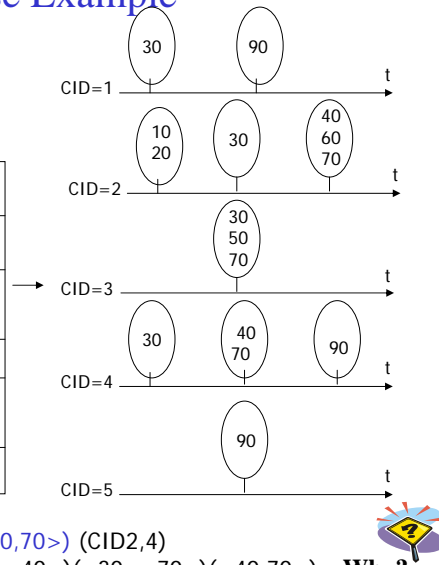
Where $p.\text{itemset}_1 = q.\text{itemset}_1, \dots,$

$p.\text{itemset}_{k-2} = q.\text{itemset}_{k-2}$

For example:
 $\{1,2,3\} \times \{1,2,4\}$
 $=$
 $\{1,2,3,4\}$
and
 $\{1,2,4,3\}$

Sequence Database Example

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?**

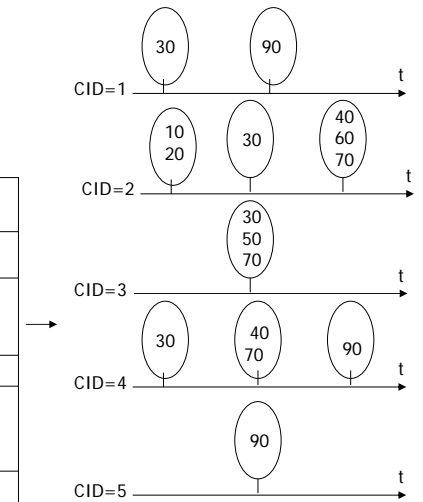


Sort Phases

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



Litemset Phase

Litemset Phase:

- Find all large itemset
(Why? Because each itemset in a large sequence has to be a large itemset.)
- To get large (frequent) itemsets → Use Apriori algorithm
- Need to modify support counting. (For sequential patterns, support is measured by fraction of customers.)

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

Litemset Result:

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

Difference from Apriori:

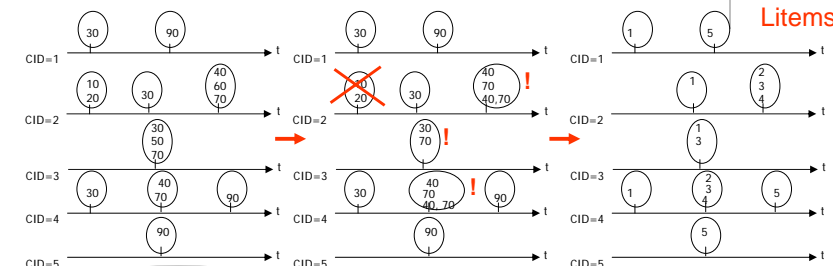
- the support count should be incremented only once per customer

Transformation Phase

Transformation Phase:

- Each large itemset is then mapped to a set of contiguous integers
(Why? to be able to compare two frequent itemsets in constant time)
- Represent transactions as sets of large itemsets.

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



Sequence database

→

Transformed database

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

Sequence Phase

- Sequence Phase:

- 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.

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

Itemsets

MinSupport =40%, i.e. 2 customers

```
1, <(1), {5}>
2, <(1), {2, 3, 4}>
3, <(1, 3)>
4, <(1), {2, 3, 4}, {5}>
5, <(5)>
```

Apriori

F.Seq.	Sup.
{1}	4
{2}	2
{3}	3
{4}	2
{5}	3
{(1), {2}}	2
{(1), {3}}	3
{(1), {4}}	2
{(1), {5}}	2

Re-mapping

```
<30>
<40>
<70>
<40, 70>
<90>
<30><40>
<30><70>
<30, (40, 70)>
<30><90>
```

Maximal phase

- Maximum Phase:

- Find the maximal sequences among the set of frequent sequences
- delete all sequences that are sub-sequences of other frequent sequences.

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

```
<30>
<40>
<70>
<40, 70>
<90>
<30><40>
<30><70>
<30, (40, 70)>
<30><90>
```

Summary for AprioriAll

- Algorithm wastes much time in counting non-maximal sequences, which can not be sequential patterns
- There are other variations of AprioriAll that reduce the candidates that are not maximal: AprioriSome and DynamicSome
- Absence of time window constraints
- *AprioriALL* is the basis of many efficient algorithm developed later. GSP is among them.

GSP—A Generalized Sequential Pattern Mining Algorithm

- GSP (Generalized Sequential Pattern) mining algorithm
 - proposed by Agrawal and Srikant, EDBT'96
- Outline of the method
 - Initially, every item in DB is a candidate of length-1
 - for each level (i.e., sequences of length-k) do
 - scan database to collect support count for each candidate sequence
 - generate candidate length-(k+1) sequences from length-k frequent sequences using Apriori
 - repeat until no frequent sequence or no candidate can be found
- Major strength: Candidate pruning by Apriori

A Basic Property of Sequential Patterns: Apriori

- A basic property: Apriori (Agrawal & Sirkant'94)
 - If a sequence S is not frequent
 - Then none of the super-sequences of S is frequent
 - E.g, <hb> is infrequent → so do <hab> and <(ah)b>

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Given *support threshold*
min_sup = 2

The GSP Algorithm

- Take sequences in form of <x> as length-1 candidates
- Scan database once, find F_1 , the set of length-1 sequential patterns
- Let $k=1$; while F_k is not empty do
 - Form C_{k+1} , the set of length-(k+1) candidates from F_k ;
 - If C_{k+1} is not empty, scan database once, find F_{k+1} , the set of length-(k+1) sequential patterns
 - Let $k=k+1$;

Finding Length-1 Sequential Patterns

- Examine GSP using an example
- Initial candidates: all singleton sequences
 - <a>, , <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once, count support for candidates

Cand	Sup
<a>	3
	5
<c>	4
<d>	3
<e>	3
<f>	2
<g>	1
<h>	1

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

min_sup = 2

Generating Length-2 Candidates

51 length-2 Candidates

	<a>		<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>		<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Without Apriori property,
 $8*8+8*7/2=92$
candidates

Apriori prunes
44.57% candidates

Generating Length-2 Candidates

$min_sup = 2$

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

	<a>		<c>	<d>	<e>	<f>
<a>	<aa>:2	<ab>:2	<ac>:1	<ad>:1	<ae>:1	<af>:1
	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>		<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
			<(bc)>:0	<(bd)>:2	<(be)>:1	<(bf)>:2
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

SID: 30, 50

SID: 20, 30

Generating Length-2 Candidates

$min_sup = 2$

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

	<a>		<c>	<d>	<e>	<f>
<a>	<aa>:2	<ab>:2	<ac>:1	<ad>:1	<ae>:1	<af>:1
	<ba>:3	<bb>:4	<bc>:4	<bd>:2	<be>:3	<bf>:2
<c>	<ca>:2	<cb>:3	<cc>:1	<cd>:2	<ce>:1	<cf>:1
<d>	<da>:2	<db>:2	<dc>:2	<dd>:0	<de>:1	<df>:0
<e>	<ea>:0	<eb>:1	<ec>:0	<ed>:1	<ee>:1	<ef>:1
<f>	<fa>:1	<fb>:2	<fc>:1	<fd>:0	<fe>:1	<ff>:2

	<a>		<c>	<d>	<e>	<f>
<a>		<(ab)>:0	<(ac)>:1	<(ad)>:1	<(ae)>:1	<(af)>:0
			<(bc)>:0	<(bd)>:2	<(be)>:1	<(bf)>:2
<c>				<(cd)>:0	<(ce)>:2	<(cf)>:0
<d>					<(de)>:1	<(df)>:0
<e>						<(ef)>:0
<f>						

Length-2 Sequential Patterns

- After scanning the database to collect support count for each length-2 candidate
- There are 19 length-2 candidates which pass the minimum support threshold
 - They are length-2 sequential patterns
 - 16 of them in the pattern of <xy>
 - 3 of them in the pattern of <(xy)>

Generating Length-3 Candidates and Finding Length-3 Patterns

- Generate Length-3 Candidates
 - Self-join length-2 sequential patterns
 - Based on the Apriori property
 - <ab>, <aa> and <ba> are all length-2 sequential patterns → <aba> is a length-3 candidate
 - 54 candidates are generated
 - <(bd)>, <bb> and <db> are all length-2 sequential patterns → <(bd)b> is a length-3 candidate
 - 27 candidates are generated
- Find Length-3 Sequential Patterns
 - Scan database once more, collect support counts for candidates
 - 19 out of 73 candidates pass support threshold

a(bd), (bd)a, b(bd), (bd)b, (bd)c, (bd)d, (bd)e, (bd)f, c(bd), d(bd), f(bd), a(bf), (bf)a, (bf)b, b(bf), (bf)c, (bf)d, (bf)e, (bf)f, c(bf), d(bf), f(bf), b(ce), (ce)a, (ce)b, (ce)d, d(ce)

Generating Length-3 Candidates

	<a>		<c>	<d>	<e>	<f>
<a>	<aa>:2	<ab>:2	<ac>:1	<ad>:1	<ae>:1	<af>:1
	<ba>:3	<bb>:4	<bc>:4	<bd>:2	<be>:3	<bf>:2
<c>	<ca>:2	<cb>:3	<cc>:1	<cd>:2	<ce>:1	<cf>:1
<d>	<da>:2	<db>:2	<dc>:2	<dd>:0	<de>:1	<df>:0
<e>	<ea>:0	<eb>:1	<ec>:0	<ed>:1	<ee>:1	<ef>:1
<f>	<fa>:1	<fb>:2	<fc>:1	<fd>:0	<fe>:1	<ff>:2

- <aaa>:0, <aab>:0
- <aba>:2, <abb>:2, <abc>:1, <abd>:1, <abe>:1, <abf>:1
- <baa>, <bab>
- <bba>, <bbb>, <bbc>, <bbd>, <bbe>, <bbf>
- <bca>, <bcb>, <bcd>
- <bda>, <bdb>, <bdc>
- <bfb>, <bff>
- <caa>, <cab>
- <cba>, <cbb>, <cbc>, <cbd>, <cbe>, <cbf>
- <cda>, <cdb>, <cdc>
- <daa>, <dab>
- <dba>, <ddb>, <dbc>, <dbd>, <dbe>, <dbf>
- <dca>, <dcb>, <dcd>
- <fba>, <fbb>, <fbc>, <fbd>, <fbe>, <fbf>
- <ffb>, <fff>

Example of generating <xyz> pattern for <ag>:

- Need to concatenate another Length-2 frequent itemset
- Concatenating another frequent itemsets that **start with a** to form <aaa> and <aab>

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)cb(ade)>

min_sup = 2

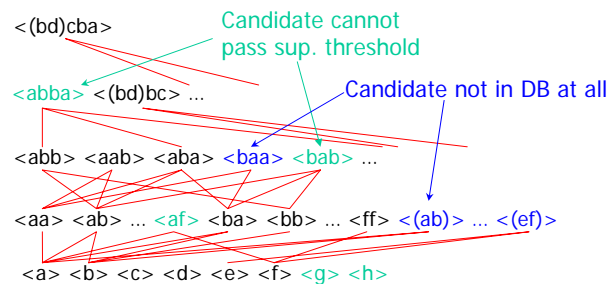
Generating Length-3 Candidates

	<a>		<c>	<d>	<e>	<f>				
<a>	<aa>:2	<ab>:2	<ac>:1	<ad>:1	<ae>:1	<af>:1				
	<ba>:3	<bb>:4	<bc>:4	<bd>:2	<be>:3	<bf>:2	<c>	<d>	<e>	<f>
<c>	<ca>:2	<cb>:3	<cc>:1	<cd>:2	<ce>:1	<cf>:1	<(ac)>:1	<(ad)>:1	<(ae)>:1	<(af)>:0
<d>	<da>:2	<db>:2	<dc>:2	<dd>:0	<de>:1	<df>:0	<(bc)>:0	<(bd)>:2	<(be)>:1	<(bf)>:2
<e>	<ea>:0	<eb>:1	<ec>:0	<ed>:1	<ee>:1	<ef>:1		<(cd)>:0	<(ce)>:2	<(cf)>:0
<f>	<fa>:1	<fb>:2	<fc>:1	<fd>:0	<fe>:1	<ff>:2			<(de)>:1	<(df)>:0
										<(ef)>:0

Example of generating <(xy)z> pattern for <(bd)>:

- Need to concatenate another Length-2 frequent itemset
- Concatenating those patterns that **end with b or d** to form something like <a(bd)>, <b(bd)>, <c(bd)>, <d(bd)>, <f(bd)>
- Concatenating those patterns that **starts with b or d** to form something like <(bd)a>, <(bd)b>, <(bd)c>, <(bd)d>, <(bd)e>, <(bd)f>

The GSP Mining Process



min_sup = 2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)cb(ade)>

Bottlenecks of GSP

- A huge set of candidates could be generated
 - 1,000 frequent length-1 sequences generate $1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500$ length-2 candidates!
- Multiple scans of database
- Real challenge: mining long sequential patterns
 - An exponential number of short candidates
 - A length-100 sequential pattern needs 10^{30} candidate sequences!

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

Lecture Outline

Part I: Concepts (30 minutes)

- Basic concepts

Part II: Apriori-based Approaches (45 minutes)

- Apriori-All
- GSP

Part III: Pattern-Growth-based Approaches (45 minutes)

- **Free-Span** (Frequent Pattern-Projected Sequential Pattern Mining)
- **Prefix-Span** (Prefix-Projected Sequential Pattern)

FreeSpan (Generalities)

- J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. **FreeSpan: Frequent pattern-projected sequential pattern mining**. KDD'00, pages 355-359.
- A divide-and-conquer approach
 - Recursively **project** a sequence database into a set of smaller databases
 - Mine each projected database to find the subset of patterns

FreeSpan (example)

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

- Given a sequence database S and min_support = 2
 - Step 1: find length-1 sequential patterns and list them in support descending order
 - f_list = a:4,b:4,c:4,d:3,e:3,f:3;~~g:1~~
 - Step 2: divide search space. The complete set of seq. patterns can be partitioned into 6 disjoint subsets (move down the f_list):
 - ones only contain item **a**
 - ones contain item **b** but no items after b in f_list
 - ones contain item **c** but no items after c in f_list
 - ones contain item **d** but no items after d in f_list
 - ones contain item **e** but no items after e in f_list
 - ones contain item **f**
- find subsets of sequential patterns. They can be mined by constructing projected databases and mining each recursively

FreeSpan (con't)

- Finding Seq. Patterns containing item b but no items after b in f_list
 - -projected database:
 - 10:<a(ab)a>, 20:<aba>, 30:<(ab)b>, 40:<ab>
 - Find all the length-2 seq. patterns containing item b but no items after b in f_list :
 - <ab>:4, <ba>:2, <(ab)>:2
 - Further partition and mining

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

f_list = a:4,b:4, c:4,d:3,e:3,f:3

From FreeSpan to PrefixSpan


- Freespan:
 - Projection-based: No candidate sequence needs to be generated
 - But, projection can be performed at any point in the sequence, and the projected sequences may not shrink much. For example, the size of f-projected database is the same as the original sequence database
- PrefixSpan
 - Projection-based
 - But only prefix-based projection: less projections and quickly shrinking sequences

• J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", Proc. 2001 Int. Conf. on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001.

Prefix of a Sequence


- Given two sequences $\alpha = \langle a_1 a_2 \dots a_n \rangle$ and $\beta = \langle b_1 b_2 \dots b_m \rangle$, $m \leq n$
- Sequence β is called a **prefix** of α if and only if:
 - $b_i = a_i$ for $i \leq m-1$;
 - $b_m \subseteq a_m$;
 - All the items in $(a_m - b_m)$ are alphabetically after those in b_m

Given an alphabetical order of items in each itemset (element)

 $\alpha = \langle a(abc)(ac)d(cf) \rangle$ $\beta = \langle a(abc)a \rangle$ $\beta = \langle a \rangle$ $\beta = \langle a(ab) \rangle$	 $\alpha = \langle a(abc)(ac)d(cf) \rangle$ $\beta = \langle a(ab)a \rangle$ $\beta = \langle a(abc)c \rangle$
---	---

Projection


- Given sequences α and β , such that β is a subsequence of α .
- A subsequence α' of sequence α is called a **projection** of α w.r.t. β prefix if and only if
 - α' has prefix β ;
 - There exist no proper super-sequence α'' of α' such that α'' is a subsequence of α and also has prefix β



$\alpha = \langle a(abc)(ac)d(cf) \rangle$
 $\beta = \langle (bc)a \rangle$
 $\alpha' = \langle (bc)(ac)d(cf) \rangle$

Postfix

- Let $\alpha' = \langle a_1 a_2 \dots a_n \rangle$ be the projection of α w.r.t. prefix $\beta = \langle a_1 a_2 \dots a_{m-1} a'_m \rangle$ ($m \leq n$)
- Sequence $\gamma = \langle a''_m a_{m+1} \dots a_n \rangle$ is called the **postfix** of α w.r.t. prefix β , denoted as $\gamma = \alpha / \beta$, where $a''_m = (a_m - a'_m)$
- We also denote $\alpha = \beta \cdot \gamma$



$\alpha' = \langle a(abc)(ac)d(cf) \rangle$
 $\beta = \langle a(abc)a \rangle$
 $\gamma = \langle _c d(cf) \rangle$

PrefixSpan – Algorithm

- **Input:** A sequence database S, and the minimum support threshold min_sup
- **Output:** The complete set of sequential patterns
- **Method:** Call PrefixSpan($\langle \rangle, 0, S$)
- **Subroutine** PrefixSpan($\alpha, l, S|_{\alpha}$)
- **Parameters:**
 - α : sequential pattern,
 - l: the length of α ;
 - $S|_{\alpha}$: the α -projected database, if $\alpha \neq \langle \rangle$; otherwise; the sequence database S.

PrefixSpan – Algorithm (2)

- **Method**
 1. Scan $S|_{\alpha}$ once, find the set of frequent items b such that:
 - a) b can be assembled to the last element of α to form a sequential pattern; or
 - b) $\langle b \rangle$ can be appended to α to form a sequential pattern.
 2. For each frequent item b, append it to α to form a sequential pattern α' , and output α' ;
 3. For each α' , construct α' -projected database $S|_{\alpha'}$, and call PrefixSpan($\alpha', l+1, S|_{\alpha'}$).

PrefixSpan - Example

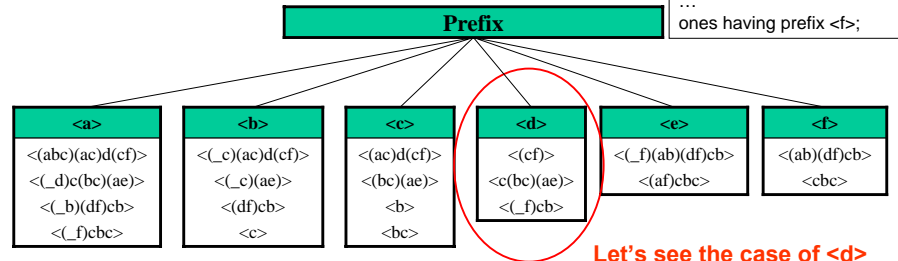
1. Find length-1 sequential patterns

<a>		<c>	<d>	<e>	<f>	<g>
4	4	4	3	3	3	1

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

min_support = 2

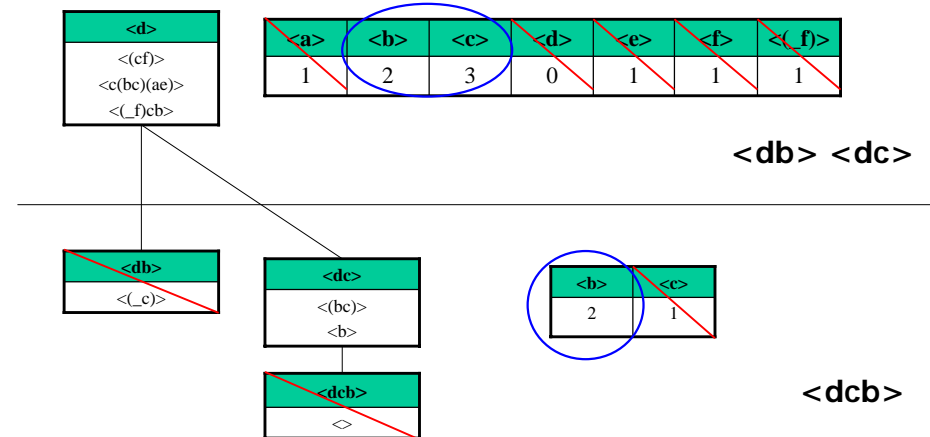
2. Divide search space



PrefixSpan – Example (2)

Projected database for <d>

3. Find subsets of sequential patterns



PrefixSpan - characteristics

- No candidate sequence needs to be generated by PrefixSpan
- Projected databases keep shrinking
- The major cost of PrefixSpan is the construction of projected databases



How to reduce this cost?

Different projection methods

- Bi-level projection**
 - reduces the number and the size of projected databases
- Pseudo-Projection**
 - reduces the cost of projection when projected database can be held in main memory



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

Bi-level Projection

min_support = 2

- Scan to get 1-length sequences
- Construct a **triangular matrix** instead of **projected databases** for each length-1 patterns

	a	b	c	d	e	f
b	(4,2,2)	1				
c	(4,2,1)	(3,3,2)	3			
d	(2,1,1)	(2,2,0)	(1,3,0)	0		
e	(1,2,1)	(1,2,0)	(1,2,0)	(1,1,0)	0	
f	(2,1,1)	(2,2,0)	(1,2,1)	(1,1,1)	(2,0,1)	1

ALL length-2 sequential pattern

Support(<ac>) = 4
Support(<ca>) = 2
Support(<ac>) = 1

Support(<cc>) = 3

Bi-level projection (2)

- For each length-2 sequential pattern α , construct the α -projected database and find the frequent items
- Construct corresponding S-matrix



<ab>	a	b	c	(c)	d	(d)	e	(e)	f	(f)
<(c)(ac)(cf)>	2	0	2	2	0	1	0	0	1	0
<(c)a>										
<c>										

<aba> <abc> <a(bc)>

a	0		
c	(1,0,1)	1	
(c)	(ϕ ,2, ϕ)	(ϕ ,1, ϕ)	ϕ
	a	c	(c)

<a(bc)a>

Bi-level projection (3) - optimization

- “Do we need to include every item in a postfix in the projected databases?”
- NO!** Item pruning in projected database by 3-way Apriori checking



<ac> is not frequent

Any super-sequence of it can never be a sequential pattern

c can be excluded from construction of <ab>-projected database

<a(bd)> is not frequent

To construct <a(bc)>-projected database, sequence <a(bcde)df> should be projected to <(c)df> instead of <(c)de)df>

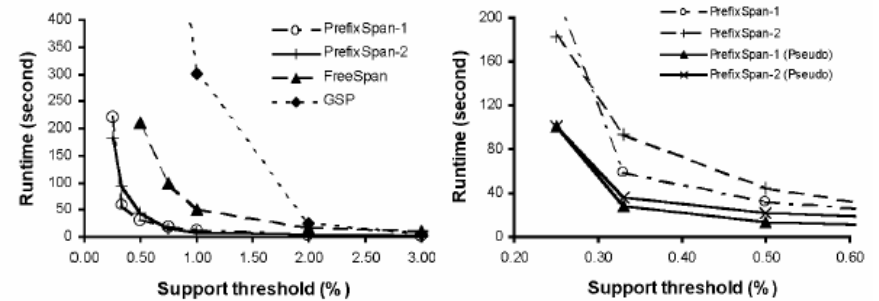
Pseudo-Projection

- **Observation:** postfixes of a sequence often appear repeatedly in recursive projected databases
- **Method:** instead of constructing *physical* projection by collecting all the postfixes, we can use pointers referring to the sequences in the database as a pseudo-projection
- Every projection consists of two pieces of information: **pointer** to the sequence in database and **offset** to the postfix in the sequence

s1 = <a(abc)(ac)d(cf)>

Pointer	Offset	Postfix
s1	2	<(abc)(ac)d(cf)>
s1	5	<(ac)d(cf)>
s1	6	<_c)d(cf)>

Efficiency of Prefix-Span and Effect of Pseudo-Projection



Summary

- Sequential Pattern Mining is useful in many application, e.g. weblog analysis, financial market prediction, BioInformatics, etc.
- It is similar to the frequent itemsets mining, *but* with consideration of ordering.
- We have looked at different approaches that are descendants from two popular algorithms in mining frequent itemsets
 - Candidates Generation: AprioriAll and GSP
 - Pattern Growth: FreeSpan and PrefixSpan