Mining Sequences

Examples of Sequence

- Web sequence:

  ⟨ {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} ⟩

- Purchase history of a given customer

  ⟨{Java in a Nutshell, Intro to Servlets} {EJB Patterns},…⟩

- Sequence of classes taken by a computer science major:

  ⟨ {Algorithms and Data Structures, Introduction to Operating Systems} {Database Systems, Computer Architecture} {Computer Networks, Software Engineering} {Computer Graphics, Parallel Programming} …⟩

Formal Definition of a Sequence

- A sequence is an ordered list of **elements** (transactions)

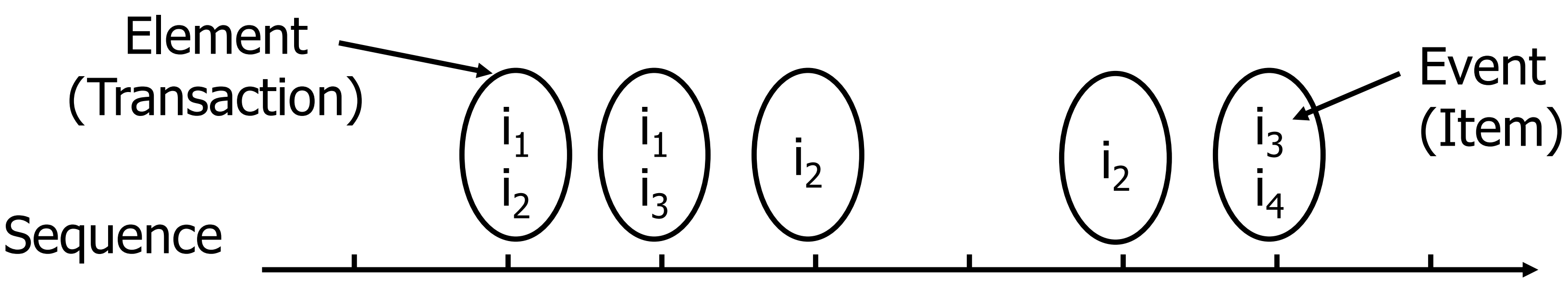
$$s = < e_1 \, e_2 \, e_3 \, \ldots >$$

  - Each element contains a collection of **events** (items)

$$e_i = \{i_1, i_2, \ldots, i_k\}$$

  - Each element is attributed to a specific time or location

- A *k*-sequence is a sequence that contains *k* events (items)



Formal Definition of a Subsequence

- A sequence $\langle a_1 a_2 \ldots a_n \rangle$ is contained in another sequence $\langle b_1 b_2 \ldots b_m \rangle$ ($m \geq n$) if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, …, $a_n \subseteq b_{in}$

Data sequence	Subsequence	Contained?
$\langle \{2,4\}\ \{3,5,6\}\ \{8\} \rangle$	$\langle \{2\}\ \{3,5\} \rangle$	Yes
$\langle \{1,2\}\ \{3,4\} \rangle$	$\langle \{1\}\ \{2\} \rangle$	No
$\langle \{2,4\}\ \{2,4\}\ \{2,5\} \rangle$	$\langle \{2\}\ \{4\} \rangle$	Yes

- Support of a subsequence w is the fraction of data sequences that contain w
- A *sequential pattern* is a frequent subsequence (i.e., a subsequence whose support is $\geq$ *minsup*)

Sequential Pattern Mining: Example

E.g.
A: <{1,2,4}, {2,3}, {5}>
B: <{1,2}, {2,3,4}>

Group	Timestamp	Events
A	1	1,2,4
A	2	2,3
A	3	5
B	1	1,2
B	2	2,3,4
C	1	1,2
C	2	2,3,4
C	3	2,4,5
D	1	2
D	2	3, 4
D	3	4, 5
E	1	1, 3
E	2	2, 4, 5

Minsup = 50% *i.e.* min. sup. count = 2

Examples of Frequent Subsequences:

< {1,2} >	s=60%
< {2,3} >	s=60%
< {2,4}>	s=80%
< {3} {5}>	s=80%
< {1} {2} >	s=80%
< {2} {2} >	s=60%
< {1} {2,3} >	s=60%
< {2} {2,3} >	s=60%
< {1,2} {2,3} >	s=60%

Sequential Pattern Mining: Definition

- Given:
  - a database of sequences
  - a user-specified minimum support threshold, *minsup*

- Task:
  - Find all subsequences with support ≥ *minsup*

- Challenge:
  - Many more candidate sequential patterns than candidate itemsets.

Extracting Sequential Patterns

- Given *n* events (items): $i_1, i_2, i_3, \ldots, i_n$
- Candidate 1-subsequences:

  $<\{i_1\}>, <\{i_2\}>, <\{i_3\}>, \ldots, <\{i_n\}>$

- Candidate 2-subsequences:

  $<\{i_1, i_2\}>, <\{i_1, i_3\}>, \ldots, <\{i_1\} \{i_1\}>, <\{i_1\} \{i_2\}>, \ldots, <\{i_{n-1}\} \{i_n\}>$

- Candidate 3-subsequences:

  $<\{i_1, i_2 , i_3\}>, <\{i_1, i_2 , i_4\}>, \ldots, <\{i_1, i_2\} \{i_1\}>, <\{i_1, i_2\} \{i_2\}>, \ldots,$
  $<\{i_1\} \{i_1 , i_2\}>, <\{i_1\} \{i_1 , i_3\}>, \ldots, <\{i_1\} \{i_1\} \{i_1\}>, <\{i_1\} \{i_1\} \{i_2\}>, \ldots$

APRIORI-like Algorithm

- Make the first pass over the sequence database to yield all the 1-element frequent sequences

- Repeat until no new frequent sequences are found

  Candidate Generation:
  - Merge pairs of frequent subsequences found in the (k-1)th pass to generate candidate sequences that contain k items

  Candidate Pruning:
  - Prune candidate k-sequences that contain infrequent (k-1)-subsequences

  Support Counting:
  - Make a new pass over the sequence database to find the support for these candidate sequences
  - Eliminate candidate k-sequences whose actual support is less than *minsup*

Candidate Generation

- Base case (k=2):
  - Merging two frequent 1-sequences $<\{i_1\}>$ and $<\{i_2\}>$ will produce four candidate 2-sequences:
  - $<\{i_1\}, \{i_2\}>$, $<\{i_2\}, \{i_1\}>$, $<\{i_1, i_2\}>$, $<\{i_2, i_1\}>$
- General case (k>2):
  - A frequent (k-1)-sequence w_1 is merged with another frequent (k-1)-sequence w_2 to produce a candidate k-sequence if the subsequence obtained by removing the first event in w_1 is the same as the subsequence obtained by removing the last event in w_2
  - The resulting candidate after merging is given by the sequence w_1 extended with the last event of w_2.
    - If the last two events in w_2 belong to the same element, then the last event in w_2 becomes part of the last element in w_1
    - Otherwise, the last event in w_2 becomes a separate element appended to the end of w_1

Candidate Generation Examples

- Merging the sequences
  w_1=<{1} {2 3} {4}> and w_2 =<{2 3} {4 5}>
  will produce the candidate sequence < {1} {2 3} {4 5}> because the last two events in w_2 (4 and 5) belong to the same element

- Merging the sequences
  w_1=<{1} {2 3} {4}> and w_2 =<{2 3} {4} {5}>
  will produce the candidate sequence < {1} {2 3} {4} {5}> because the last two events in w_2 (4 and 5) do not belong to the same element

- Finally, the sequences <{1}{2}{3}> and <{1}{2, 5}> don't have to be merged (Why?)
- Because removing the first event from the first sequence doesn't give the same subsequence as removing the last event from the second sequence.
- If <{1}{2,5}{3}> is a viable candidate, it will be generated by merging a different pair of sequences, <{1}{2,5}> and <{2,5}{3}>.

Example

Frequent
3-sequences

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

Candidate
Generation

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

Candidate
Pruning

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

Timing Constraints

Buyer A: < {TV} … {DVD Player} >

Buyer B: < {TV} … {DVD Player} >

…

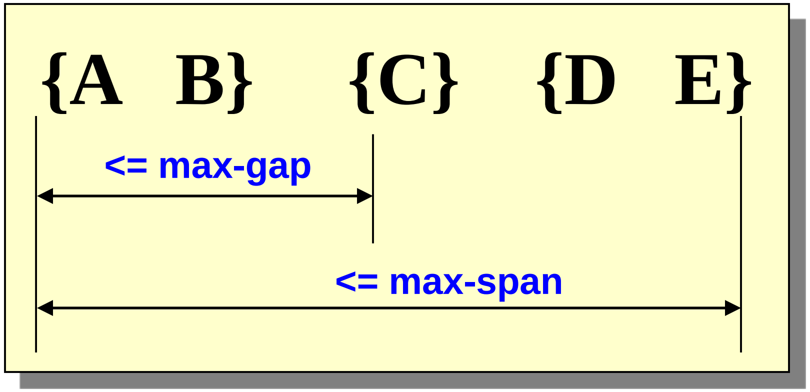
- The sequential pattern of interest is

  <{TV}{DVD Player}>

  which suggests that people who buy TV will also soon buy DVD player.

- A person who bought a TV ten years earlier should not be considered as supporting the pattern because the time gap between the purchases is too long.

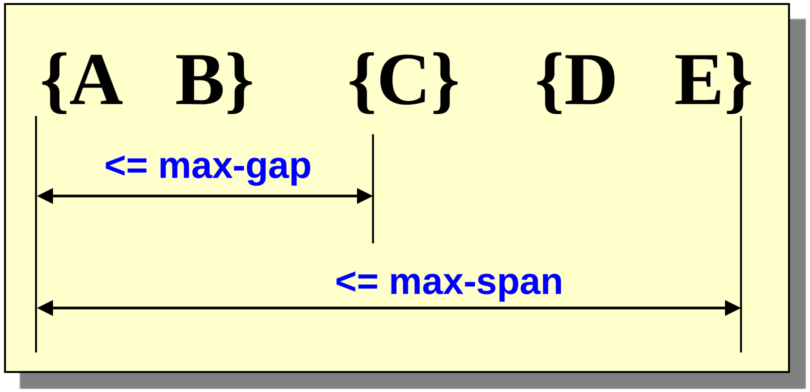
Timing Constraints



max-gap = 2, max-span= 4

Data sequence	Subsequence	Contained?
<{2,4} {3,5,6} {4,7} {4,5} {8}>	< {6} {5} >	
<{1} {2} {3} {4} {5}>	< {1} {4} >	
<{1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	
<{1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	

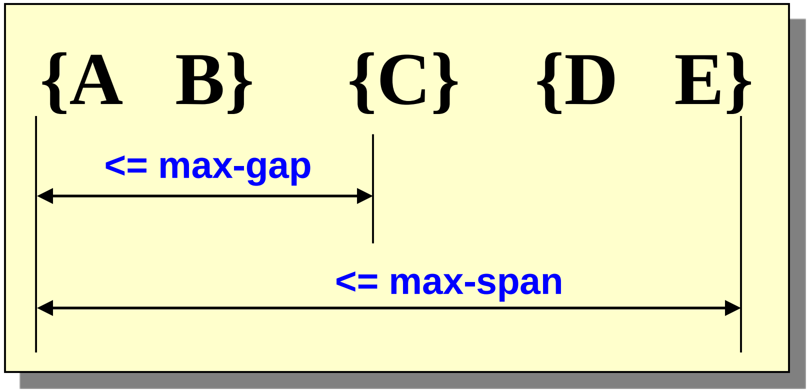
Timing Constraints



max-gap = 2, max-span= 4

Data sequence	Subsequence	Contained?
<{2,4} {3,5,6} {4,7} {4,5} {8}>	< {6} {5} >	Yes
<{1} {2} {3} {4} {5}>	< {1} {4} >	
<{1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	
<{1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	

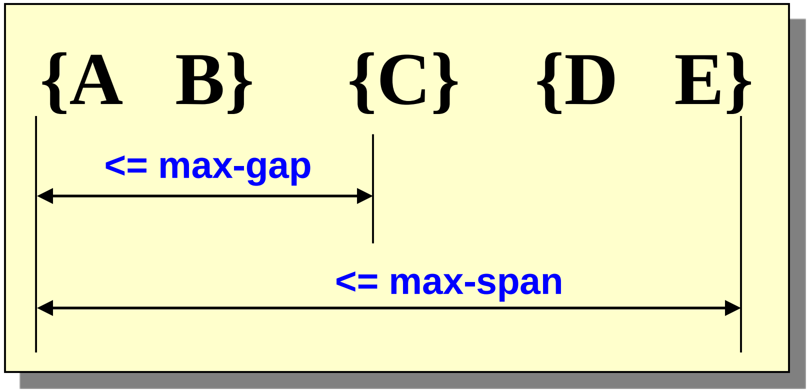
Timing Constraints



max-gap = 2, max-span= 4

Data sequence	Subsequence	Contained?
<{2,4} {3,5,6} {4,7} {4,5} {8}>	< {6} {5} >	Yes
<{1} {2} {3} {4} {5}>	< {1} {4} >	No
<{1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	
<{1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	

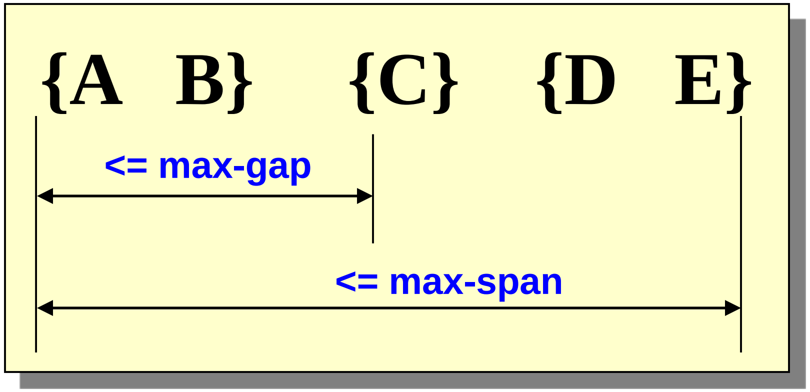
Timing Constraints



max-gap = 2, max-span= 4

Data sequence	Subsequence	Contained?
<{2,4} {3,5,6} {4,7} {4,5} {8}>	< {6} {5} >	Yes
<{1} {2} {3} {4} {5}>	< {1} {4} >	No
<{1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	Yes
<{1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	

Timing Constraints



max-gap = 2, max-span= 4

Data sequence	Subsequence	Contained?
<{2,4} {3,5,6} {4,7} {4,5} {8}>	< {6} {5} >	Yes
<{1} {2} {3} {4} {5}>	< {1} {4} >	No
<{1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	Yes
<{1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	No

Mining Sequential Patterns with Timing Constraints

- Approach 1:
  - Mine sequential patterns without timing constraints
  - Postprocess the discovered patterns

- Approach 2:
  - Modify algorithm to directly prune candidates that violate timing constraints
  - Question:
    - Does APRIORI principle still hold?

APRIORI Principle for Sequence Data

Object	Timestamp	Events
A	1	1,2,4
A	2	2,3
A	3	5
B	1	1,2
B	2	2,3,4
C	1	1, 2
C	2	2,3,4
C	3	2,4,5
D	1	2
D	2	3, 4
D	3	4, 5
E	1	1, 3
E	2	2, 4, 5

Suppose:

max-gap = 1

max-span = 5

<{2} {5}>

support = 40%

but

<{2} {3} {5}>

support = 60%

Problem exists because of max-gap constraint

This problem can avoided by using the concept of a contiguous subsequence.

Contiguous Subsequences

- s is a contiguous subsequence of

  $w = \langle e_1, e_2, \ldots, e_k \rangle$

  if any of the following conditions holds:

  1. s is obtained from w by deleting an item from either e_1 or e_k
  2. s is obtained from w by deleting an item from any element e_i that contains at least 2 items
  3. s is a contiguous subsequence of s' and s' is a contiguous subsequence of w (recursive definition)

- Examples: s = < {1} {2} >
  – is a contiguous subsequence of
    < {1} {2 3}>, < {1 2} {2} {3}>, and < {3 4} {1 2} {2 3} {4} >
  – is not a contiguous subsequence of
    < {1} {3} {2}> and < {2} {1} {3} {2}>

Modified Candidate Pruning Step

- Modified APRIORI Principle
  - If a k-sequence is frequent, then all of its contiguous (k-1)-subsequences must also be frequent

- Candidate generation doesn't change. Only pruning changes.

- Without maxgap constraint:
  - A candidate k-sequence is pruned if at least one of its (k-1)-subsequences is infrequent

- With maxgap constraint:
  - A candidate k-sequence is pruned if at least one of its **contiguous** (k-1)-subsequences is infrequent