Sequential Pattern Mining

Frequent patterns for sequences

Slides edited from the "Lecture Notes for Chapter 7" by Tan, Steinbach, Kumar

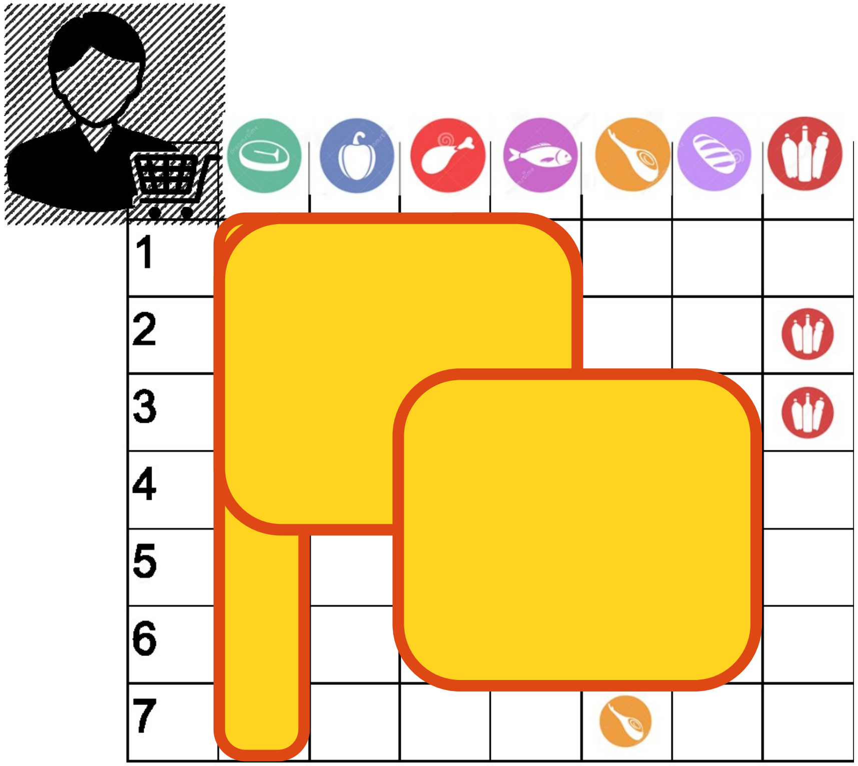
From itemsets to sequences

- Frequent itemsets and association rules focus on transactions and the items that appear there
- Databases of transactions usually have a temporal information
  - Sequential patter exploit it
- Example data:
  - Market basket transactions
  - Web server logs
  - Tweets
  - Workflow production logs

Frequent patterns

Events or combinations of events that appear frequently in the data

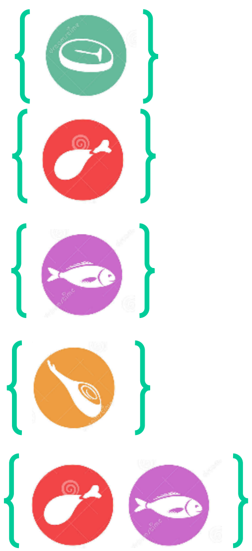
E.g. items bought by customers of a supermarket

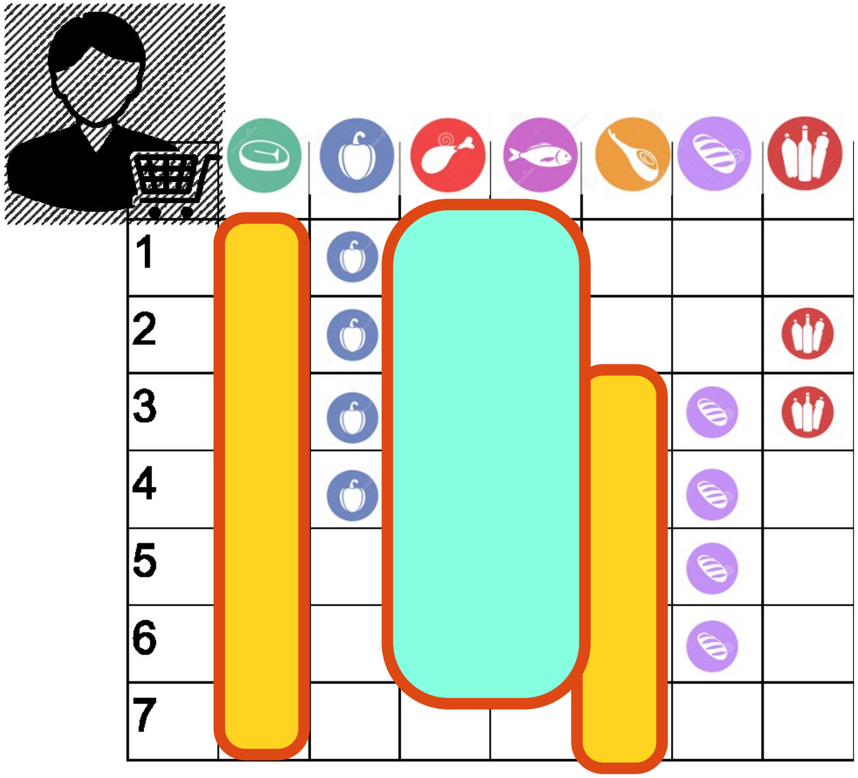


Frequent patterns

Frequent itemsets w.r.t. minimum threshold

E.g. with Min_freq = 5

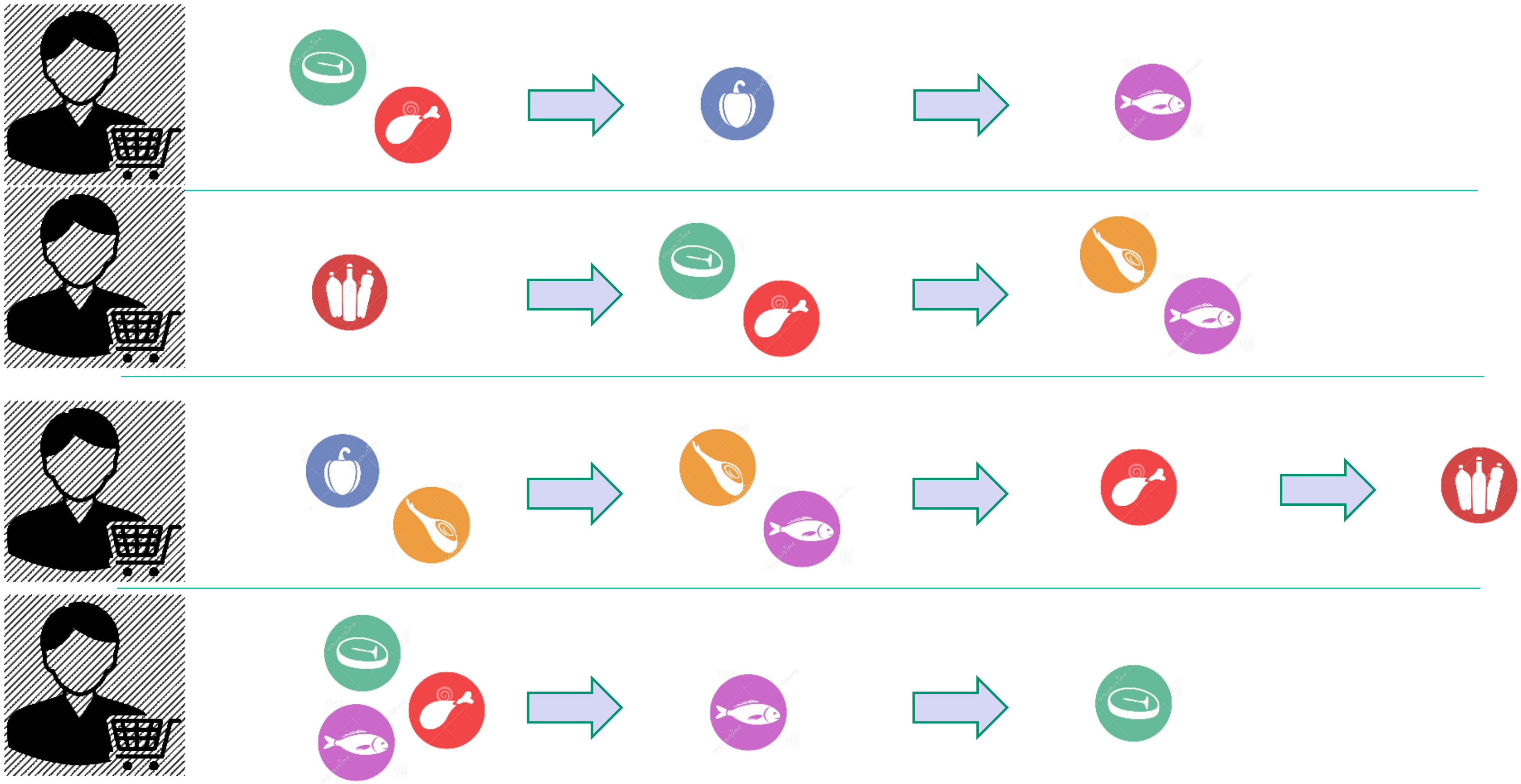




Frequent patterns
Complex domains

Frequent sequences (a.k.a. Sequential patterns)

Input: sequences of events (or of groups)

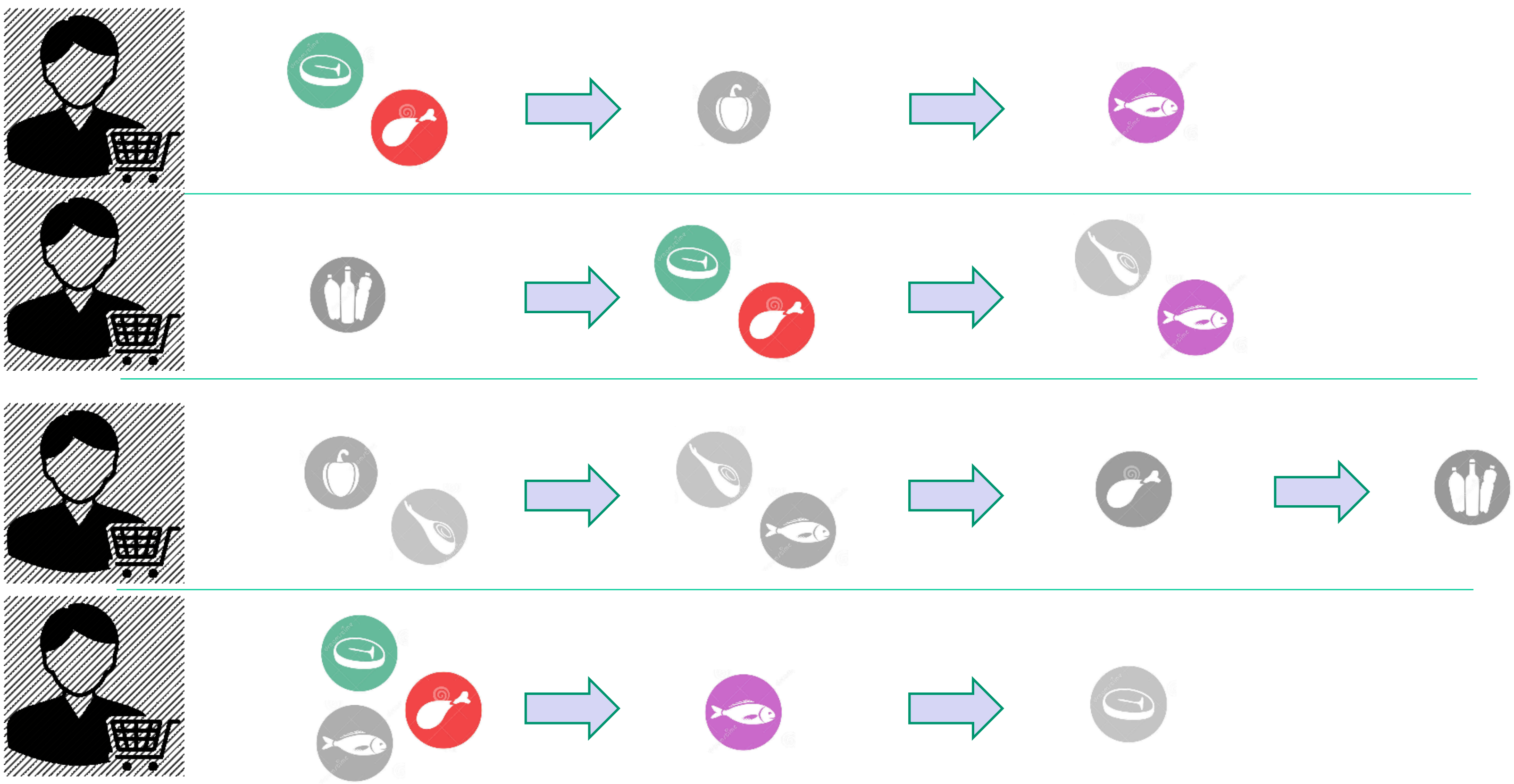


Frequent patterns

Complex domains

Objective: identify sequences that occur frequently

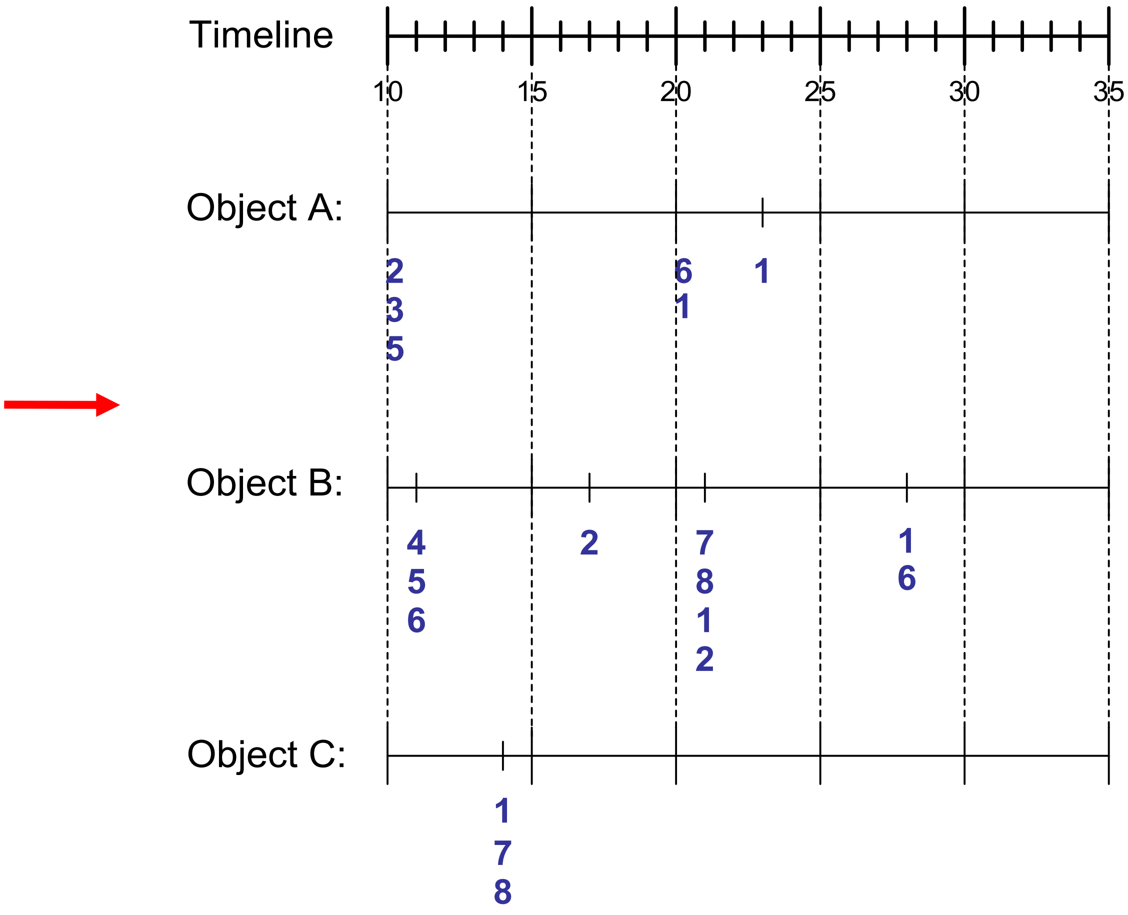
- Sequential pattern: 



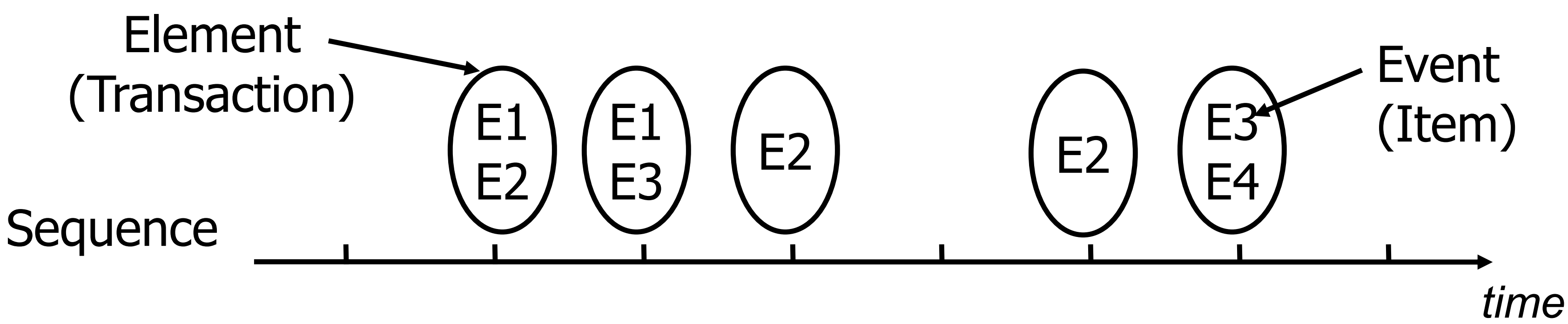
Sequence Data

Sequence Database:

Object	Timestamp	Events
A	10	2, 3, 5
A	20	6, 1
A	23	1
B	11	4, 5, 6
B	17	2
B	21	7, 8, 1, 2
B	28	1, 6
C	14	1, 8, 7



Terminology



Sequence Database	Sequence	Element (Transaction)	Event (Item)
Customer	Purchase history of a given customer	A set of items bought by a customer at time t	Books, diary products, CDs, etc
Web Data	Browsing activity of a particular Web visitor	A collection of files viewed by a Web visitor after a single mouse click	Home page, index page, contact info, etc
Event data	History of events generated by a given sensor	Events triggered by a sensor at time t	Types of alarms generated by sensors
Genome sequences	DNA sequence of a particular species	An element of the DNA sequence	Bases A,T,G,C

Formal Definition of a Sequence

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

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

  - Each element is attributed to a specific time or location
  - Each element contains a collection of events (items)

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

- Length of a sequence, |s|, is given by the number of elements of the sequence
- A k-sequence is a sequence that contains k events (items)

Formal Definition of a Sequence

- Example

  s = < {A,B}, {B,E,F}, {A}, {E,F,H} >

  - Length of s: |s| = 4 elements
  - s is a 9-sequence
  - Times associated to elements:
    - {A,B} → time=0
    - {B,E,F} → time = 120
    - {A} → time = 130
    - {E,F,H} → time = 200

Sequences without explicit time info

- Default: time of element = position in the sequence
- Example

    s = < {A,C}, {E}, {A,F}, {E,G,H} >

    - Default times associated to elements:
        - {A,C} → time=0
        - {E} → time = 1
        - {A,F} → time = 2
        - {E,G,H} → time = 3

Examples of Sequence

- Web sequence:

Singleton elements

< {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} >

- Sequence of initiating events causing the nuclear accident at 3-mile Island:
(http://stellar-one.com/nuclear/staff_reports/summary_SOE_the_initiating_event.htm)

< {clogged resin **&** outlet valve closure} {loss of feedwater}
{condenser polisher outlet valve shut} {booster pumps trip}
{main waterpump trips **&** main turbine trips **&** reactor pressure increases}>

Complex elements

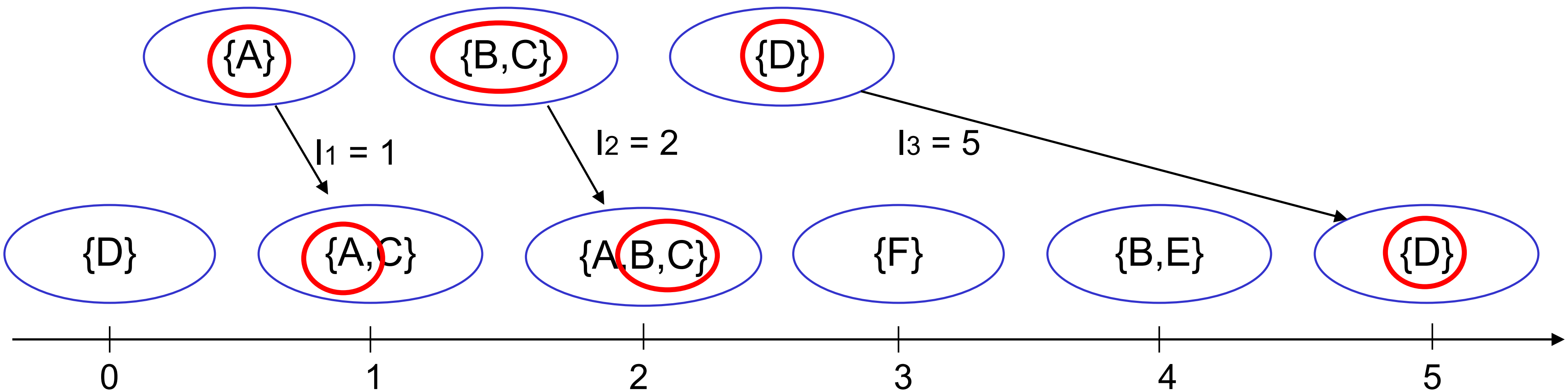
- Sequence of books checked out at a library:

<{Fellowship of the Ring} {The Two Towers} {Return of the King}>

Singleton elements

Formal Definition of a Subsequence

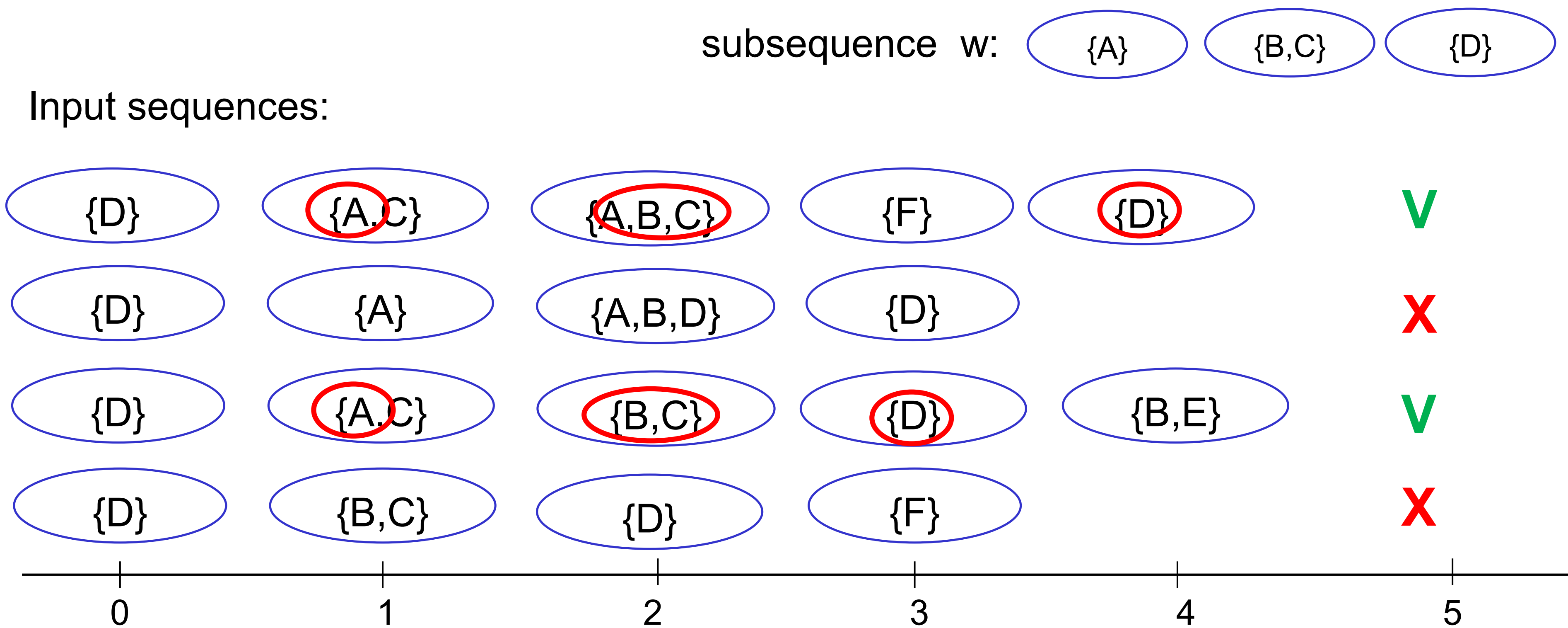
- A sequence $<a_1\ a_2 \ldots a_n>$ is contained in another sequence $<b_1\ b_2 \ldots b_m>$ $(m \geq n)$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i1}$, …, $a_n \subseteq b_{in}$



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

Formal Definition of Sequential Pattern

- The **support** of a subsequence w
  - is the fraction of data sequences that contain w

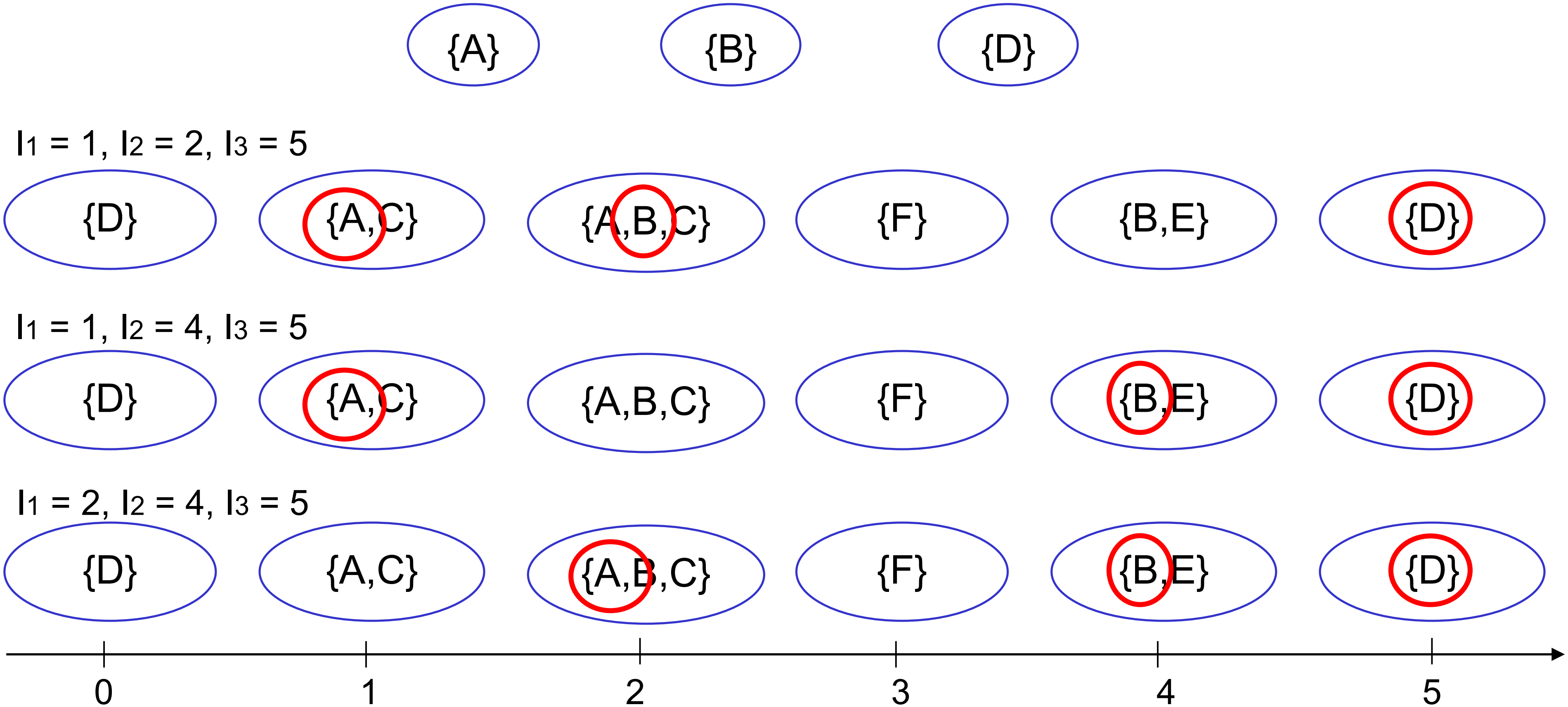


support of w: 2/4 = 0.50 (50%)

- A **sequential pattern**
  - is a **frequent** subsequence
  - i.e., a subsequence whose support is ≥ ***minsup***

Formal Definition of Sequential Pattern

- Remark: a subsequence (i.e. a candidate pattern) might be mapped into a sequence in several different ways
  - Each mapping is an **instance** of the subsequence
  - In mining sequential patterns we need to find only one instance



Exercises

- find instances/occurrence of the following patterns

  <{C}{H}{C}>

  <{A} {F} >

  <{A}{A}{D}>

  <{A}{A,B}{F}>

- in the input sequence below

<	{A,C}	{C,D}	{F,H}	{A,B}	{B,C,D}	{E}	{A,B,D}	{F}	>
	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	

Exercises

- find instances/occurrence of the following patterns

  <{C} {H} {C}>
  <{A} {B} >
  <{C} {C} {E}>
  <{A} {E}>

- in the input sequence below

<	{A,C}	{C,D,E}	{F}	{A,H}	{B,C,D}	{E}	{A,B,D}	>
	t=0	t=1	t=2	t=3	t=4	t=5	t=6	

Sequential Pattern Mining: Definition

- Given:
  - a database of sequences
  - a user-specified minimum support threshold, *minsup*
- Task:
  - Find all subsequences with support ≥ *minsup*

Sequential Pattern Mining: Challenge

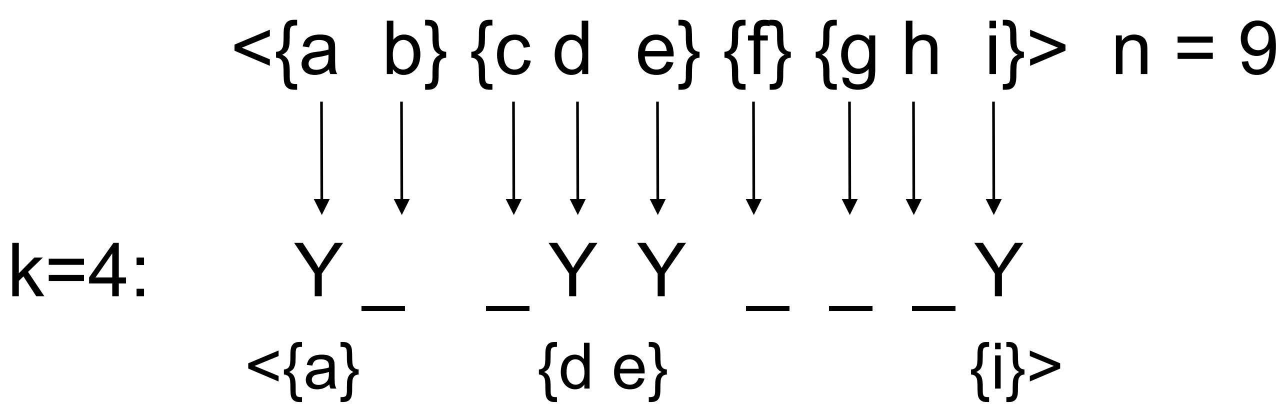
- Trivial approach: generate all possible k-subsequences, for k=1,2,3,… and compute support
- Combinatorial explosion!
  - With frequent itemsets mining we had:
    - N. of k-subsets = $\binom{n}{k}$ *n = n. of distinct items in the data*
  - With sequential patterns:
    - N. of k-subsequences = n^k
    - The same item can be repeated:
      - < {A} {A} {B} {A} … >

Sequential Pattern Mining: Challenge

- Even if we generate them from input sequences
  - E.g.: Given a n-sequence: <{a b} {c d e} {f} {g h i}>
    - Examples of subsequences:

      <{a} {c d} {f} {g} >, < {c d e} >, < {b} {g} >, etc.

  - Number of k-subsequences can be extracted from it



Answer :

$$\binom{n}{k} = \binom{9}{4} = 126$$

Sequential Pattern Mining: Example

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

Minsup = 50%

Examples of Frequent Subsequences:

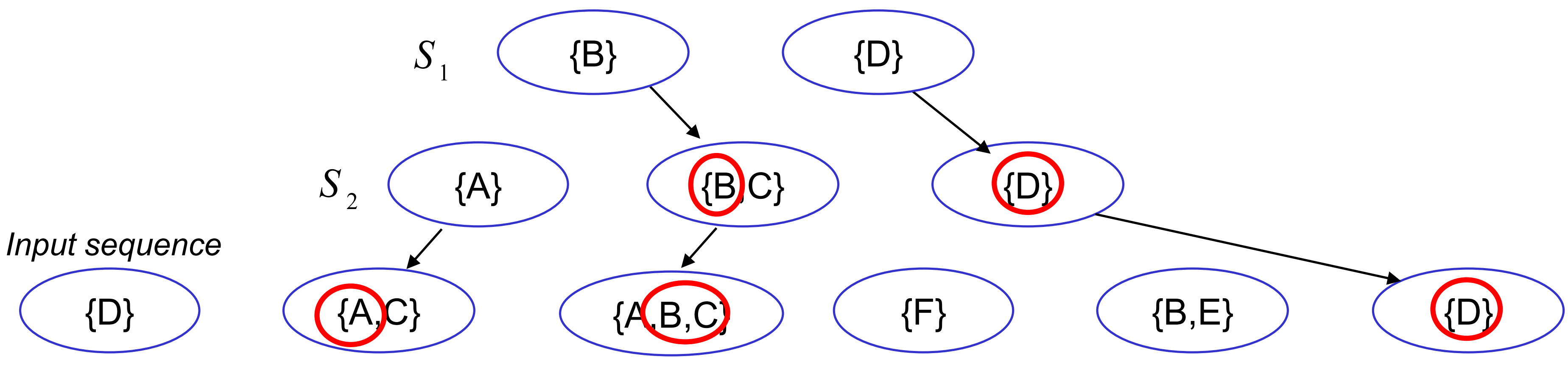
< {1,2} > s=60%
< {2,3} > s=60%
< {2,4}> s=80%
< {3} {5}> s=80%
< {1} {2} > s=80%
< {2} {2} > s=60%
< {1} {2,3} > s=60%
< {2} {2,3} > s=60%
< {1,2} {2,3} > s=60%

Generalized Sequential Pattern (GSP)

- **Follows the same structure of Apriori**
  - Start from short patterns and find longer ones at each iteration
- **Based on “Apriori principle” or “anti-monotonicity of support”**
  - If one sequence S1 is contained in sequence S2, then the support of S2 cannot be larger than that of S1:

$$S_1 \subseteq S_2 \Rightarrow \sup(S_1) \geq \sup(S_2)$$

- **Intuitive proof**
  - Any input sequence that contains S2 will also contain S1



Generalized Sequential Pattern (GSP)

- **Follows the same structure of Apriori**
  - Start from short patterns and find longer ones at each iteration

- **Step 1**:
  - Make the first pass over the sequence database D to yield all the 1-element frequent sequences

- **Step 2**:

  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 D to find the support for these candidate sequences
  - **Candidate Elimination**:
    - Eliminate candidate *k*-sequences whose actual support is less than *minsup*

Extracting Sequential Patterns

- Given n events: $i_1, i_2, i_3, \ldots, i_n$
  - 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:

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

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

- Remark: events within a element are ordered

  YES: $<\{i_1, i_2 , i_3\}>$ NO: $<\{i_3, i_1 , i_2\}>$

Candidate Generation

- Base case (k=2):
  - Merging two frequent 1-sequences <{i_1}> and <{i_2}> will produce two candidate 2-sequences: <{i_1} {i_2}> and <{i_1 i_2}>
  - Special case: i_1 can be merged with itself: <{i_1} {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 one 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 last two events in w_2 belong to the same element => last event in w_2 becomes part of the last element in w_1: <{d}{a}{b}> + <{a}{b,c}> = <{d}{a}{b,c}>
      - Otherwise, the last event in w_2 becomes a separate element appended to the end of w_1: <{a,d}{b}> + <{d}{b}{c}> = <{a,d}{b}{c}>
  - Special case: check if w_1 can be merged with itself
    - Works when it contains only one event type: < {a} {a}> + <{a} {a}> = < {a} {a} {a}>

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

- We **do not have** to merge the sequences
  w_1 =<{1} {2 6} {4}> and w_2 =<{1} {2} {4 5}>
  to produce the candidate < {1} {2 6} {4 5}>
  - Notice that if the latter is a viable candidate, it will be obtained by merging w_1 with < {2 6} {4 5}>

Candidate Pruning

- Based on Apriori principle:
  - If a k-sequence W contains a (k-1)-subsequence that is not frequent, then W is not frequent and can be pruned
- Method:
  - Enumerate all (k-1)-subsequence:
    - {a,b}{c}{d} → {b}{c}{d} , {a}{c}{d} , {a,b}{d} , {a,b}{c}
  - Each subsequence generated by cancelling 1 event in W
    - Number of (k-1)-subsequences = k
  - Remark: candidates are generated by merging two “mother” (k-1)-subsequences that we know to be frequent
    - Correspond to remove the first event or the last one
    - Number of significant (k-1)-subsequences to test = k – 2
    - Special cases: at step k=2 the pruning has no utility, since the only (k-1)-subsequences are the “mother” ones

GSP 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} >

GSP Exercise

- Given the following dataset of sequences

ID	Sequence				
1	a b	→	a	→	b
2	b	→	a	→	c d
3	a	→	b		
4	a	→	a	→	b d

- Generate sequential patterns if min_sup = 35%

GSP Exercise - solution

Sequential pattern	Support
a	100 %
b	100 %
d	50 %
a → a	50 %
a → b	75 %
a → d	50 %
b → a	50 %
a → a → b	50 %

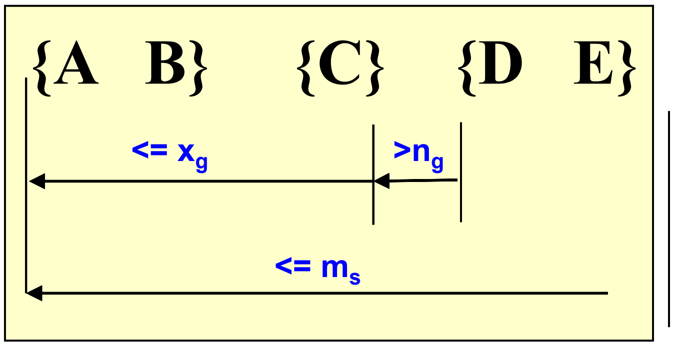
Timing Constraints

Motivation by examples:

- Sequential Pattern {milk} → {cookies}
  - It might suggest that cookies are bought to better enjoy milk
  - Yet, we might obtain it even if all customers by milk and **after 6 months** buy cookies, in which case our interpretation is wrong
- {cheese A} → {cheese B}
  - Does it mean that buying and eating cheese A induces the customer to try also cheese B (e.g. by the same brand)?
  - Maybe, yet if they are bought within 20 minutes it is like that they were to be bought together (and the customer forgot it)
- {buy PC} → {buy printer}→{ask for repair}
  - Is it a good or bad sign?
  - It depends on how much time the whole process took:
    - Short time => issues, Long time => OK, normal life cycle

Timing Constraints

- Define 3 types of constraint on the instances to consider
  - E.g. ask that the pattern instances last no more than 30 days



x_g: max-gap → Each element of the pattern instance must be **at most** x_g time after the previous one

n_g: min-gap → Each element of the pattern instance must be **at least** n_g time after the previous one

m_s: maximum span → The overall duration of the pattern instance must be at most $\mathbf{m_s}$

$x_g = 2, n_g = 0, m_s= 4$ → consecutive elements at most distance 2 & overall duration at most 4 time units

Data sequence	Subsequence	Contain?
< {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

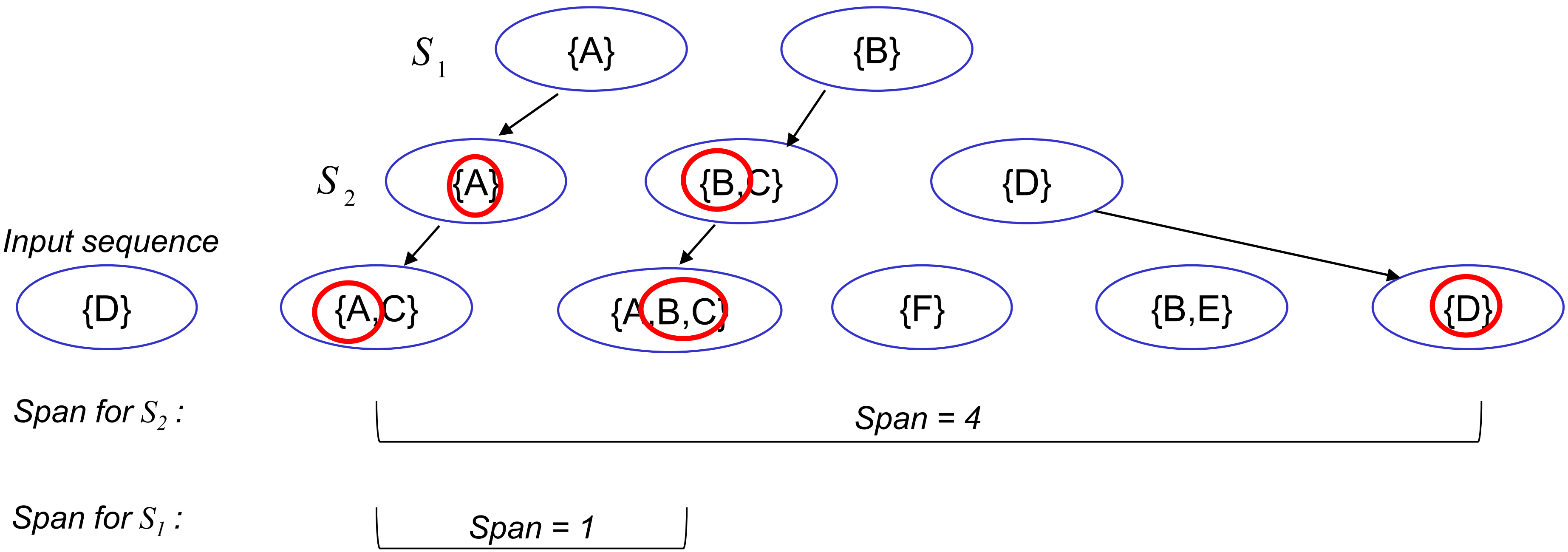
- Approach 1:
  - Mine sequential patterns without timing constraints
  - Postprocess the discovered patterns
  - Dangerous: might generate billions of sequential patterns to obtain only a few time-constrained ones

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

Apriori principle with time constraints

- **Case 1: max-span**

- **Intuitive check**
  - Does any input sequence that contains S2 will also contain S1 ?



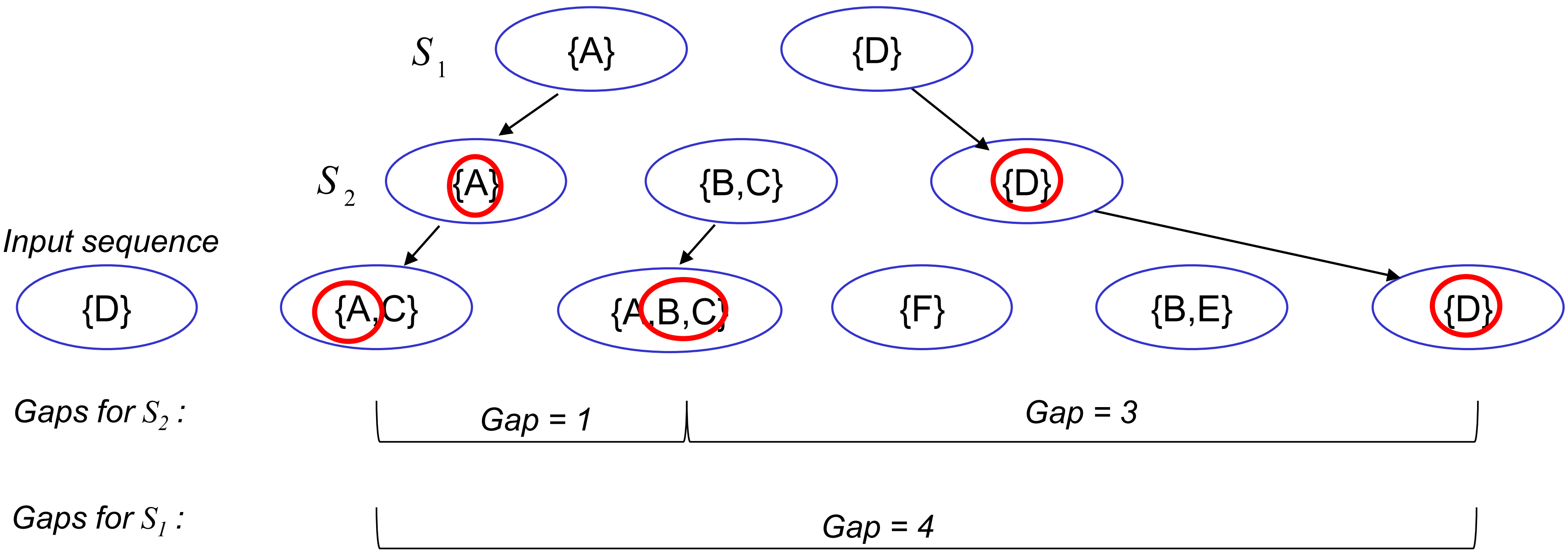
- **When S1 has less elements, S1 span can (only) decrease**
  - If S2 span is OK, then also S1 span is OK

V

Apriori principle with time constraints

- **Case 2: min-gap**

- **Intuitive check**
  - Does any input sequence that contains S2 will also contain S1 ?



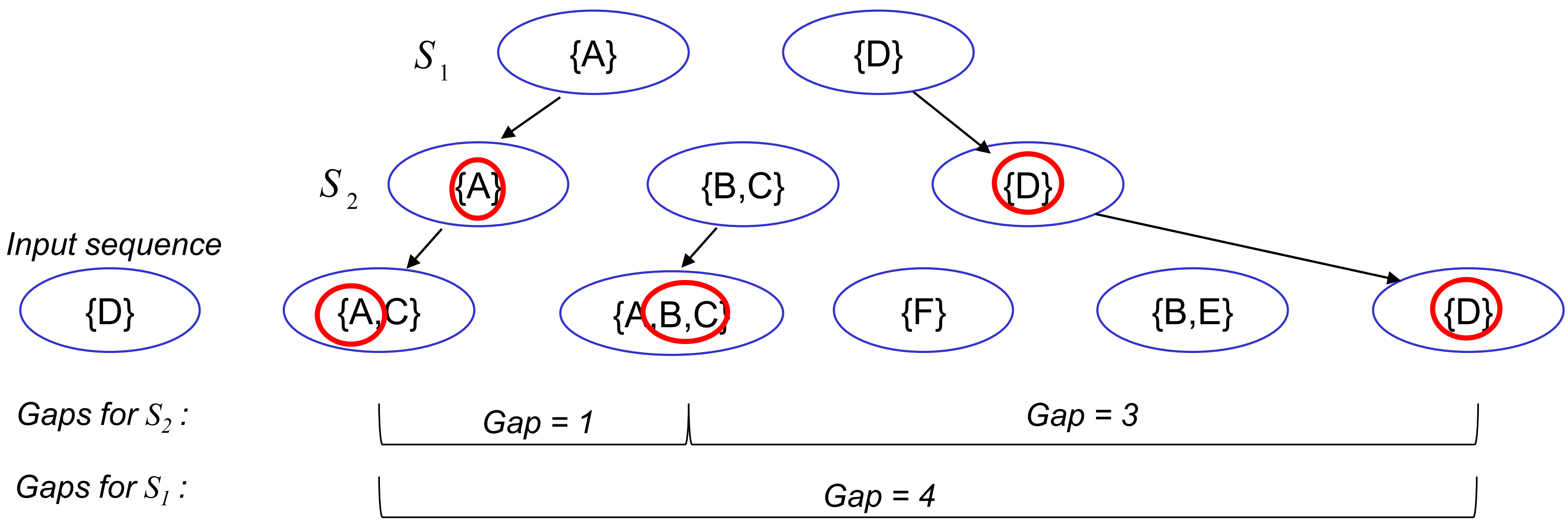
- **When S1 has less elements, gaps for S1 can (only) increase**
  - If S2 gaps are OK, they are OK also for S1

V

Apriori principle with time constraints

- **Case 3: max-gap**

- **Intuitive check**
  - Does any input sequence that contains S2 will also contain S1 ?



- **When S1 has less elements, gaps for S1 can (only) increase**
  - Happens when S1 has lost an internal element w.r.t. S2
  - Even if S2 gaps are OK, S1 gaps might grow too large w.r.t. max-gap

X

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:

$x_g = 1$ (max-gap)

$n_g = 0$ (min-gap)

$m_s = 5$ (maximum span)

minsup = 60%

<{2} {5}> support = 40%

but

<{2} {3} {5}> support = 60%

Problem exists because of max-gap constraint

No such problem if max-gap is infinite

Contiguous Subsequences

- s is a contiguous subsequence of

  w = $<e_1>< e_2>\ldots< e_k>$

  if any of the following conditions hold:

  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 more than 2 items — *Key point: avoids internal "jumps"*
  3. s is a contiguous subsequence of s' and s' is a contiguous subsequence of w (recursive definition) — *Not interesting for our usage*

- 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}>

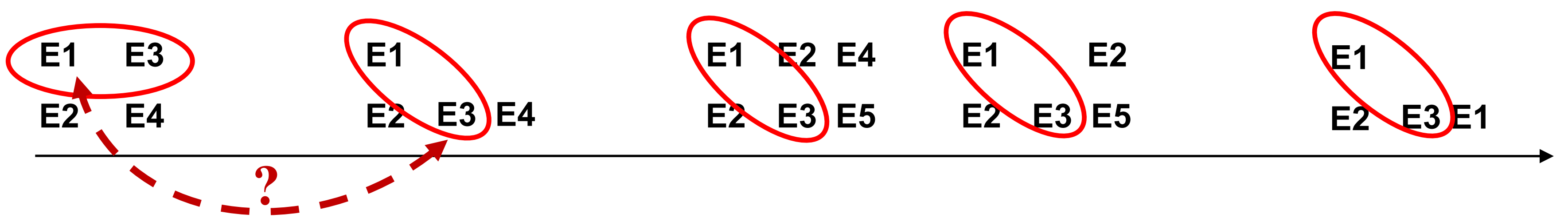
Modified Candidate Pruning Step

- 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
  - Remark: the "pruning power" is now reduced
    - Less subsequences to test for "killing" the candidate
  - Question: what is the "pruning power" when all elements are singletons?

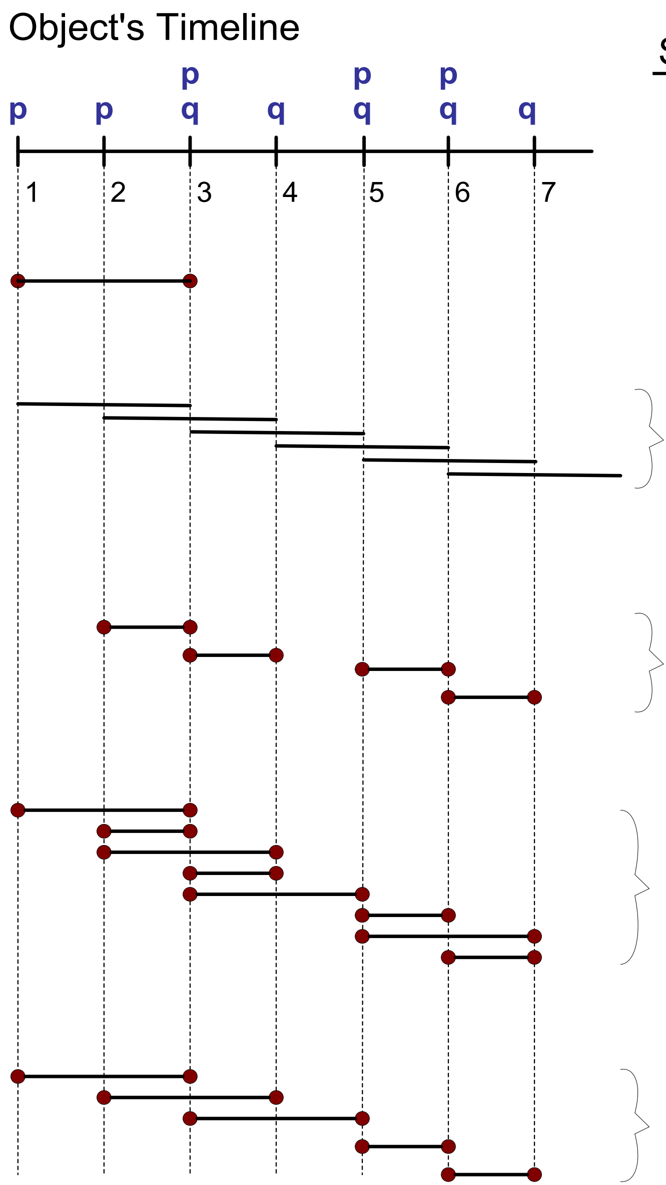
Other kinds of patterns for sequences

- In some domains, we may have only one very long time series
  - Example:
    - monitoring network traffic events for attacks
    - monitoring telecommunication alarm signals
- Goal is to find frequent sequences of events in the time series
  - Now we have to count “instances”, but which ones?
  - This problem is also known as frequent episode mining

Pattern: <E1> <E3>



General Support Counting Schemes



Sequence: (p) (q)

Method	Support Count
COBJ	1
CWIN	6
CMINWIN	4
CDIST_O	8
CDIST	5

Assume:

$x_g = 2$ (max-gap)

$n_g = 0$ (min-gap)

$m_s = 2$ (maximum span)