Pattern Mining for Complex Data

(DMV Lecture, M2 SIF)

Peggy Cellier

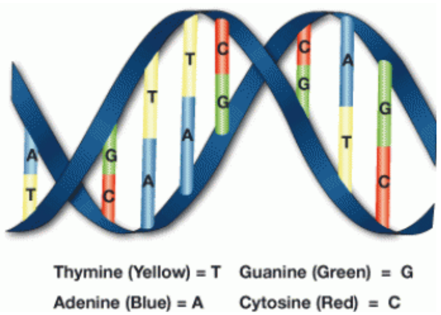
peggy.cellier@irisa.fr

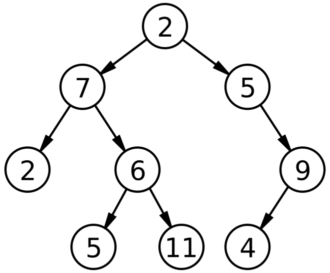
Last revision: October 2021

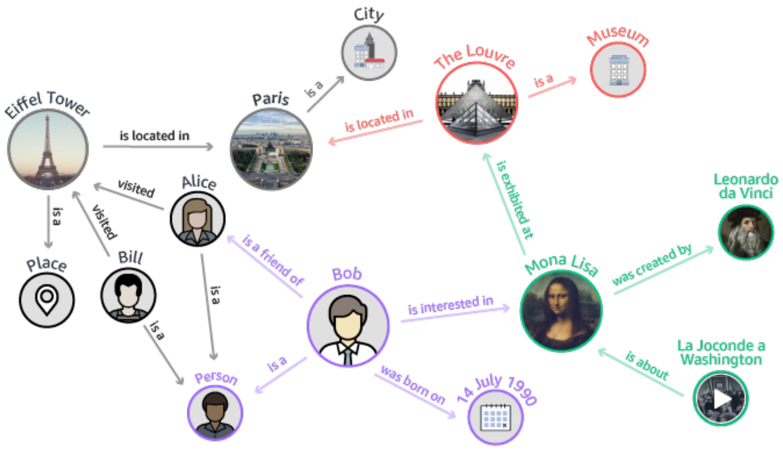
Introduction

- **Reminder (Alexandre’s lecture):**
  - Patterns = local regularities in data
  - Frequent itemsets = regularities in transactional data (sets of elements)

- **Other data?**
  - Many types: sequences, trees, graphs, intervals…
  - More structured than sets (i.e. *more relations between elements*)
    - Also have regularities !







→ **need to extend pattern mining to structured data**

Problems due to data complexity

- **Pattern identification in data**
  - FIS: simple set inclusion operation $\subseteq$
  - Structured data:
    - **Many possible inclusion definitions** for sequences, trees, graphs…
    - Inclusions may be computationally **expensive**

- **Support counting**
  - Possible overlap between found occurrences
  - → **how to count support?**

- **Complexity**
  - FIS: $O(2^{\#items})$
  - Structure data: **search space may be exponentially bigger**!
    - *More precise values depend on problem*

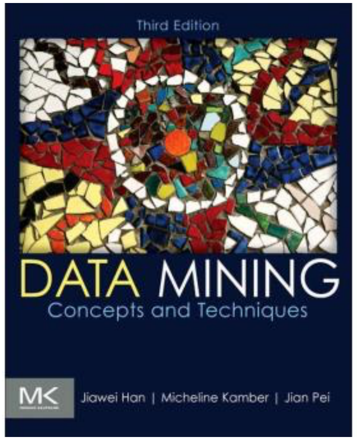
Schedule of this lecture

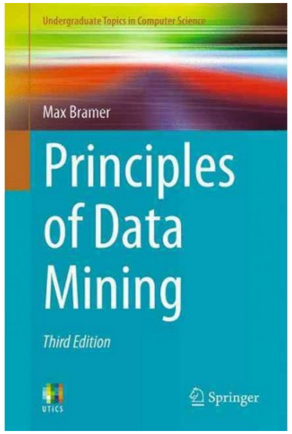
- **Sequential Pattern Mining**

- **Graph Mining**

References and course material

- [1] « Data mining, Concepts and techniques 2nd/3rd edition» - J. Han, M. Kamber and J. Pei (2011)
- [2] « The data mining and knowledge discovery handbook » - Oded Maimon and Lior Rokach (2005)
- [3] Marc Plantevit’s lectures (2009)
- [4] « Principle of data mining » - M. Bramer (2007)
- [5] « Apprentissage artificiel » - A. Cornuéjols and L. Miclet (2003)
- [6] « Relational Data Mining » - S. Dzeroski and N. Lavrac (2001)
- [7] Alexandre Termier’s lectures (2017)
- [8] Davide Mottin, Anton Tstitsulin’s lectures (2017) – Hasso Plattner Institute




Undergraduate Topics in Computer Science
Max Bramer
Principles of Data Mining
Third Edition
UTiCS
Springer

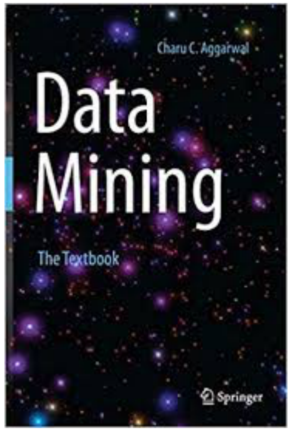


Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

II. Definitions
   - I. Vocabulary
   - II. Sequence database
   - III. Frequent sequential patterns
   - IV. Problem definition
   - V. Discussion about time parameters

III. Sequence mining algorithms
   - I. Search space
   - II. Apriori based approaches (Generate & Prune)
      - I. GSP
      - II. SPADE
   - III. Pattern growth Approach
      - I. PrefixSpan

IV. To go further
   - I. Closed and Maximal Sequential Patterns
   - II. Mining sequential patterns with gap constraints
   - III. Episode mining (Winepi)

Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

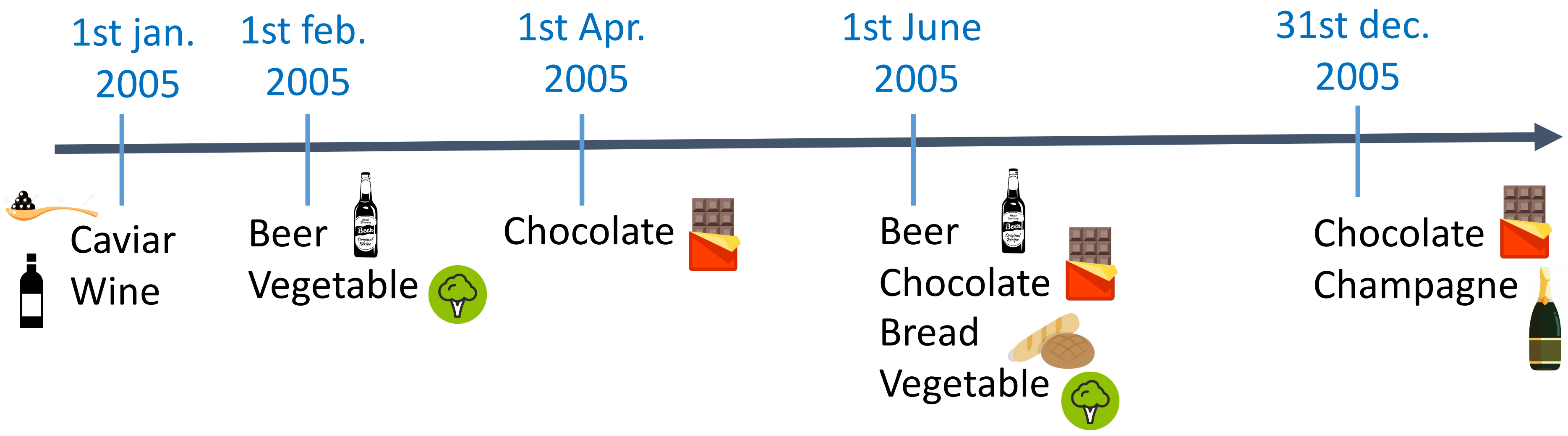
II. Definitions
- I. Vocabulary
- II. Sequence database
- III. Frequent sequential patterns
- IV. Problem definition
- V. Discussion about time parameters

III. Sequence mining algorithms
- I. Search space
- II. Apriori based approaches (Generate & Prune)
  - I. GSP
  - II. SPADE
- III. Pattern growth Approach
  - I. PrefixSpan

IV. To go further
- I. Closed and Maximal Sequential Patterns
- II. Mining sequential patterns with gap constraints
- III. Episode mining (Winepi)

Sequential data: What are we looking for?

- **Example: Let us consider data from retail**
  - Products bought by a customer



1st jan. 2005
Caviar
Wine

1st feb. 2005
Beer
Vegetable

1st Apr. 2005
Chocolate

1st June 2005
Beer
Chocolate
Bread
Vegetable

31st dec. 2005
Chocolate
Champagne

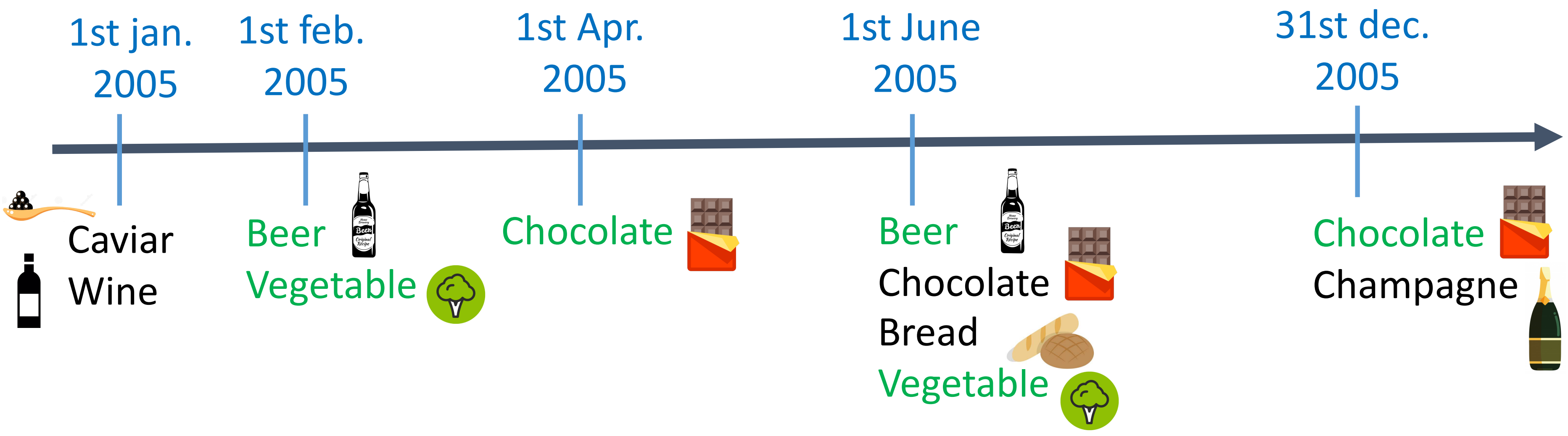
What are we looking for?

Repetitions

considering chronology between transactions

Sequential data: What are we looking for?

- **Example: Let us consider data from retail**
  - Products bought by a customer



1st jan. 2005
Caviar
Wine

1st feb. 2005
Beer
Vegetable

1st Apr. 2005
Chocolate

1st June 2005
Beer
Chocolate
Bread
Vegetable

31st dec. 2005
Chocolate
Champagne

What are we looking for?

Example: <(**Beer Vegetable**) (**Chocolate**)>

Sequential patterns

Informally

<(A B) C (D E)>

↓

$$\underbrace{A,B \rightarrow C \rightarrow D,E}_{\text{1 month}}$$

Read as:

people who buy **A** and **B**

then buy **C**

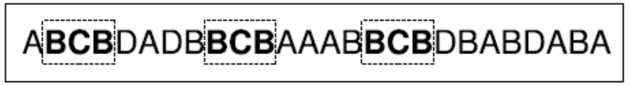
and then buy **D** and **E**

in a month

(Some) types of sequential patterns

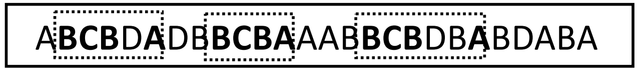
- **Substrings**

B → C → B



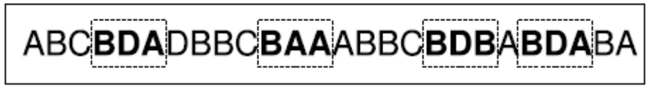
- **Sequences with gaps**

B → C → B → A



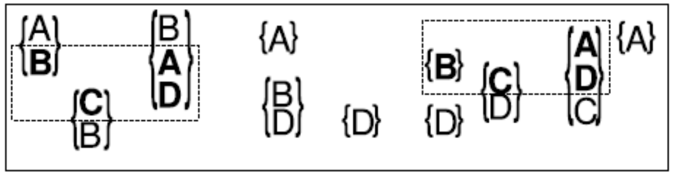
- **Regular expressions**

B → ¬C → A|B

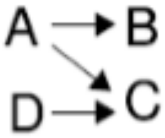


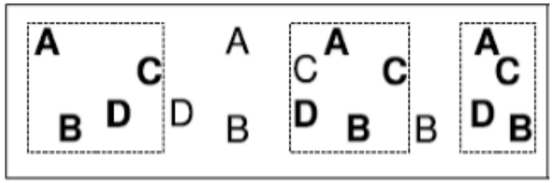
- **Sequences of itemsets**

{B} → {C} → {A,D}



- **Episodes**





Images from F. Moerchen

Application area

- **Bioinformatics**
  - ex: patterns = parts of DNA sequences
- **Health**
  - ex: patterns = health care pathways
- **Debugging**
  - ex: patterns = sequences of instructions / functions calls
- **Marketing**
  - ex: patterns = customer buying habits in time
- **…**

Table of Contents

Sequential Pattern Mining

- I. Introduction: what are we looking for
- II. Definitions
  - I. Vocabulary
  - II. Sequence database
  - III. Frequent sequential patterns
  - IV. Problem definition
  - V. Discussion about time parameters
- III. Sequence mining algorithms
  - I. Search space
  - II. Apriori based approaches (Generate & Prune)
    - I. GSP
    - II. SPADE
  - III. Pattern growth Approach
    - I. PrefixSpan
- IV. To go further
  - I. Closed and Maximal Sequential Patterns
  - II. Mining sequential patterns with gap constraints
  - III. Episode mining (Winepi)

Vocabulary

- **Vocabulary (reminder)**
  - Let $I=\{i_1,\ldots,i_n\}$ be the set of all items.
  - An itemset is a subset of I and denoted $(i_1 i_2 \ldots i_m)$ where $i_k \in I$

- **Sequence**
  - A sequence s is an **ordered** list of itemsets denoted by $<s_1 s_2 \ldots s_p>$
  - Order can be:
    - **Implicit**: position of elements
      - Ex: DNA - ACCGT ⇔ <A, C, C, G, T>
    - **Explicit**: elements + timestamps
      - Ex: Log - <(1, pushButton), (2, endOfWorld)>

- **k-sequence**
  - A k-sequence is a sequential pattern of length k (k items).
  - Examples
    - <(a b) (c) (d e)> is a **5**-sequence.
    - <(a) (c) (d e)> is a **?**-sequence.
    - <(a) (c) (d) (z) (y)> is a **?**-sequence.

Sequence Database

- A sequence database consists of ordered elements or events

transaction database vs sequence database

TID	itemsets
10	a b d
20	a c d
30	a d e f
40	e f

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

Note: **Implicit** timestamp here

Sequence Database

- **Dataset**
  - Transactions → Sequences of itemsets with **timestamp** (date)

- **Example**

Date / SeqId	Monday	Tuesday	Wednesday	Thursday
S_1	abc	bde	abf	ad
S_2	abc	abc	-	bcf
S_3	bce	-	adf	abc
S_4	acf	bd	abf	e

seq. inclusion – sub-sequence – super-sequence

- **Sequence inclusion**
  - Let S_1=<a_1,…, a_n> *and* S_2=<b_1,…,b_m> be two sequences.
  - S_1 is a sub-sequence of S_2 or S_2 is a super-sequence of S_1
  - denoted by $S_1 \subseteq S_2$
  - If there are integers $1 \leq i1 < i2 < \ldots < in \leq m$ s.t. $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, …, $a_n \subseteq b_{in}$
  - Examples
    - S1=<(10) (20 30) (40) (20)>
    - S2=<(20) (40)> $\subseteq$ **S1 ?**
    - S3=<(20) (30)> $\subseteq$ **S1 ?**

Sequential Patterns

- **Sequential pattern**
  - A **sequential pattern** is defined as a sequence <$X_1, \dots, X_n$>
  - where X_i is an itemset.
  - Example
    - <(a b) (c) (d e)>
      - a and b are synchronous
      - d and e are synchronous

      ===> they share the same timestamp
      - c happens after a and b
      - d and e happen after c
- **Support**
  - A sequence *S* supports a sequential pattern *P* if $\boldsymbol{P \subseteq S}$.
  - The **support** value of P, denoted by ***sup(P)*** is then defined as the proportion of sequences supporting P.
- **Frequent sequential pattern**
  - A sequential pattern S is **frequent** if ***sup(S) >= minsup***
    - where minsup is a given threshold

Example of sequential patterns

seq./date	d_1	d_2	d_3	d_4
S_1	abc	bde	abf	ad
S_2	abc	abc	-	bcf
S_3	bce	-	adf	abc
S_4	acf	bd	abf	e

- sup(<(ac) (b) (bf)>)

- **Exercise:** Compute the support value of the following sequential patterns
  - <(a) (bd) (a)>
  - <(b) (b) (f)>
  - <(b) (d) (f)>
  - <(cf) (b)>

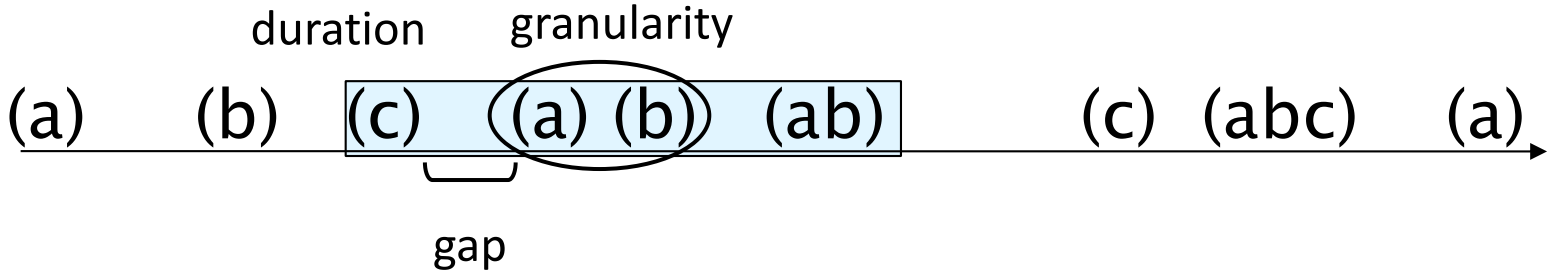
Sequential pattern mining: problem definition

- **Given**
  - a sequence database: *D*
  - the minimum support threshold: *minsup*

- **Problem definition**
  - The problem of sequential patern mining is to find the set of **all** frequent subsequences from *D* wrt *minsup*.

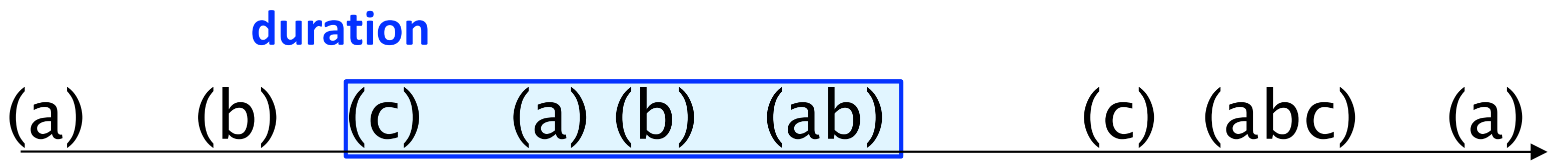
Discussion about time parameters

- **3 main time parameters/constraints**
  1. Duration of sequences (data preparation)
  2. Granularity of itemsets (data preparation)
  3. Time gap between itemsets



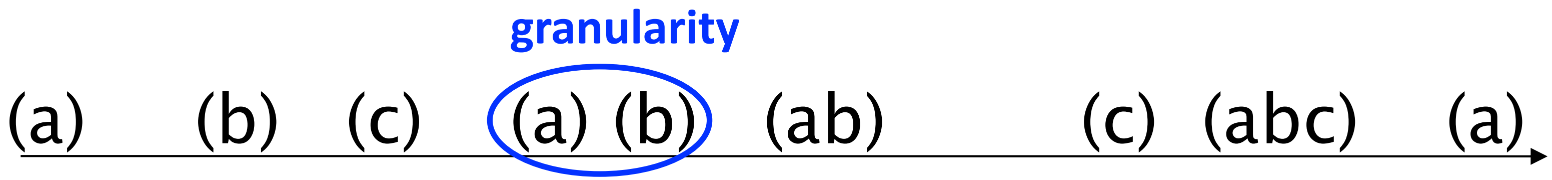
Duration of sequences

- **Duration of sequences**
  - Chunking size of target sequences
  - Preprocessing
  - Examples
    - Complete sequences
    - Specified time interval
    - Split into years, months…
  - Last chunking strategy enables periodical sequential patterns
    - “Each year, a wet spring results in increased bookings of travels abroad in summer”



Event folding window

- **Event folding window**
  - Atomicity of transactions happening within a given time interval
  - Preprocessing
  - “Which time unit?”
  - Examples
    - Grocery: sales of a week
    - Travel agency: travels purchased during a year



granularity

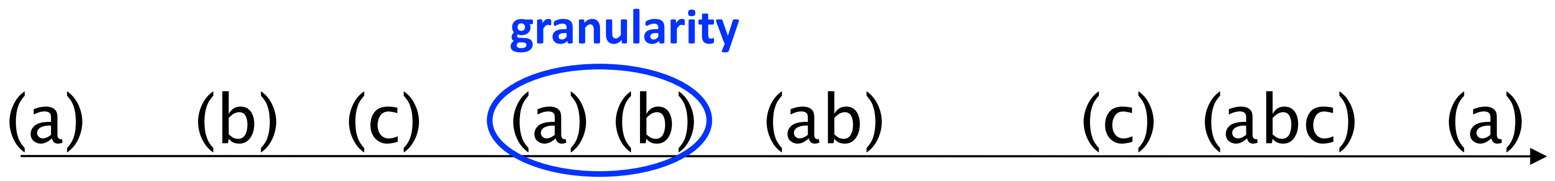
(a) (b) (c) (a) (b) (ab) (c) (abc) (a)

Event folding window

Event folding window => Important choice

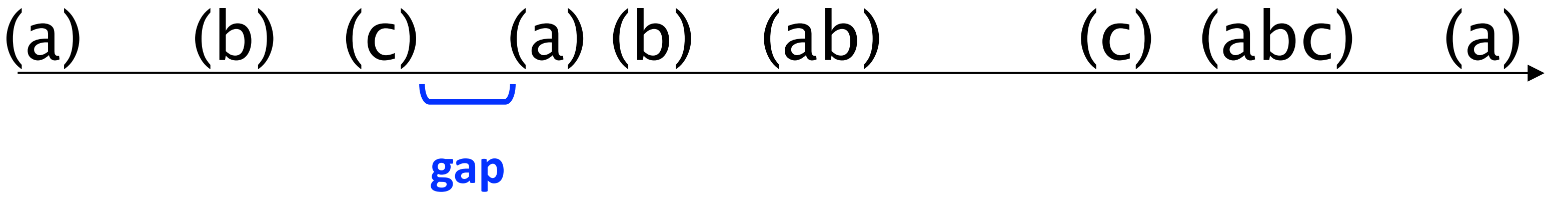
- Too short interval ⇒ low support sequences
  - Example: sequences with a too fine grain
    - <A,B,C> or <B,A,C> instead of having <AB,C>

- Too long interval ⇒ no more (or less) sequentiality
  - Example: Sequence with a big grain
    - <AB> instead of <A,B>
    - ordering between A and B has disappeared



Gap constraint

- **Time gap between itemsets**
  - Number of time units between successive itemsets of sequential patterns
    - Until which time gap do one still consider that there is sequentiality?
  - Intuitively, delete too far events

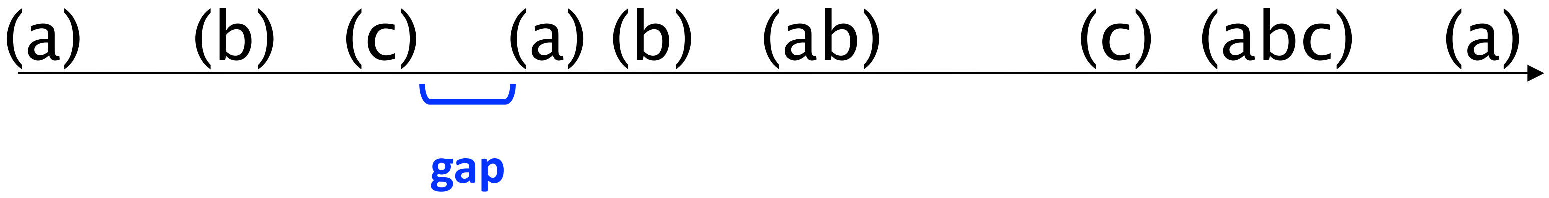


(a) (b) (c) (a) (b) (ab) (c) (abc) (a)

gap

Gap constraint

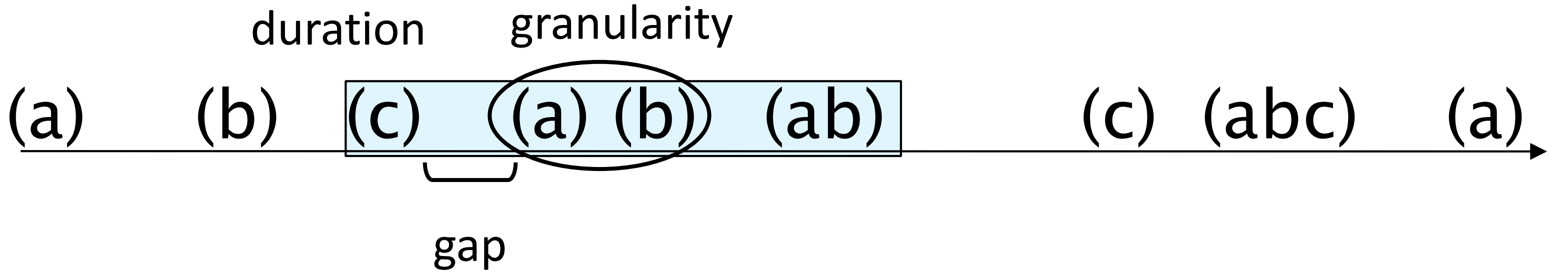
- **Time gap between itemsets**
  - Number gap=0 => contiguous
    - transactions succeed immediately
    - E.g., "sales of A, B, C in 3 successive weeks" (time unity is the week)
  - $\text{gap}_{\text{min}} \leq \text{gap} \leq \text{gap}_{\text{max}}$
    - Transaction cannot be too close nor to far
    - E.g., "If someone rents the movie *Matrix reloaded*, he may probably also rent *Matrix revolutions* within the 15 days" (time unity is the day)
  - Infinite gap
    - Only sequentiality



About constraints

- **Application of time constraints**
  - **Duration** and **granularity** are usually applied **before** the extraction
    - To prepare the sequence database
  - Whereas **gap** is used when mining
    - To extract the sequential patterns
- **Other constraints**
  - Time-relative constraints are only some of possible constraints

  => **Other constraints**
    - incompatibility between items
    - templates (regular expressions)
    - length of patterns
    - ...



Exercise

- **Exercise**
  - Consider the following parameter to extract patterns
    - Time gap = [0,1]
  - Compute the support values of
    - <(a) (bd) (a)> = <a (bd) a>
    - <(b) (b) (f)> = <b b f>
    - <(b) (d) (f)> = <b d f>
    - <(cf) (b)> = <(cf) b>

Seq./t	t=1	t=2	t=3	t=4	t=5	t=6
S_1	abc	b	de	af	b	ad
S_2	abc	bc	a	bcf		
S_3	bce	adf	e	abc	f	
S_4	acf	bd	abf	e		

Table of Contents

Sequential Pattern Mining

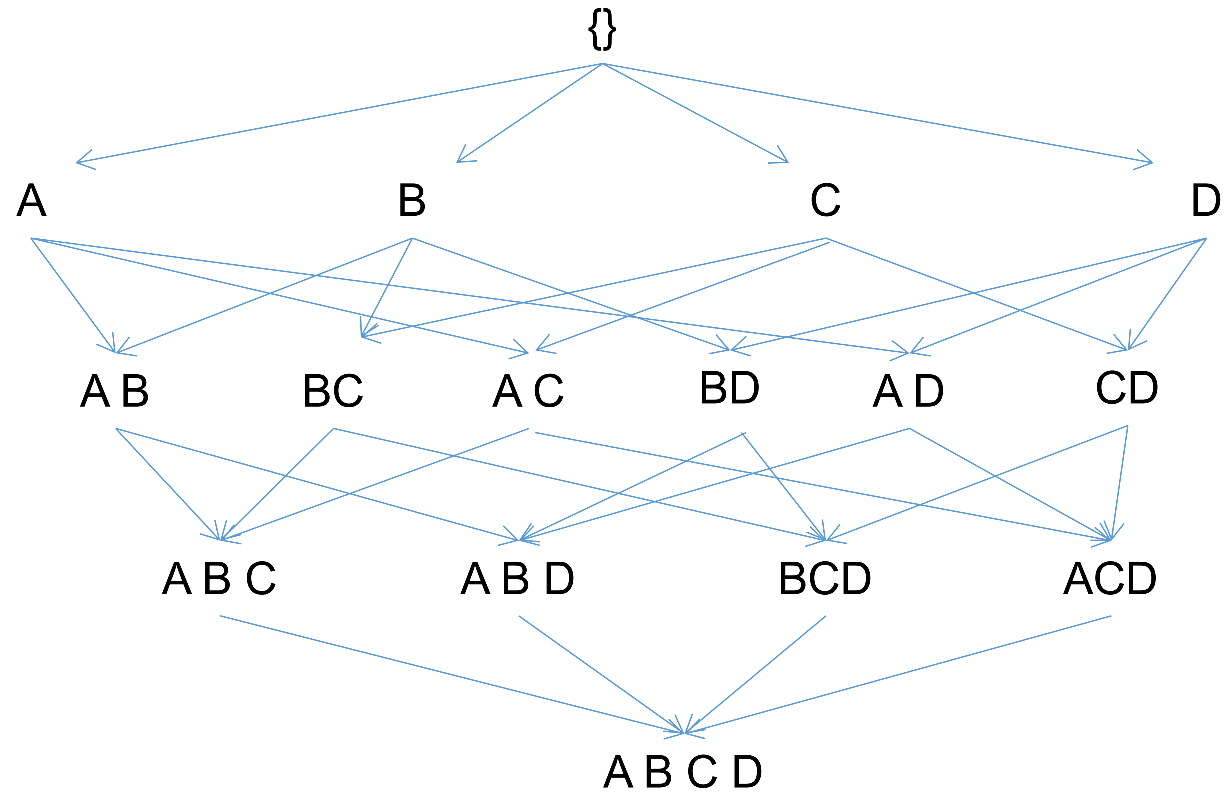
I. Introduction: what are we looking for

II. Definitions
   - I. Vocabulary
   - II. Sequence database
   - III. Frequent sequential patterns
   - IV. Problem definition
   - V. Discussion about time parameters

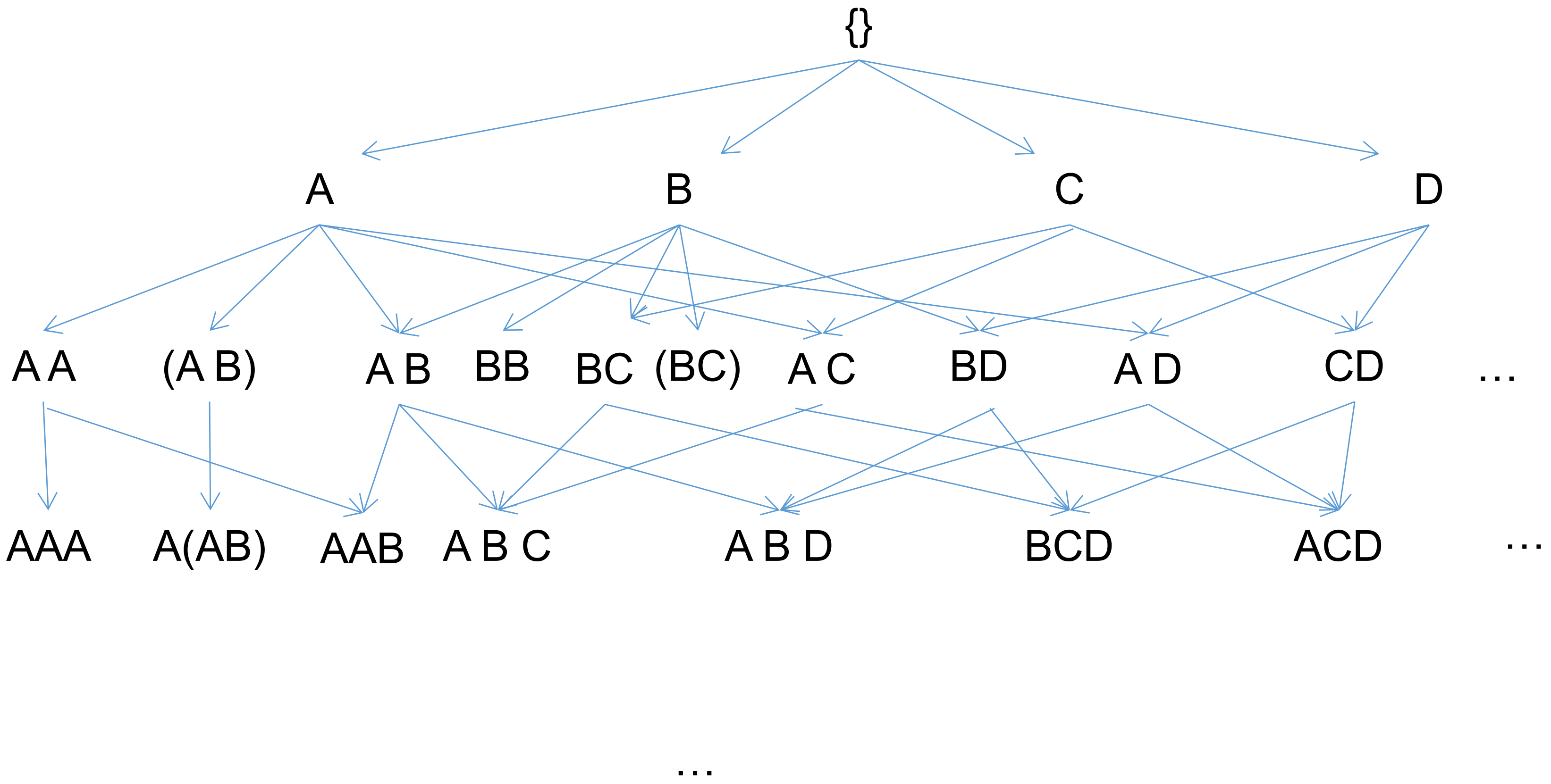
III. Sequence mining algorithms
   - I. Search space
   - II. Apriori based approaches (Generate & Prune)
     - I. GSP
     - II. SPADE
   - III. Pattern growth Approach
     - I. PrefixSpan

IV. To go further
   - I. Closed and Maximal Sequential Patterns
   - II. Mining sequential patterns with gap constraints
   - III. Episode mining (Winepi)

Search Space for itemset mining (lattice)



Search Space for sequential pattern mining



Sequential Pattern Mining Algorithms

- **Apriori-based Algorithms (also named Generate & Prune)**
  - Horizontal Data Format Algorithms
    - **GSP** (hash tree)
    - PSP (prefix tree – less memory)
  - Vertical Data Format Algorithms
    - **SPADE**
    - SPAM
    - LAPIN-SPAM

- **Pattern Growth Algorithms**
  - FreeSpan
  - **PrefixSpan**

- **Extensions**
  - Closure
    - CloSpan
    - BIDE
    - Gap-BIDE
    - Clasp
  - Episode Mining
    - Minepi, **Winepi**
  - Constraints
    - SPIRIT
    - SDMC

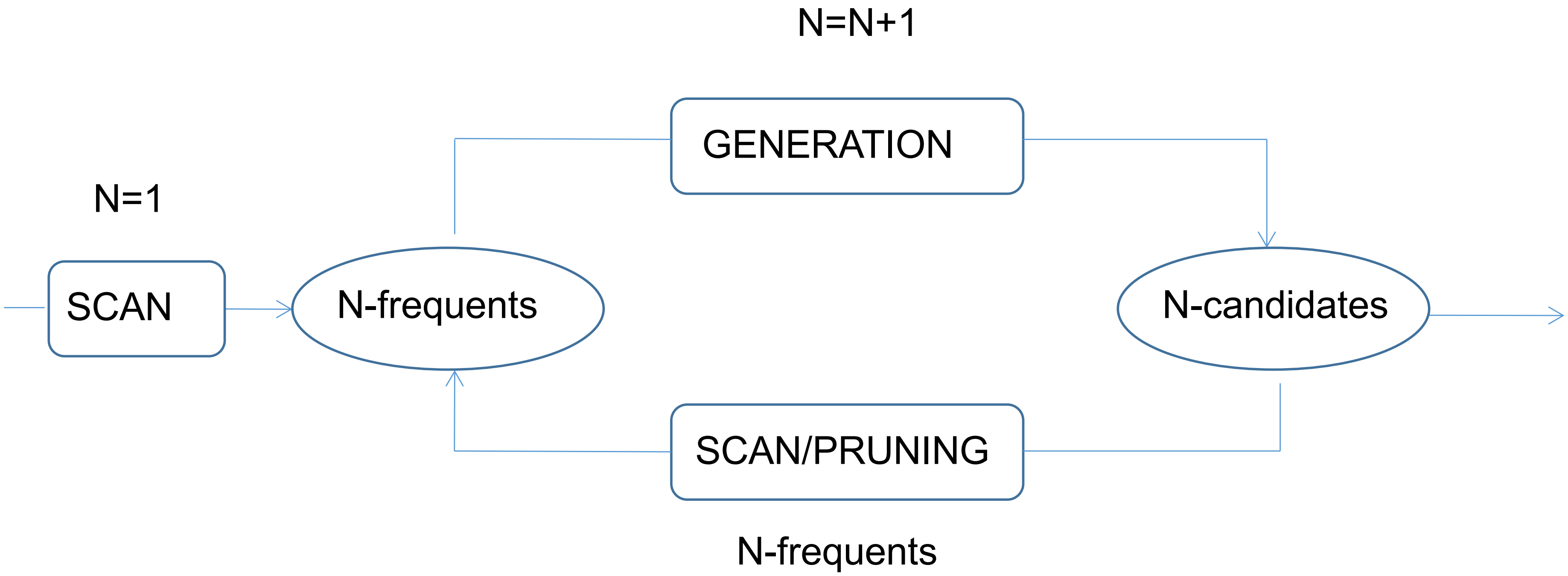
Table of Contents

Sequential Pattern Mining

- I. Introduction: what are we looking for
- II. Definitions
  - I. Vocabulary
  - II. Sequence database
  - III. Frequent sequential patterns
  - IV. Problem definition
  - V. Discussion about time parameters
- III. Sequence mining algorithms
  - I. Search space
  - II. Apriori based approaches (Generate & Prune)
    - **I. GSP**
    - II. SPADE
  - III. Pattern growth Approach
    - I. PrefixSpan
- IV. To go further
  - I. Closed and Maximal Sequential
  - II. PatternsMining sequential patterns with gap constraints
  - III. Episode mining (Winepi)

General Approach: generate/prune

- **GSP (Generalized Sequential Pattern) mining algorithm**
  - [Agrawal and Srikant, EDBT' 96]
  - In the same vein as Apriori for frequent itemset mining
  - GSP is a horizontal data format based SPM algorithm.



GSP: based on Apriori

```
N=0
While ($\text{Result}_N$ != NULL)
        N = N+1
        Generate candidates ($\text{Candidates}_N$)
        Prune candidates ($\text{Result}_N$)
        Result = Result U $\text{Result}_N$

Result is the whole set of sequential patterns
```

- **Requirements:**
  - 2 kinds of extensions => to generate candidates
  - the anti-monotony property => to prune candidates

2 kinds of extension

S-extension

Add an itemset to the sequence

Example: <(a,b)(c)> → <(a,b)(c)(d)>

I-extension

Add an item into an existing itemset of the sequence

Example: <(a,b)(c)> → <(a,b)(c,d)>

Anti-monotony property

- Property:
  - If a k-sequence is not frequent
  - THEN all (k+1) sequences which contain it are not frequent too.

- **Example:**
  - IF *sup*(<(A),(B,C)>) < *minsup*
  - THEN *sup*(<(A),(B,C),(D)>) << *minsup*

- **This property allows to adapt Apriori to extract**
  - Frequent sequential patterns
  - (and thus temporal association rules)

GSP: based on Apriori

- **Method in details**
  - generate frequent length-1 candidates from frequent items in DB : <A>, <B>
  - generate frequent length-2 candidates by self-joining 2 frequent length-1 patterns: <(A) (A)>, <(A) (B)>, <(A B)>
  - 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 (self-join)
  - repeat until no frequent sequence or no candidate can be found
- **Major strength: Candidate pruning by Apriori property (anti-monotonicity)**
- **Self-join s_1 et s_2:**
  - Remove first element of s1 (s1-first$_{s1}$) and last element of s2 (s_2-last$_{s2}$)
  - If (s_1-first$_{s1}$) = (s_2-last$_{s2}$) then generate s_1+last$_{s2}$
  - Examples

< (A B) (C) >
+ < (B) (C D)>
< (A B) (C D) >

< (A B) (C) >
+ < (B) (C) (E)>
< (A B) (C) (E) >

Example (GSP)

- **Sequence database**
  - 8 items
  - 5 sequences
  - (minsup=2)

Id_seq	Séquence
1	<(bd) (c) (b) (ac)>
2	<(bf) (ce) (b) (fg)>
3	<(ah) (bf) (a) (b) (f)>
4	<(be) (ce) (d)>
5	<(a) (bd) (b) (c) (b) (ade)>

Example (GSP)

- **Sequence database**
  - 8 items
  - 5 sequences
  - (minsup=2)

Id_seq	Séquence
1	<(bd) (c) (b) (ac)>
2	<(bf) (ce) (b) (fg)>
3	<(ah) (bf) (a) (b) (f)>
4	<(be) (ce) (d)>
5	<(a) (bd) (b) (c) (b) (ade)>

- **N=1**
- **Candidate generation**
- **Pruning unfrequent patterns**
  - 6 frequent sequences with 1 item

Candidate	Support
<a>	3
<b>	5
<c>	4
<d>	3
<e>	3
<f>	2
~~<g>~~	~~1~~
~~<h>~~	~~1~~

Example (GSP)

- **N=2**
- **Candidate generation**
  - 51 sequences with 2 items

S-extension

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<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>

I-extension

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

Remark:

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

Apriori property prunes 44.57% candidates

The most time consuming step of GSP

- **Computation of the candidate support**
  - Candidates stored in main memory
- **It’s important to limit the disk access**
  - Load the sequence database in memory when it’s possible

Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

II. Definitions
  - I. Vocabulary
  - II. Sequence database
  - III. Frequent sequential patterns
  - IV. Problem definition
  - V. Discussion about time parameters

III. Sequence mining algorithms
  - I. Search space
  - II. Apriori based approaches (Generate & Prune)
    - I. GSP
    - **II. SPADE**
  - III. Pattern growth Approach
    - I. PrefixSpan

IV. To go further
  - I. Closed and Maximal Sequential Patterns
  - II. Mining sequential patterns with gap constraints
  - III. Episode mining (Winepi)

SPADE

- **SPADE (Sequential Pattern Discovery using Equivalent classes)**
  - [Zaki, ML'01]
  - SPADE is a SPM algorithm based on a vertical data format.

Vertical format

SID	Séquence
1	<(bd) c b (ac)>
2	<(bf) (ce) b (fg)>
3	<(ah) (bf) a b f>
4	<(be) (ce) d>
5	<a (bd) b c b (ade)>

a		b		c		d		e		f		g		h	
SID	EID	SID	EID	SID	EID	SID	EID	SID	EID	SID	EID	SID	EID	SID	EID
1	4	1	1	1	2	1	1	2	2	2	1	2	4	3	1
3	1	1	3	1	4	4	3	4	1	2	4				
3	3	2	1	2	2	5	2	4	2	3	2				
5	1	2	3	4	2	5	6	5	6	3	5				
5	6	3	2	5	4										
		3	4												
		4	1												
		5	2												
		5	3												
		5	5												

SPADE algorithm

- **Algorithm**
  - Scan DB and then transforms the database into the vertical format
  - Filter non frequent 1-sequences (count the number of =/= SID)
    - Example with minsup=4: Frequent 1-sequences: <b>, <c>

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

c	
SID	EID
1	2
1	4
2	2
4	2
5	4

- Repeat until no more sequences can be generated
  - Join k-sequences such that they share SID and the EIDs follow the sequential ordering

SPADE algorithm

- **Algorithm**
  - Scan DB and then transforms the database into the vertical format
  - Filter non frequent 1-sequences (count the number of =/= SID)
    - Example with minsup=4: Frequent 1-sequences: <b>, <c>
  - Repeat until no more sequences can be generated
    - Join k-sequences such that they share SID and the EIDs follow the sequential ordering
    - Filter non frequent (k+1)-sequences (count the number of =/= SID)
- **To reduce space memory**
  - Join two k-sequences that have all subsequences in common except the last element (cf itemset => lexicographical improvement)
  - store only one EID, the one of the last element
  - lattice decomposition (class of sequences)

Drawbacks of generate/prune approaches

- **A lot of irrelevant candidates are generated**
  - For instance, for 1000 frequent sequences with 1 item, the number of candidate sequences with 2 items is:
    - 1000 x 1000 x (1000 x 999)/2 = 1 499 500
  - Several readings of the sequence database
  - Beam search approach is memory-consuming

- **To extract long sequences, that kind of approaches is not adapted**
  - Exponential number of candidate subsequences are generated
    - E.g., for a 100-sequence: 2^{100} - 1 ≈10^{30}

Table of Contents

Sequential Pattern Mining

- I. Introduction: what are we looking for
- II. Definitions
  - I. Vocabulary
  - II. Sequence database
  - III. Frequent sequential patterns
  - IV. Problem definition
  - V. Discussion about time parameters
- III. Sequence mining algorithms
  - I. Search space
  - II. Apriori based approaches (Generate & Prune)
    - I. GSP
    - II. SPADE
  - III. Pattern growth Approach
    - **I. PrefixSpan**
- IV. To go further
  - I. Closed and Maximal Sequential Patterns
  - II. Mining sequential patterns with gap constraints
  - III. Episode mining (Winepi)

General Idea of Pattern Growth Approaches

- No candidate generation
- Frequent items are extracted from projected bases
- Greedy algorithm
- [Pei et al, ICDE’01]

General Idea: PrefixSpan (Pei *et al.* @ICDE' 01)

- Use frequent prefix to divide the search space and compute projected bases

- Look for only relevant sequences

Definition

- **Definition: suffix**
  - Let S=<I1, …, In> be a sequence.
  - Let S'=<I'1, …, I'm> be a subsequence of S.
  - S'' =<Jo, …, Jn> is a suffix of S w.r.t. S' if:
    - <I1, …, Io> is the smallest prefix that contains S'
    - And all items from (Jo – I'm) are ordered after element of I'm in Io.

- **Examples**
  - S = <(a) (abc) (ac) (d) (cf)>

  - Suffix(<a>) = <(abc) (ac) (d) (cf)>
  - Suffix(<(a)(b)>) = <(c) (ac) (d) (cf)>

Projected base

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

Prefix	Projection
<a>	<(abc)(ac)d(cf)>
	<(_d)c(bc)(ae)>
	<(_b)(df)cb>
	<(_f)cbc>

PrefixSpan (Pei *et al.* @ICDE' 01)

- **Informal algorithm**
  - **Step 1**:
    - Extraction of **frequent 1-sequences**
      - Example: <a>, <b>, <c>, <d>, <e>, <f>, <g>
      - The set of sequential patterns is thus divided into 7 subsets
        - Ones that start with <a>
        - Ones that start with <b>
        - Ones that start with <c>
        - Ones that start with <d>
        - Ones that start with <e>
        - Ones that start with <f>
        - Ones that start with <g>
  - **Step 2**:
    - Computation of the **projected base** for each prefix
  - **Step 3**:
    - For each prefix, computation of candidates to be **an extension**.
    - The frequent candidates are added and the extension becomes a new prefix.
    - Go to Step 2
  - **End**: No more prefix can be generated

Projected base

- **Exercise**
  - minsup=4 (absolute support) equivalent to relative support 4/4=1 (100%)
  - Apply PrefixSpan on the following database

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

Projected base

- **Exercise**

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

- Step 1: frequent 1-sequences
  - Sup(<a>)=4
  - Sup(<b>)=4
  - Sup(<c>)=4
  - ~~Sup(<d>)=3~~
  - ~~Sup(<e>)=3~~
  - ~~Sup(<f>)=3~~
  - ~~Sup(<g>)=1~~

Projected base

- Step 2(1): Projected databases
  - Prefix: <a>

Id_seq	Projected DB
10	<(abc)(ac)d(cf)>
20	<(_d)c(bc)(ae)>
30	<(_b)(df)ccb>
40	<(_f)cbc>

  - Prefix: <b>

Id_seq	Projected DB
10	<(_c)(ac)d(cf)>
20	<(_c)(ae)>
30	<(df)ccb>
40	<c>

  - Prefix: <c>

Id_seq	Projected DB
10	<(ac)d(cf)>
20	<(bc)(ae)>
30	<cb>
40	<bc>

Projected base

- Step 3(1): item-extensions
  - Prefix: <a>
    - b
    - c
  - Prefix: <b>
    - ∅
  - Prefix: <c>
    - c

Id_seq	Projected DB
10	<(abc)(ac)d(cf)>
20	<(_d)c(bc)(ae)>
30	<(_b)(df)ccb>
40	<(_f)cbc>

Id_seq	Projected DB
10	<(_c)(ac)d(cf)>
20	<(_c)(ae)>
30	<(df)ccb>
40	<c>

Id_seq	Projected DB
10	<(ac)d(cf)>
20	<(bc)(ae)>
30	<cb>
40	<bc>

Projected base

- Step 2(2): projected database
  - Prefix: <ab>

Id_seq	Projected DB
10	<(_c)(ac)d(cf)>
20	<(_c)(ae)>
30	<>
40	<c>

  - Prefix: <ac>

Id_seq	Projected DB
10	<(ac)d(cf)>
20	<(bc)(ae)>
30	<cb>
40	<bc>

  - Prefix: <cc>

Id_seq	Projected DB
10	<d(cf)>
20	<(ae)>
30	<b>
40	<>

Projected base

- Step 3(2): item-extensions
  - Prefix: <ab>
    - ø
  - Prefix: <ac>
    - c
  - Prefix: <cc>
    - ø

Id_seq	Projected DB
10	<(_c)(ac)d(cf)>
20	<(_c)(ae)>
30	<>
40	<c>

Id_seq	Projected DB
10	<(ac)d(cf)>
20	<(bc)(ae)>
30	<cb>
40	<bc>

Id_seq	Projected DB
10	<d(cf)>
20	<(ae)>
30	<b>
40	<>

Projected base

- Step 2(3): projected database
  - Prefix: <acc>

Id_seq	Projected DB
10	<d(cf)>
20	<(ae)>
30	<b>
40	<>

Projected base

- Step 2(3): projected database
  - Prefix: <acc>
    - ∅

Id_seq	Projected DB
10	<d(cf)>
20	<(ae)>
30	<b>
40	<>

- **END**

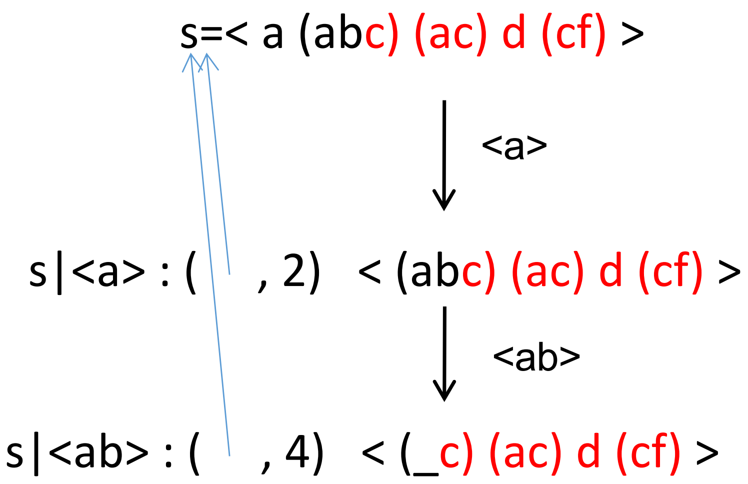
- **Result**
  - <a>, <b>, <c>
  - <a b>, <a c>, <c c>
  - <a c c>

Advantages of PrefixSpan

- **No candidate generation**
- **The projected sequence database is smaller at each step**
- **The most consuming step**
  - Projected database building
    - Improvement thanks to pseudo-projections

Pseudo Projection

- **Instead of copy sequence database at each step, use**
  - pointers on the sequence
  - and offset to identify the suffix



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

<a>

s|<a> : (, 2) < (abc) (ac) d (cf) >

<ab>

s|<ab> : (, 4) < (_c) (ac) d (cf) >

Table of Contents

Sequential Pattern Mining

- I. Introduction: what are we looking for
- II. Definitions
  - I. Vocabulary
  - II. Sequence database
  - III. Frequent sequential patterns
  - IV. Problem definition
  - V. Discussion about time parameters
- III. Sequence mining algorithms
  - I. Search space
  - II. Apriori based approaches (Generate & Prune)
    - I. GSP
    - II. SPADE
  - III. Pattern growth Approach
    - I. PrefixSpan
- IV. To go further
  - I. Closed and Maximal Sequential Patterns
  - II. Mining sequential patterns with gap constraints
  - III. Episode mining (Winepi)

Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

II. Definitions
   - I. Vocabulary
   - II. Sequence database
   - III. Frequent sequential patterns
   - IV. Problem definition
   - V. Discussion about time parameters

III. Sequence mining algorithms
   - I. Search space
   - II. Apriori based approaches (Generate & Prune)
     - I. GSP
     - II. SPADE
   - III. Pattern growth Approach
     - I. PrefixSpan

IV. To go further
   - **I. Closed and Maximal Sequential Patterns**
   - II. Mining sequential patterns with gap constraints
   - III. Episode mining (Winepi)

Closed and Maximal Sequential Patterns

- **Definition**
  - A sequential pattern s is **closed** over a set of patterns S
  - Iff $\nexists s' \in S, s \subseteq s'$ (or $\forall s' \in S, s \not\subseteq s'$)
  - s.t. sup(s)=sup(s')

- **Definition**
  - A sequential pattern s is **maximal** over a set of patterns S
  - Iff $\nexists s' \in S, s \subseteq s'$ (or $\forall s' \in S, s \not\subseteq s'$)

- **Example**
  - Let us consider the following set of sequences

Pattern	Support	Maximal ?	Closed ?
<(ab) (c) (e)>	2		
<(a) (c) (d)>	4		
<(a) (c) (e)>	3		
<(c) (d) (e)>	5		
<(a) (c)>	4		
<(b)>	7		

Closed and Maximal Sequential Patterns

- **How to compute those patterns?**
  - As postprocessing
  - With specific algorithms (e.g., CloSpan, BIDE)

Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

II. Definitions
- I. Vocabulary
- II. Sequence database
- III. Frequent sequential patterns
- IV. Problem definition
- V. Discussion about time parameters

III. Sequence mining algorithms
- I. Search space
- II. Apriori based approaches (Generate & Prune)
  - I. GSP
  - II. SPADE
- III. Pattern growth Approach
  - I. PrefixSpan

IV. To go further
- I. Closed and Maximal Sequential Patterns
- **II. Mining sequential patterns with gap constraints**
- III. Episode mining (Winepi)

Mining sequential patterns with gap constraints

How to take into account gap constraints ?

- **Approach 1:**
  - Mine sequential patterns without gap constraints
  - Postprocess the discovered patterns
- **Approach 2:**
  - Modify GSP to directly prune candidates that violate gap constraints
  - Question:
    - Does Apriori principle (anti-monotonicity) still hold?

Mining sequential patterns with gap constraints

- **Does Apriori principle (anti-monotonicity) still hold?**

Seq. ID	Sequence
10	<(abd)(bc)(e)>
20	<(ab)(bcd) >
30	<(ab)(bcd)(bde)>
40	<(b)(c)(d)(de)>
50	<(ac)(bde) >

Suppose:

***maxgap*= 1**

***minsup* = 50%**

<(b) (e)> support = 40% (10, 30)

but

<(b) (c) (e)> support = 60% (10, 30, 40)

Problem exists because of *maxgap* **constraint**

No such problem if *maxgap* **is** infinite

Mining sequential patterns with gap constraints

Contiguous subsequences

- **Definition: contiguous**
  - s is a **contiguous subsequence** of w = <e1>< e2>…< ek>
  - if any of the following conditions hold:
    - s is obtained from w by deleting an item from either e1 or ek
    - s is obtained from w by deleting an item from any element e_i that contains at least 2 items
    - s is a contiguous subsequence of s' and s' is a contiguous subsequence of w (recursive definition)

- **Example:**
  - s = < (a) (b) >
  - is a contiguous subsequence of
    < (**a**) (**b** c)>, < (**a** b) (**b**) (c)>, and < (c d) (**a** b) (**b** c) (d) >
  - is not a contiguous subsequence of
    < (**a**) (c) (**b**)> and < (b) (**a** **b**) (c) (b)>

Mining sequential patterns with gap constraints

Contiguous subsequences [Gap-Bide]

- **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

For candidate <(b) (c) (e)>

Check 2 contigous 2-subsequences:

- <(b) (c)>
- <(c) (e)>

Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

II. Definitions
   - I. Vocabulary
   - II. Sequence database
   - III. Frequent sequential patterns
   - IV. Problem definition
   - V. Discussion about time parameters

III. Sequence mining algorithms
   - I. Search space
   - II. Apriori based approaches (Generate & Prune)
     - I. GSP
     - II. SPADE
   - III. Pattern growth Approach
     - I. PrefixSpan

IV. To go further
   - I. Closed and Maximal Sequential Patterns
   - II. Mining sequential patterns with gap constraints
   - **III. Episode mining (Winepi)**

Episode Mining

Episode mining

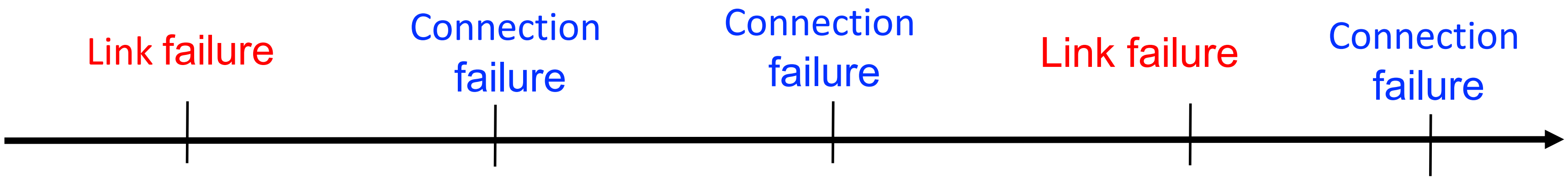
=

analysing sequences of events to discover recurrent episodes

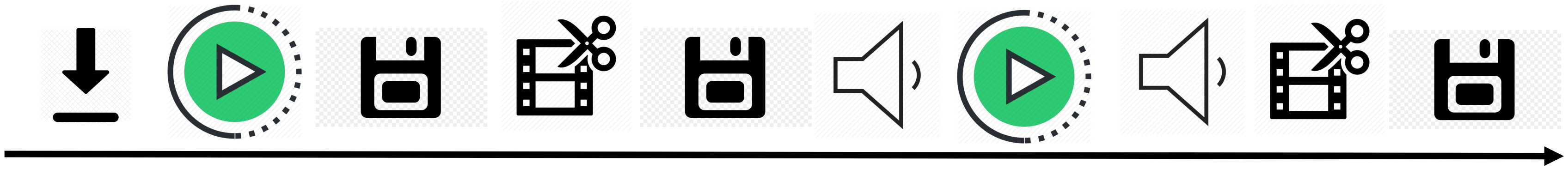
[Mannila et al. DMKD’97]

Episode Mining

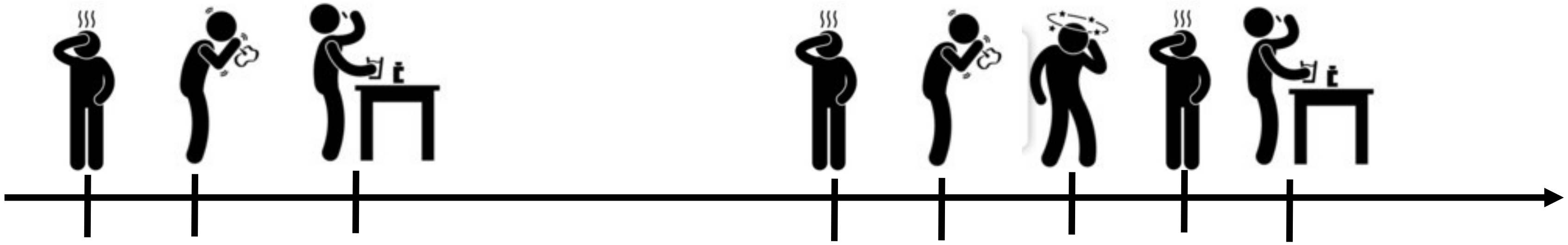
- **Event sequence**
  - Alarms in telecommunication network



  - User interface actions

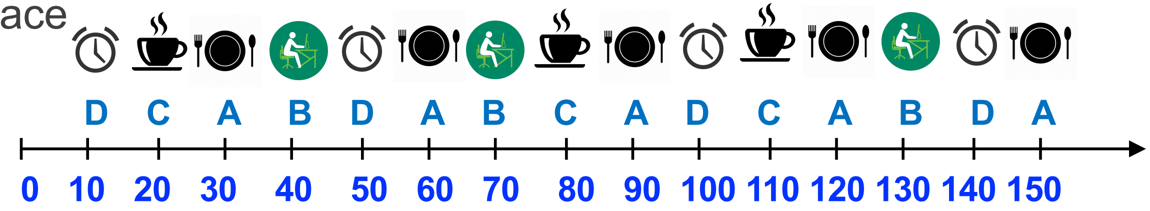


  - Occurrences of recurrent illnesses



Episode Mining

- **Event sequence**
  - Example: human trace



- Event types
  - R = {A=‘eat’, B=‘work’, C=‘prepare coffee’, D=‘wake up’}
- Occurrence times
  - integer → 10 … 150
- Event: pair (E, t)
  - E: event type
  - t: occurrence time
  - Example: (A,30)
- Sequence on R: S = (s, T_s, T_e)
  - Example:
    - s= <(D,10), (C,20), …, (A,150)>
    - starting time: T_s = 10
    - ending time: T_e = 150
- A time slot may contain 0, 1 or several events

Episode Mining

Episode

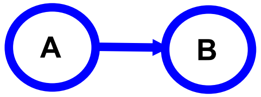
- Informally, an episode is a partially ordered collection of events occurring together
- E = (V, ≤)
  - V: collection of event types
  - ≤: partial order

Occurences

- Episode *E* occurs in a sequence *S*
- if it's possible to match event types of *E* on events of *S*
- so that the partial order ≤ is respected

Partial orders

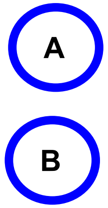
- Total order: serial episode



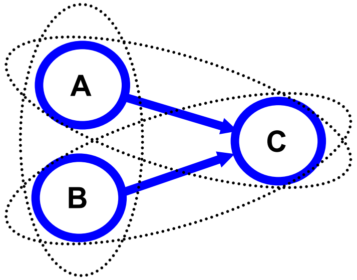
Serial episode

Note: in the sequence there can be other events occurring between A and B

- No order: parallel episode



Parallel episode



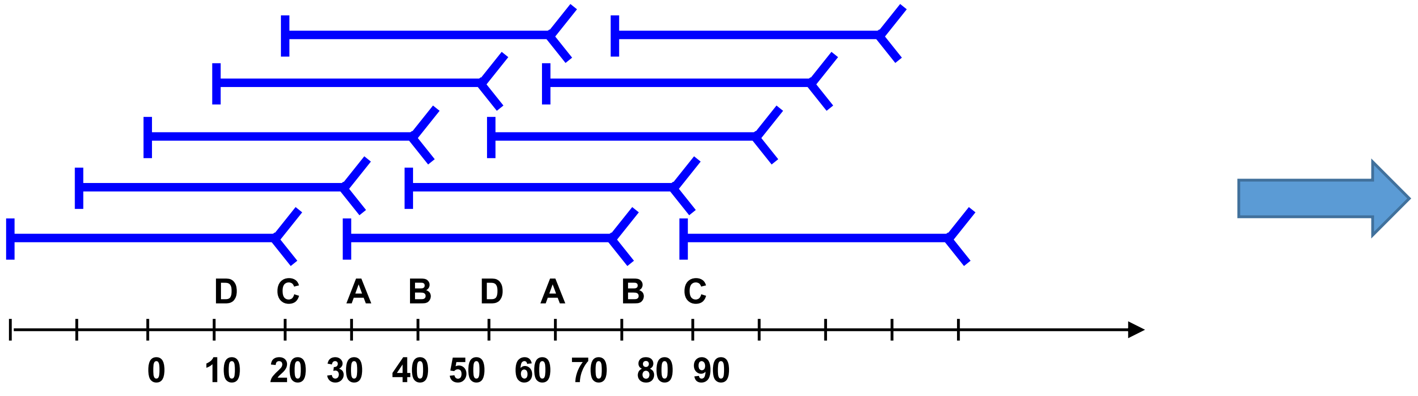
More complex episode with serial and parallel

Note: We mostly consider the discovery of serial and parallel episodes

WINEPI: sliding window

- **The name of the WINEPI method comes from the technique it uses: a sliding window**

- **Sliding window**
  - A window is slided through the event-based data sequence
  - Each window "snapshot" is like a row in a database
  - The collection of these "snapshots" forms the rows in the database



N°	Sequence
1	D
2	DC
3	DCA
4	DCAB
5	CABD
6	ABDA
7	BDAB
8	DABC
9	ABC
10	BC
11	C

Window width: 40 s

- last point excluded

First (last) window contains first (last) point:

- 11 possible windows on the example

WINEPI: frequency

- **The frequency/support of an episode α is**
  - « the fraction of windows in which the episode occurs »
  - defined as $fr(\alpha, S, w) = \dfrac{|\{S_w \in W(S, w) \mid \alpha \text{ occurs in } S_w\}|}{|W(S, w)|}$
    - w: window width
    - Where $W(S, w)$ is the set of all windows of S w.r.t w
- **An episode is frequent if**
  - $fr(\alpha, S, w) \geq$ ***min_freq*** (threshold)
- **Anti-monotonicity**
  - if episode α is frequent then all subepisodes $\beta \subseteq \alpha$ are frequent.

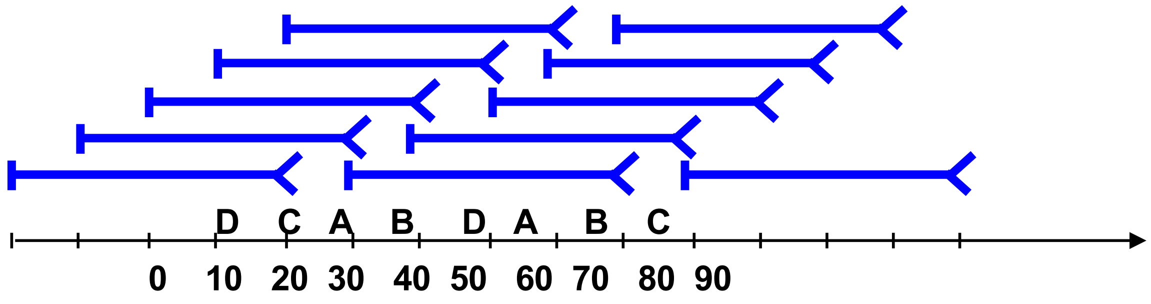
WINEPI algorithm

- **Input**:
  - A set ***R*** of event types,
  - an event sequence **s** over *R*,
  - a set **E** of episodes, // parall or serial
  - a window width ***win***,
  - and a frequency threshold ***min_fr***
- **Output**:
  - The collection of frequent episodes: **F(s, *win*, *min_fr*)**

```
1. compute $C_1 \leftarrow \{\alpha \in E \mid |\alpha| = 1\}$;
2. $i = 1$;
3. while $C_i \neq \emptyset$ do
4.         // Database pass
           compute $F_i(s, win, min\_fr) \leftarrow \overbrace{\{\alpha \in C_i \mid fr(\alpha, s, win) \geq min\_fr\}}^{\text{Test of frequency}}$;
5.         $i \leftarrow i+1$;
6.         // Candidate generation
           compute $C_i \leftarrow \{\alpha \in E \mid |\alpha| = i, \text{ and } \underbrace{\forall \beta \in E \text{ s.t. } \beta \subseteq \alpha \text{ and } \beta \in F_{|\beta|}(s, win, min\_fr),\}}_{\text{All subepisodes have to be frequent}}$;
7. for all i do ouptut $F_i(s, win, min\_fr)$
```

WINEPI algorithm: generation of candidate episodes

- **Example: find all parallel episodes with frequency > 40 % (present in at least 5 windows)**
  - Create singletons, i.e., parallel episodes of size 1
    - ***A, B, C, D***
  - Select the frequent singletons
    - **here all are**
  - From those frequent episodes, build candidate episodes of size 2
    - ***AB, AC, AD, BC, BD, CD***
  - Select the frequent parallel episodes of size 2
    - **here all are**
  - From those frequent episodes, build candidate episodes of size 3
    - ***ABC, ABD, ACD, BCD***
  - Select the frequent episodes of size 3
    - only **ABD** occurs in more than four windows
  - There are no candidate episodes of size four



N°	Sequence
1	D
2	DC
3	DCA
4	DCAB
5	CABD
6	ABDA
7	BDAB
8	DABC
9	ABC
10	BC
11	C

Alternative: MINEPI

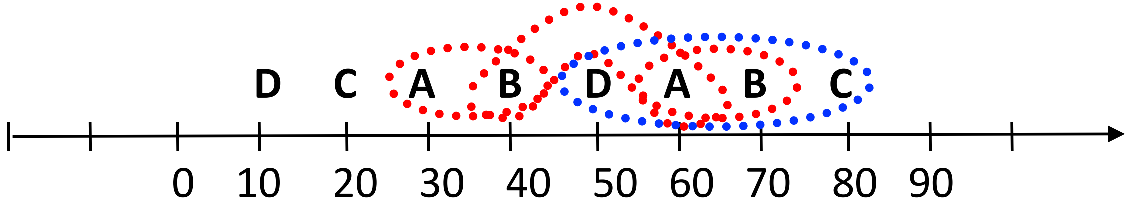
- **[Mannila et al. DMKD’97]**
- **Alternative approach to discover episodes**
  - No sliding windows
  - For each potentially interesting episode, find out the exact occurrences
- **Minepi is based of the notion of minimal occurrences**

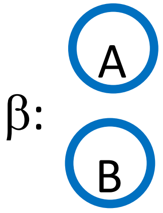
Minimal occurrence

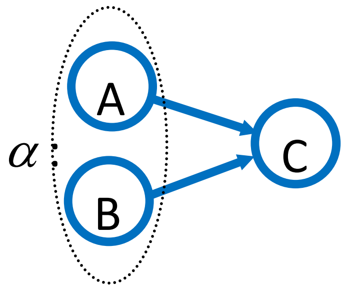
- **Formally, given a episode α and an event sequence S, the interval [t_s,t_e] is a minimal occurrence α of S,**
  - If α occurs in the window corresponding to the interval
  - And If α does not occur in any proper subinterval

- **The set of minimal occurrences of an episode α in a given event sequence is denoted by *mo(α):***
  - *mo(α)* = { [t_s,t_e] | [t_s,t_e] is a minimal occurrence of α }

- **Example**
  - β consisting of event types A and B has three minimal occurrences in s: *mo*(β) = {[30,40], [40,60], [60,70]}
    - Note: ([30,70] is not minimal)
  - α has one occurrence in s: *mo*(α) = {[60,80]}
    - Note: ([30,80] is not minimal)







Minepi

- **Task: Find all serial episodes**
  - Using maximum time bound of 40 secs
  - min_fr=1

- **Create singletons, i.e., episodes of size 1**
  - (A, B, C, D)
- **Create an occurrence table**
  - will use inverse tables
  - A: 30, 60 ; B: 40, 70 ; C: 20, 80 ; D: 10, 50
- **Recognize the frequent singletons**
  - here all are
- **From frequent episodes of size 1 build candidate episodes of size 2**
  - AB, BA, AC, CA, AD, DA, BC, CB, BD, DB, CD, DC
- **Use the inverse table to create minimal occurrences for the candidates**
  - Mo(AB)={[30,40], [60,70]}
    - Read the first occurrence of A (30-30), and find the first following B (40-40)
    - Read the second occurrence of A (60-60), and find the first following B (70-70)
  - Continue with BA, AC etc
- **Recognize the frequent episodes of size 2**
  - here almost are
- **From frequent episodes of size 2 build candidate episodes of size 3**
- **And so on**

Table of Contents

Sequential Pattern Mining

I. Introduction: what are we looking for

II. Definitions
  - I. Vocabulary
  - II. Sequence database
  - III. Frequent sequential patterns
  - IV. Problem definition
  - V. Discussion about time parameters

III. Sequence mining algorithms
  - I. Search space
  - II. Apriori based approaches (Generate & Prune)
    - I. GSP
    - II. SPADE
  - III. Pattern growth Approach
    - I. PrefixSpan

IV. To go further
  - I. Closed and Maximal Sequential Patterns
  - II. Mining sequential patterns with gap constraints
  - III. Episode mining (Winepi)