Mining patterns in collections of structured data

CM2 – Complex domains of patterns

Thomas Guyet

Inria – AIstroSight

M1 ENS – DBDM – 2023

Outline

1. Basics of frequent sequential pattern mining
   - Backgrounds
   - Algorithms for frequent sequential patterns extraction
   - Reducing the number of patterns
   - Available implementations for sequential pattern mining
   - Sum up
2. Sequence mining in long sequences
   - Problem setting
   - Different enumerations
3. Negative sequential patterns
   - Syntax and semantics of NSP
   - Survey results
   - Relationships between containment relations
   - Properties of containment relations
   - Conclusion
4. Graph mining
5. Temporal pattern mining

Sequence dataset

- A sequence dataset consists of ordered elements or events
- Transation dataset vs. sequence dataset

SID	sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ab)c(bc)(ae)(ad)\rangle$
30	$\langle (ef)(ab)(df)cbc\rangle$
40	$\langle eg(af)cbc(de)\rangle$
50	$\langle aa(ad)b\rangle$

Sequence dataset

- A sequence dataset consists of ordered elements or events
- Transation dataset vs. sequence dataset

SID	sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle(ab)c(bc)(ae)(ad)\rangle$
30	$\langle(ef)(ab)(df)cbc\rangle$
40	$\langle eg(af)cbc(de)\rangle$
50	$\langle aa(ad)b\rangle$

Sequence dataset

- A sequence dataset consists of ordered elements or events
- Transation dataset vs. sequence dataset

SID	sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ab)c(bc)(ae)(ad)\rangle$
30	$\langle (ef)(ab)(df)cbc\rangle$
40	$\langle eg(af)cbc(de)\rangle$
50	$\langle aa(ad)b\rangle$

- Sequential pattern mining methods are not made for Xomics data

Subsequence vs. super sequence

- A sequence is an ordered list of elements, denoted $\langle e_1, e_2, \ldots, e_l \rangle$
  - An element (so called itemsets) may contain a set of items
  - items are so called events
  - $\mathcal{I}$ is the set of items/events
  - Items within an itemset are unordered and we list them alphabetically (convention)
  - Sequentially represents occurrence order in time of events
- Given two sequences $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$ and $\beta = \langle b_1, b_2, \ldots, b_m \rangle$
  - α is called a subsequence of β, denoted as $\alpha \preceq \beta$, if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}$, $\ldots$, $a_n \subseteq b_{j_n}$
  - in such case, we say that β is a super-sequence of α
  - $(j_i)_{i \in \mathbb{N}_n}$ is called an embedding of α in β

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

What is Sequential Pattern Mining (SPM)?

Given a set of sequences, $\mathcal{D}$

- support of pattern P: number of sequences in $\mathcal{D}$ which are supersequences of P:

$$supp(P) = |\{s \in \mathcal{D} | P \preceq s\}|$$

- Frequent pattern mining task: Given a support threshold σ, find the complete set of frequent subsequences
  - ↪ Patterns are sequences
  - ↪ σ is a key parameter
    - too high: no interesting patterns are extracted (too much frequent)
    - too low: huge number of patterns (difficult to interpret and possibly time-consuming to extract)

SID	sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ab)c(bc)(ae)(ad)\rangle$
30	$\langle (ef)(ab)(df)cbc\rangle$
40	$\langle eg(af)cbc(de)\rangle$
50	$\langle aa(ad)b\rangle$

Some frequent patterns:

- $\langle abd\rangle$ (but not $\langle a(bc)d\rangle$)
- $\langle (ab)c\rangle$

Itemset mining vs Sequential pattern mining

TID	itemset
10	a,b,c
20	a,c,d
30	a,d,e
40	b,e,f

Patterns are subset (itemsets) of the set of items

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

Patterns are sequences

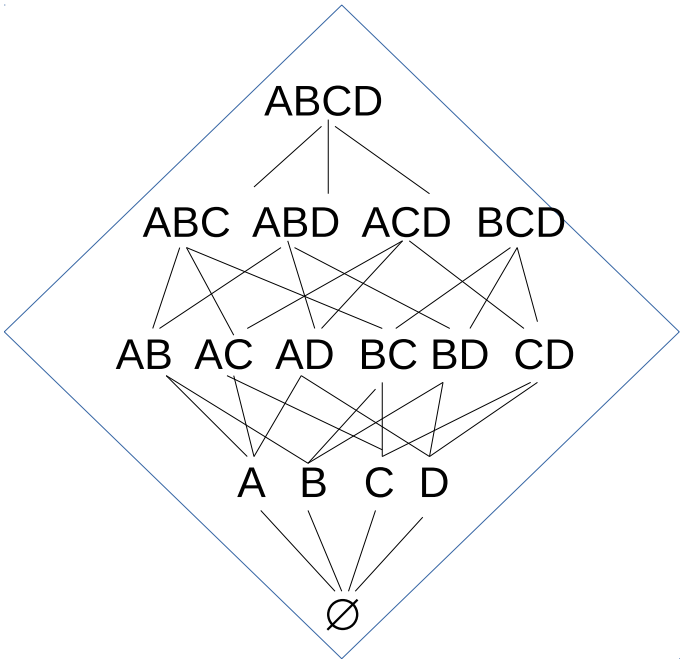
Algorithms

Classical algorithms for sequential pattern mining

- Apriori-based method: GSP (Generalized Sequential Patterns: [SA96])
- Pattern-growth methods: FreeSpan & PrefixSpan [PHMA+04]
- Vertical format-based mining: SPADE [Zak01b]
- Constraint-based sequential pattern mining ([GRS99, PHW02])
- Mining closed sequential patterns: CloSpan [YHA03], LCMseq, sequential version of LCM [UAUA04]

$\Rightarrow$ all these algorithms are based on the antimonotocity to prune efficiently the search space

Search space

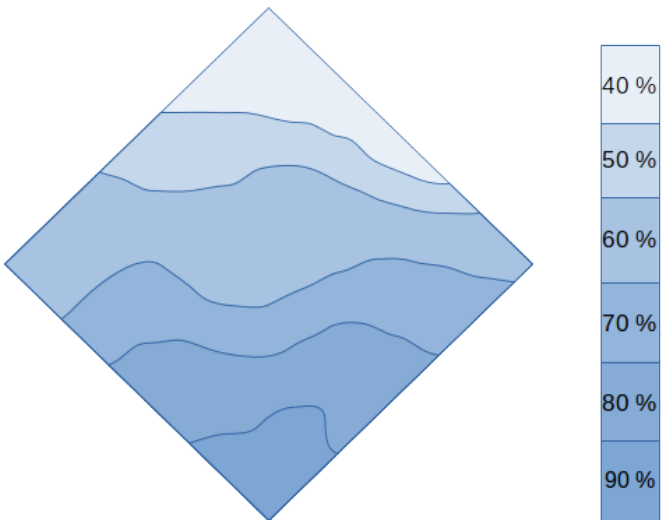


- Search space of sequences of length at most k
  - size: $\sum_{l=1}^{|\mathcal{I}|} l^k$
  - structured as lattice of subsequences (subsequence partial order)

Naive algorithm

$\mathcal{FP} \leftarrow \emptyset$
for P a pattern **do**
    $sup \leftarrow supp_{\mathcal{D}}(P)$
    if $sup \geq \sigma$ **then**
        $\mathcal{FP} \leftarrow \mathcal{FP} \cup \{P\}$

Search space & anti-monotonicity



- Search space of sequences of length at most k
  - size: $\sum_{l=1}^{|\mathcal{I}|} l^k$
  - structured as lattice of subsequences (subsequence partial order)
- Anti-monotonicity
  - $P \preceq Q \implies supp(P) > supp(Q)$
  - The support measure is decreasing monotonic on the lattice of sequences
- More interesting

$$supp(P) < \sigma \implies \forall Q,\ P \preceq Q,\ supp(Q) < \sigma$$

  - if P is not frequent, any larger pattern Q will be frequent

GSP: Generalized Sequential Pattern [SA96]

$C \leftarrow \mathcal{I}$
for $l \in [1, k]$ **do**
  $L_l \leftarrow \emptyset$ ▷ *Evaluate support (DB scan)*
  for $P \in C$ **do** ▷ *Test candidates*
    $sup \leftarrow supp_{\mathcal{D}}(P)$
    if $sup \geq \sigma$ **then**
      $L_l \leftarrow L_l \cup \{P\}$
  $C \leftarrow \emptyset$
  for $P \in L_l$ **do** ▷ *Generate candidates*
    for $i \in \mathcal{I}$ **do**
      $C \leftarrow C \cup \{P \oplus i, P \otimes i\}$ ▷ *Last itemset add/new itemset*
return $\bigcup_l L_l$

Running example

Seq. Id	Tr. time	Itemset
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H

minimal frequency threshold: $\sigma = 2$

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

Running example

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

- 1-frequent sequences: A, B, D, F

Running example

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

- 1-frequent sequences: A, B, D, F
  - $\rightarrow$ Candidates: $\langle AA \rangle$, $\langle AB \rangle$, $\langle AD \rangle$, $\langle AF \rangle$, $\langle (AB) \rangle$, $\langle (AD) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BB \rangle$, $\langle BD \rangle$, $\langle BF \rangle$, $\langle (BD) \rangle$, $\langle (BF) \rangle$, ...

Running example

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

- 1-frequent sequences: A, B, D, F
  - $\rightarrow$ Candidates: $\langle AA \rangle$, $\langle AB \rangle$, $\langle AD \rangle$, $\langle AF \rangle$, $\langle (AB) \rangle$, $\langle (AD) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BB \rangle$, $\langle BD \rangle$, $\langle BF \rangle$, $\langle (BD) \rangle$, $\langle (BF) \rangle$, ...
- 2-frequent sequences: $\langle (AB) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BF \rangle$, $\langle DA \rangle$, $\langle DB \rangle$, $\langle DF \rangle$, $\langle FA \rangle$

Running example

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

- 1-frequent sequences: A, B, D, F
  - $\rightarrow$ Candidates: $\langle AA \rangle$, $\langle AB \rangle$, $\langle AD \rangle$, $\langle AF \rangle$, $\langle (AB) \rangle$, $\langle (AD) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BB \rangle$, $\langle BD \rangle$, $\langle BF \rangle$, $\langle (BD) \rangle$, $\langle (BF) \rangle$, ...
- 2-frequent sequences: $\langle (AB) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BF \rangle$, $\langle DA \rangle$, $\langle DB \rangle$, $\langle DF \rangle$, $\langle FA \rangle$
  - $\rightarrow$ Candidates: $\langle (AB)A \rangle$, $\langle (AB)B \rangle$, $\langle (AB)D \rangle$, $\langle (AB)F \rangle$, $\langle (ABD) \rangle$, $\langle (ABF) \rangle$, ...

Running example

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

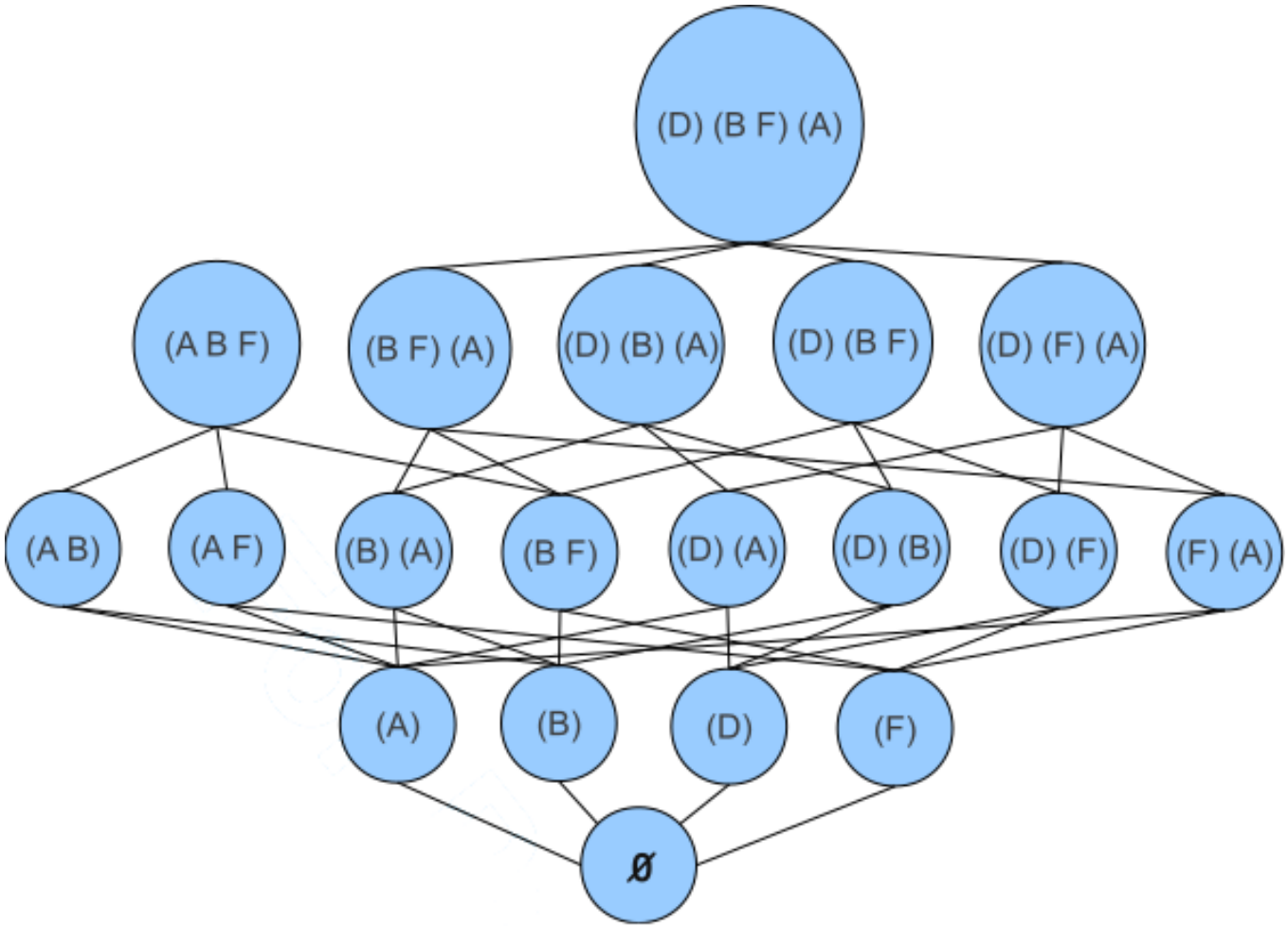
- 1-frequent sequences: A, B, D, F
  - $\rightarrow$ Candidates: $\langle AA \rangle$, $\langle AB \rangle$, $\langle AD \rangle$, $\langle AF \rangle$, $\langle (AB) \rangle$, $\langle (AD) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BB \rangle$, $\langle BD \rangle$, $\langle BF \rangle$, $\langle (BD) \rangle$, $\langle (BF) \rangle$, ...
- 2-frequent sequences: $\langle (AB) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BF \rangle$, $\langle DA \rangle$, $\langle DB \rangle$, $\langle DF \rangle$, $\langle FA \rangle$
  - $\rightarrow$ Candidates: $\langle (AB)A \rangle$, $\langle (AB)B \rangle$, $\langle (AB)D \rangle$, $\langle (AB)F \rangle$, $\langle (ABD) \rangle$, $\langle (ABF) \rangle$, ...
- 3-frequent sequences: $\langle (ABF) \rangle$, $\langle (BF)A \rangle$, $\langle DBA \rangle$, $\langle DBF \rangle$, $\langle D(BF) \rangle$, $\langle DFA \rangle$,

Running example

SID	Sequence
1	$\langle (CD)(ABC)(ABF)(ACDF) \rangle$
2	$\langle (ABF)(E) \rangle$
3	$\langle (ABF) \rangle$
4	$\langle (DGH)(BF)(AGH) \rangle$

- 1-frequent sequences: A, B, D, F
  - $\rightarrow$ Candidates: $\langle AA \rangle$, $\langle AB \rangle$, $\langle AD \rangle$, $\langle AF \rangle$, $\langle (AB) \rangle$, $\langle (AD) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BB \rangle$, $\langle BD \rangle$, $\langle BF \rangle$, $\langle (BD) \rangle$, $\langle (BF) \rangle$, ...
- 2-frequent sequences: $\langle (AB) \rangle$, $\langle (AF) \rangle$, $\langle BA \rangle$, $\langle BF \rangle$, $\langle DA \rangle$, $\langle DB \rangle$, $\langle DF \rangle$, $\langle FA \rangle$
  - $\rightarrow$ Candidates: $\langle (AB)A \rangle$, $\langle (AB)B \rangle$, $\langle (AB)D \rangle$, $\langle (AB)F \rangle$, $\langle (ABD) \rangle$, $\langle (ABF) \rangle$, ...
- 3-frequent sequences: $\langle (ABF) \rangle$, $\langle (BF)A \rangle$, $\langle DBA \rangle$, $\langle DBF \rangle$, $\langle D(BF) \rangle$, $\langle DFA \rangle$,
- 4-frequent sequence: $\langle D(BF)A \rangle$

Running example: Lattice of frequent sequential patterns



GSP: Generalized Sequential Pattern [SA96]

- First algorithm to use the anti-monotonicity property to efficiently extract sequential patterns
- Breadth-first search strategy: level by level
- Require to compute all patterns at the same time: memory consuming
- Easy to implement
- Require a lot of dataset scans: not efficient
- A lot of algorithms adapted from GSP (APriori) have been proposed (*e.g.* to generate less candidates based on frequent sub-patterns)

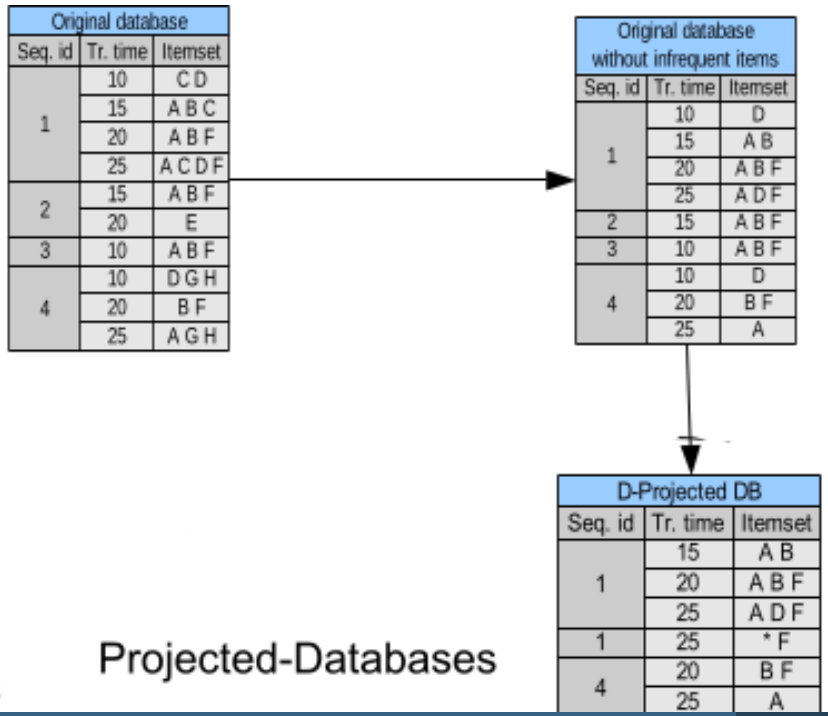
A better alternative $\Rightarrow$ a depth-first search strategy

PrefixSpan [PHMA+04]

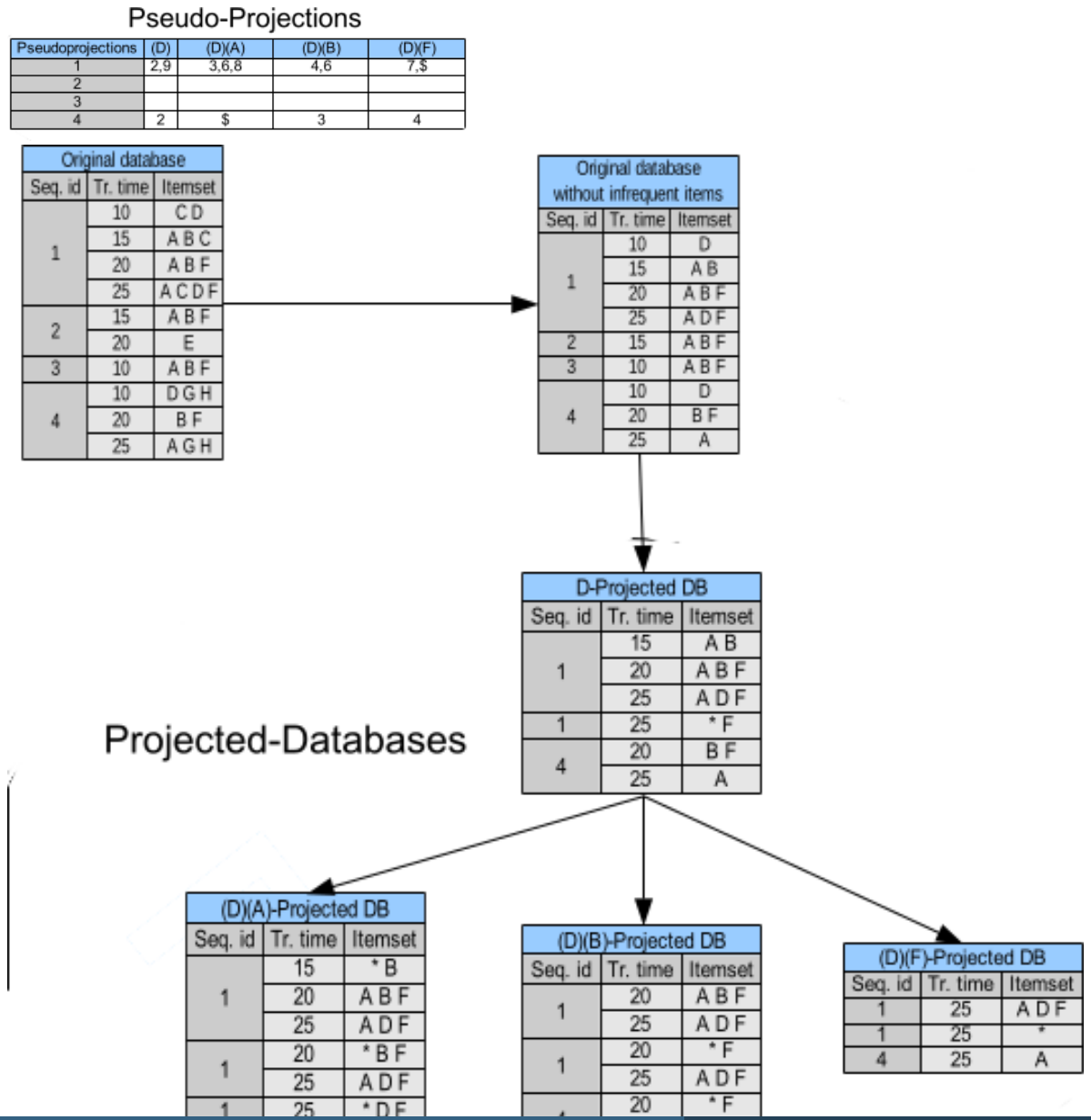
- Depth-first search strategy: recursive implementation (divide and conquer principle)
- More efficient in memory: avoid generating candidates
- Use a projected databases
  - at a recursion level (associated to a pattern P),
    - the database holds only the sequences in which P occurs
    - for each sequence, we keep only the events after to last position of the first pattern embedding (only those events are useful to search for possible extension of the pattern)
  - it limits the number of dataset scans

Running example

Seq. Id	Tr. time	Itemset
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H



Running example – pseudo projection principle



PrefixSpan

- it obtains the whole set of frequent sequences without generating candidates
- low memory consumption (thanks to pseudo projected database)
- easy to distribute/parallelize (shared data, distributed pseudo-projections)

$\Rightarrow$ a frequently used algorithm for sequential pattern mining

SPADE [Zak01a]

Seq. Id	Tr. time	Itemset
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H

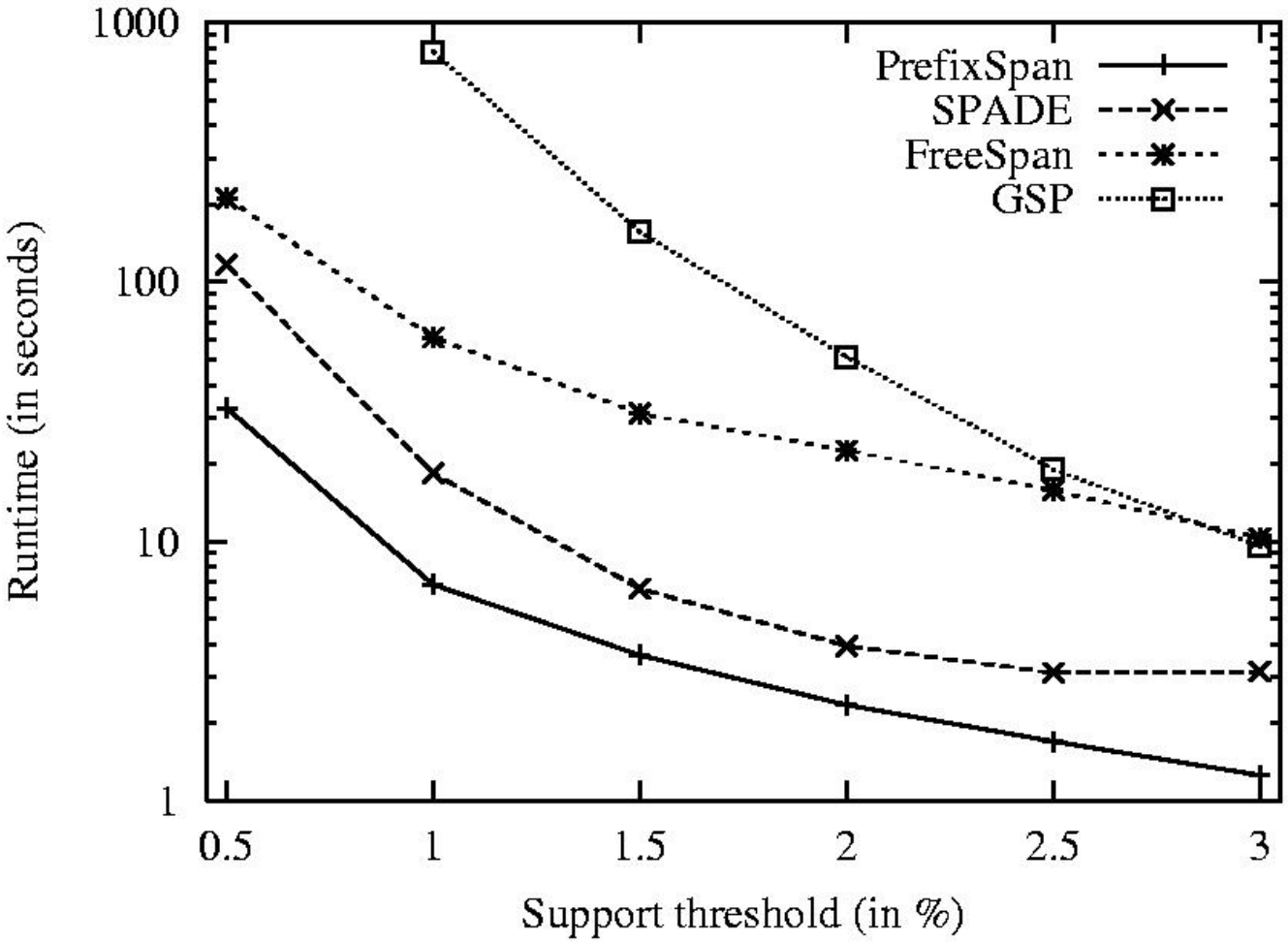
ID-LISTS

VERTICAL DATABASE

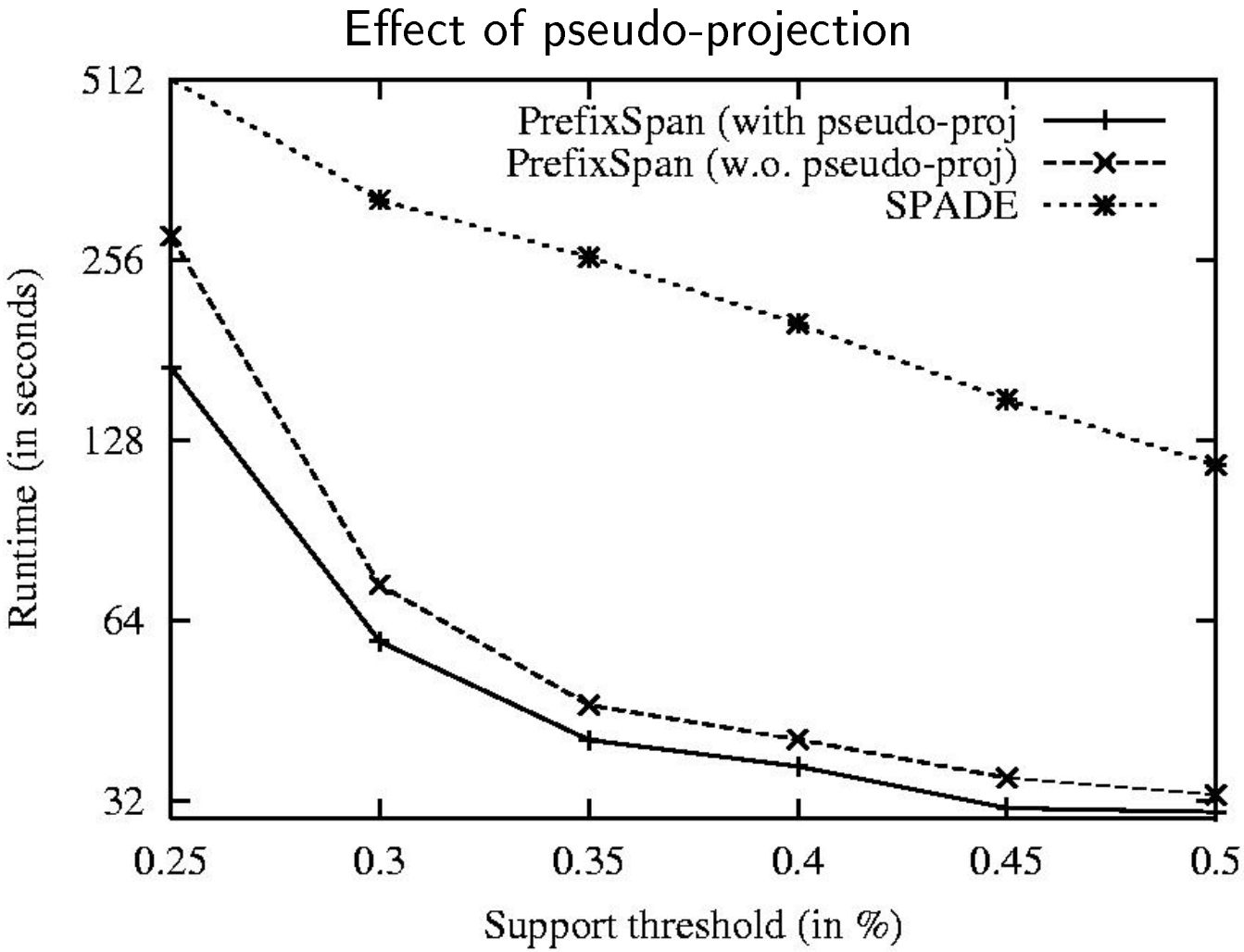
A		B		D		F	
Sid	Tid	Sid	Tid	Sid	Tid	Sid	Tid
1	15	1	15	1	10	1	20
	20		20		25		25
	25	2	15	4	10	2	15
2	15	3	10			3	10
3	10	4	20			4	20
4	25						

- Use a *Vertical Format Database* (previously introduced in ECLAT)
  - it makes possible to easily count the support of each sequence
  - it allows an efficient way to create a common supersequence from two candidate frequent sequences
- Both DFS and BFS strategy
- The algorithm decomposes the search space in different pieces and solve them separately
  - → equivalence relation between sequences (prefix sharing)

Comparative study

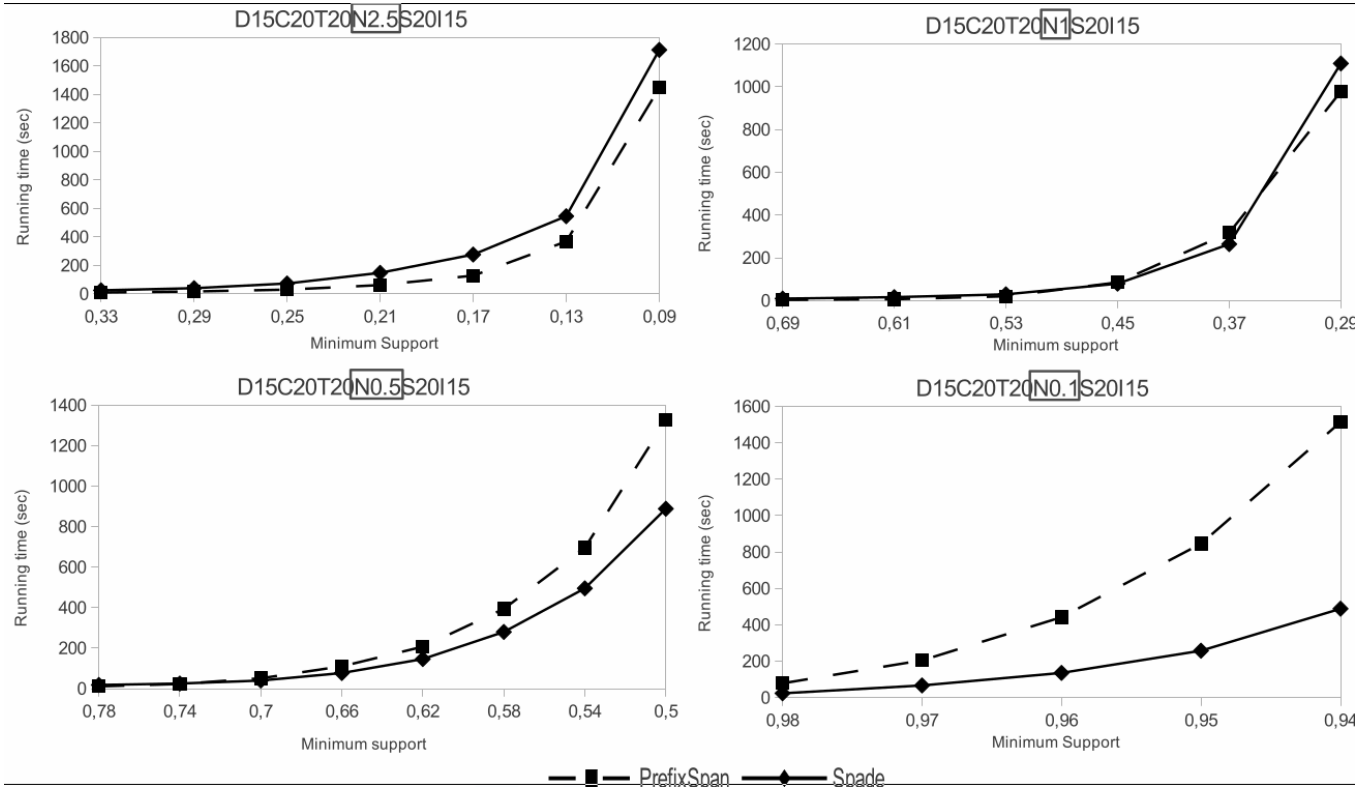


Comparative study



Comparative study

Effect of dataset density



N: number of different items (/100)

From frequent patterns to decision rules

Mining sequential predictive/decision rules

- A sequential rules is of the form $P \implies Q$, where $P = \langle p_1, \ldots, p_n \rangle$ and $Q = \langle q_1, \ldots, q_m \rangle$ are sequential patterns
- Two measures of interest
  - support: $supp(\langle P, Q \rangle) = supp(\langle p_1, \ldots, p_n, q_1, \ldots, q_m \rangle)$
  - accuracy: $supp(\langle P, Q \rangle)/supp(\langle P \rangle)$
- Mining decision rules with a minimal support (σ) and a minimal accuracy ($\kappa \in [0, 1]$) can be easily done in post-processing

$\mathcal{FP} \leftarrow extract(\mathcal{D}, \sigma)$
$\mathcal{R} \leftarrow \emptyset$ ▷ *Set of rules*
for $R = \langle p_1, \ldots, p_n \rangle \in \mathcal{FP}$ **do**
  for $i \in [2, n-1]$ **do**
    $P \leftarrow \langle p_1, \ldots, p_i \rangle$, $Q \leftarrow \langle p_{i+1}, \ldots, p_n \rangle$
    if $supp(R)/supp(P) > \kappa$ **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$
return $\mathcal{R}$

The BIG problem: too much patterns

- frequent (sequential) patterns are numerous ... actually overwhelming!
- the lower is the threshold, the higher is the number
- it is not reasonable to present this deluge of pattern to data analyst

$\Rightarrow$ are such method useful??

A well-known problem

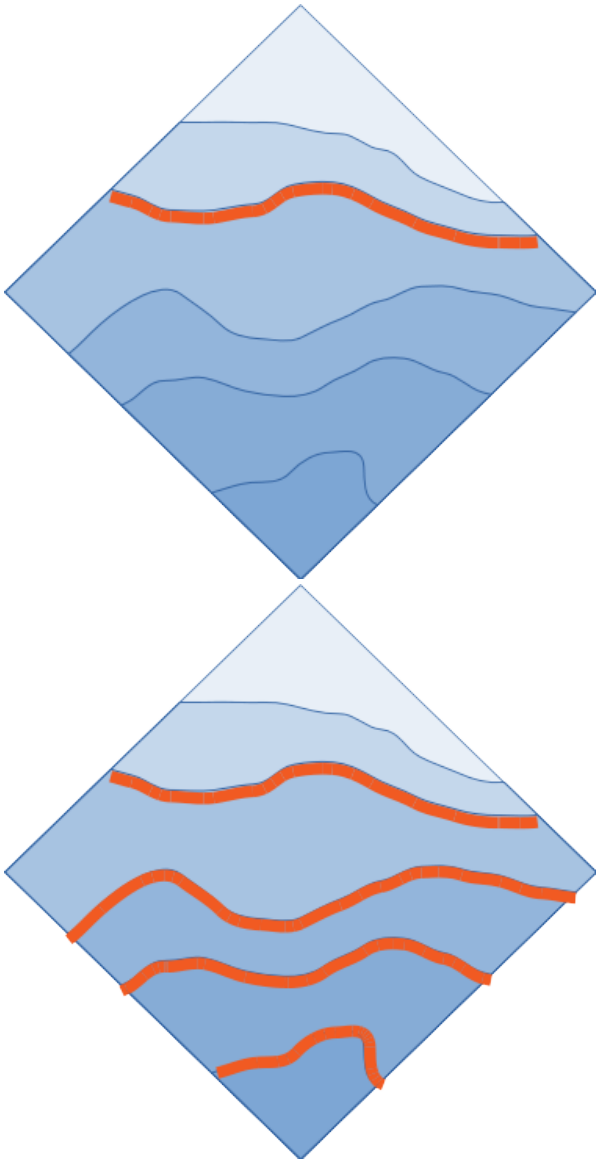
- A current challenge of pattern mining: extract *less* but *meaningful patterns*
- A lot of different approaches have been proposed
  - alternative tasks
    - condensed patterns
    - alternative interestingness measures: high utility patterns, rare patterns, ...
  - database sampling
  - constrained based patterns
  - ranking patterns

Reducing the number of patterns

We briefly remind some alternative tasks:

1. condensed patterns: closed and maximal patterns
2. constrained patterns

Closed and maximal patterns



Definitions

- Maximal: a frequent pattern P is maximal iff not any pattern Q, $P \prec Q$ is frequent
- Closed: a frequent pattern P is closed iff not any pattern Q, $P \prec Q$ has the same support as P ($supp(P) = supp(Q)$)

- Closed and maximal patterns are less numerous and often considered as significant
- The set of closed patterns captures information of all frequent patterns
- Set theory results enables to design very efficient algorithms for closed patterns (LCM)

Backward closure

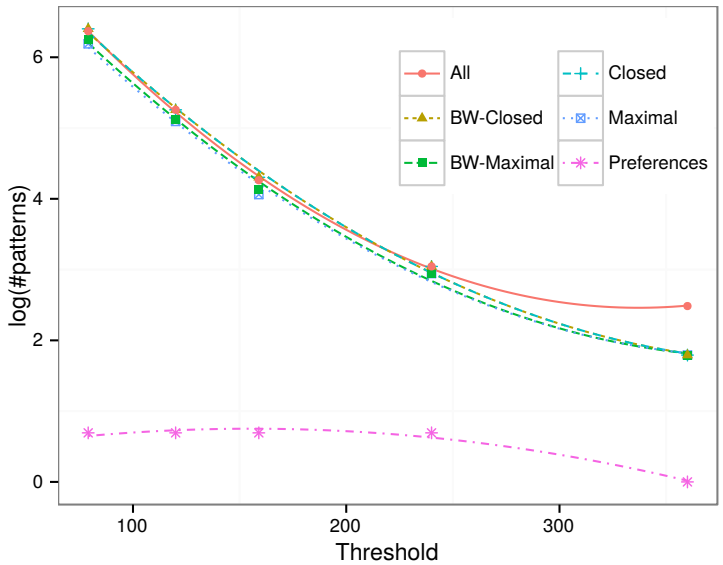
- The (full-)closure maybe difficult to compute
- using the prefix relation instead of the sub-sequence relation leads to a simplified lattice (less edges)
- A pattern P is backward closed iff there is any pattern Q such that P is a prefix of Q and Q and P have the same support
- According to the backward extension strategies, the backward closure is efficiently computable
- There is much more backward-closed patterns than (fully-)closed patterns

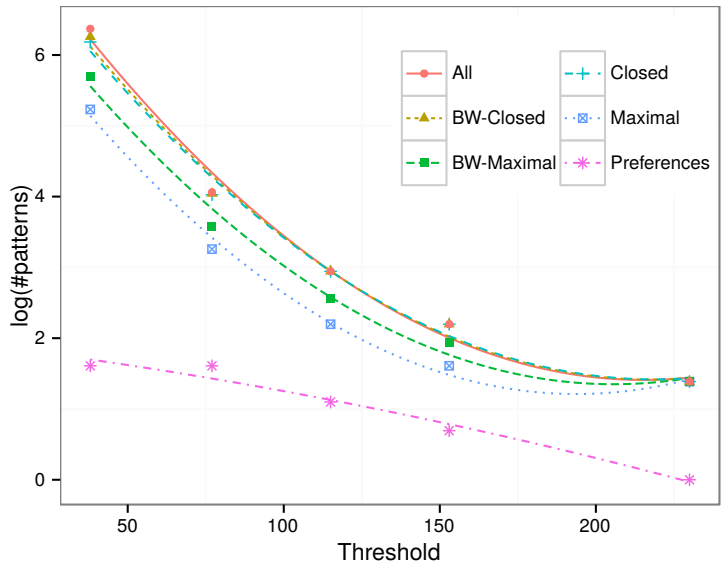
- Algorithms that extract "closed" patterns usually consider only the backward closure
- BIDE algorithm use a bidirectional closure (backward and frontward)

Pattern set reduction results

Datasets

- UNIX: sequence of UNIX commands
- JMLR: abstracts of JMLR (natural language)





Adding constraints on sequential patterns

Objective

- Data mining should be an interactive process
  - → User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility/versatility: provides constraints on what to be mined
  - System optimization: explores such constraints for efficient constraint-based mining
- The more you constraint your patterns, the less they are (and the more interesting they are)

Some classical constraints

Classical sequential pattern constraints classification [PHW02, NDGN13]

- pattern structure constraints:
  - → items (inclusion/exclusion), regular expression, sub-patterns, costs ...
- pattern occurrences constraints
  - → gaps/span constraints, ...
- pattern set constraints
  - → closed/maximal, sky-patterns, etc.

Approaches that are constraints-aware

- Algorithms: SPIRIT [GRS99], DM4SP [Rig]
- Constrained pattern mining "frameworks" [BL07, BJ05]
  - → query-like systems
- Declarative approaches CP/SAT/ASP [NDGN13, CJSS12, MLC13, GDT+13, GGQ+16]
  - → mining constraint frequent patterns is seen has a

Sequential pattern mining tools

- SPMF, Java, large collection of sequential pattern mining tools, P. Fournier-Viger
  `http://www.philippe-fournier-viger.com/spmf/`
- SPADE/cSPADE/PRISM,
  `http://www.cs.rpi.edu/~zaki/www-new/pmwiki.php/Software/Software`
  - SPADE is available in R
- not reference library for Python
  - we developed our own implementation of PrefixSpan (C++) with Python interface
- More effort have to be done to give a broad access to such algorithms

Sequential pattern mining: sum-up

- Mining task: Extract all frequent subsequences from a structured dataset $\mathcal{D}$
  - $\mathcal{D}$: a dataset of sequences
  - frequent? a minimal frequency threshold: σ
- Algorithms are based on an anti-monotonicity property to extract all patterns efficiently
- Alternative tasks for mining less but more interesting patterns
  - condensed patterns: closed, maximal
  - constrained patterns

Outline

1. Basics of frequent sequential pattern mining
   - Backgrounds
   - Algorithms for frequent sequential patterns extraction
   - Reducing the number of patterns
   - Available implementations for sequential pattern mining
   - Sum up
2. Sequence mining in long sequences
   - Problem setting
   - Different enumerations
3. Negative sequential patterns
   - Syntax and semantics of NSP
   - Survey results
   - Relationships between containment relations
   - Properties of containment relations
   - Conclusion
4. Graph mining
5. Temporal pattern mining

Mining a dataset *vs* frequent episode mining

Frequent episode mining is a different task [MTV97] !!

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

Frequent episode mining

- Mine a unique *long* (temporal) sequence of itemsets

$$\mathbb{D} = \langle (E_1, t_1), \dots (E_n, t_n) \rangle$$

  - A functioning trace of a unique system
  - Often encountered in streaming environment (but here, only few works have been proposed in the field of incremental frequent episode mining)
- The task complexity is significantly higher

Reveal a new challenge: evaluate the support

1. the first problem is to efficiently evaluate a support
2. the second problem is to define a support anti-monotone

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

$\Rightarrow$ How many occurrences of the pattern $\langle A,B,C\rangle$?

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

$\Rightarrow$ How many occurrences of the pattern $\langle A, B, C\rangle$?

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

$\Rightarrow$ How many occurrences of the pattern $\langle A, B, C\rangle$?

Counting all occurrences is not anti-monotone

$\mathbb{D} = \langle(A, _),(A, _),(B, _),(B, _),(A, _),(A, _),(B, _),(A, _)\rangle$

- occurrences of $\langle A, B\rangle$ in $\mathbb{D}$: (1,3), (1,4), (1,7), (2,3), (2,4), (2,7), (6,7)
- occurrences of $\langle A, B, A\rangle$ in $\mathbb{D}$: (1,3,5), (1,3,6), (1,3,8), (2,3,5), ...

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

Definition (Windows-based frequency (WINEPI))

A window on an event sequence, $\mathbb{D}$, is a time interval $[t_s, t_e]$, where t_s and t_e are positive integers such that $t_s \leq t_n$ and $t_e \geq t_1$. The window width of $[t_s, t_e]$ is given by $(t_e - t_s)$. Given a user-defined window width T_X, the **windows-based frequency** of α is the number of windows of width T_X which contain at least one occurrence of α.

How many Windows-based occurrences of $\alpha = \langle A, B, C \rangle$ with window width 6?

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

Definition (Minimal occurrences-based frequency (MINEPI))

The time-window of an occurrence, h, of α is given by $[t_{h(v_1)}, t_{h(v_N)}]$. A minimal window of α is a time-window which contains an occurrence of α, such that no proper sub-window of it contains an occurrence of α. An occurrence in a minimal window is called a minimal occurrence. The **minimal occurrences-based frequency** of α in $\mathbb{D}$ (denoted f_{mi}) is defined as the number of minimal windows of α in $\mathbb{D}$.

How many minimal occurrences of $\langle A, B, C \rangle$?

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

Definition (head-frequency)

Given a window-width k, the **head frequency** of α is the number of windows of width k which contain an occurrence of α starting at the left-end of the window and is denoted as $f_h(\alpha, k)$.

Definition (Total frequency)

Given a window width k, the **total frequency** of α, denoted as $f_{tot}(\alpha, k)$, is defined as follows.

$$f_{tot}(\alpha, k) = \min_{\beta \preceq \alpha} f_h(\beta, k)$$

What is the total/head frequency of $\langle A, B, C \rangle$ with width $k = 6$?

How to enumerate subsequences occurrences

$(A,1),(A,2),(B,3),(A,6),(A,7),(C,8),(B,9),(D,11),(C,12),(A,13),(B,14),(C,15)$

Definition (Non-overlapped frequency)

Two occurrences h_1 and h_2 of α are said to be non-overlapped if either $t_{h_1(v_N)} < t_{h_2(v_1)}$ or $t_{h_2(v_N)} < t_{h_1(v_1)}$. A set of occurrences is said to be non-overlapped if every pair of occurrences in the set is non-overlapped. A set H, of non-overlapped occurrences of α in $\mathbb{D}$ is maximal if $|H| \geq |H'|$, where H' is any other set of non-overlapped occurrences of α in $\mathbb{D}$. The **non-overlapped frequency** is defined as the cardinality of a maximal non-overlapped set of occurrences of α in $\mathbb{D}$.

What is the maximal set of non-overlapped occurrences of $\langle A, B, C \rangle$?

Counting occurrences in long sequences

How to choose the enumeration technique?

- Anti-monotonicity: enables to have efficient and correct algorithms
- Semantics: different count $\Rightarrow$ different sets of frequent patterns

Example $\langle AABBAABA \rangle$

Enumerations of $\langle AB \rangle$

- All : (1,3), (1,4), (1,7), (2,3), (2,4), (2,7), (6,7)
- N-over : (1,3), (2,3), (6,7)
- S-N-over : (1,3), (2,4), (6,7)
- OccMin : (2,3), (6,7)
- Rigid : (1,2), (6,7)

Enumerations of $\langle ABA \rangle$

- All : (1,3,5), (1,3,6), (1,3,8), (2,3,5), ...
- N-over : (1,3,5), (2,4,6), (6,7,8)
- S-N-over : (1,3,5), (6,7,8)
- OccMin : (2,3,5), (6,7,8)
- Rigid : (6,7,8)

Outline

1. Basics of frequent sequential pattern mining
   - Backgrounds
   - Algorithms for frequent sequential patterns extraction
   - Reducing the number of patterns
   - Available implementations for sequential pattern mining
   - Sum up
2. Sequence mining in long sequences
   - Problem setting
   - Different enumerations
3. Negative sequential patterns
   - Syntax and semantics of NSP
   - Survey results
   - Relationships between containment relations
   - Properties of containment relations
   - Conclusion
4. Graph mining
5. Temporal pattern mining

Motivation for Negative Sequential Patterns

Dataset with hidden frequent patterns

- $\text{drug}_1 \rightsquigarrow \ldots \rightsquigarrow \text{drug}_n \rightsquigarrow \text{disease}$
- $\text{drug}_1 \rightsquigarrow \ldots \rightsquigarrow \text{new_drug} \rightsquigarrow \ldots \rightsquigarrow \text{drug}_n$

  $\rightarrow$ disease appears only when no new_drug have been delivered

Frequent sequential pattern mining

- **positivism**: frequent pattern mining algorithms extract only subsequences that occur !
  - $\text{drug}_1 \rightsquigarrow \ldots \rightsquigarrow \text{drug}_n$

$\rightarrow$ not really useful for our problem

What kind of pattern would be interesting?

$\rightarrow$ patterns that may highlight the **absence** of an item (so called *negative items*)

- $\text{drug}_1 \rightsquigarrow \ldots \rightsquigarrow$ **no** new_drug $\rightsquigarrow \ldots \rightsquigarrow \text{drug}_n \rightsquigarrow$ **no** disease

Motivation: examples of some interesting patterns

In monitoring

A system becames faulty while some maintenance operations have not been performed before despite an alert event.

Customer-Basket

For a lot of customers, If some market-place customer has not received special offers or coupons for a long time then she/he has a high probability of churning while if she/he were provided such special offers she/he should remain loyal to her/his market-place

CRM

There is a log of clients that make a phone-call for reason R, without previously consulting the dedicated web page even if he/she visited the compagny in website.

Pharmaco-epidemiology

There is a lot of patients that have an treatment for a disease but does not visit a specialist have a longer cure

Negative sequential patterns in the State of the Art

- Few algorithms extract negative sequential patterns
  - eNSP [CDZ16] and its variants
  - NegGSP [ZZZC09]
  - Gong et al [GLD15]
  - PNSP [HLC08]
- but ...
  - Incomplete algorithms ...
  - Zheng et *al.* [ZZZC09]: "the [*anti-monotonicity*] doesn't apply to negative sequential pattern"

1. Do state of the art algorithms really extract the same patterns?
2. Is there an efficient, complete and correct algorithm to extract some frequent NSP?

Negative patterns: a syntactic definition I

Definition (Negative sequential patterns (NSP))

A negative pattern $\mathrm{p} = \langle p_1 \ \neg q_1 \ p_2 \ \neg q_2 \ \ldots \ p_{n-1} \ \neg q_{n-1} \ p_n \rangle$ is a finite sequence where $p_i \subseteq \mathcal{I} \setminus \emptyset$ for all $i \in [n]$ and $q_i \subseteq \mathcal{I}$ for all $i \in [n-1]$. The *length* of a NSP, denoted $|\mathrm{p}|$ is n, its number of itemsets (negative or positive). $\mathrm{p}^+ = \langle p_1 \ \ldots \ p_n \rangle$ is so-called the positive part of the NSP.

Syntactic limitations on negative sequential patterns

- a NSP cannot start neither finish by a negative pattern,
- a NSP cannot have two successive negative itemsets.
- a NSP cannot specify positive and negative items at the same time.

Containment relation: a survey I

If the sequential pattern p $= \langle (ca)\ d\ e \rangle$, indicate in which sequences of the Table the pattern p appears

id	*Sequence*
p_0	$\langle e\ (caf)\ d\ b\ e\ d \rangle$
p_1	$\langle c\ a\ d\ b\ e\ d \rangle$
p_2	$\langle e\ (ca)\ d \rangle$
p_3	$\langle d\ e\ (ca)\ b\ d\ b\ e\ f \rangle$
p_4	$\langle c\ e\ b\ (fac)\ d\ e\ c \rangle$

Containment relation: a survey II

Let be the sequential pattern p $= \langle c \neg d\ e \rangle$. Indicate the sequences of the base below in which, according to you, p appears.

id	*Sequence*
s_0	$\langle f\ f\ c\ b\ d\ a\ e \rangle$
s_1	$\langle f\ c\ b\ f\ a\ e \rangle$
s_2	$\langle b\ f\ c\ b\ a \rangle$
s_3	$\langle b\ c\ b\ e\ d \rangle$
s_4	$\langle f\ a\ c\ e\ b \rangle$

Containment relation: a survey III

Let the sequential pattern p $= \langle d \neg(af)\ b \rangle$. Indicate the sequences of the base below in which, according to you, p appears.

id	*Sequence*
i_0	$\langle e\ e\ d\ a\ b\ e \rangle$
i_1	$\langle d\ (af) b\ c \rangle$
i_2	$\langle e\ d\ (fc)\ b \rangle$
i_3	$\langle e\ c\ d\ (ec)\ b \rangle$
i_4	$\langle d\ (fa)\ b\ e \rangle$

Containment relation: a survey IV

Let be the sequential pattern p $= \langle f \neg(ea)\ d \rangle$. Indicate the sequences from the base below in which you think p appears.

id	*Sequence*
e_0	$\langle b\ b\ f\ c\ e\ d\ b \rangle$
e_1	$\langle b\ f\ e\ a\ c\ b\ d \rangle$
e_2	$\langle f\ c\ (ea)\ b\ c\ d\ c \rangle$
e_3	$\langle b\ c\ f\ b\ c\ c\ d \rangle$

Containment relation: a survey V

Let the sequence pattern p $= \langle b\ \neg e\ f \rangle$. Indicate the sequences of the base below in which you think p appears.

id	*Sequence*
o_0	$\langle b\ a\ f\ d\ b\ d\ f \rangle$
o_1	$\langle b\ a\ f\ d\ e\ b\ d\ f \rangle$
o_2	$\langle d\ b\ e\ c\ a\ d\ f\ b\ d\ e\ f \rangle$
o_3	$\langle b\ a\ f\ b\ a\ e\ f \rangle$

Semantics of NSP

The **containment relation** between a NSP p and a sequence s defines the **semantic** of NSP

Semantic of negative patterns: itemset non-inclusion

Definition (non inclusion)

We introduce two operators relating two itemsets $P \subseteq \mathcal{I} \setminus \emptyset$ and $I \subseteq \mathcal{I}$:

- partial non inclusion: $P \not\preceq I \Leftrightarrow \exists e \in P,\ e \notin I$
- total non inclusion: $P \not\sqsubseteq I \Leftrightarrow \forall e \in P, e \notin I$

$\mathcal{D}_1 =$

$s_1 = \langle b\ f\ a \rangle$
$s_2 = \langle b\ (cf)\ a \rangle$
$s_3 = \langle b\ (df)\ a \rangle$
$s_4 = \langle b\ (ef)\ a \rangle$
$s_4 = \langle b\ (cdef)\ a \rangle$

	partial non inclusion $\not\preceq$	total non inclusion $\not\sqsubseteq$
$\langle b \neg ca \rangle$	1, 3, 4	1, 3, 4
$\langle b \neg (cd) a \rangle$	1, 2, 3, 4	1, 4
$\langle b \neg (cde) a \rangle$	1, 2, 3, 4	1
$\langle b \neg (cdeg) a \rangle$	1, 2, 3, 4, 5	1
	monotonic	anti monotonic

Soft/strict-embeddings I

Definition (Strict and soft embeddings of negative patterns)

Let s $= \langle s_1 \ \ldots \ s_n \rangle$ be a sequence and p $= \langle p_1 \ \neg q_1 \ \ldots \ \neg q_{m-1} \ p_m \rangle$ be a negative sequential pattern.
e $= (e_i)_{i \in [m]} \in [n]^m$ is a **soft-embedding** ($\circ$) of pattern p in sequence s iff:

- $p_i \subseteq s_{e_i}, \forall i \in [m]$
- $q_i \not\subseteq s_j, \ \forall j \in [e_i + 1, e_{i+1} - 1]$ for all $i \in [m-1]$

e $= (e_i)_{i \in [m]} \in [n]^m$ is a **strict-embedding** ($\bullet$) of pattern p in sequence s iff:

- $p_i \subseteq s_{e_i}, \forall i \in [m]$
- $q_i \not\subseteq \bigcup_{j \in [e_i+1, e_{i+1}-1]} s_j$ for all $i \in [m-1]$

Soft/strict-embeddings II

Example (*soft*/*strict*-embeddings)

$\mathcal{D}_2 =$

$s_1 = \langle a\ c\ b\ e\ d \rangle$
$s_2 = \langle a\ (bc)\ e\ d \rangle$
$s_3 = \langle a\ b\ e\ d \rangle$
$s_4 = \langle a\ e\ d \rangle$

- $np_3 = \langle a \neg(bc) d \rangle$

$\circ$-embedding
$(\forall j \in [e_{i-1} + 1, e_{i+1} - 1],\ p_i \not\subseteq s_j)$:

$s_1 = \langle a\ c\ b\ e\ d \rangle$
~~$s_2 = \langle a\ (bc)\ e\ d \rangle$~~
$s_3 = \langle a\ b\ e\ d \rangle$
$s_4 = \langle a\ e\ d \rangle$

$\bullet$-embedding $(p_i \not\subseteq \bigcup_{j \in [e_{i-1}+1, e_{i+1}-1]} s_j)$:

~~$s_1 = \langle a\ c\ b\ e\ d \rangle$~~
~~$s_2 = \langle a\ (bc)\ e\ d \rangle$~~
$s_3 = \langle a\ b\ e\ d \rangle$
$s_4 = \langle a\ e\ d \rangle$

Negative pattern occurrence I

Definition (Negative pattern occurrence)

Let s be a sequence, p be a negative sequential pattern, and p^+ the positive part of p. Let $\not\sqsubseteq \in \{\not\sqsubseteq, \not\preceq\}$ be a itemset non-inclusion operator, and $\bullet \in \{\circ, \bullet\}$ correspond to the embedding strategy.

- Pattern p *softly-occurs* in sequence s, denoted $p \preceq_{\bullet}^{\not\sqsubseteq} s$, iff there exists at least one embedding of p in s.
- Pattern p *strictly-occurs* in sequence s, denoted $p \sqsubseteq_{\bullet}^{\not\sqsubseteq} s$, iff for each embedding e of p^+ in s, e is also an embedding of p in s, and there exists at least one embedding e of p^+.

($\circ$: soft-embedding, and $\bullet$: strict-embedding)

Negative pattern occurrence II

Example

$\mathcal{D}_3 =$

$s_1 = \langle a\ b\ e\ d \rangle$
$s_2 = \langle a\ b\ c\ a\ d\ e\ b\ d \rangle$
$s_3 = \langle a\ b\ c\ d\ b\ d \rangle$
$s_4 = \langle a\ e\ d\ b\ e\ d\ d \rangle$
$s_5 = \langle a\ e\ d\ b\ c\ e\ d \rangle$

- $np_4 = \langle ab\neg cd \rangle$

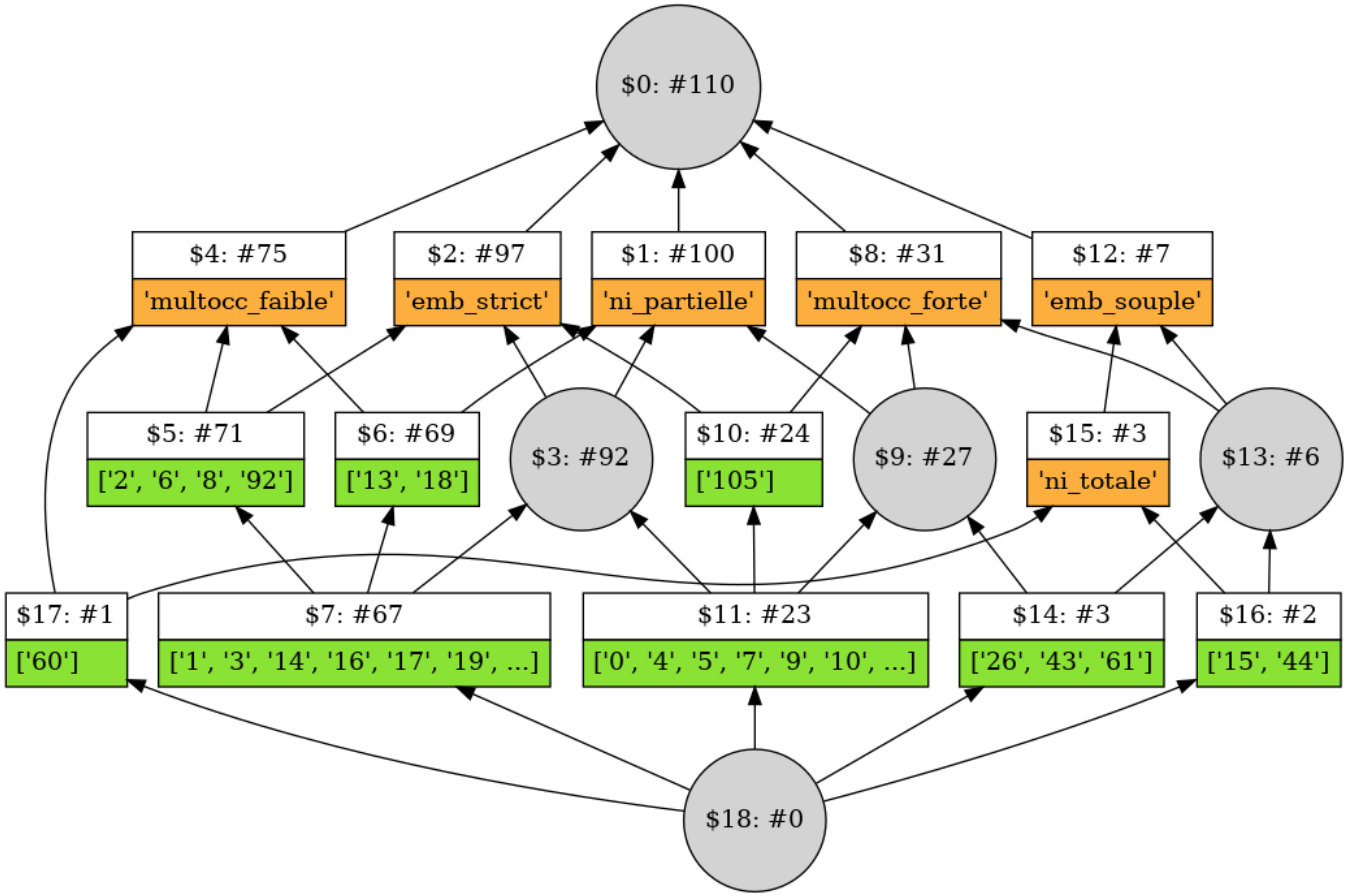
softly-occur (*there exists*):

$s_1 = \langle a\ b\ e\ d \rangle$
$s_2 = \langle a\ b\ c\ a\ d\ e\ b\ d \rangle$
$s_3 = \langle a\ b\ c\ d\ b\ d \rangle$
$s_4 = \langle a\ e\ d\ b\ e\ d\ d \rangle$
~~$s_5 = \langle a\ e\ d\ b\ c\ e\ d \rangle$~~

strict-occur (*for all positives*):

$s_1 = \langle a\ b\ e\ d \rangle$
~~$s_2 = \langle a\ b\ c\ a\ d\ e\ b\ d \rangle$~~
~~$s_3 = \langle a\ b\ c\ d\ b\ d \rangle$~~
$s_4 = \langle a\ e\ d\ b\ e\ d\ d \rangle$
~~$s_5 = \langle a\ e\ d\ b\ c\ e\ d \rangle$~~

Results of the survey



- that two semantics are mainly used : $\sqsubset_{\circ}^{\not z}$ and $\prec_{\circ}^{\not z}$

Dominance relation

We denote Θ the set of containment relations:

$$\Theta = \left\{ \preceq_{\circ}^{\not\sqsubseteq}, \preceq_{\bullet}^{\not\sqsubseteq}, \preceq_{\circ}^{\not\preceq}, \preceq_{\bullet}^{\not\preceq}, \sqsubseteq_{\circ}^{\not\sqsubseteq}, \sqsubseteq_{\bullet}^{\not\sqsubseteq}, \sqsubseteq_{\circ}^{\not\preceq}, \sqsubseteq_{\bullet}^{\not\preceq} \right\}$$

Is there relationships between containment relations?

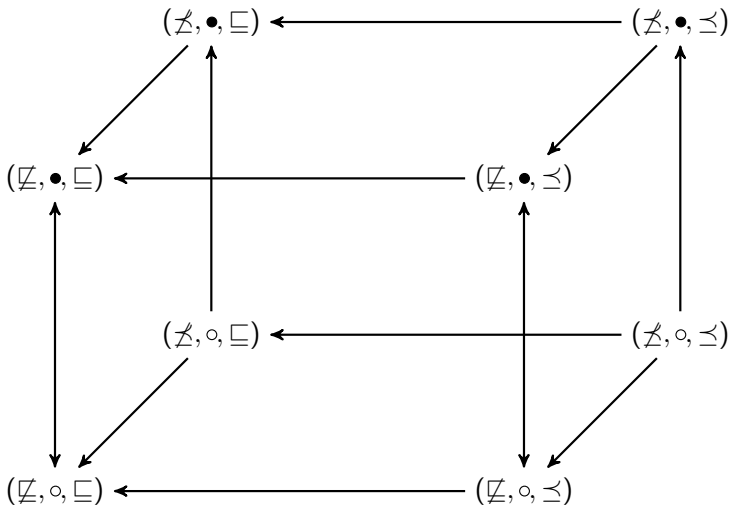


Figure: Summary of the dominance relation between relations of Θ. An arrow shows that the operator at the destination is dominated by the operator at the origin. Unidirectional arrows indicates that the dominance relation holds in one direction but not in the other direction.

Interesting properties: partial orders and anti-monotonicity

Requirements for pattern mining algorithms

→ Anti-monotonicity of the containment relation

→ A partial order on the set of negative patterns

Definition (Anti-monotonicity on $(\mathcal{N}, \ltimes)$)

Let $\theta \in \Theta$ be a containment relation, θ is anti-monotonic on $(\mathcal{N}, \ltimes)$ iff for all $p, p' \in \mathcal{N}$ and all sequence s:

$$p \ltimes p' \implies (p'\theta s \implies p\theta s)$$

where $\ltimes$ is a partial order.

1. Is there interesting partial orders on NSP?
2. For which containment relations?

Interesting properties: partial orders and anti-monotonicity

1. **Is there interesting partial orders on the set of NSP ($\mathcal{N}$)?**
2. For which containment relations?

Some results with $\sqsubseteq$ (sum up)

- Proposal of three partial orders on $\mathcal{N}$
  - $p \lhd p'$: inclusion of positives and negatives itemsets
  - $p \lhd p'$: extension by positive itemsets only at the end
  - $p \lhd\!\!\cdot\, p'$: same positive, inclusion in negatives
- Properties
  - not any anti-monotonic containment relation on $(\mathcal{N}, \lhd)$
  - $\preceq_{\bullet}^{\sqsubseteq}$ is anti-monotonic on $(\mathcal{N}, \lhd)$, where $\bullet \in \{\circ, \bullet\}$.
  - $\dashv\!\!\preceq_{\bullet}^{\sqsubseteq}$ is anti-monotonic on $(\mathcal{N}, \lhd\!\!\cdot)$, where $\dashv\!\!\preceq \in \{\preceq, \sqsubseteq\}$ and $\bullet \in \{\circ, \bullet\}$.
- similar partial orders/properties with $\not\sqsubseteq$

Interesting properties: partial orders and anti-monotonicity

1. Is there interesting partial orders on the set of NSP ($\mathcal{N}$)?
2. **For which containment relations?**

Some results with $\sqsubseteq$ (sum up)

- Proposal of three partial orders on $\mathcal{N}$
  - $p \lhd p'$: inclusion of positives and negatives itemsets
  - $p \lhd p'$: extension by positive itemsets only at the end
  - $p \lhd\!\!\cdot\, p'$: same positive, inclusion in negatives
- Properties
  - not any anti-monotonic containment relation on $(\mathcal{N}, \lhd)$
  - $\preceq_{\bullet}^{\sqsubseteq}$ is anti-monotonic on $(\mathcal{N}, \lhd)$, where $\bullet \in \{\circ, \bullet\}$.
  - $\dashv\!\!\preceq_{\bullet}^{\sqsubseteq}$ is anti-monotonic on $(\mathcal{N}, \lhd\!\!\cdot)$, where $\dashv\!\!\preceq \in \{\preceq, \sqsubseteq\}$ and $\bullet \in \{\circ, \bullet\}$.
- similar partial orders/properties with $\not\preceq$

Come back to pattern mining

For all pairs of NSPs $p, p' \in \mathcal{N}$:

$$p \lhd p' \implies supp_{\sqsubseteq,\bullet,\preceq}(p') \leq supp_{\sqsubseteq,\bullet,\preceq}(p) \quad (1)$$

$$p \lhd\!\!\!\!\cdot\; p' \implies supp_{\sqsubseteq,\bullet,\dashv}(p') \leq supp_{\sqsubseteq,\bullet,\dashv}(p) \quad (2)$$

Toward new pattern mining algorithms

- $\lhd\!\!\!\!\cdot$ can not be practically used
- $\lhd$ is a partial order that can be used to traverse the search space
  - $\rightarrow$ NegPSpan algorithm [?]
- What's about the other semantics?
  - $\rightarrow$ use dominance relation to extract patterns within an upper-bound

Conclusions

Our initial questions

1. **Do state of the art algorithms really extract the same patterns?**
   - → No! The state of the art algorithm does not use the same semantics of NSP
   - → The problem comes from the overload of the $\neg$ symbol
2. **Is there an efficient, complete and correct algorithm to extract some frequent NSP?**
   - → Yes! it is possible to use the classical anti-monotonicity trick to design new algorithms
   - → We can benefit from years of research in frequent pattern mining to improve our algorithms
   - → But only works for two containment relation (two semantics) ...

Lessons learnt to understand pattern mining algorithms

- Positiveness of the pattern mining algorithm makes the analyst blind about important pieces of information in the dataset
- The **containment relation** defines the semantics of the patterns
  - Simple notations does not mean that it is simple to understand
- Anti-monotonicity comes from **both**:
  1. the structure of the pattern set (order)
  2. the enumeration definition (cf episodes)

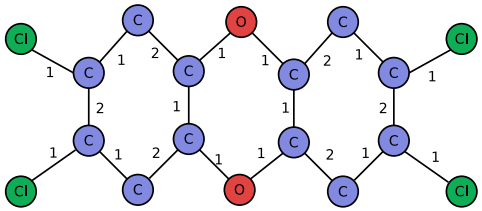
Outline

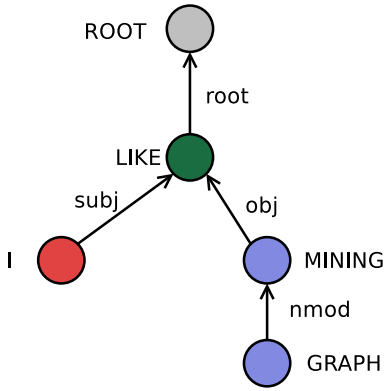
1. Basics of frequent sequential pattern mining
   - Backgrounds
   - Algorithms for frequent sequential patterns extraction
   - Reducing the number of patterns
   - Available implementations for sequential pattern mining
   - Sum up
2. Sequence mining in long sequences
   - Problem setting
   - Different enumerations
3. Negative sequential patterns
   - Syntax and semantics of NSP
   - Survey results
   - Relationships between containment relations
   - Properties of containment relations
   - Conclusion
4. Graph mining
5. Temporal pattern mining

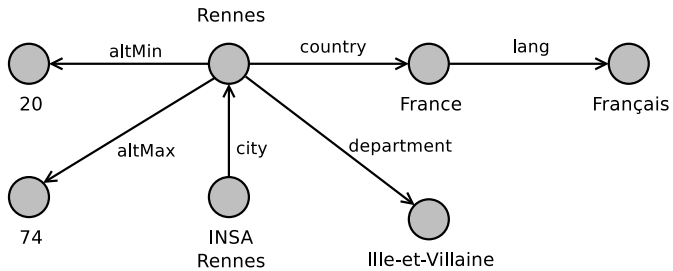
1. Graphs notions and problem statement
   - Graph definitions
   - Support in graphs
2. Pattern-merging algorithms (Apriori-based, BFS)
3. Pattern-growth algorithms (DFS)
4. Canonical codes
5. Conclusion

- Graph $G = (V, E)$: data structure with a set of *vertices* V and a set of *edges* $E \subseteq V \times V$ connecting them
  - *Undirected* graph: edges (u, v) and (v, u) are the same.
  - *Labeled* graph $G = (V, E, l)$: labeling function l associating labels to vertices and edges.
  - Most graph mining approaches focus on undirected labeled graphs.
- Graphs are sometimes called *networks* depending on the domain

- Graphs are a powerful and expressive structure to represent data, used in many domains: molecules, physical networks (telco), social networks, text corpora, program traces (call graph), semantic web...

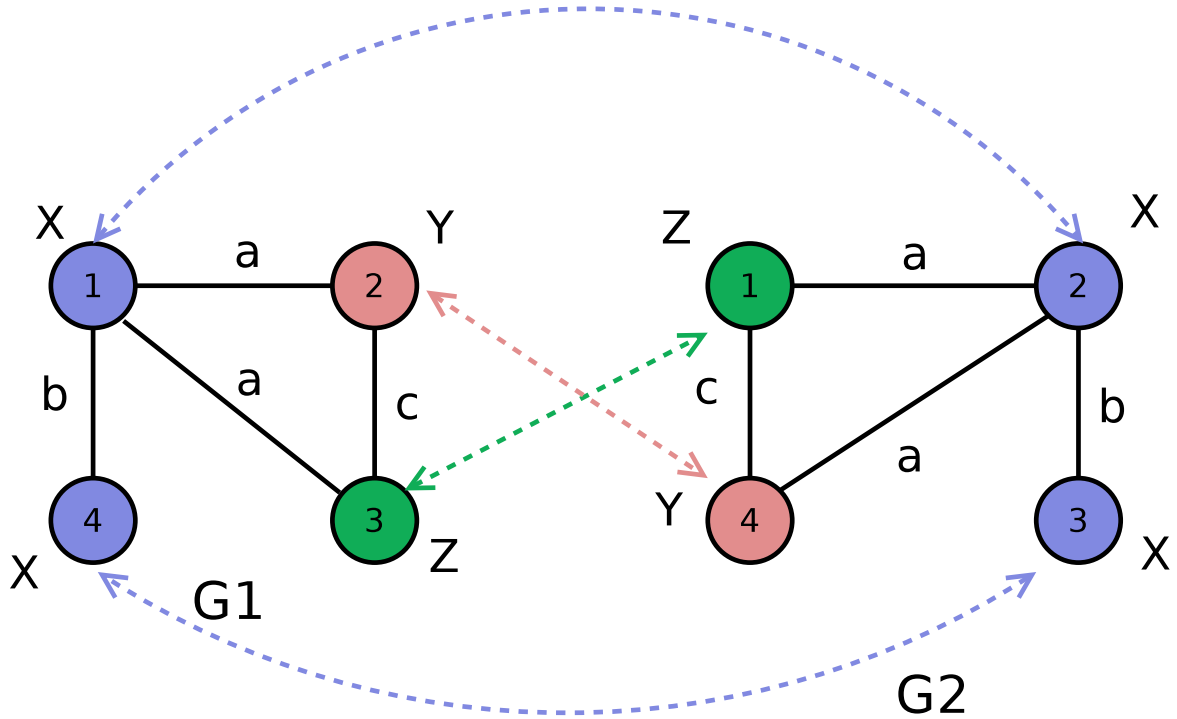






(Sub)graph isomorphism

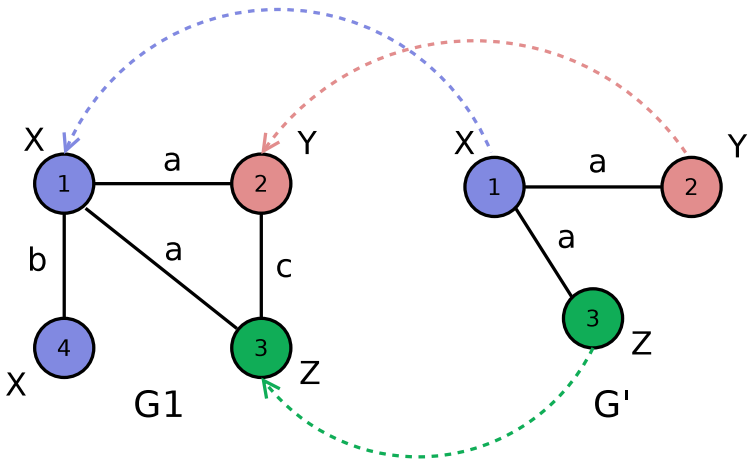
- Graphs can appear different but actually have the same structure
- *Graph isomorphism*: recognizing if two graphs are the same
  - Isomorphism: bijective function mapping vertices of G1 to vertices of G2 so that edges and labels are preserved



G1 and G2 are isomorphic

(Sub)graph isomorphism

- *Subgraph isomorphism*: recognizing if a graph is part of another graph
  - A graph G' is subgraph isomorphic to a graph G if there exist an injective function $\varepsilon \in V' \rightarrow V$ such that $\forall e = (u, v) \in E' \;\; (\varepsilon(u), \varepsilon(v)) \in E$; $\forall v \in V' \;\; l(\varepsilon(v)) = l(v)$; $\forall e \in E' \;\; l(\varepsilon(e)) = l(e)$.
  - We call ε an *embedding* or *occurrence* of G' in G



G' is a subgraph of G1

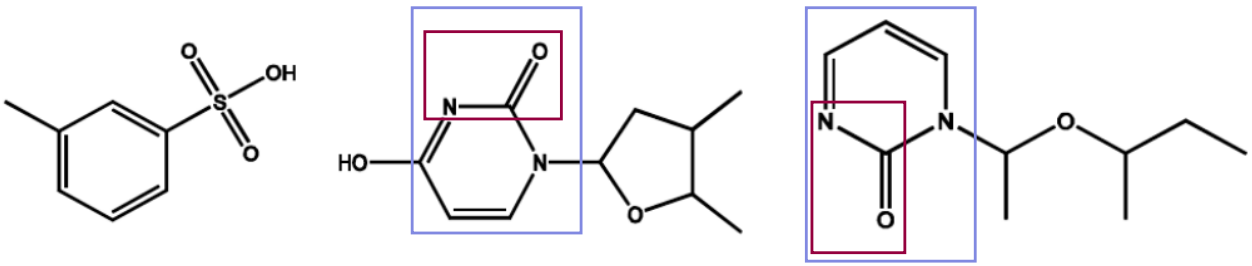
- Subgraph isomorphism search is NP-complete!
  - In practice if labels are diverse enough, it can be computed in reasonable time. *But sometimes data does not play nice*

Graph pattern mining

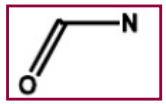
Graph pattern mining is essentially the problem of discovering frequent subgraphs (patterns) occurring in the input data graph(s).

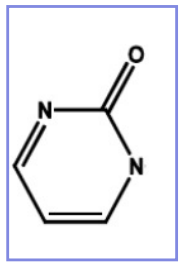
- Find structures describing interesting concepts in the data
- Abstract parts of the data as instances of patterns
- Learn about the data by looking at what is frequent in it

Data:



Example patterns:





What is frequent?

- Discovering frequent subgraphs = discovering subgraphs with a support greater than user-given parameter *minsup*
- Support definition depends on the kind of graph data
  - Base idea similar to other pattern mining domains: "how often is the pattern found in the input data?"

Two families of graph data:

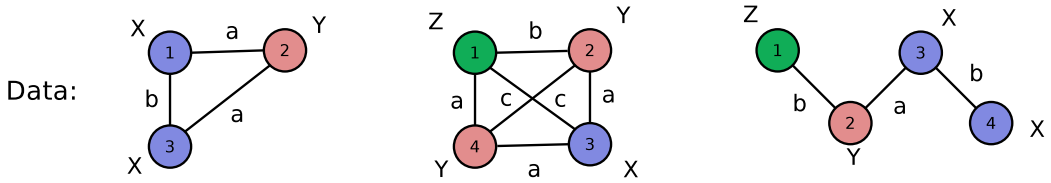
- Graph collection: a (generally large) set of (generally small) graphs.
  - e.g. molecules, sentences
- Single graph: the data is a unique (generally large) graph
  - e.g. semantic web, social networks, DNA

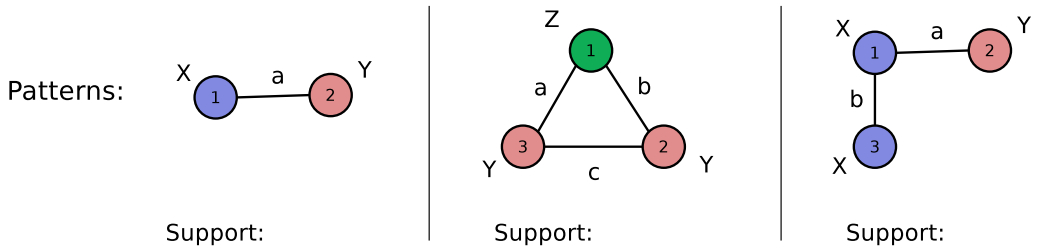
What is frequent in a graph collection?

Let $\mathcal{D}$ be a graph collection and P a graph pattern,

$$support(P) = \frac{|\{g \in \mathcal{D} \mid P \text{ is subgraph-isomorphic to } g\}|}{|\mathcal{D}|}$$

- Each graph of the collection can only contribute once to the support, even if it has multiple occurrences of the pattern



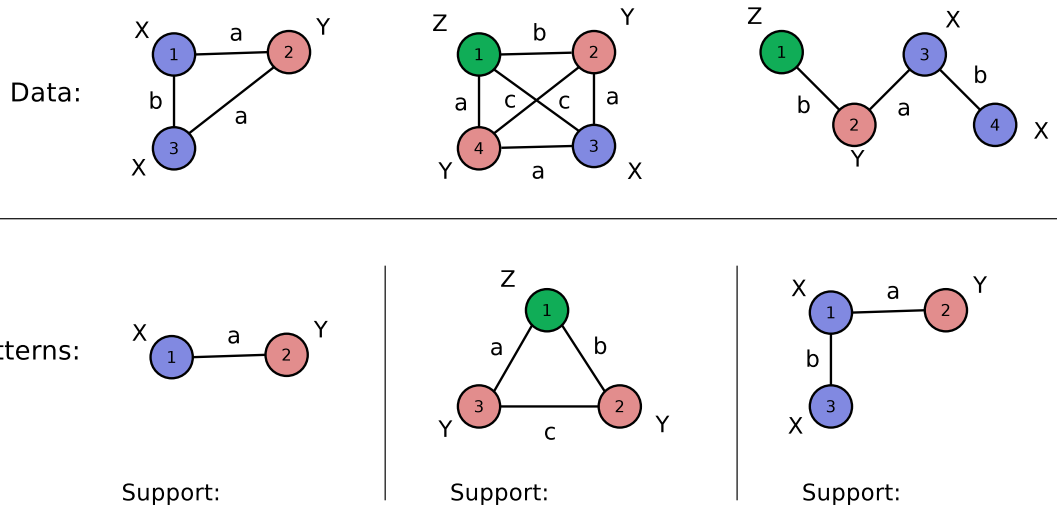


What is frequent in a graph collection?

Let $\mathcal{D}$ be a graph collection and P a graph pattern,

$$support(P) = \frac{|\{g \in \mathcal{D} \mid P \text{ is subgraph-isomorphic to } g\}|}{|\mathcal{D}|}$$

- Each graph of the collection can only contribute once to the support, even if it has multiple occurrences of the pattern

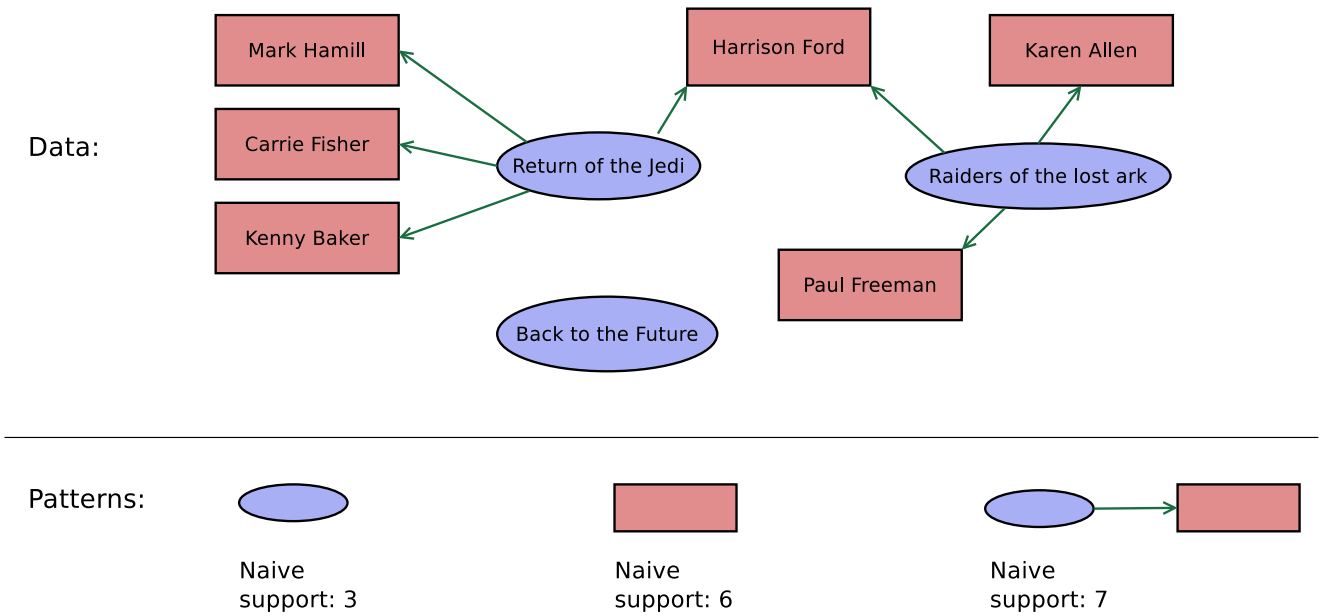


- This measure is **anti-monotonic**: support of a graph is lower or equal to support of its subgraphs

What is frequent in a single graph?

Naive solution

Count how many occurrences the pattern has in the graph.



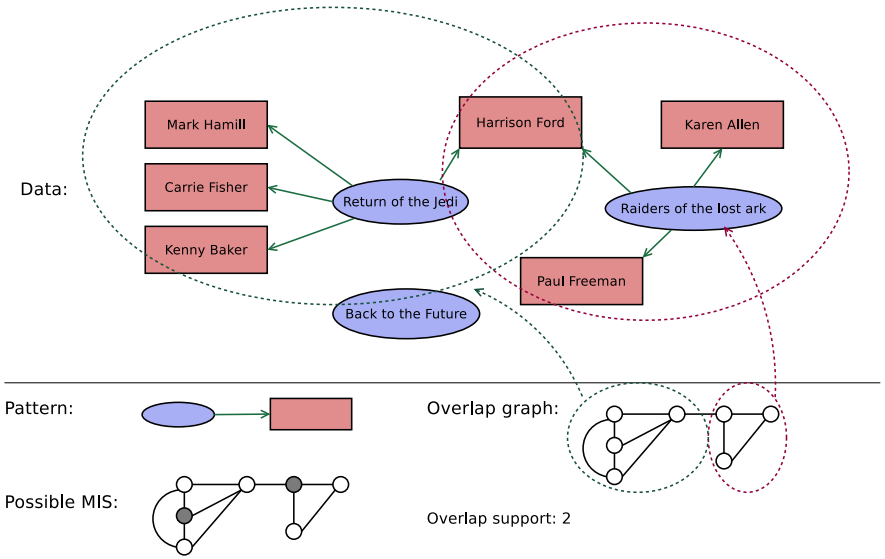
Naive support is **not** anti-monotonic

What is frequent in a single graph?

Overlap-based approaches [Kuramochi and Karypis, 2004]

1. Compute overlap graph of pattern embeddings
2. Support is size of MIS (Maximum Independent Set) of overlap graph

- i.e. maximum number of non-overlapping embeddings of the pattern



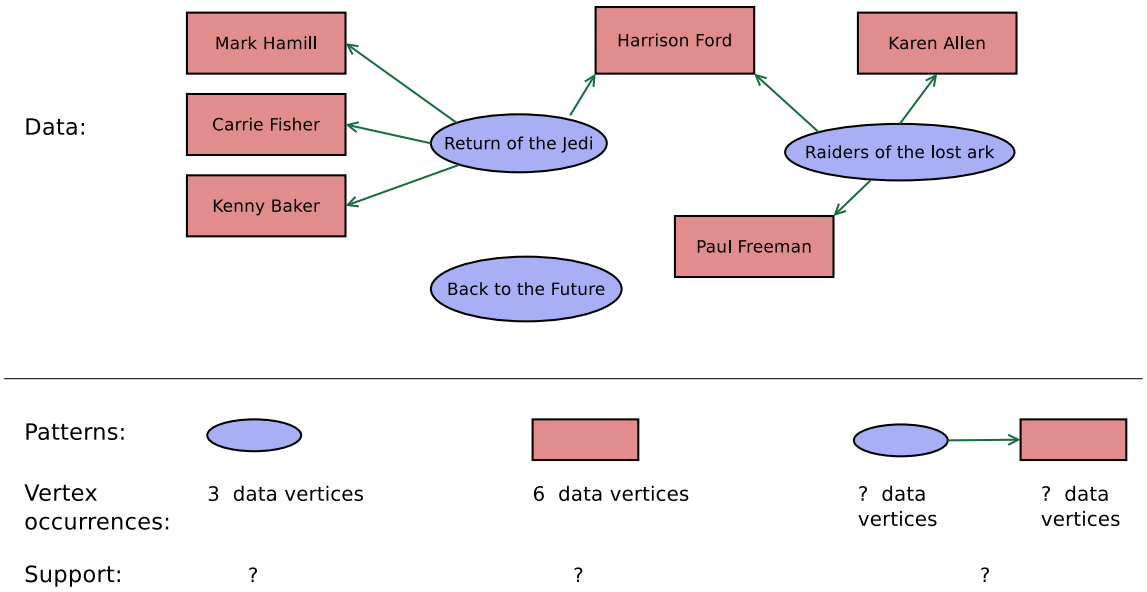
Overlap-based support **is** anti-monotonic
MIS computation is NP-complete

What is frequent in a single graph?

Minimum image based support [Bringmann and Nijssen, 2008]

Let $\mathcal{D}$ be a single data graph and P a graph pattern,

$$support(P) = \min_{v \in V^P} |\{\varepsilon(v) \mid \varepsilon \text{ is an embedding of } P \text{ in } \mathcal{D}\}|$$



Minimum image based support **is** anti-monotonic
Does not need to compute a NP-complete problem

1 Graphs notions and problem statement
- Graph definitions
- Support in graphs

2 Pattern-merging algorithms (Apriori-based, BFS)

3 Pattern-growth algorithms (DFS)

4 Canonical codes

5 Conclusion

Pattern-merging algorithms

If the support measure is anti-monotonic, for a k-size pattern to be frequent, all its (k-1)-size elements must be frequent.

Pattern-merging graph mining algorithms work similarly to Apriori:

0. Given L_k the set of k-size *frequent* patterns
1. Merge *compatible* k-size patterns to create C_{k+1} the set of candidate (k+1)-size patterns
   - Compatible k-size patterns: patterns that have a common (k-1)-size core (i.e. differ in only one element)
2. Prune C_{k+1}: only retain patterns whose *all* (k-1)-size elements are frequent
3. Create L_{k+1} by computing support of all patterns in C_{k+1}
   - If $L_{k+1} = \emptyset$, stop

- Main pattern-merging graph mining algorithms:
  - AGM/AcGM [Inokuchi et al., 2000]
  - FSG [Kuramochi and Karypis, 2001]
  - DPMine [Gudes et al., 2006]
- Main difference is the definition of a k-size pattern: k vertices, k edges, k edge-disjoint paths, ...

Drawbacks of pattern-merging approaches

- Merging k-size patterns to create (k+1)-size pattern requires finding patterns that share a common (k-1)-size core → subgraph isomorphism
- Pruning step: need to verify if (k-1)-size elements of a pattern are frequent → subgraph isomorphism
- Support computation: need to find occurrences of pattern → subgraph isomorphism
  - Can be skipped by storing information in memory → memory consumption
- Breadth-First Search (BFS): need to store all frequent k-size patterns → high memory consumption

Remember that subgraph isomorphism is NP-complete!

1 Graphs notions and problem statement
- Graph definitions
- Support in graphs

2 Pattern-merging algorithms (Apriori-based, BFS)

3 Pattern-growth algorithms (DFS)

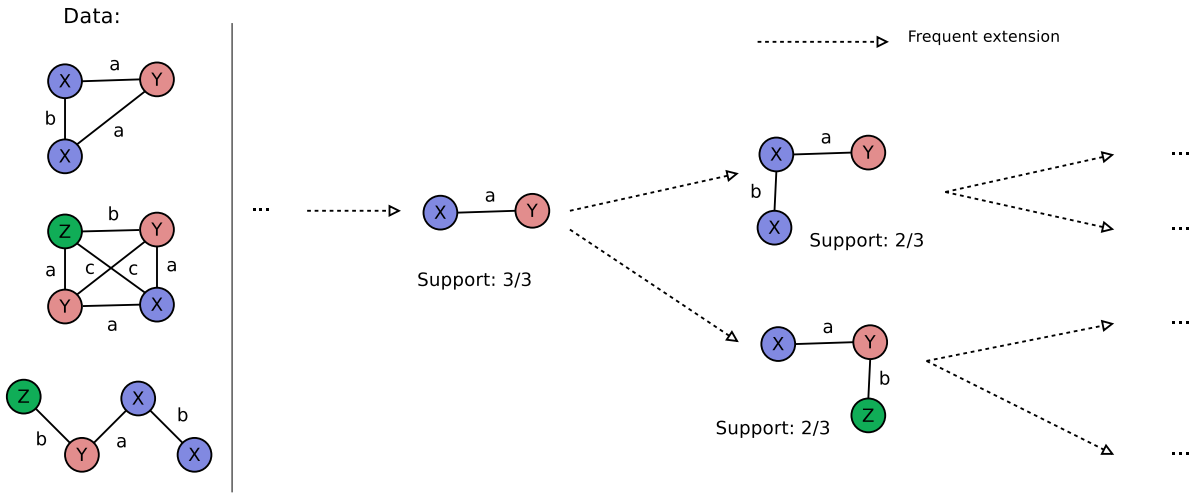
4 Canonical codes

5 Conclusion

Pattern-growth algorithms

Solve drawbacks of pattern-merging algorithms:

- Expand frequent patterns by looking at possible frequent extensions of their embeddings
  - No need to merge patterns → avoid subgraph-isomorphism check: time gain
  - No need to store all k-size patterns to generate (k+1)-size patterns: memory gain
  - Only generate frequent patterns → avoid testing non-frequent candidates: time gain
- Most algorithms in this family use depth-first search to generate patterns → often called DFS algorithms



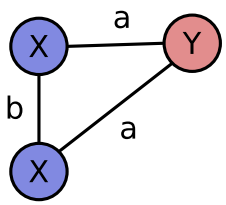
Pattern-growth algorithms

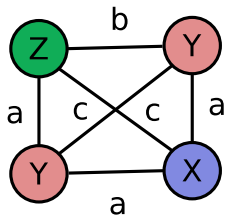
Most used/cited pattern-growth algorithms [Wörlein et al., 2005]:

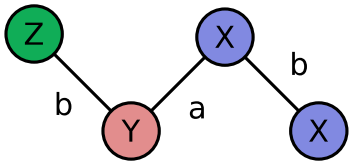
- MoFa [Borgelt and Berthold, 2002]
  - Developed to find substructures in collection of molecules
  - Least efficient of the four because it generates many times the same patterns
- gSpan [Yan and Han, 2002]
  - The most cited
  - Introduces techniques to avoid generating multiple times the same patterns (canonical labeling, DFS with rightmost path expansion)
- FFSM [Wang et al., 2003]
  - Uses both pattern extension and a special efficient join operation
- Gaston [Nijssen and Kok, 2005]
  - Works in phases to avoid subgraph isomorphism as much as possible: starts with simple patterns (paths), used to mine slightly more complex patterns (trees), then graphs.
  - The fastest of the four

Exercise: DFS search

Data:







1 Graphs notions and problem statement
- Graph definitions
- Support in graphs

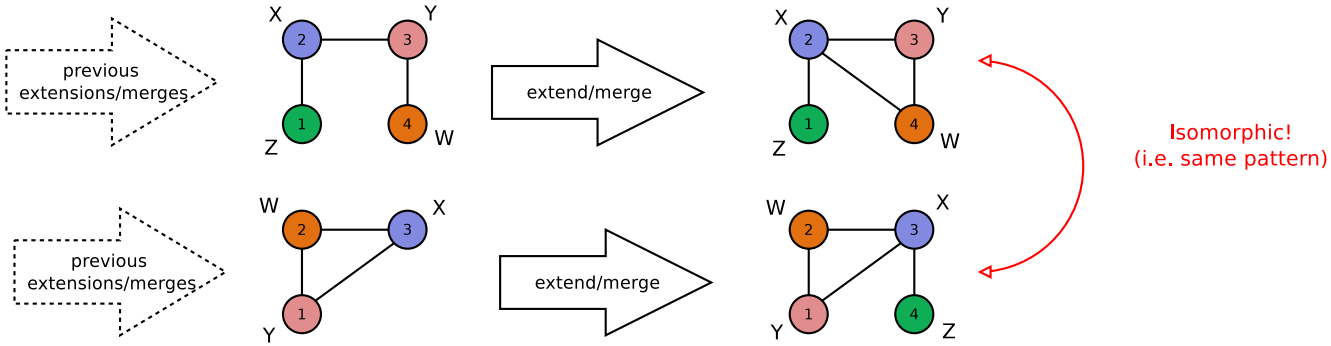
2 Pattern-merging algorithms (Apriori-based, BFS)

3 Pattern-growth algorithms (DFS)

4 Canonical codes

5 Conclusion

Canonical codes



Different search paths may lead to the same pattern!

How to avoid exploring multiple times the same patterns?

- Have a generation strategy that limits duplicates
  - E.g. always expand from the latest expanded vertex (Mofa, gSpan, ...)
  - Does not suffice by itself: see image above
- Detect if a pattern can be found following another search path
  - Naive approach: compare with all generated patterns $\rightarrow$ infeasible in reasonable time and memory
  - **Canonical codes** (gSpan, FFSM, Gaston)

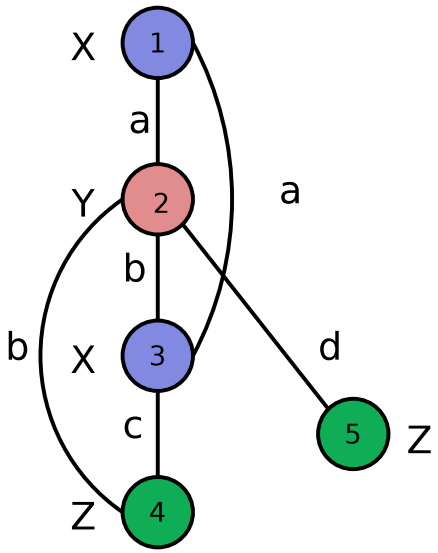
Canonical codes

1. Map each graph (2-dimensions) to a code (1-dimension) such that if two graph have equal codes they are isomorphic
2. Make codes comparable
   - The *minimum possible code* for a graph is called the **canonical code** of the graph[1]
   - Same canonical code $\iff$ isomorphic graphs
   - Canonical code uniquely identifies a graph
3. Only extend patterns on search paths that yield the canonical code for the pattern

[1] The maximum could also be used, it's arbitrary

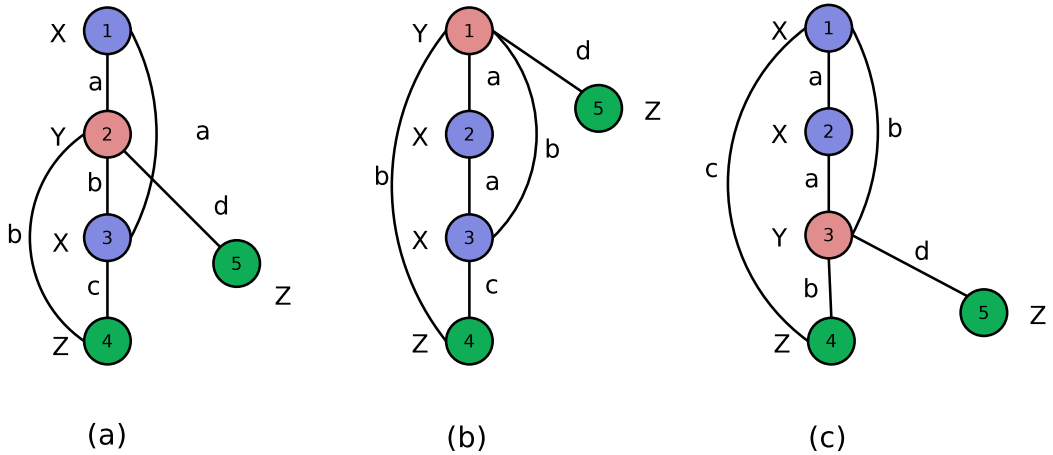
gSpan canonical code

- Code based on DFS construction of the graph (called DFS code)
- Each edge $e = (u, v)$ added to the graph is represented by a code element $(u, v, l(u), l(e), l(v))$



Code
$(1, 2, X, a, Y)$
$(2, 3, Y, b, X)$
$(3, 1, X, a, X)$
$(3, 4, X, c, Z)$
$(4, 2, Z, b, Y)$
$(2, 5, Y, d, Z)$

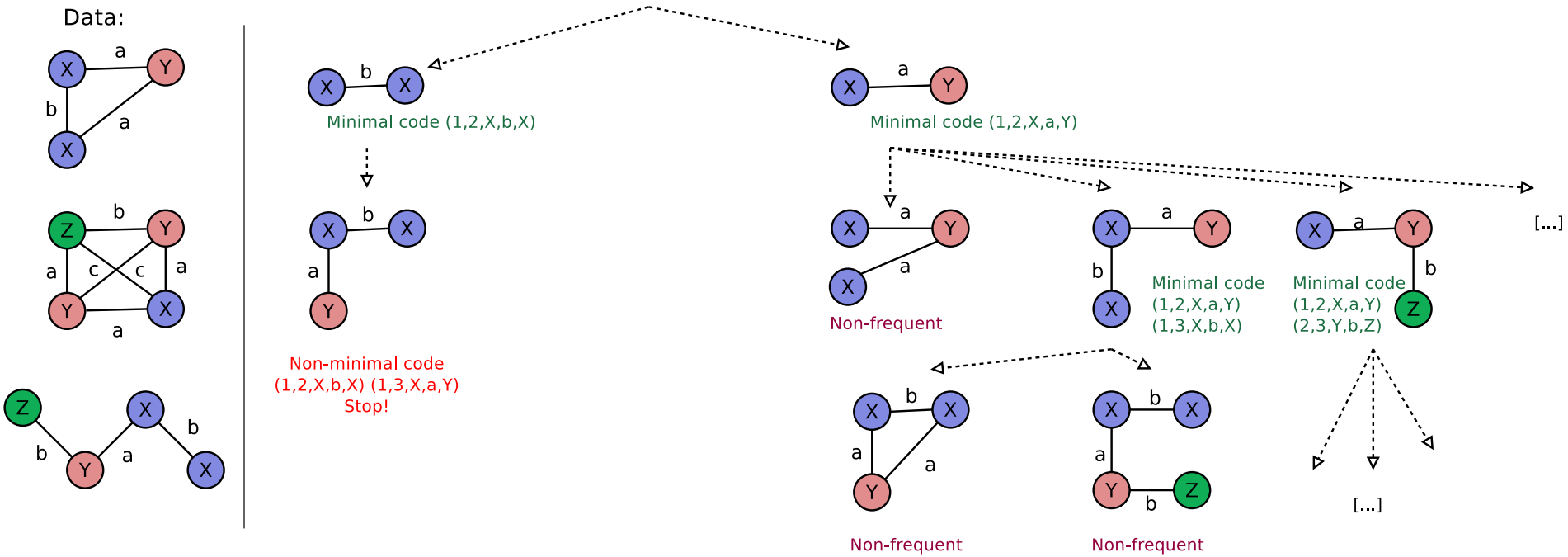
gSpan canonical code



(a)	(b)	(c)
$(1,2,X,a,Y)$	$(1,2,Y,a,X)$	$(1,2,X,a,X)$
$(2,3,Y,b,X)$	$(2,3,X,a,X)$	$(2,3,X,a,Y)$
$(3,1,X,a,X)$	$(3,1,X,b,Y)$	$(3,1,Y,b,X)$
$(3,4,X,c,Z)$	$(3,4,X,c,Z)$	$(3,4,Y,b,Z)$
$(4,2,Z,b,Y)$	$(4,1,Z,b,Y)$	$(4,1,Z,c,X)$
$(2,5,Y,d,Z)$	$(1,5,Y,d,Z)$	$(3,5,Y,d,Z)$

- Same graph can have different DFS codes depending on starting vertices
- Order defined on codes: lexicographic order of code elements
- When a pattern is generated during DFS search, decide if it could have a smaller DFS code. In that case, do not extend the pattern
  - It will be extended in the DFS branch where it has a minimal code
  - Assumes that the DFS search will eventually visit branches with minimal DFS code for any pattern

DFS pruning with canonical codes



1. Graphs notions and problem statement
   - Graph definitions
   - Support in graphs
2. Pattern-merging algorithms (Apriori-based, BFS)
3. Pattern-growth algorithms (DFS)
4. Canonical codes
5. Conclusion

Conclusion

- Graphs are a generic data structure that allows to express a large quantity of *structured* data
- However, graphs have additional complexity w.r.t. simpler data such as itemsets and sequential patterns, which can not be ignored when developing and using graph mining approaches
  - Pattern matching being a NP-complete subgraph isomorphism problem
  - Support computation
  - Recognizing if two graphs are the same (graph isomorphism)
  - ...
- Existing pattern mining approaches are constructed on the same basis as itemset mining (Apriori, pattern-growth), but need additional concepts to avoid too much complexity (e.g. canonical codes)

In pattern mining

The more generic the pattern/data language, the more it allows for expressiveness, but the more pattern mining tends to be difficult

Outline

1. Basics of frequent sequential pattern mining
   - Backgrounds
   - Algorithms for frequent sequential patterns extraction
   - Reducing the number of patterns
   - Available implementations for sequential pattern mining
   - Sum up
2. Sequence mining in long sequences
   - Problem setting
   - Different enumerations
3. Negative sequential patterns
   - Syntax and semantics of NSP
   - Survey results
   - Relationships between containment relations
   - Properties of containment relations
   - Conclusion
4. Graph mining
5. Temporal pattern mining

Toward temporal data mining

Sequential pattern mining

- Sequential pattern mining capture only the **sequentiality** in events
  - dataset of sequences (timestamped are not useful)
  - patterns are subsequent patterns
- more temporal information can be capture through constraints
  - *maxgap* constraint: can be applied on position or on timestamps
  - *maxspan* constraint: idem

  $\rightarrow$ very useful to prevent from extracting useless patterns (*e.g.* relating events very distant in time)

$\Rightarrow$ is it possible to discover patterns with more temporal information?

YES: Temporal pattern mining

- It is possible to define new **pattern domains** that hold temporal information

$\rightarrow$ there will be a hard trade-off: the more complex is the pattern model, the harder it is to extract them

General principles

Defining a new pattern domain

- a pattern domain specifies a **pattern-model**, *i.e.* the shape of the patterns that can be extracted
- the pattern domain defines a **search-space**
- Ideally, we expect to define a search-space such that:
  - it is possible to define a lattice-structure
  - the interestingness measure (frequency) is anti-monotonic
  - → not so trivial: enumerating complex patterns maybe difficult (*cf.* long sequences)
- a **strategy** to explore
  - a complete algorithm (extract all patterns)
    - ideal from the algorithm point of view
    - but the temporal dimension may generate numerous similar patterns: not necessarily useful to extract them all (*cf.* chronicles)
  - often, an heuristic-search that extracts a potentially interesting subset of temporal patterns

Why temporal patterns?

We are studying dynamic systems

- We are interested in how thinks evolves
  - sequence seems a bit simple to model evolutions
  - modeling complex evolution (including parallel events) may be interesting
  - modeling the speed of the evolution may also be insightful
- We are interested in better prediction
  - temporal dimension may help to better discriminate different situations
  - pattern including time may predict accurately when thinks will happens

Examples

- In CRM: predicting when a client will make new purchases
- In care pathway analysis: determining the duration of a treatment that co-occurs with disease(s)

Example: discriminating with temporal information

- Quantitative temporal sequence models the behavior of a dynamic system by sequences of events (the states)
- Events are discrete (finite set)
- Date and duration of events are real-valued

ECG patterns motivation

Given two different behaviors

- Sequences of events are identical: {'P','QRS','T','P'}
- Only temporal information can distinguish the states

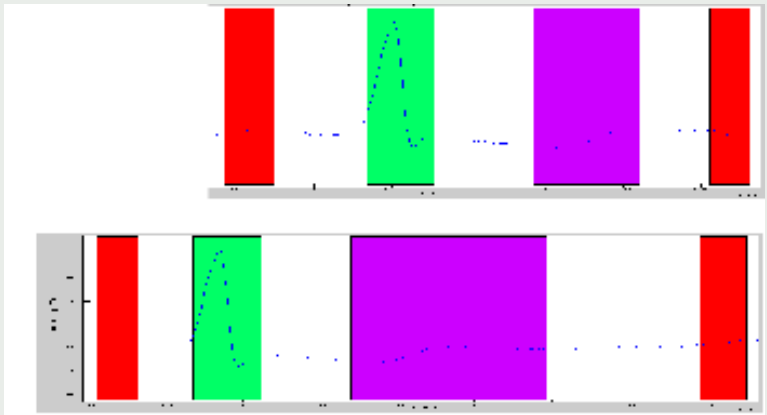
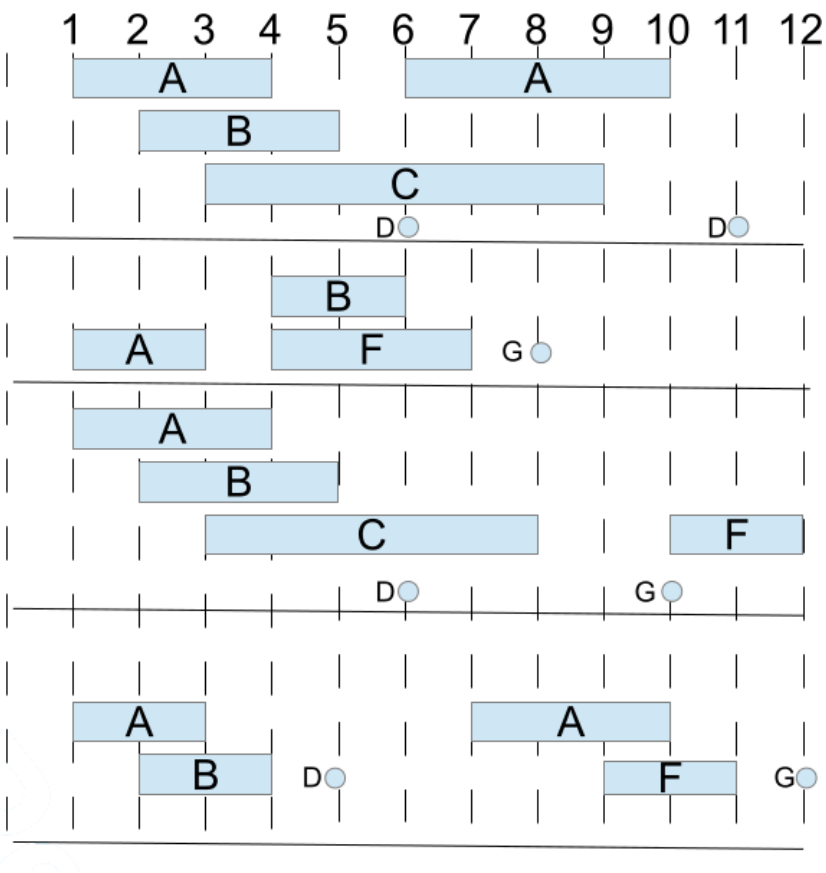


Figure: Two ECG cycles, same patient same time scale

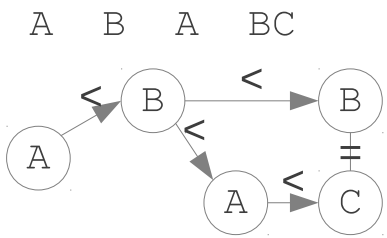
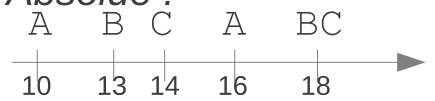
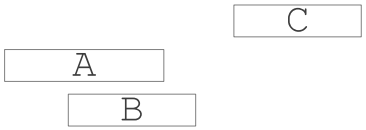
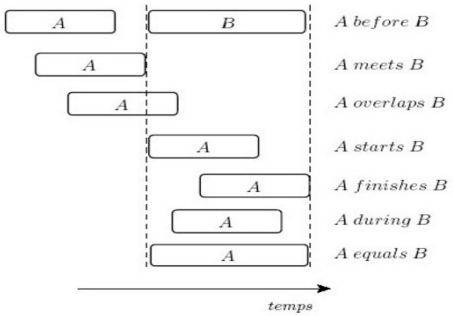
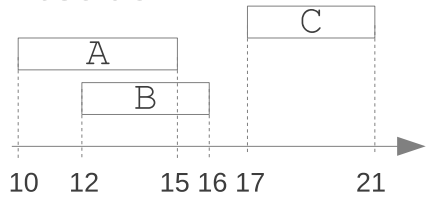
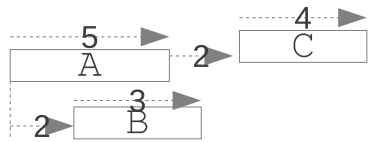
Temporal databases

Seq. Id	Tr. time	Itemset
1	1	A[3]
	2	B[3]
	3	C[6]
	6	A[4] D[0]
	11	D[0]
2	1	A[2]
	4	B[2] F[3]
	8	G[0]
3	1	A[3]
	2	B[3]
	3	C[5]
	6	D[0]
	10	F[2] G[0]
4	1	A[2]
	2	B[2]
	5	D[0]
	7	A[3]
	9	F[2]
	12	G[0]



Temporal databases

Temporal information in sequences

	Ordre/succession	Information temporelle qualitative	Information temporelle quantitative
Ponctuelles	ABCABCABC		*Absolue :*  *Relative :*
Intervalles		 Logique d'Allen (1983)	*Absolue :*  *Relative :* 

State of the art

Brief (incomplete) state of the art

Type de séquence \ Type de motif	Ponctuel, qualitatif	Ponctuel, quantitatif	Intervalle, qualitatif	Intervalle, quantitatif
Ponctuelle, qualitative	*GSP* *PrefixSpan* *SPADE* (Zaki) *DiffSet*, ...			
Ponctuelle, quantitative	Contraintes *MinGap*, *MaxGap* (Han et al.)	*WinEPI*, *MinEPI*, (Manilla et al.)	*TAS*, Giannotti et al.	*Chronicles*, (Dousson et Duong, Cram et al.) *delta-patterns*, Yoshida et al.
Intervalle, qualitative			*ARMADA* (Winarko et al.) *CTMiner* (Chen et al.)	
Intervalle, quantitative			*TprefixSpan* (Kam et Fu) *IEMiner* (Patel et al.)	*QTPrefixSpan* (Nakagaito et al.) *QtempIntMiner* (Guyet, Quiniou) ***QTIPrefixSpan*** ***QTIAPriori***

Thanks for listening

Questions ?

References I

Jean-Francois Boulicaut and Baptiste Jeudy, *Constraint-based data mining*, Data Mining and Knowledge Discovery Handbook (Oded Maimon and Lior Rokach, eds.), Springer US, 2005, pp. 399–416.

Francesco Bonchi and Claudio Lucchese, *Extending the state-of-the-art of constraint-based pattern discovery*, Data Knowl. Eng. 60 (2007), no. 2, 377–399.

Longbing Cao, Xiangjun Dong, and Zhigang Zheng, *e-NSP: Efficient negative sequential pattern mining*, Artificial Intelligence 235 (2016), 156–182.

Emmanuel Coquery, Said Jabbour, Lakhdar Saïs, and Yakoub Salhi, *A SAT-Based approach for discovering frequent, closed and maximal patterns in a sequence*, 20th European Conference on Artificial Intelligence (ECAI'12), 2012, pp. 258–263.

Tias Guns, Anton Dries, Guido Tack, Siegfried Nijssen, and Luc De Raedt, *MiningZinc: A modeling language for constraint-based mining*, Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, AAAI Press, 2013, pp. 1365–1372.

Martin Gebser, Thomas Guyet, René Quiniou, Javier Romero, and Torsten Schaub, *Knowledge-based sequence mining with ASP*, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016, pp. 1497–1504.

Yongshun Gong, Chuanlu Liu, and Xiangjun Dong, *Research on typical algorithms in negative sequential pattern mining*, Open Automation and Control Systems Journal 7 (2015), 934–941.

Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim, *SPIRIT: Sequential pattern mining with regular expression constraints*, Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1999, pp. 223–234.

References II

Sue-Chen Hsueh, Ming-Yen Lin, and Chien-Liang Chen, *Mining negative sequential patterns for e-commerce recommendations*, Proceedings of Asia-Pacific Services Computing Conference, IEEE, 2008, pp. 1213–1218.

Jean-Philippe Métivier, Samir Loudni, and Thierry Charnois, *A constraint programming approach for mining sequential patterns in a sequence database*, CoRR abs/1311.6907 (2013), 1.

H. Mannila, H. Toivonen, and A. I. Verkamo, *Discovering frequent episodes in event sequences*, Journal of Data Mining and Knowledge Discovery 1 (1997), no. 3, 210–215.

Benjamin Negrevergne, Anton Dries, Tias Guns, and Siegfried Nijssen, *Dominance programming for itemset mining*, International Conference on Data Mining (ICDM), 2013.

J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, *Mining sequential patterns by pattern-growth: the prefixspan approach*, IEEE Transactions on Knowledge and Data Engineering 16 (2004), no. 11, 1424–1440.

Jian Pei, Jiawei Han, and Wei Wang, *Mining sequential patterns with constraints in large databases*, Proceedings of the eleventh international conference on Information and knowledge management, ACM, 2002, pp. 18–25.

Christophe Rigotti, http://liris.cnrs.fr/ crigotti/dmt4sp.html.

R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements*, Proceedings of the 5th Int. Conf. on Extending Database Technology, 1996, pp. 3–17.

Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura, *An efficient algorithm for enumerating closed patterns in transaction databases*, International Conference on Discovery Science, Springer, 2004, pp. 16–31.

References III

Xifeng Yan, Jiawei Han, and Ramin Afshar, *Clospan: Mining: Closed sequential patterns in large datasets*, Proceedings of the 2003 SIAM International Conference on Data Mining, SIAM, 2003, pp. 166–177.

M. Zaki, *SPADE: An efficient algorithm for mining frequent sequences*, Machine Learning 42 (2001), no. 1/2, 31–60.

M. J. Zaki, *SPADE: An efficient algorithm for mining frequent sequences*, Machine Learning 42 (2001), no. 1/2, 31–60.

Zhigang Zheng, Yanchang Zhao, Ziye Zuo, and Longbing Cao, *Negative-GSP: An efficient method for mining negative sequential patterns*, Proceedings of the Australasian Data Mining Conference, 2009, pp. 63–67.