

## Part 2

# Mining Patterns in Sequential Data

## Sequential Pattern Mining: Definition

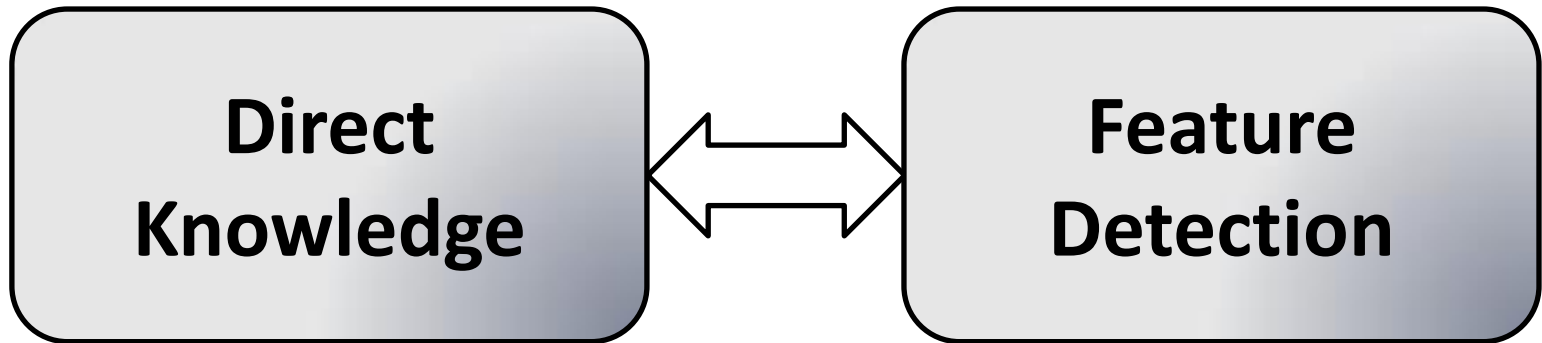
“Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min\_support threshold, sequential pattern mining is to find all of the frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min\_support.”

~ [Agrawal & Srikant, 1995]<sup>1</sup>

“Given a set of data sequences, the problem is to discover sub-sequences that are frequent, i.e., the percentage of data sequences containing them exceeds a user-specified minimum support.”

~ [Garofalakis, 1999]

# Why Sequential Patterns?

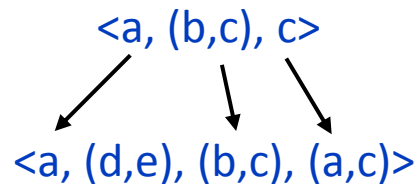


## Notation & Terminology


- Data:
  - Dataset: set of *sequences*
  - Sequence: an ordered list of *itemsets (events)*  $\langle e_1, \dots, e_n \rangle$
  - Itemset: an (unordered) set of items  $e_i = \{i_{i_1}, \dots, i_{i_z}\}$
- $S_{\text{sub}} = \langle s_1, \dots, s_n \rangle$  is a **subsequence** of sequence  $S_{\text{ref}} = \langle r_1, \dots, r_n \rangle$  if:

$$\exists i_1 < \dots < i_n : s_k \subseteq r_{i_k}$$


Example:




is subsequence

More Examples:  jupyter

- Length of a sequence: # items used in the sequence (not unique):  
Example: length  $\langle a, (b,c), a \rangle = 4$

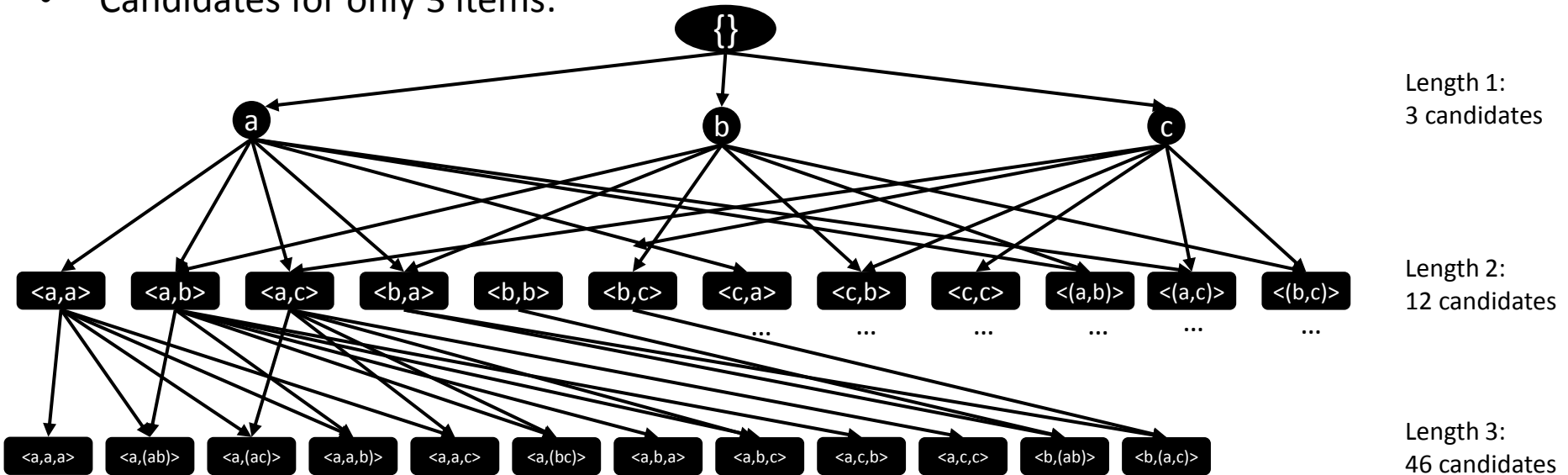
More Examples:  jupyter

## Frequent Sequential Patterns

- Support  $\text{sup}(S)$  of a (sub-)sequence  $S$  in a dataset:  
Number of sequences in the dataset that have  $S$  as a subsequence
- Examples:  jupyter
- Given a user chosen constant  $\text{minSupport}$ :  
Sequence  $S$  is **frequent** in a dataset if  $\text{sup}(S) \geq \text{minSupport}$
  - Task: Find all frequent sequences in the dataset
  - If all sequences contain exactly one event:  
Frequent itemset mining!

## Pattern Space

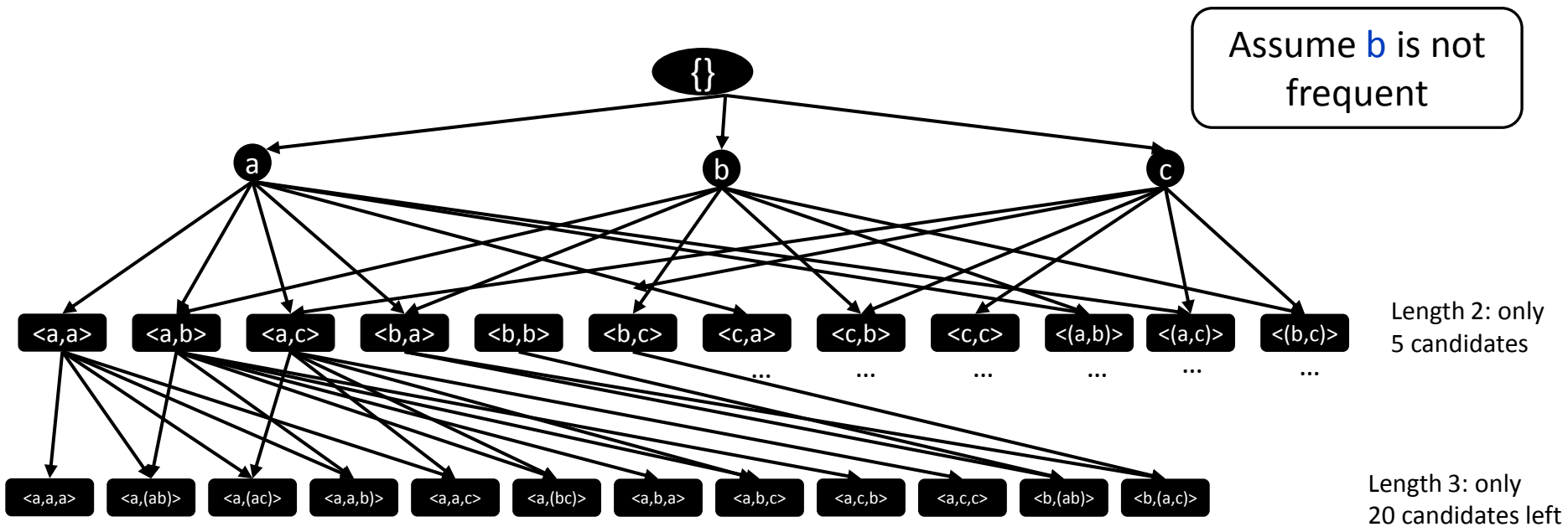
- General approach: enumerate candidates and count
- Problem: “combinatorial explosion”: Too many candidates
- Candidates for only 3 items:



- Candidates for 100 items:
  - Length 1: 100;
  - Length 2:  $100 * 100 * \frac{100 * 99}{2} = 14,950$
  - Length 3:  $\sum_i^{100} \#candidates \text{ for length } i = 2^{100} - 1 \approx 10^{30}$

## Monotonicity and Pruning

- If  $S$  is a subsequence of  $R \rightarrow$  then  $\text{sup}(S)$  is at most as large as  $\text{sup}(R)$
- Monotonicity:  
If  $S$  is not frequent, then it is impossible that  $R$  is frequent!  
E.g.  $\langle a \rangle$  occurs only 5 times, then  $\langle a, b \rangle$  can occur at most 5 times
- Pruning:  
If we know that  $S$  is not frequent, we do not have to evaluate any supersequence of  $S$ !



# Apriori Algorithm (for Sequential Patterns) [Agrawal & Srikant, 1995]

- Evaluate pattern “levelwise” according to their length:
    - Find frequent patterns with length 1
    - Use these to find frequent patterns with length 2
    - ...
  
  - First find frequent single items
  - At each level do:
    - Generate candidates from frequent patterns of the last level
      - For each pair of candidate sequences (A, B):
        - Remove first item of A and the last item of B
        - If these are then equal:  
generate a new candidate by adding the last item of b at the end of a
      - E.g.:  $A = \langle a, (b,c), d \rangle$ ,  $B = \langle (b,c), (d,e) \rangle \rightarrow$  new candidate  $\langle a, (b,c), (d,e) \rangle$
- 
- Prune the candidates (check if all subsequences are frequent)
- Check the remaining candidates by counting

More Examples:  jupyter



## Extensions based on Apriori:

- **Generalized Sequential Patterns (GSP):** [Srikant & Agrawal 1996]
  - Adds max/min gaps,
  - Taxonomies for items,
  - Efficiency improvements through hashing structures
  
- **PSP:** [Masseglia et al. 1998]  
 Organizes candidates in a prefix tree
  
- **Maximal Sequential Patterns using Sampling (MSPS):** Sampling  
 [Luo & Choung 2005]
- ...
- **See Mooney / Roddick for more details** [Mooney & Roddick 2013]

## SPaDE: Sequential Pattern Discovery using Equivalence Classes

[Zaki 2001]

- Uses a vertical data representation:

SID	Time	Items
1	10	a, b, d
1	15	b, d
1	20	c
2	15	a
2	20	b, c, d
3	10	b, d

→

**a**

SID	Time
1	10
2	15

**b**

SID	Time
1	10
1	15
2	20
3	10

**c**

SID	Time
1	20
2	20

**d**

SID	Time
1	10
1	15
2	20
3	10

(Original) Horizontal database layout

Vertical database layout

- ID-lists for longer candidates are constructed from shorter candidates
- Exploits **equivalence classes**:  
 $\langle b \rangle$  and  $\langle d \rangle$  are equivalent  $\rightarrow \langle b, x \rangle$  and  $\langle d, x \rangle$  have the same support
- Can traverse search space with depth-first or breadth-first search


## Extensions based on SPaDE

- **SPAM: Bitset representation** [Ayres et al. 2002]
- **LAPIN:** [Yang & et al. 2007]  
 Uses last position of items in sequence to reduce generated candidates
- **LAPIN-SPAM: combines both ideas** [Yang & Kitsuregawa 2005]
- **IBM:** [Savary & Zeitouni 2005]  
 Combines several datastructures (bitsets, indices, additional tables)

## PrefixSpan [Pei et al. 2001]

- Similar idea to Frequent Pattern Growth in FIM
- Determine frequent single items (e.g., **a**, **b**, **c**, **d**, **e**):
  - First mine all frequent sequences starting with *prefix* **<a...>**
  - Then mine all frequent sequences starting with *prefix* **<b...>**
  - ...
- Mining all frequent sequences starting with **<a...>** does not require complete dataset!
- Build projected databases:
  - Use only sequences containing **a**
  - For each sequence containing **a** only use the part “after” **a**

Given Sequence	Projection to <b>a</b>
<b>&lt; b, (c,d), a, (b d), e &gt;</b>	<b>&lt;a, (b,d), e&gt;</b>
<b>&lt;c, (a,d), b, (d,e)&gt;</b>	<b>&lt;(a,d), b, (d,e)&gt;</b>
<b>&lt;b, (de), c&gt;</b>	<b>[will be removed]</b>

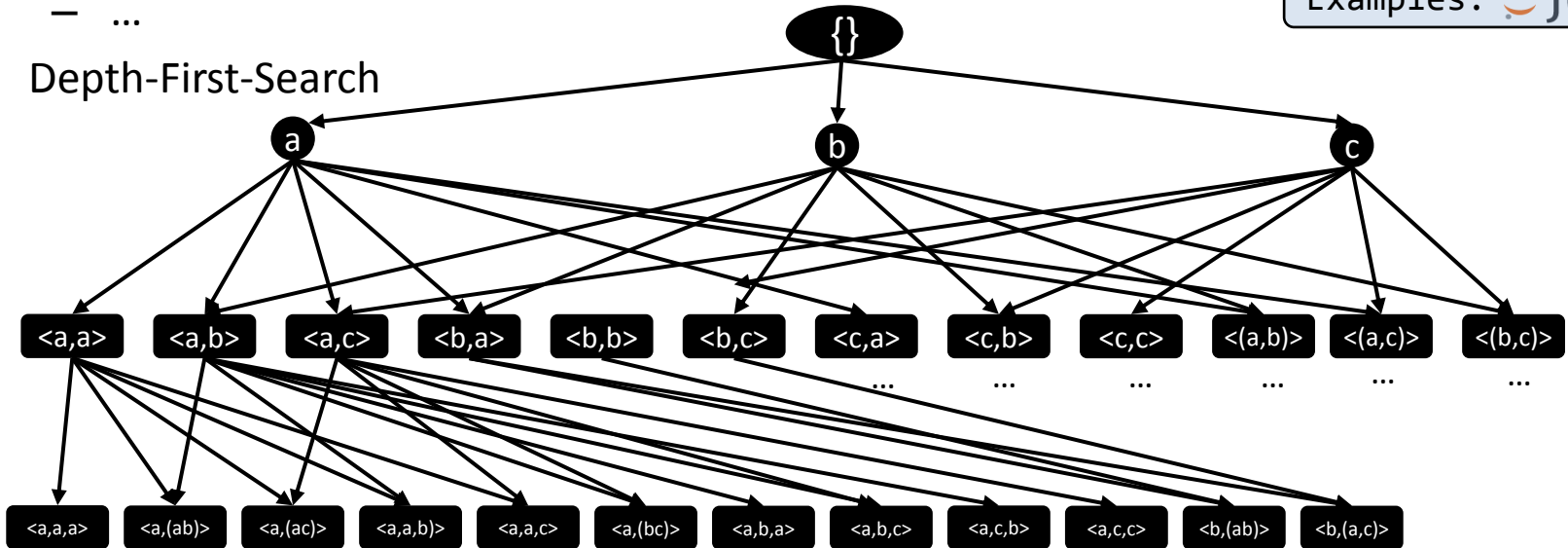
More Examples:  **Jupyter**

## PrefixSpan (continued)

- Given prefix  $a$  and projected database for  $a$ : mine recursively!
  - Mine frequent single items in projected database (e.g.,  $b, c, d$ )
  - Mine frequent sequences with prefix  $\langle a, b \rangle$
  - Mine frequent sequences with prefix  $\langle a, c \rangle$
  - ...
  - Mine frequent sequences with prefix  $\langle (a,b) \rangle$
  - Mine frequent sequences with prefix  $\langle (a,c) \rangle$
  - ...

Examples:  jupyter

- Depth-First-Search



## Advantages of PrefixSpan

- Advantages compared to Apriori:
  - ⊕ No explicit candidate generation, no checking of not occurring candidates
  - ⊕ Projected databases keep shrinking
- Disadvantage:
  - ⊖ Construction of projected database can be costly

## So... which algorithm should you use?

- All algorithm give the same result
- Runtime / memory usage varies
- Current studies are inconclusive
- Depends on dataset characteristics:
  - Dense data tends to favor SPaDE-like algorithms
  - Sparse data tends to favor PrefixSpan and variations
- Depends on implementations



## The Redundancy Problem

- The result set often contains many and many similar sequences
- Example: find frequent sequences with  $\text{minSupport} = 10$ 
  - Assume  $\langle a, (bc), d \rangle$  is frequent
  - Then the following sequence also MUST be frequent:
    - $\langle a \rangle, \langle b \rangle, \langle c \rangle,$
    - $\langle a, b \rangle, \langle a, c \rangle, \langle a, d \rangle, \langle b, d \rangle, \langle c, d \rangle, \langle (b,c) \rangle,$
    - $\langle a, (b,c) \rangle, \langle a, b, d \rangle, \langle a, c, d \rangle, \langle (b,c), d \rangle$
- Presenting all these as frequent subsequences carries little additional information!



## Closed and Maximal Patterns

- Idea: Do not use all patterns, but only...
  - ... frequent *closed* sequences:  
all super-sequences have a smaller support
  - ... frequent *maximal* sequences :  
All super-sequences are not frequent

• Example:


Dataset
<a, b, c, d, e, f>
<a, c, d>
<c, b, a>
<b, a, (de)>
<b, a, c, d, e>

$\text{sup} (<a,c>) = 3 \rightarrow$  frequent

$\text{sup} (<a,c,d>) = 3 \rightarrow$  frequent, closed

$\text{sup} (<a,c,d,e>) = 2 \rightarrow$  frequent, closed, max.

$\text{sup} (<a,c,d,e,f>) = 1 \rightarrow$  not frequent

Try this example:  jupyter

- Set of all frequent sequences can be derived from the maximal sequences
- Count of all frequent sequences can be derived from the closed sequences

# Mining Closed & Maximal Patterns

- In principle: can filter resulting frequent itemsets
- Specialized algorithms
  - Apply pruning during the search process
  - Much faster than mining all frequent sequences
- Some examples
  - Closed:
    - CloSpan: PrefixTree with additional pruning [Yan et al. 2003]
    - BIDE: Memory-efficient forward/backward checking [Wang&Han 2007]
    - ClaSP: Based on SPaDE [Gomariz et al. 2013]
  - Maximal:
    - AprioriAdjust: Based on Apriori [Lu & Li 2004]
    - VMSP: Based on vertical data structures [Fournier-Viger et al. 2014]
    - MaxSP: Inspired by PrefixSpan,  
maximal backward and forward extensions [Fournier-Viger et al. 2013]
    - MSPX: approximate algorithm using samples [Luo & Chung 2005]

# Beyond Frequency

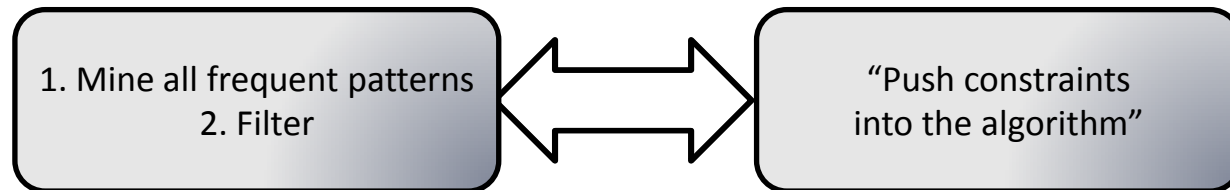
- Frequent sequence  $\neq$  interesting sequence
- Example for text sequences:  
Most frequent sequences in “Adventures from Tom Sawyer”<sup>1</sup>:

Sequence	Support (in %)
<and, and>	13%
<and, to>	9.8%
<to, and	9.1%
<of, and>	8.6%

- Two options:
  - Add constraints (filter)
  - Use interestingness measures

## Constraints

- **Item constraints:** e.g., high-utility items: Sum all items in the sequence > 1000\$
- **Length constraint:** Minimum/maximum number of events/transactions
- **Model-based constraints:** Sub-/supersequences of a given sequence
- **Gap constraints:** Maximum gap between events of a sequence
- **Time constraints:** Given timestamps, maximum time between events of a sequence
- **Closed or maximal** sequences only
- ...
- Computation:

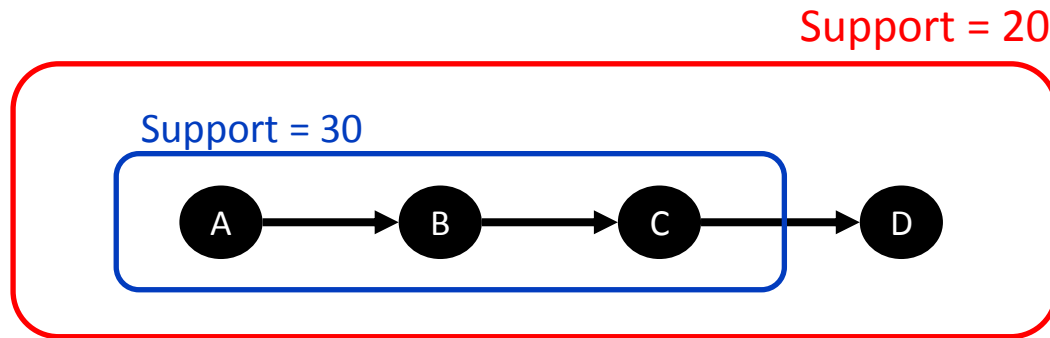


# Interestingness Measures and top-k Search

- Use *interestingness measures*
  - Function that assign a numeric value (score) to each sequence
  - Should reflect the “assumed interestingness” for users
  - Desired properties:  
conciseness, generality, reliability, diversity,  
novelty, surprisingness, applicability
  
- New goal: search for the  $k$  sequences that achieve the highest score
  
- Interestingness measure also implies a ranking of the result
  
- Simple mining approach:
  1. Compute all frequent patterns
  2. Compute the score of each pattern

## Confidence

- Typical measure for association rule mining
- Can easily be adapted for sequential pattern
- Split sequence into a rule (e.g., with the last event as rule head)
- Confidence = accuracy of this rule



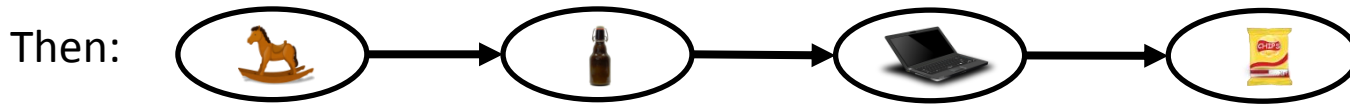
$$\text{Confidence } (\langle A, B, D \rangle \rightarrow F) = \frac{20}{30}$$

- Can be used as a constraint or as an interestingness measure

## Leverage [Petitjean et al. 2015]

- Compare support of a sequence with “expected support”:  

$$Score(S) = sup(S) - expectedSupport(S)$$
- Idea of expected support?



Is also more likely to be frequent than the average 4-sequence  
 It should be reported only, if its frequency exceeds expectation

- Formalization for 2-sequences:

$$expectedSupport(\langle a, b \rangle) = \frac{sup(\langle a, b \rangle) + sup(\langle b, a \rangle)}{2}$$

- Formalization for larger sequences generalizes this

## Other Interestingness Measures

- Information theoretic approaches: [Tatti & Vreeken 2012], [Lam et al. 2014]
  - Use minimum description length
  - Find sequence (sets) that best explain/compress the dataset
  
- Model-based approaches [Gwadera & Crestani 2010], [Lam et al. 2014]
  - Build a reference model (e.g., learn a markov chain model)
  - Determine which sequences are most unlikely given that model
  - (Compute statistical significance)
  
- Include time information
- ...



# Efficient top-k Sequential Pattern Mining [Petitjean et al. 2015]

- Example Algorithm SkOPUS:
- Depth First Search
- Pruning:
  - Interestingness measures like leverage/confidence are not directly monotone (unlike support)  
E.g.: score (  $\langle a, b, c \rangle$  ) can be higher than score (  $\langle a, b \rangle$  )
  - Use *upper bounds* (“optimistic estimates”)  $oe(S)$   
For each sequence  $S$  this is threshold,  
such that no super-sequence of  $S$  has a higher score
  - Has to be determined for each interestingness measure separately
  - Often easy to compute for a single interestingness measure

## Case Study Web Log Mining [Soares et al. 2006]

- Portuguese web portal for business executives:
- Data: 3,000 users; 70,000 session; 1.7M accesses
- Navigation patterns found on page level:
  - Too many
  - Not very useful
- On type level (“news”, “navigation”)
  - More interesting findings

# Mining Web logs to Improve Website Organization

[Srikant & Yang 2001]

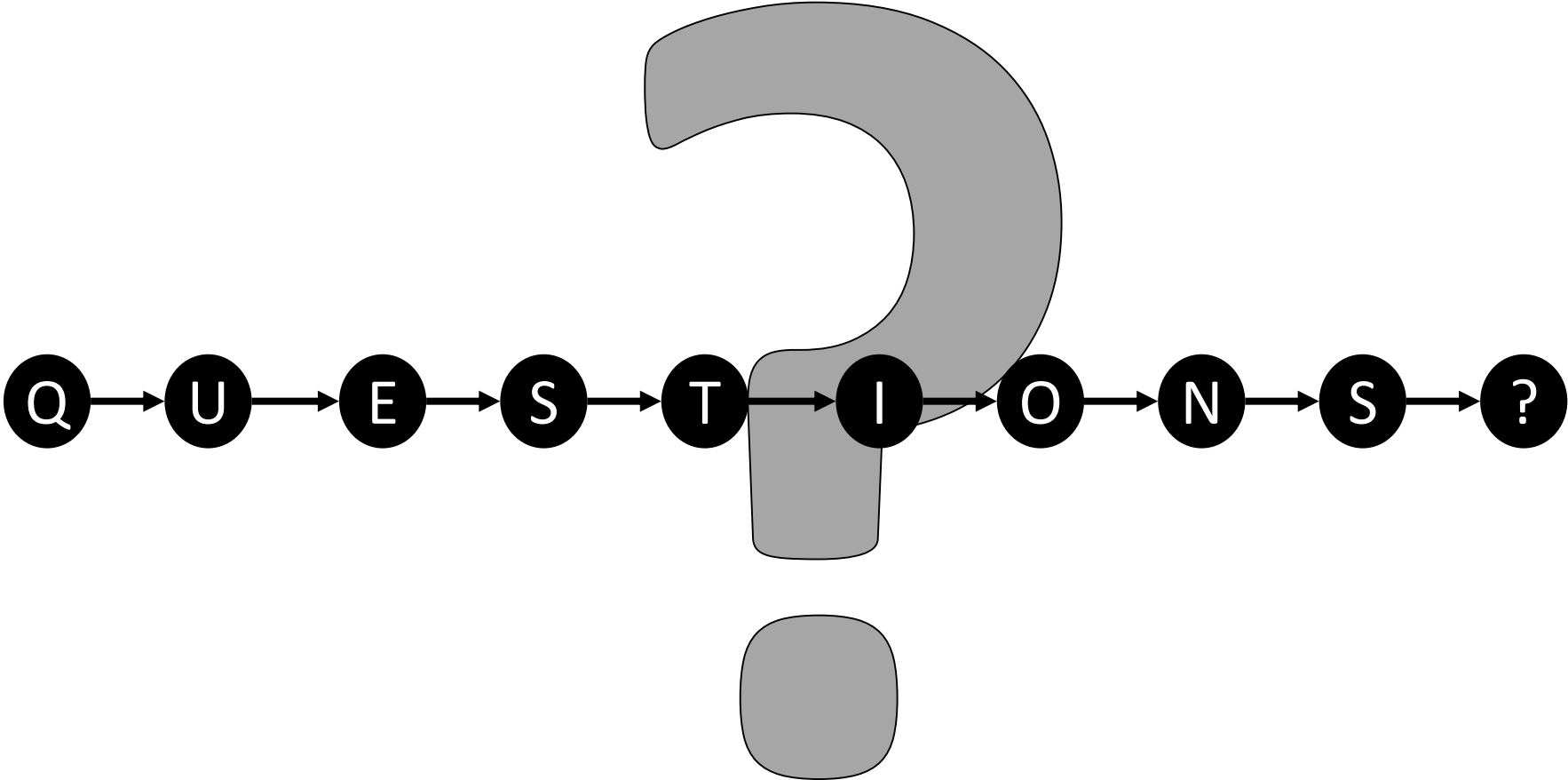
- Given link structure of a web page, visitor log
- Build sequences for each visitor
- Define target page
  
- Find frequent paths to the target page
- Identify links that could shorten user paths

# Available Software Libraries

- Java:
  - SPMF (most extensive library)  
<http://www.philippe-fournier-viger.com/spmf/>
  - Basic support in RapidMiner, KNIME
- R
  - arulesSequences package
  - TraMiner package
- Python
  - Multiple basic implementations
  - The implementations for this tutorial (mainly educational, not efficient)
- Spark: PrefixSpan available

## What we did not talk about...

- Episode mining [Mannila et al. 1997]
  - Given long sequences: find recurring patterns
  - Mining: candidate generation vs. pattern growth
- Discriminative sequential pattern
- Incremental mining / data streams
- Pattern in time-series



## References (1/2)

- Agrawal, R., & Srikant, R. (1995, March). Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on* (pp. 3-14). IEEE.
- Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002, July). Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 429-435). ACM.
- Fournier-Viger, P., Wu, C. W., Gomariz, A., & Tseng, V. S. (2014). VMSP: Efficient vertical mining of maximal sequential patterns. In *Advances in Artificial Intelligence* (pp. 83-94). Springer International Publishing.
- Fournier-Viger, P., Wu, C. W., & Tseng, V. S. (2013). Mining maximal sequential patterns without candidate maintenance. In *Advanced Data Mining and Applications* (pp. 169-180). Springer Berlin Heidelberg.
- Garofalakis, M. N., Rastogi, R., & Shim, K. (1999, September). SPIRIT: Sequential pattern mining with regular expression constraints. In *VLDB* (Vol. 99, pp. 7-10).
- Gomariz, A., Campos, M., Marin, R., & Goethals, B. (2013). Clasp: An efficient algorithm for mining frequent closed sequences. In *Advances in knowledge discovery and data mining* (pp. 50-61). Springer Berlin Heidelberg.
- Gwadera, R., & Crestani, F. (2010). Ranking sequential patterns with respect to significance. In *Advances in Knowledge Discovery and Data Mining* (pp. 286-299). Springer Berlin Heidelberg.
- Lam, H. T., Mörchen, F., Fradkin, D., & Calders, T. (2014). Mining compressing sequential patterns. *Statistical Analysis and Data Mining*, 7(1), 34-52.
- Luo, C., & Chung, S. M. (2005, April). Efficient Mining of Maximal Sequential Patterns Using Multiple Samples. In *SDM* (pp. 415-426).
- Mannila, H., Toivonen, H., & Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3), 259-289.
- Maseglier, F., Cathala, F., & Poncelet, P. (1998). The PSP approach for mining sequential patterns. In *Principles of Data Mining and Knowledge Discovery* (pp. 176-184). Springer Berlin Heidelberg.
- Mooney, C. H., & Roddick, J. F. (2013). Sequential pattern mining--approaches and algorithms. *ACM Computing Surveys (CSUR)*, 45(2), 19.

## References (2/2)

- Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. C. (2001, April). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *icccn* (p. 0215). IEEE.
- Soares, C., de Graaf, E., Kok, J. N., & Kusters, W. A. (2006). Sequence mining on web access logs: A case study. In *Belgian/Netherlands Artificial Intelligence Conference, Namur*.
- Srikant, R., & Agrawal, R. (1996). *Mining sequential patterns: Generalizations and performance improvements* (pp. 1-17). Springer Berlin Heidelberg.
- Srikant, R., & Yang, Y. (2001, April). Mining web logs to improve website organization. In *Proceedings of the 10th international conference on World Wide Web* (pp. 430-437). ACM.
- Tatti, N., & Vreeken, J. (2012, August). The long and the short of it: summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 462-470). ACM.
- Wang, J., Han, J., & Li, C. (2007). Frequent closed sequence mining without candidate maintenance. *Knowledge and Data Engineering, IEEE Transactions on*, 19(8), 1042-1056.
- Yan, X., Han, J., & Afshar, R. (2003, May). CloSpan: Mining closed sequential patterns in large datasets. In *In SDM* (pp. 166-177).
- Yang, Z., Wang, Y., & Kitsuregawa, M. (2007). LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases. In *Advances in Databases: Concepts, Systems and Applications* (pp. 1020-1023). Springer Berlin Heidelberg.
- Yang, Z., & Kitsuregawa, M. (2005, April). LAPIN-SPAM: An improved algorithm for mining sequential pattern. In *Data Engineering Workshops, 2005. 21st International Conference on* (pp. 1222-1222). IEEE.
- Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2), 31-60.