DATABASE SYSTEMS GROUP

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

LMU

Knowledge Discovery in Databases
SS 2016

Chapter 3b: Sequential Pattern Mining

Lecture: Prof. Dr. Thomas Seidl

Tutorials: Julian Busch, Evgeniy Faerman,
Florian Richter, Klaus Schmid

Overview of chapter 3b – Sequential Pattern Mining

1. Sequential Pattern Mining Basics
2. Sequential Pattern Mining Algorithms
3. Interval-based Sequential Pattern Mining
4. Process Mining

Sequential pattern mining

- Task 1: find all subsets of items that occur with a specific sequence in many transactions.
  - E.g.: 97% of transactions contain the *sequence* {jogging → high ECG → sweating}
- Task 2: find all rules that correlate the order of one set of items after that of another set of items in the transaction database.
  - E.g.: 72% of users who perform a web search *then* make a long eye gaze over the ads *follow that* by a successful add-click
- The order of the items matters, thus all possible permutations of items must be considered when checking possible frequent sequences, not only the combinations of items
- Applications: data with temporal order (streams), e.g.: bioinformatics, Web mining, text mining, sensor data mining, process mining etc.

Sequential Pattern Mining vs. Frequent Itemset Mining

- Both can be applied on similar dataset
  - Each customer has a customer id and aligned with transactions.
  - Each transaction has a transaction id and belongs to one customer.
  - Based on the transaction id, each customer also aligned to a transaction sequence.

Cid	Tid	Item
1	1	{butter}
	2	{milk}
	3	{sugar}
2	4	{butter, sugar}
	5	{milk, sugar}
	6	{butter, milk, sugar}
	7	{eggs}
3	8	{sugar}
	9	{butter, milk}
	10	{eggs}
	11	{milk}

Cid	Item
1	{butter} ,{milk}, {sugar}
2	{butter, sugar}, {milk, sugar}, {butter, milk, sugar}, {eggs}
3	{sugar}, {butter, milk}, {eggs}, {milk}

Frequent itemset mining

- No temporal importance in the order of items happening together

items	frequency
{butter}	4
{milk}	5
{butter, milk}	2
…	

Sequential pattern mining

- The order of items matters

sequences	frequency
{butter}	4
{butter, milk}	2
{butter},{milk}	4
{milk},{butter}	1
{butter},{butter,milk}	1
…	

Sequential pattern mining: Basic Notions (1/2)

- *Alphabet* Σ is set symbols or characters (denoting items)
- *Sequence* $\boldsymbol{S} = s_1 s_2 \dots s_k$ is an ordered list of a *length* $|\boldsymbol{S}| = k$ items where $s_i \in \Sigma$ is an item at position i also denoted as $\boldsymbol{S}[i]$
- *K-sequence* is a sequence of length k
- *Consecutive subsequence* $\boldsymbol{R} = r_1 r_2 \dots r_m$ of $\boldsymbol{S} = s_1 s_2 \dots s_n$ is also a sequence in Σ such that $r_1 r_2 \dots r_m = s_j s_{j+1} \dots s_{j+m-1}$ with $1 \leq j \leq n - m + 1$. we say $\boldsymbol{S}$ contains $\boldsymbol{R}$ and denote this by $\boldsymbol{R} \subseteq \boldsymbol{S}$
- In the more general: *subsequence* $\boldsymbol{R}$ of $\boldsymbol{S}$ we allow for gaps between the items of $\boldsymbol{R}$, i.e. the items of the subsequence $\boldsymbol{R} \subseteq \boldsymbol{S}$ must have the same order of the ones in $\boldsymbol{S}$ but there can be some other items between them
- A *prefix* of a sequence $\boldsymbol{S}$ is any consecutive subsequence of the form $\boldsymbol{S}[1:i] = s_1 s_2 \dots s_i$ with $0 \leq i \leq n$, $\boldsymbol{S}[1:0]$ is the empty prefix
- A *suffix* of a sequence $\boldsymbol{S}$ is any consecutive subsequence of the form $\boldsymbol{S}[i:n] = s_i s_{i+1} \dots s_n$ with $1 \leq i \leq n + 1$, $\boldsymbol{S}[n+1:n]$ is the empty suffix

Sequential pattern mining: Basic Notions (2/2)

- Given a database $D = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_N}\}$ of N sequences, the *support* of a sequence $\boldsymbol{R}$ in D is defined as the number of sequences in D that contain:
$$sup(\boldsymbol{R}) = |\{\boldsymbol{S_i} \in D | \boldsymbol{R} \subseteq \boldsymbol{S_i}\}|$$
- The *relative support* of $\boldsymbol{R}$ is the fraction of sequences that contain $\boldsymbol{R}$:
$$rsup(\boldsymbol{R}) = \sup(\boldsymbol{R}) / N$$
- Given a user-defined $minSup$ threshold, a sequence $\boldsymbol{R}$ is called *frequent* (or *sequential*) in database D iff : $sup(\boldsymbol{R}) \geq minSup$
- A frequent sequence is *maximal* if it is not a subsequence of any other frequent sequence
- A frequent sequence is *closed* if it is not a subsequence of any other frequent sequence with the same support

Sequential Pattern Mining Algorithm

- Breadth-first search based
  - GSP (*Generalized Sequential Pattern*) algorithm[1]
  - SPADE[2]
  - …
- Deepth-first search based
  - PrefixSpan[3]
  - SPAM[4]
  - …

[1]Sirkant & Aggarwal: *Mining sequential patterns: Generalizations and performance improvements.* EDBT 1996
[2]Zaki M J. *SPADE: An efficient algorithm for mining frequent sequences[J].* Machine learning, 2001, 42(1-2): 31-60.
[3]Pei at. al.: *Mining sequential patterns by pattern-growth: PrefixSpan approach.* TKDE 2004
[4]Ayres, Jay, et al: *Sequential pattern mining using a bitmap representation.* SIGKDD 2002.

GSP algorithm

- GSP (*Generalized Sequential Pattern*) algorithm[1] performs a level-wise (breadth-first) search within the prefix tree
- Given the set of frequent sequences at level k, generate *all* possible sequence extensions or candidates at level $k+1$
- Uses the Apriori principle (anti-monotonicity)
- Next compute the support of each candidate and prune the ones with $sup < minSup$
- Stop the search when no more frequent extensions are possible

[1]Sirkant & Aggarwal: *Mining sequential patterns: Generalizations and performance improvements.* EDBT 1996

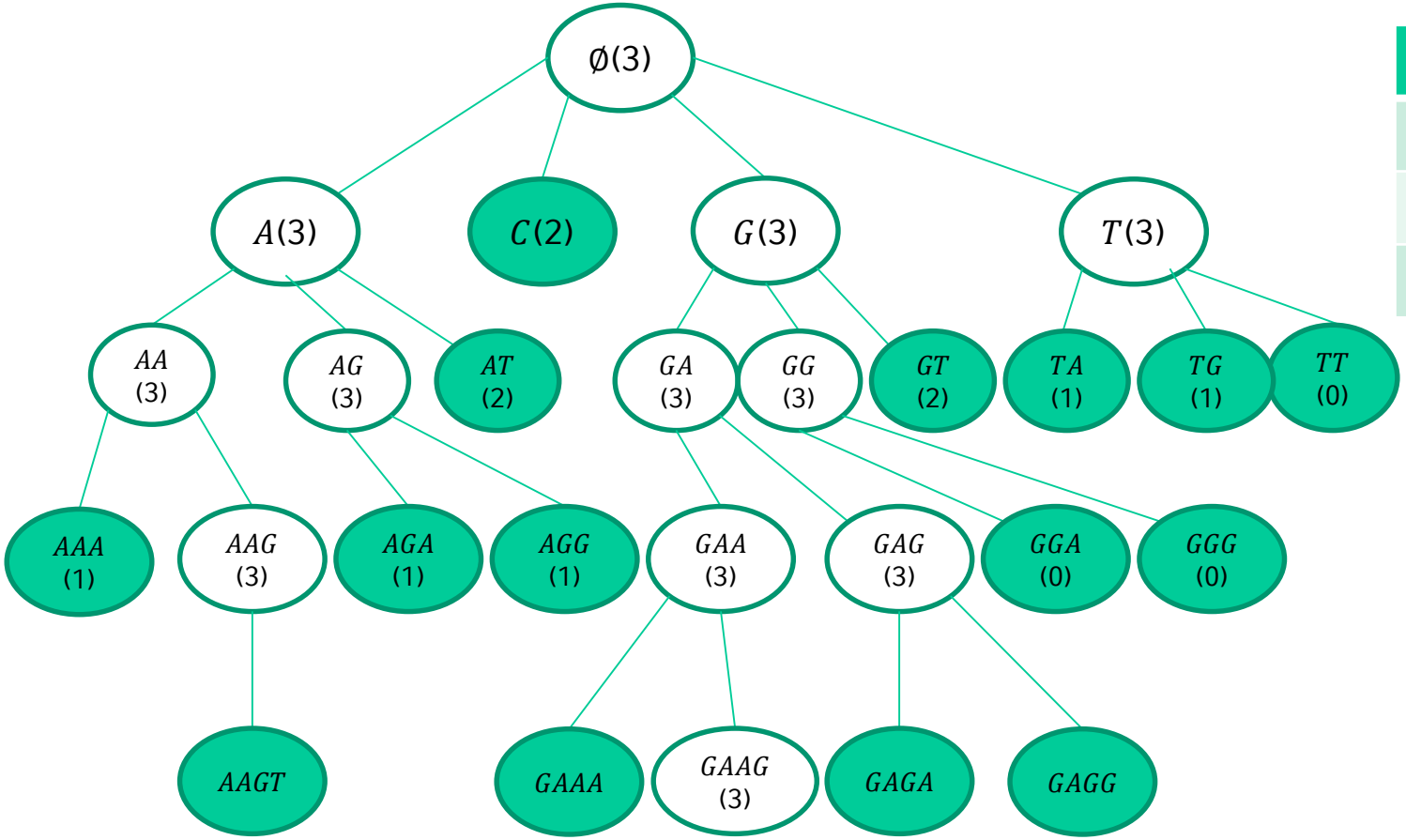
Projection-based sequence mining: Representation

- The sequence search space can be organized in a prefix search tree
- The root (Level 0) contains the empty sequence with each item $x \in \Sigma$ as one of its children
- A node labeled with sequence: $\boldsymbol{S} = s_1 s_2 \ldots s_k$ at Level k has children of the form $\boldsymbol{S}' = s_1 s_2 \ldots s_k s_{k+1}$ at Level $k+1$
  - Thus $\boldsymbol{S}$ is a prefix of $\boldsymbol{S}'$ (or $\boldsymbol{S}'$ is an extension of $\boldsymbol{S}$)
- Example: let $\Sigma = \{A, C, G, T\}$ and let D consist of the following three sequences:

Id	Sequence
$\boldsymbol{S_1}$	$CAGAAGT$
$\boldsymbol{S_2}$	$TGACAG$
$\boldsymbol{S_3}$	$GAAGT$

- The prefix search tree is:

Projection-based sequence mining: Representation



Id	Sequence
S_1	$CAGAAGT$
S_2	$TGACAG$
S_3	$GAAGT$

- Sequence (sup)
- Shaded sequences are infrequent ($minSup = 3$)
- Sequences without sup can be pruned based on infrequent subsequences

Projection-based sequence mining

- For a database D and an item $s \in \Sigma$, the *projected database* w.r.t. s is denoted D_s and is found as follows:
- For each sequence $\boldsymbol{S_i} \in D$ do
  - Find the first occurrence of s in $\boldsymbol{S_i}$, say at position p
  - $\text{suff}_{S_i,s} \leftarrow \text{suffix}(\boldsymbol{S_i})$ starting at position $p+1$
  - Remove infrequent items from $\text{suff}_{S_i,s}$
  - $D_s = D_s \cup \text{suff}_{S_i,s}$

- Example for the following D:
  - $D_G = \{AAGT, AAG, AAGT\}$

Id	Sequence
$\boldsymbol{S_1}$	$CAGAAGT$
$\boldsymbol{S_2}$	$TGACAG$
$\boldsymbol{S_3}$	$GAAGT$

Projection-based sequence mining

- The *PrefixSpan* algorithm computes the support for only the individual items in the projected databased D_s
- Then performs recursive projections on the frequent items in a depth-first manner
- Initialization: $D_R \leftarrow D, \boldsymbol{R} \leftarrow \emptyset, \mathcal{F} \leftarrow \emptyset$

$PrefixSpan(D_R, \boldsymbol{R}, minSup, \mathcal{F})$
For each $s \in \Sigma$ such that $\sup(s, D_R) \geq minSup$ do

- $\boldsymbol{R_s} = \boldsymbol{R} + s$ //append s to the end of $\boldsymbol{R}$
- $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\boldsymbol{R_s}, \sup(s, D_R))\}$ //calculate the support of s for each $\boldsymbol{R_s}$within D_R
- $D_s \leftarrow \emptyset$ //create projected data for s
- For each $\boldsymbol{S_i} \in D_R$ do
  - $\boldsymbol{S_i'} \leftarrow$ projection of $\boldsymbol{S_i}$ w.r.t. item s
  - Remove an infrequent symbols from $\boldsymbol{S_i'}$
  - If $\boldsymbol{S_i'} \neq \emptyset \; then \; D_s = D_s \cup \boldsymbol{S_i'}$
- If $D_s \neq \emptyset$ then $PrefixSpan(D_s, \boldsymbol{R_s}, minSup, \mathcal{F})$

PrefixSpan: Example

Id	Sequence
S_1	$CAGAAGT$
S_2	$TGACAG$
S_3	$GAAGT$

$D_\emptyset$	
S_1	$CAGAAGT$
S_2	$TGACAG$
S_3	$GAAGT$
$A(3), \sout{C(2)}, G(3), T(3)$	

D_A	
S_1	$GAAGT$
S_2	AG
S_3	AGT
$A(3), G(3), \sout{T(2)}$	

D_T	
S_2	$GAAG$
$\sout{A(1)}, \sout{G(1)}$	

D_G	
S_1	$AAGT$
S_2	AAG
S_3	$AAGT$
$A(3), G(3), \sout{T(2)}$	

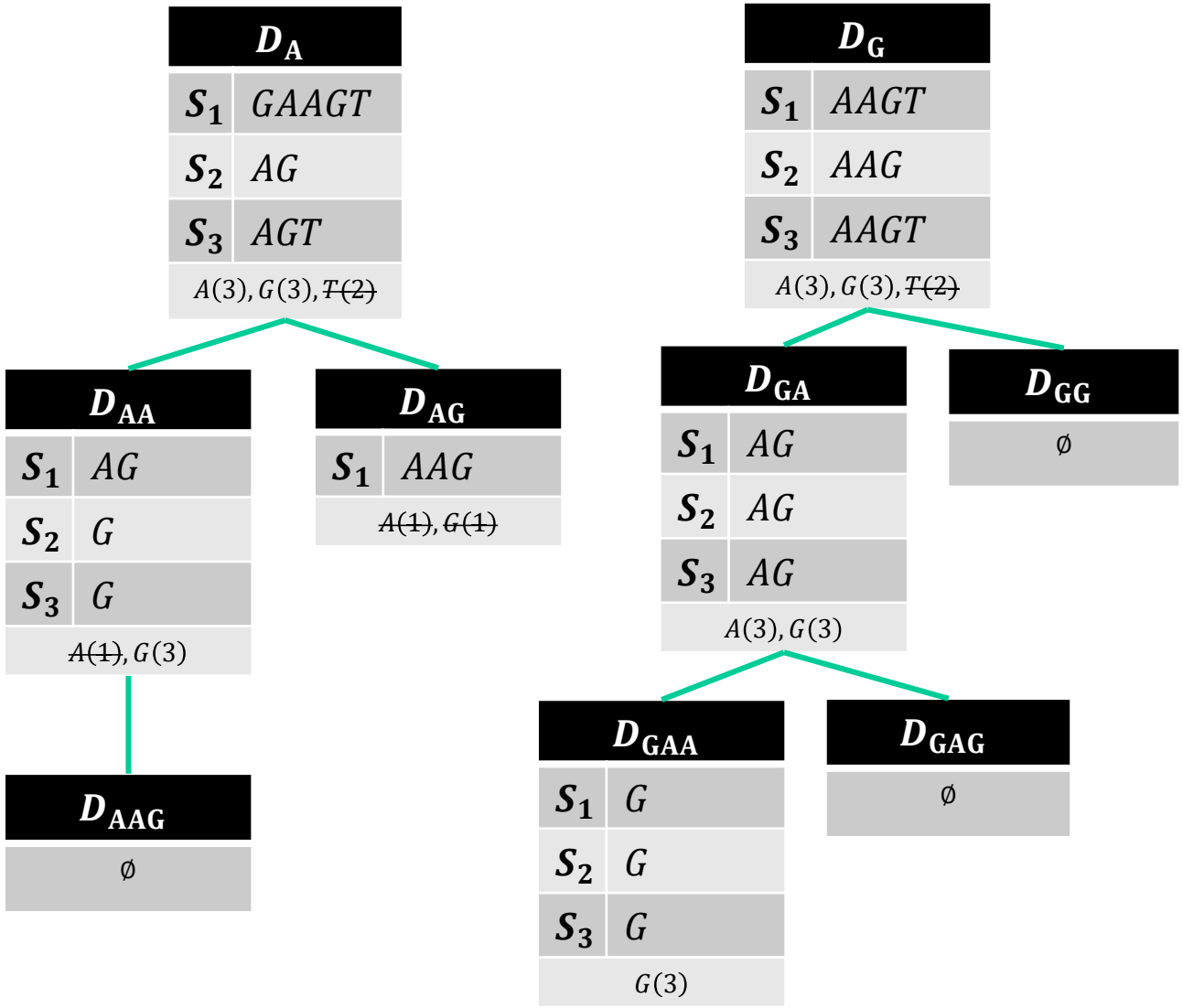
D_{AA}	
S_1	AG
S_2	G
S_3	G
$\sout{A(1)}, G(3)$	

D_{AG}	
S_1	AAG
$\sout{A(1)}, \sout{G(1)}$	

D_{GA}	
S_1	AG
S_2	AG
S_3	AG
$A(3), G(3)$	

D_{GG}
$\emptyset$

PrefixSpan: Example (cont'd)



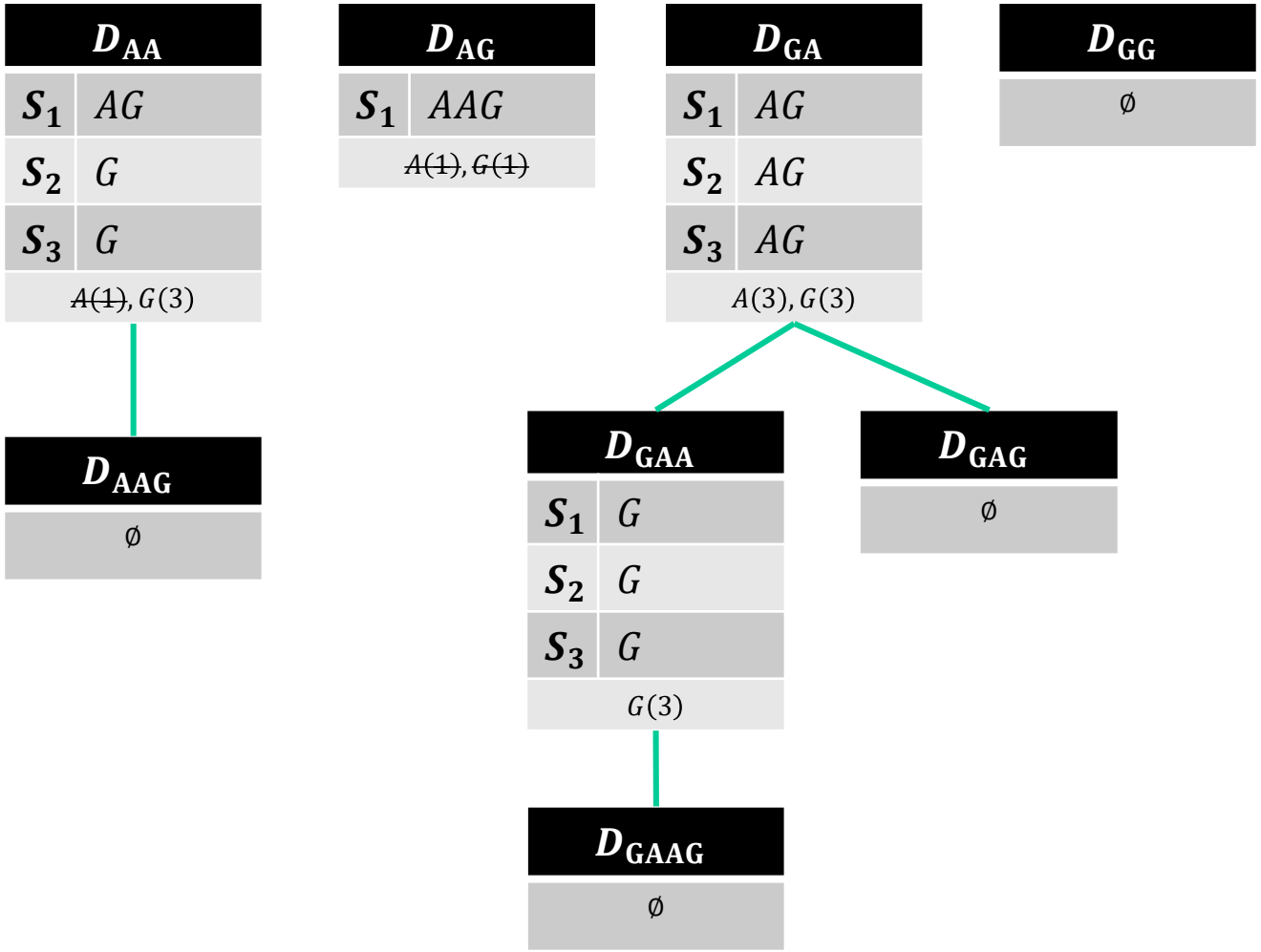
Id	Sequence
S_1	$CAGAAGT$
S_2	$TGACAG$
S_3	$GAAGT$

D_T

S_2	$GAAG$
$\sout{A(1)}, \sout{G(1)}$	

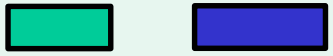
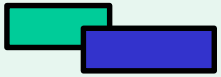
PrefixSpan: Example (cont'd)

Id	Sequence
S_1	$CAGAAGT$
S_2	$TGACAG$
S_3	$GAAGT$



Interval-based Sequential Pattern Mining

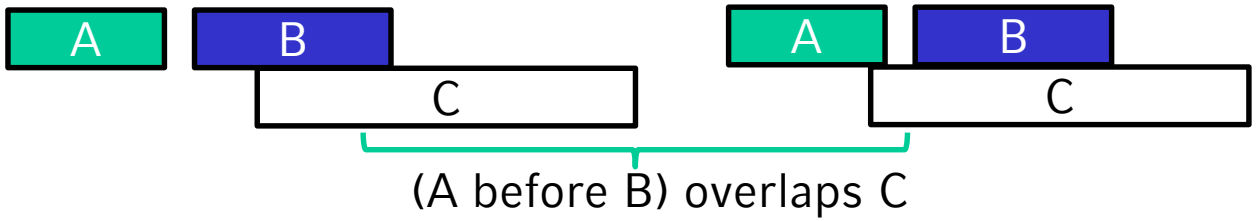
- Deals with the more common interval-based items s (or events).
- Each event has a starting and an ending time point, where $t_{s_{start}} < t_{s_{end}}$.
- Application: Health data analysis, Stock market data analysis, etc.
- Predefined relationships between items are more complex.
  - Point-based relationships: before, after, same time.
  - Interval-based relationships: Allen's relations [1], End point representation [2], etc.

Before	Overlaps	Contains	Starts
After	Overlapped-by	During	Started-by
			
Finished-by	**Meets**	**Equal**	
Finishes	Met-by	Equal	
			

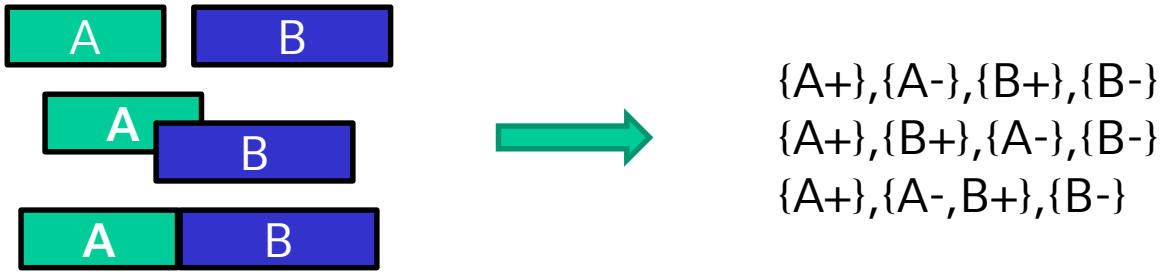
[1] Allen: *Maintaining knowledge about temporal intervals.* In Communications of the ACM 1983
[2] Wu, Shin-Yi, and Yen-Liang Chen: *Mining nonambiguous temporal patterns for interval-based events*. TKDE 2007

Interval-based Sequential Pattern Mining

- *Allen's relationships* is ambiguous when describing interval patterns with more than two events.



- *TPrefixSpan* algorithm [1] converts interval-based sequences into point-based sequences:



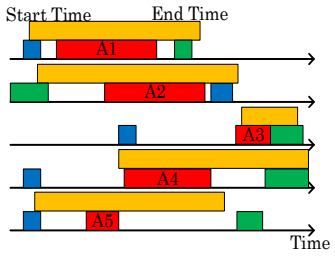
- Similar prefix projection mining approach as *PrefixSpan* algorithm.
- Validation checking is necessary in each expanding iteration to make sure that the appended time point can form an interval with a time point in the prefix.

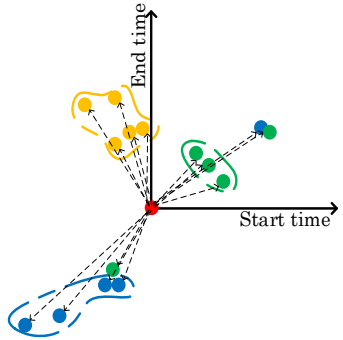
[1] Wu, Shin-Yi, and Yen-Liang Chen: *Mining nonambiguous temporal patterns for interval-based events*. TKDE 2007

An Open Issues: Mining without Predefined Patterns

- Predefined patterns quantize the relationship of items into limited number of categories:

  - Timing information ignored
  - Accuracy of support affected by noise
- Idea: learn pattern from data set by clustering
  - QTempIntMiner [1]
  - Event Space Miner [2]
  - PIVOTMiner [3]

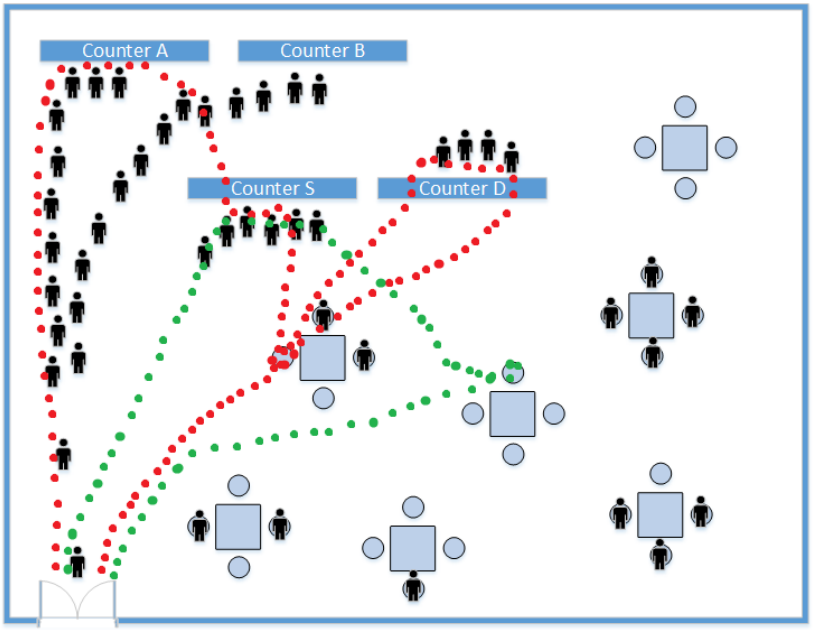


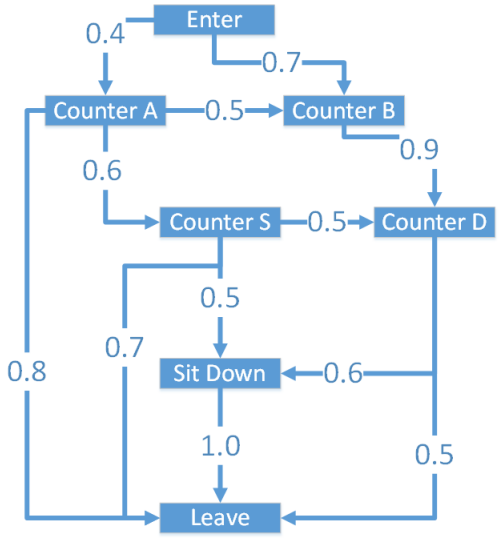


[1] Guyet, T., & Quiniou, R.: *Mining temporal patterns with quantitative intervals*. ICDMW 2008
[2] Ruan, G., Zhang, H., & Plale, B.: *Parallel and quantitative sequential pattern mining for large-scale interval-based temporal data*. IEEE Big Data 2014
[3] Hassani M., Lu Y. & Seidl T.: A Geometric Approach for Mining Sequential Patterns in Interval-Based Data Streams. FUZZ-IEEE 2016

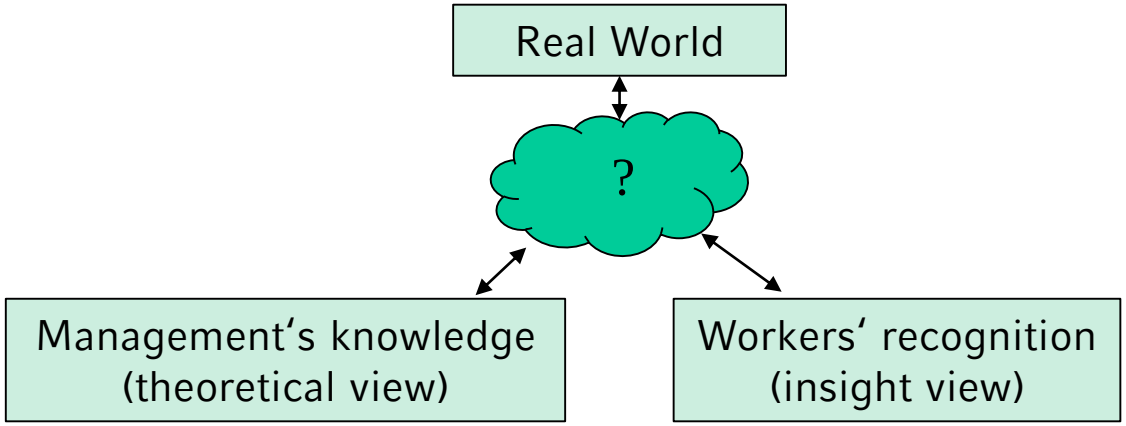
Process Mining

- *Processes* consist of (business) actions (sign document, customer call, financial transaction, delivery of goods,…)
- A *process instance* (case) is a certain sequence of actions in a process
  e.g. customer call → registration → sending package → invoice
- Processes can be modelled as graphs
- Process Mining creates and enriches models by observing real-world workflows





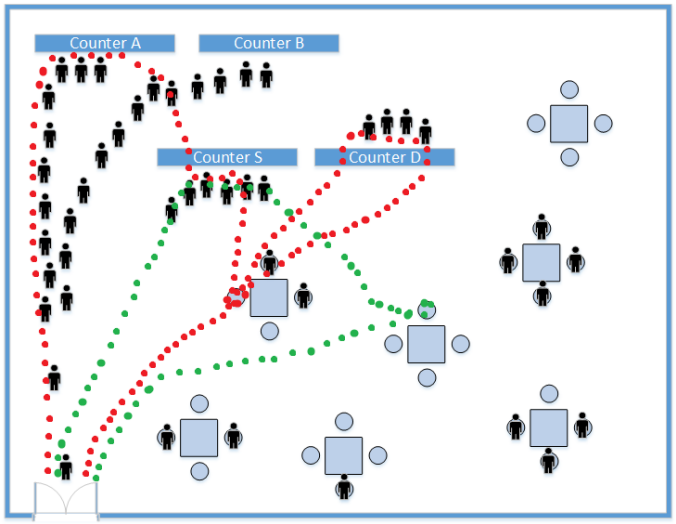
Process Mining



- Experience often bends the theoretical assumptions
- Many views, but what is the most useful/appropriate one?
- Manager: Good overview, but too far from the details
- Workers: Great detail knowledge with recent information, but missing links to other departments
- Real world information: Not naturally given, hard to acquire

Process Mining – First Task: Discovery

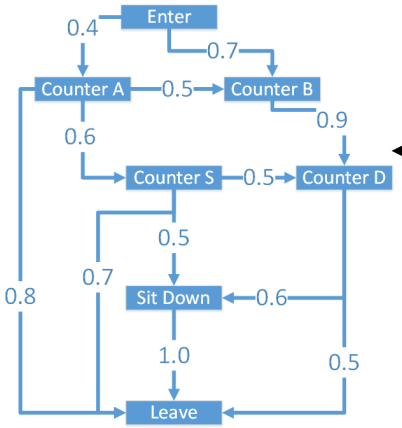
- Mine multiple sequences of actions to derive a workflow pattern
- Raw material: *event logs*, e.g. produced by ERP systems (enterprise resource planning)
- Each *event* belongs to a certain *case*
- Events are labeled with an *activity* and ordered using a *timestamp* at least
- Aims:
  - Improve understanding of "business" workflow
  - Model a simple but precise representation
  - Reveal organizational and social insights



Process Log

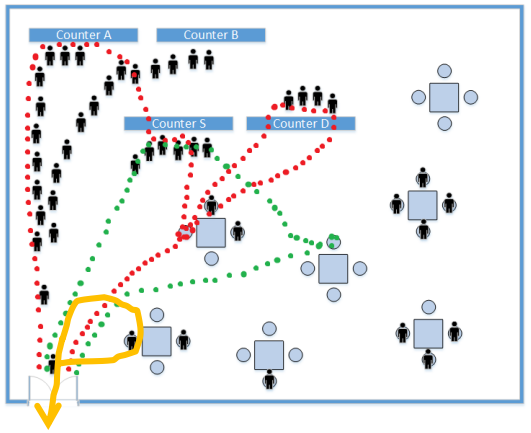
Case	Activity	Time
John	Counter A	11:13
Mary	Counter B	11:15
Mary	Receives meal	11:21
Bob	Enters facility	11:22
John	Counter S	11:22
Bob	Leaves facility	11:24
Mary	Sits down	11:25
John	Receives meal	11.28

Discovery Algorithm

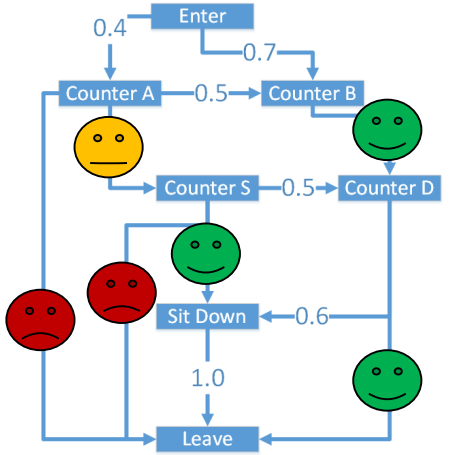


Process Mining – Other Tasks

- Task 2: Conformance Checking
  - Use previously mined model to judge the validity of a new case
  - Classification task is based on different types of models: procedural models, organizational models, business rules, laws,…
  - Aims: Model reasoning, auditing, security (fraud detection)



- Task 3: Enhancement
  - Business processes change over time, so adapt to new circumstances
  - Evolve models with new data, find deviations
  - Extend/enrich models, using a wider spectrum of data attributes
  - Aims: highlight bottlenecks, combine different models, analyze performance



Basis Discovery: α-Algorithm [1]

- Scan the log for all activities A
- For each pair of activities a and b, we define the relations
  - $a > b$ iff for some case a is immediately followed by b (direct succession)
  - $a \rightarrow b$ iff $a > b$ and not $b > a$ (causality)
  - $a||b$ iff $a > b$ and $b > a$ (parallelism)
  - $a\#b$ iff not $a > b$ and not $b > a$
- All activities, having only # or → in their row are starting activities. They are collected in T_{in}.
- Analogously, # or ← determine T_{out}.

Example $L = \{abcd, acbd, aed\}$

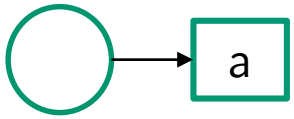
	a	b	c	d	e
a	#	→	→	#	→
b	←	#	||	→	#
c	←	||	#	→	#
d	#	←	←	#	←
e	←	#	#	→	#

$T_{in} = \{a\}, T_{out} = \{d\}$

[1] van der Aalst, W M P and Weijters, A J M M and Maruster, L (2003). "Workflow Mining: Discovering process models from event logs", *IEEE Transactions on Knowledge and Data Engineering*, vol 16

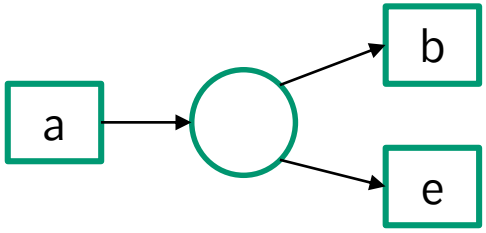
Basis Discovery: α-Algorithm [1]

- Prepare a Petri net. The set of transitions is equal to A, so each activity represents a transition
- A starting place is created and connected to each node in T_{in}
- Also, a final place is created and each node in T_{out} is connected to it



- Determine all pairs of sets A and B, such that
  - $\forall a_1, a_2 \in A: a_1 \# a_2$
  - $\forall b_1, b_2 \in B: b_1 \# b_2$
  - $\forall a_1 \in A, \forall b_1 \in B: a_1 \rightarrow b_1$
- A place is added in between A and B and connected accordingly

$A = \{a\}, B = \{b, e\}$



[1] van der Aalst, W M P and Weijters, A J M M and Maruster, L (2003). "Workflow Mining: Discovering process models from event logs", *IEEE Transactions on Knowledge and Data Engineering*, vol 16

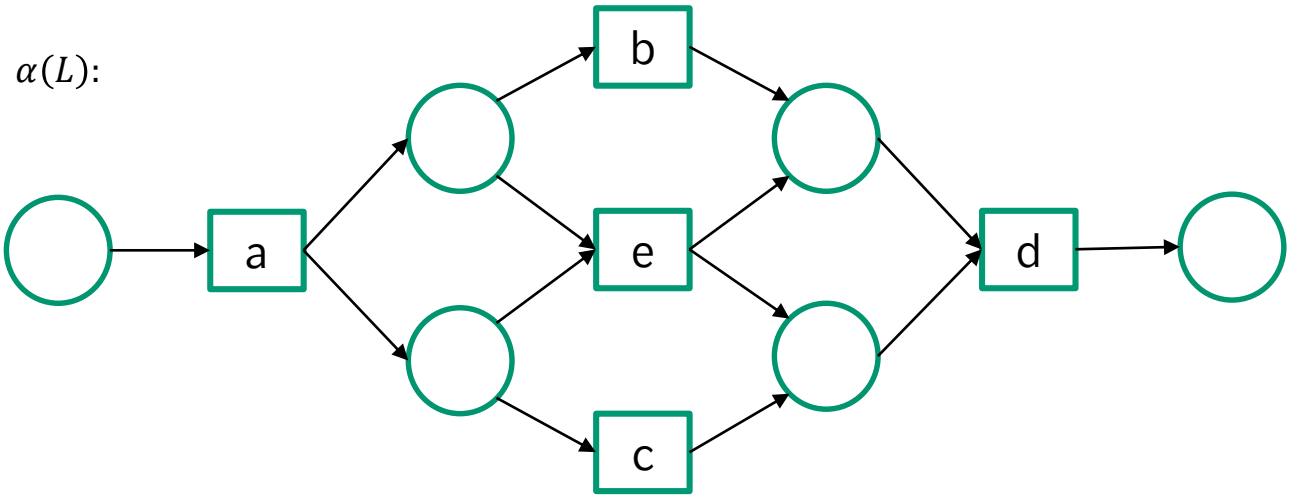
Basis Discovery: α-Algorithm [1]

- In this example, valid set pairs are:
$$(\{a\},\{be\}),(\{a\},\{ce\}),(\{be\},\{d\}),(\{ce\},\{d\})$$

- This yields the following Petri net

$L = \{abcd, acbd, aed\}$

$\alpha(L)$:



[1] van der Aalst, W M P and Weijters, A J M M and Maruster, L (2003). "Workflow Mining: Discovering process models from event logs", *IEEE Transactions on Knowledge and Data Engineering*, vol 16