Lecture 6 – Frequent Pattern mining in Sequences and Graphs

Data Mining, Spring 2016

Anders Hartzen, andershh@itu.dk

IT UNIVERSITY OF COPENHAGEN

Some slides adapted from Hector Martinez of University of Malta and Julian Togelius of New York University (both former ITU).

Overview of today's lecture

- Intro to sequences
- Sequence Pattern Mining
  - Apriori for Sequence mining
- Intro to graphs
- Apriori for Graph mining

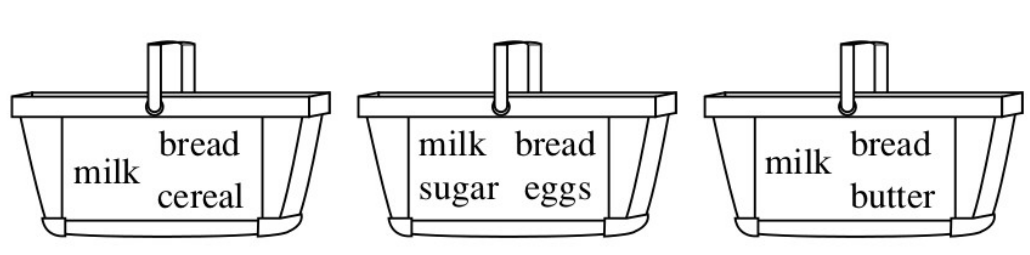
Intro to Sequences

Sequence

- Broadly speaking: an ordered list of events
- Denoted as

  $S = \langle e_1, e_2, e_3, \ldots, e_n \rangle$

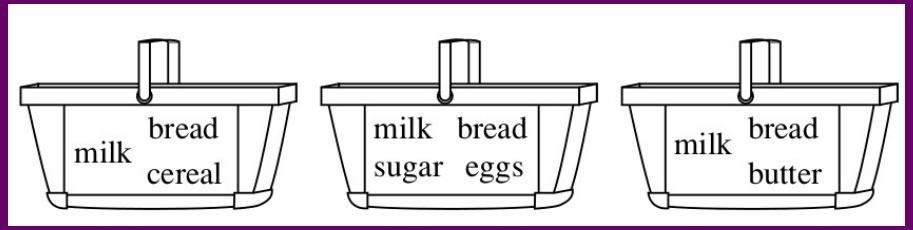
  - Event e_j also known as an element of S
- In the context of customer purchase data each event is an itemset of items bought



Sequence

- Itemset
  - Nonempty set of items
  - Denoted as

    $(x_1, x_2, x_3, \ldots, x_q)$
  - Items in an itemset can be considered as events that occur simultaneously
  - Parentheses denote the items within the same sequence element
  - An item can occur at most once in an itemset (event), but can occur many times in different itemsets (events)
- Sequence length = number of item instances in sequence
  - Sequence with length l known as l-sequence



Sequence

- Example
  - Sequence:

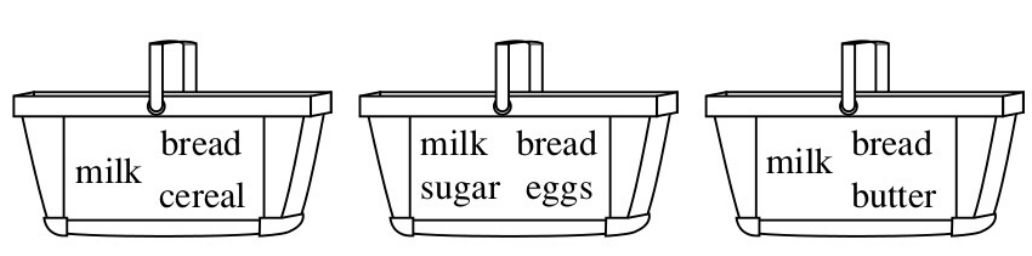
    ⟨(bread, milk, cereal) (milk, bread, sugar, eggs) (bread, milk, butter)⟩
  - Elements:

    (bread, milk, cereal)

    (milk, bread, sugar, eggs)

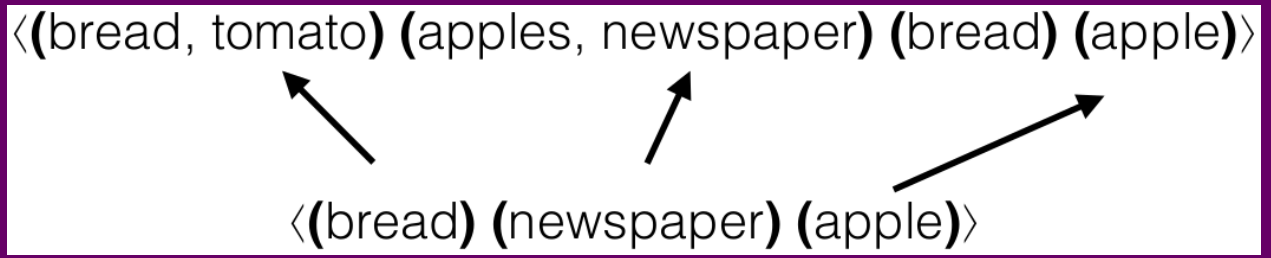
    (bread, milk, butter)
  - Items:

    bread, milk, cereal, sugar, eggs, butter,
  - Length: 10



Subsequences

- An itemset E_1 is a superset of another itemset E_2 if all the items contained in E_2 are also contained in E_1 (E_2 is a subset of E_1)
- A sequence S_1 is a subsequence of another sequence S_2 if
  - every itemset in S_1 has a superset in S_2
  - the order among itemsets in S_1 is the same as the order among their supersets in S_2
- We say that a sequence is a k-[sub]sequence if it contains k items



Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ - **No** (no super-itemset)

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ - **No** (no super-itemset)
4. ⟨(tomato) (apple)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ - **No** (no super-itemset)
4. ⟨(tomato) (apple)⟩ - **Yes**

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ - **No** (no super-itemset)
4. ⟨(tomato) (apple)⟩ - **Yes**
5. ⟨(apple) (apple)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ - **No** (no super-itemset)
4. ⟨(tomato) (apple)⟩ - **Yes**
5. ⟨(apple) (apple)⟩ - **Yes**

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

6. ⟨(apple, apple)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

6. ⟨(apple, apple)⟩ - **No** (invalid, repetition within an itemset)

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

6. ⟨(apple, apple)⟩ - **No** (invalid, repetition within an itemset)

7. ⟨(newspaper) (apple)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

6. ⟨(apple, apple)⟩ - **No** (invalid, repetition within an itemset)

7. ⟨(newspaper) (apple)⟩ - **Yes**

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

6. ⟨(apple, apple)⟩ - **No** (invalid, repetition within an itemset)
7. ⟨(newspaper) (apple)⟩ - **Yes**
8. ⟨(bread) (tomato)⟩

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

6. ⟨(apple, apple)⟩ - **No** (invalid, repetition within an itemset)
7. ⟨(newspaper) (apple)⟩ - **Yes**
8. ⟨(bread) (tomato)⟩ - **No** (no super)

Subsequence - Exercise

How many of the following sequences are valid subsequences of
⟨ **(**bread, tomato**) (**apple, newspaper**) (**apple**)** ⟩?

1. ⟨(tomato) (newspaper)⟩ - **Yes**
2. ⟨(newspaper) (tomato)⟩ - **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ - **No** (no super-itemset)
4. ⟨(tomato) (apple)⟩ - **Yes**
5. ⟨(apple) (apple)⟩ - **Yes**
6. ⟨(apple, apple)⟩ - **No** (invalid, repetition within an itemset)
7. ⟨(newspaper) (apple)⟩ - **Yes**
8. ⟨(bread) (tomato)⟩ - **No** (no super)

Sequence Database

- A sequence database consists of sequences or ordered elements/events/items recorded with or without a notion of time
  - i.e. each tuple is a data-sequence
- A data sequence is a list of transactions each defined by transaction ID, timestamp (optional) and an itemset
- The patterns we look for do not have timestamps

ds 1	02/02/2015 15:02 (bread, tomato) 02/02/2015 15:17 (apple, newspaper) 07/02/2015 09:20 (apple)
ds 2	03/02/2015 16:04 (bread) 07/02/2015 18:11 (newspaper) 10/02/2015 00:01 (apple)
ds 3	01/02/2015 19:45 (apple, bread)
ds 4	01/02/2015 19:50 (apple)

Sequence Database

- A dataset could also be created using artificial time and/or with 1-itemsets

ds 1	1 (Basement) 2 (Atrium) 3 (Elevator) 4 (Class)
ds 2	1 (Basement) 2 (Scrollbar) 3 (Basement)
ds 3	1 (Atrium) 2 (Class) 3 (Autrium)

Sequence Pattern Mining

Frequent Sequences

- A sequence is frequent if it is supported by a **minimum** number of data-sequences (minimum support)
- A sequence is supported by a data-sequence if the sequence is a subsequence of the data-sequence (time is ignored)
- Even if a sequence appears multiple times in a single data sequence, that data sequence only contributes 1 to the support (e.g. apple in ds 1)

ds 1	02/02/2015 15:02 (bread, tomato) 02/02/2015 15:17 (apple, newspaper) 07/02/2015 09:20 (apple)
ds 2	03/02/2015 16:04 (bread) 07/02/2015 18:11 (newspaper) 10/02/2015 00:01 (apple)
ds 3	01/02/2015 19:45 (apple, bread)
ds 4	01/02/2015 19:50 (apple)

Sequence	Support
(bread) (apple)	2
(apple, bread)	1
(bread)	3
(apple)	4

Frequent Sequences

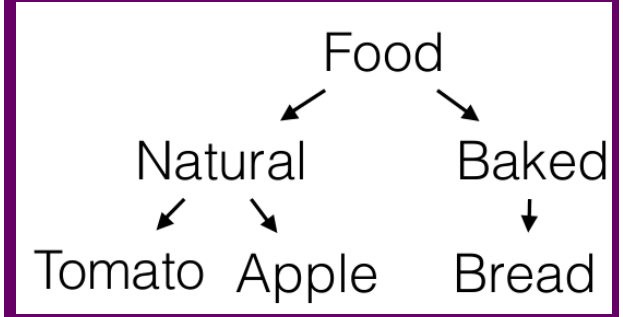
- A frequent sequence is also known as a *sequential pattern*
- A sequential pattern with length l is also known as a *l-pattern*

ds 1	02/02/2015 15:02 (bread, tomato) 02/02/2015 15:17 (apple, newspaper) 07/02/2015 09:20 (apple)
ds 2	03/02/2015 16:04 (bread) 07/02/2015 18:11 (newspaper) 10/02/2015 00:01 (apple)
ds 3	01/02/2015 19:45 (apple, bread)
ds 4	01/02/2015 19:50 (apple)

Support Constraints

- Taxonomies: allows the use of generalisations of items
  - An item (in a data-sequence) is equivalent to all its ancestors while counting supports

ds 1	02/02/2015 15:02 (bread, tomato) 02/02/2015 15:17 (apple, newspaper) 07/02/2015 09:20 (apple) 07/02/2015 09:35 (tomato)
ds 2	03/02/2015 16:04 (bread) 07/02/2015 18:11 (newspaper) 10/02/2015 00:01 (apple)
ds 3	01/02/2015 19:45 (apple, bread)
ds 4	01/02/2015 19:50 (apple)



Sequence	Suppor
(food)(newspaper)	2
(baked) (newspaper)	2
(natural, newspaper)	1
(food)	4
(natural)	4

Support Constraints

- Sliding windows / event folding window
  - Extends the concept of simultaneousness
  - Events distributed over different transactions are considered to be occuring together if transactions occur within specified time frame, i.e. window-size

ds 1	**07/02/2015 09:20** (apple) **07/02/2015 09:35** (tomato)
ds 2	03/02/2015 16:04 (bread) 07/02/2015 18:11 (newspaper) 10/02/2015 00:01 (apple)
ds 3	01/02/2015 19:45 (apple, bread)
ds 4	01/02/2015 19:50 (apple)

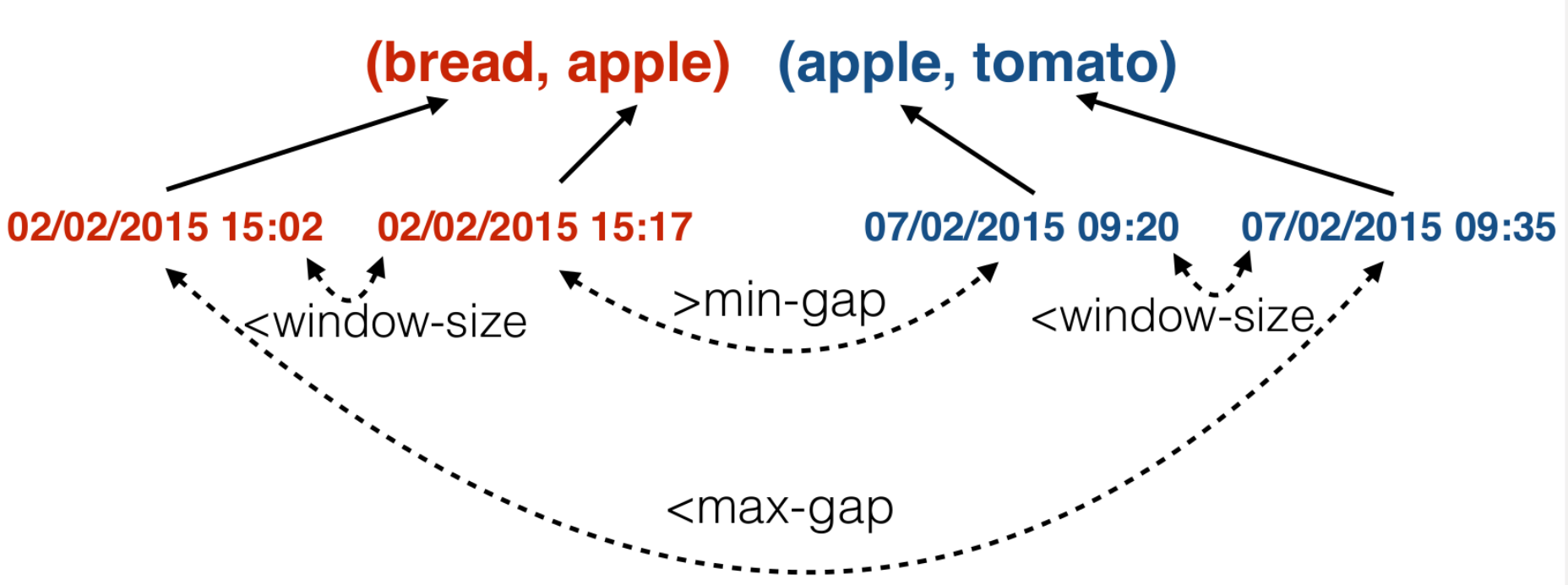
window-size= 30 min

Sequence	**Support**
(bread) (apple)	2
(apple, bread)	1
(apple, tomato)	**1**
(bread)	3
(apple)	4

Support Constraints

- Duration T of a sequence
  - i.e. either how many events in sequence or total time duration between first and last event in sequence
- Time gap
  - Restrict the time gap between consecutive itemsets
  - Min-gap: minimum time between the last item in one itemset and the first in the next itemset
  - Max-gap: maximum time between the first item in one itemset and the last in the next itemset

Support Constraints



Support Constraints

- Types of constraints
  - Anti-monotonic
    - If sequence doesn't satisfy constraint then neither will its supersequences
      - e.g. duration less or equal than 10 (since supersequences are larger)
  - Monotonic
    - If sequence does satisfy constraint then so will its supersequences
      - e.g. duration larger than 10 (since supersequences are larger)
  - Succinct
    - If we can enumerate all and only those sequences that are guaranteed to satisfy constraint (e.g. selecting sequences from year=2015)

GSP Algorithm

- The Generalised Sequential Patterns (GSP) algorithm is an adaptation of Apriori to find sequential patterns
- Old modules
  - same candidate generation-and-test approach to find all frequent patterns
  - same pruning principle: a pattern is only frequent if all its sub-patterns are
- New modules
  - Count support using a definition of sequence
  - Efficient candidate generation for sequences

GSP Algorithm

- Generate candidate set of length k+1 (sequences with k+1 items) from frequent sequences of length k
  - Step 1: generation
  - Step 2: pruning
- Count the support of the sequences in the candidate set
- Drop sequences that are supported by a number of data-sequence below the minimum threshold

GSP Candidate Generation

- Step 1: self-join the set of frequent sequences of length k
  - We join one sequence S_1 with another S_2 if and only if removing the first item of S_1 and the last item of S_2 leads to the same sequence (S_1 and S_2 share a contiguous subsequence*)
    - First and last is determined by lexicographical order within the first and last itemsets
  - The new sequence is created by adding the last item of S_2 to S_1
    - If the last item of S_2 was a separate itemset, it is appended as a new itemset. Otherwise is added to the last itemset of S_1
    - In the first iteration (k=2), create candidates with 1 and 2 itemsets

GSP Candidate Generation – Step 1 Example 1

S_1**:**⟨**(**bread, tomato**) (**apple**)**⟩ S_2**:** ⟨**(**tomato**) (**apple, newspaper**)**⟩

S_1-first item**:**⟨**(**tomato**) (**apple**)**⟩ S_2-last item**:** ⟨**(**tomato**) (**apple**)**⟩

S_1 + **last item of** S_2**:** ⟨**(**bread, tomato**) (**apple, newspaper**)**⟩

GSP Candidate Generation – Step 1 Example 2

S_1**:**⟨**(**bread, tomato**) (**apple, newspaper**)**⟩

S_1-first item**:**⟨**(**tomato**) (**apple, newspaper**)**⟩

S_2**:** ⟨**(**tomato**) (**apple, newspaper**) (**apple**)**⟩

S_2-last item**:** ⟨**(**tomato**) (**apple, newspaper**)**⟩

S_1 + **last item of** S_2**:**⟨**(**bread, tomato**) (**apple, newspaper**) (**apple**)**⟩

GSP Candidate Generation

- Step 2: prune the candidate list of (k+1)-sequences before counting support
  - A sequence can only be frequent if all its *contiguous* subsequences are frequent
  - The contiguous k-subsequences of a (k+1)-sequence are generated by
    - dropping an item from the first or last itemset (used for generation!)
    - dropping an item from any itemset with more than one item

GSP Candidate Generation

S_1**:**⟨**(**bread, tomato**) (**apple, newspaper**)**⟩

S_2**:** ⟨**(**tomato**) (**apple, newspaper**) (**apple**)**⟩

S_1 + **last item of** S_2**:**⟨**(**bread, tomato**) (**apple, newspaper**) (**apple**)**⟩

⟨**(**tomato**) (**apple, newspaper**) (**apple**)**⟩
⟨**(**bread**) (**apple, newspaper**) (**apple**)**⟩
⟨**(**bread, tomato**) (**newspaper**) (**apple**)**⟩
⟨**(**bread, tomato**) (**apple**) (**apple**)**⟩
⟨**(**bread, tomato**) (**apple, newspaper**)**⟩

GSP Counting Support

- Basic counting by scanning through data-base
  - Problem: Many scans!
- Improve GSP by using hash-tree technique (see paper for more)
- Other algorithms have attempted to solve this scanning problem
  - SPADE: Uses the vertical data format to reduce number of scans needed
  - PrefixSpan: Looks at sequence prefix and suffix to create FP-trees to avoid candidate generation

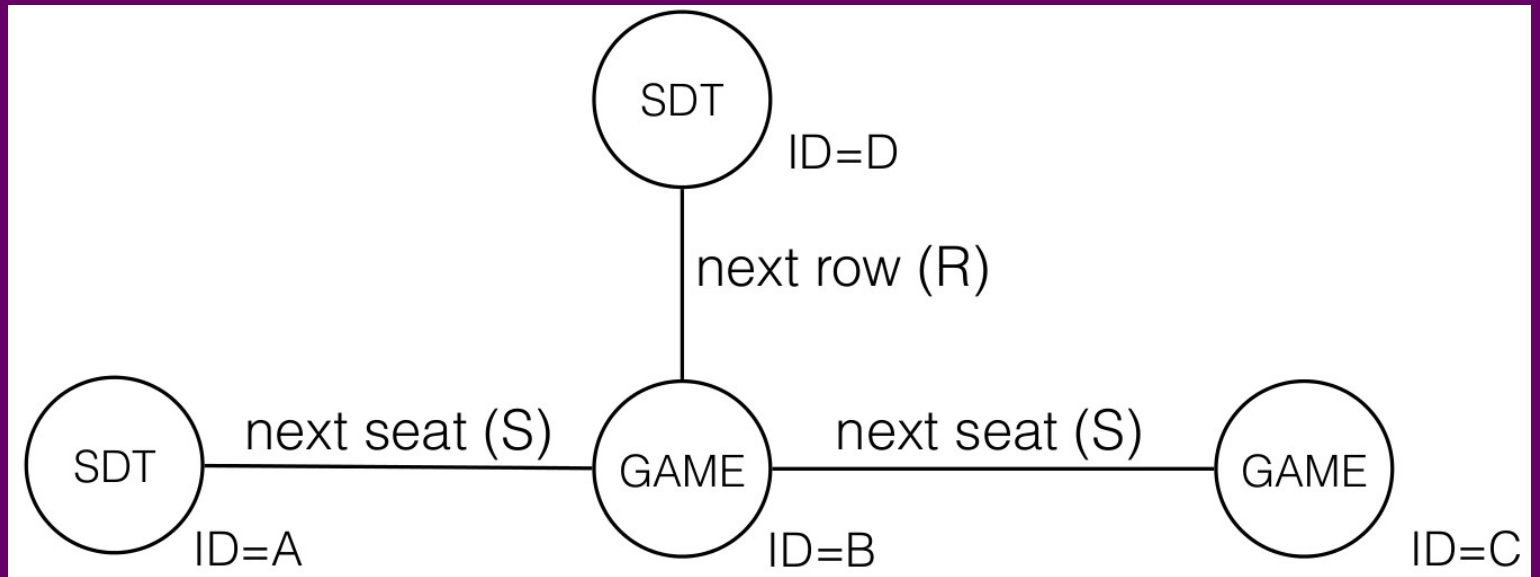
Sequential Pattern Usages

- Understanding the data (which events happen often in particular sequences, which not)
  - Association rules can be extracted in the same way as with itemsets

Graph Mining Basics

Undirected Graph

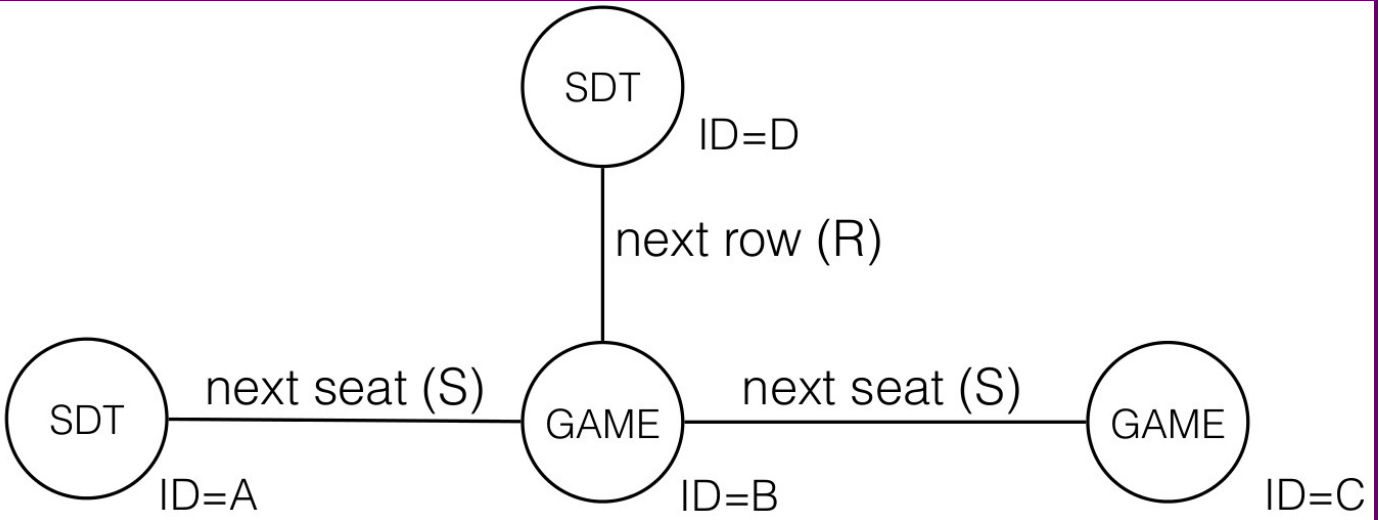
- A graph is defined by a set of vertices connected via edges
  - Vertices and edges have (non-unique) labels
  - Vertices are identified by unique (arbitrary) identifiers
  - An edge is defined by the IDs of the vertices that it connects and its label



Graph representation

- A graph can be represented as an adjacency matrix (AM)
  - Each row and each column represents a vertex ID
  - Each cell represents the edge between the row and column vertices
    - If no edge the value of 0 is used

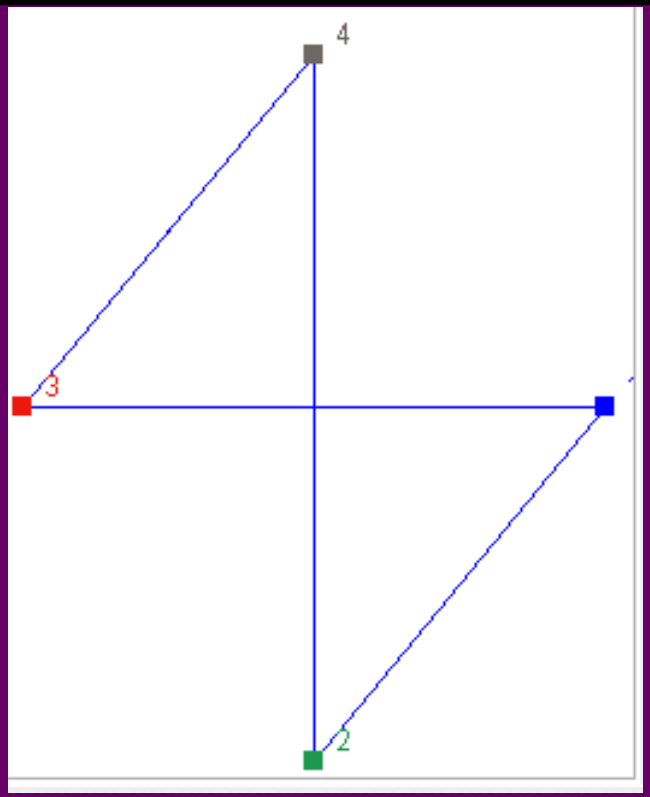
	A (SDT)	B (GAME)	C (GAME)	D (SDT)
A	0	seat	0	0
B	seat	0	seat	row
C	0	seat	0	0
D	0	row	0	0

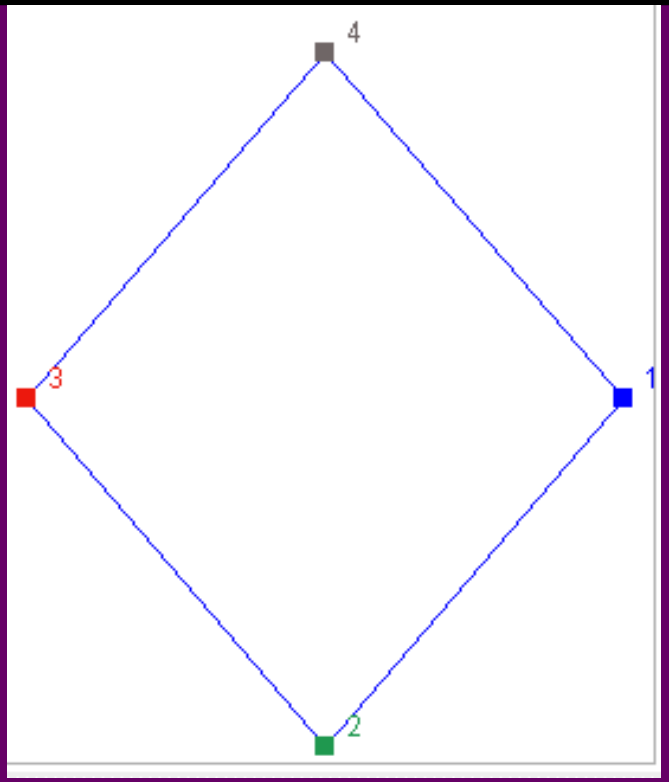


Graph Isomorphism

- When are two graphs the same?
- Consider two graphs G1 and G2, and the function f which maps the vertices of G1 to the vertices of G2
  - We suppose that
    - Function f is one-to-one
    - f(v) is adjacent to f(w) in G2 if and only if v is adjacent to w in G1
  - Then function f is an isomorphism and the two graphs G1 and G2 are isomorphic.
- Two isomorphic graphs are for all intents and purposes considered to be the same graph

Graph Isomorphism





Isomorphism: 1→1; 2→2; 3→4; 4→3
Example from:http://www.mathcove.net/petersen/lessons/get-lesson?les=3

Graph Comparison

- Problem: How do we compare two graphs?
  - e.g. when sorting in order to find frequent patterns
- The order among vertices is arbitrary (induced by arbitrary IDs), the same graph may have seemingly different adjacency matrices
- Furthermore, two different graphs may be equivalent due to symmetries (isomorphism)

Canonical Label

- A canonical label is a unique code that unequivocally represents a graph and all its isomorphic graphs
- A simple approach to obtain a canonical label consists of flattening the adjacency matrix following a set of deterministic ordering rules
  - 1.Sort vertices by their degree
  - 2.Sort vertices with the same degree by label
  - 3.Sort edges associated to the same vertex labels by edge label

Flattened Adjacency Matrix - 1

- Partition vertices by their degree (number of edges connected to them)

	A (SDT)	B (GAME)	C (GAME)	D (SDT)
A	0	seat	0	0
B	seat	0	seat	row
C	0	seat	0	0
D	0	row	0	0

	A (SDT)	C (GAME)	D (SDT)	B (GAME)
A	0	0	0	seat
C	0	0	0	seat
D	0	0	0	row
B	seat	seat	row	0

Flattened Adjacency Matrix - 2

- Sort partitions by vertex label (lexicographical order)

	A (SDT)	C (GAME)	D (SDT)	B (GAME)
A	0	0	0	seat
C	0	0	0	seat
D	0	0	0	row
B	seat	seat	row	0

	C (GAME)	A (SDT)	D (SDT)	B (GAME)
C	0	0	0	seat
A	0	0	0	seat
D	0	0	0	row
B	seat	seat	row	0

Flattened Adjacency Matrix - 3

- Represent the adjacency matrix by its elements below the diagonal (choose the permutation where edges within each partition are ordered lexicographically)

	C (GAME)	A (SDT)	D (SDT)	B (GAME)
C	0	0	0	seat
A	0	0	0	seat
D	0	0	0	row
B	seat	seat	row	0

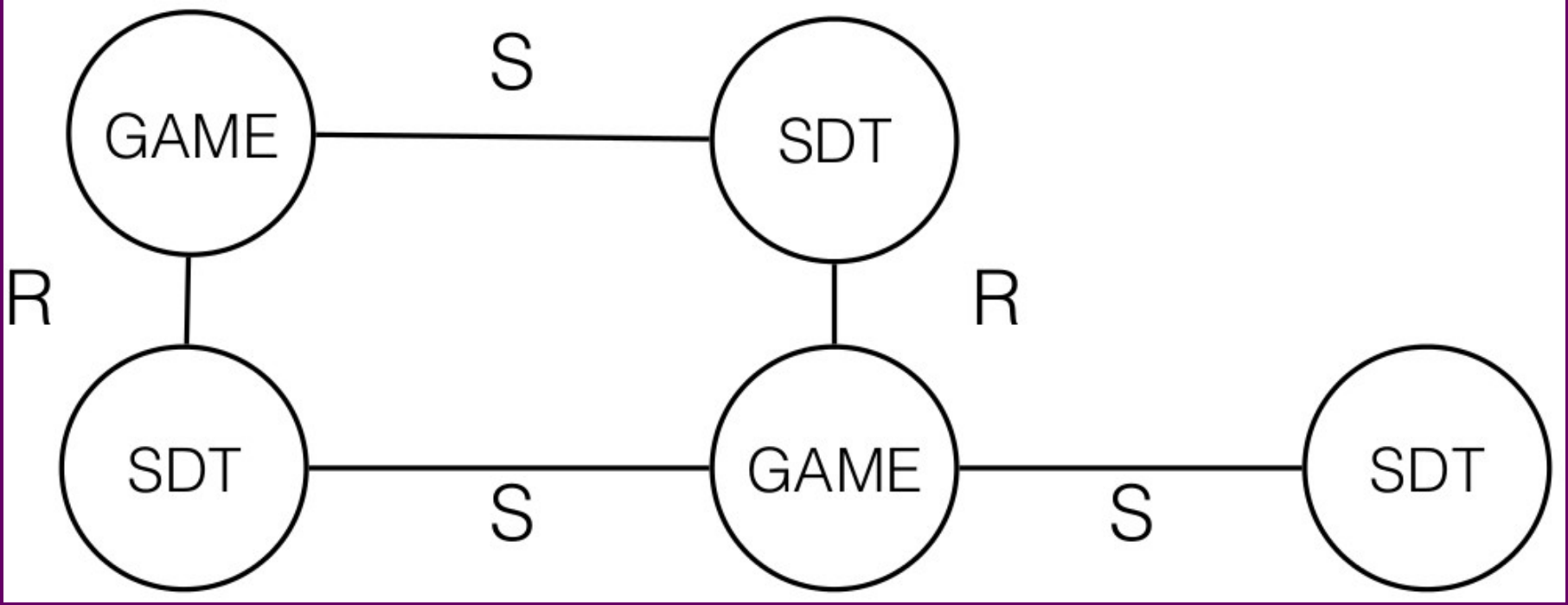
0 00 seat,seat,row

	C (GAME)	D (SDT)	A (SDT)	B (GAME)
C	0	0	0	seat
D	0	0	0	row
A	0	0	0	seat
B	seat	row	seat	0

0 00 seat,row,seat

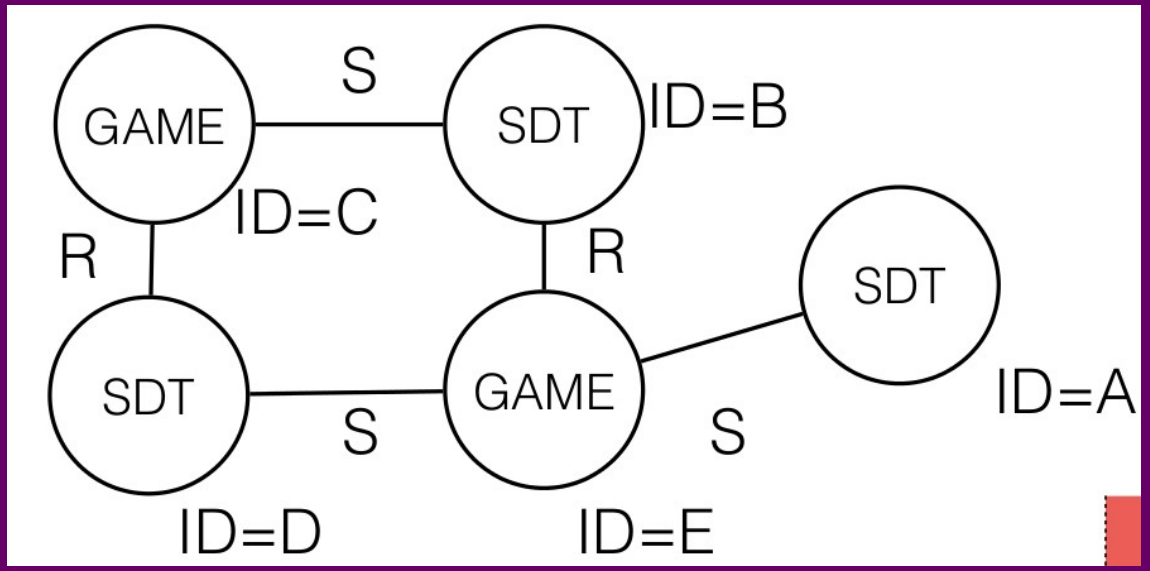
Flattened Adjacency Matrix - Example

- What is the canonical representation of?



Flattened Adjacency Matrix - Example

- First step → Partition based on degree. Three partitions made



	A	B	C	D	E
A	0	0	0	0	S
B	0	0	S	0	R
C	0	S	0	R	0
D	0	0	R	0	S
E	S	R	0	S	0

Flattened Adjacency Matrix - Example

- Step 2 → Sort each partition based on vertex label

  A = SDT, B = SDT, C = GAME, D = SDT, E = GAME

	A	B	C	D	E
A	0	0	0	0	S
B	0	0	S	0	R
C	0	S	0	R	0
D	0	0	R	0	S
E	S	R	0	S	0

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	**0**	0	S	R	0
B	**0**	**S**	0	0	R
D	**0**	**R**	**0**	0	S
E	**S**	**0**	**R**	**S**	0

Flattened Adjacency Matrix - Example

- Reading Adjacency Matrix as canonical label

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	S	R	0
B	0	S	0	0	R
D	0	R	0	0	S
E	S	0	R	S	0

Label: 0

Flattened Adjacency Matrix - Example

- Reading Adjacency Matrix as canonical label

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	S	R	0
B	0	S	0	0	R
D	0	R	0	0	S
E	S	0	R	S	0

Label: 00S

Flattened Adjacency Matrix - Example

- Reading Adjacency Matrix as canonical label

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	S	R	0
B	0	S	0	0	R
D	0	R	0	0	S
E	S	0	R	S	0

Label: 0 0S 0R0

Flattened Adjacency Matrix - Example

- Reading Adjacency Matrix as canonical label

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	S	R	0
B	0	S	0	0	R
D	0	R	0	0	S
E	S	0	R	S	0

Label: 0 0S 0R0 S0RS

Flattened Adjacency Matrix - Example

- Step 3: Fill in below diagonal, select label that is lexicographically first

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	S	R	0
B	0	S	0	0	R
D	0	R	0	0	S
E	S	0	R	S	0

Label: 0 0S 0R0 S0RS

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	R	S	0
D	0	R	0	0	S
B	0	S	0	0	R
E	S	0	S	R	0

Label: 0 0R 0S0 S0SR

Flattened Adjacency Matrix - Example

- Step 3: Fill in below diagonal, select permutation whose label is lexicographically first

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	S	R	0
B	0	S	0	0	R
D	0	R	0	0	S
E	S	0	R	S	0

Label: 0 0S 0R0 S0RS

	SDT	GAME	SDT	SDT	GAME
A	0	0	0	0	S
C	0	0	R	S	0
D	0	R	0	0	S
B	0	S	0	0	R
E	S	0	S	R	0

Label: 0 0R 0S0 S0SR

Subgraphs

- A graph G_1 is a subgraph of G_2 if all edges in G_1 are contained in G_2
- Testing if a graph G_2 contains a subgraph G_1 is equivalent to finding an isomorphic subgraph of G_1 in G_2
  - The operation consists of finding a mapping between the vertex IDs of G_1 and a subset of the vertex IDs of G_2 such that the edges in G_1 are in G_2 (and vertices maintain the same labels)
  - If $G_1 = G_2$, the operation is called automorphism (finding the isomorphisms of a graph)
- We say that a graph is a k-[sub]graph if it contains k edges

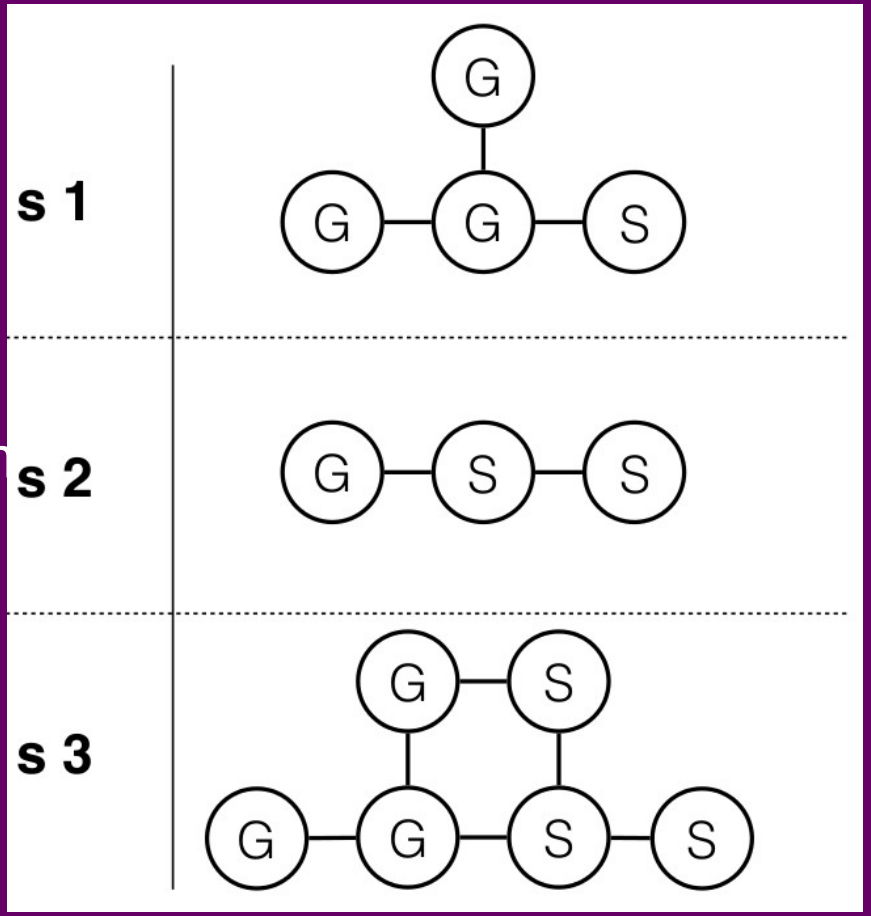
Apriori for Graph Mining

Frequent graph mining

- The Frequent Subgraph Discovery algorithm is an adaptation of Apriori to find patterns in graphs
- Old modules:
  - same candidate generation-and-test approach to find all frequent patterns
  - same pruning principle: a pattern is only frequent if all its sub-patterns are
- New modules:
  - Count support using a definition of graph
  - Efficient candidate generation for graphs

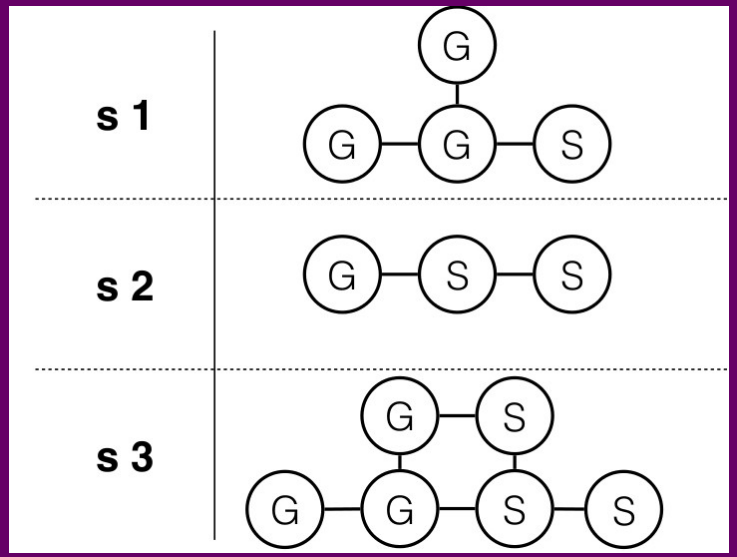
Data Set

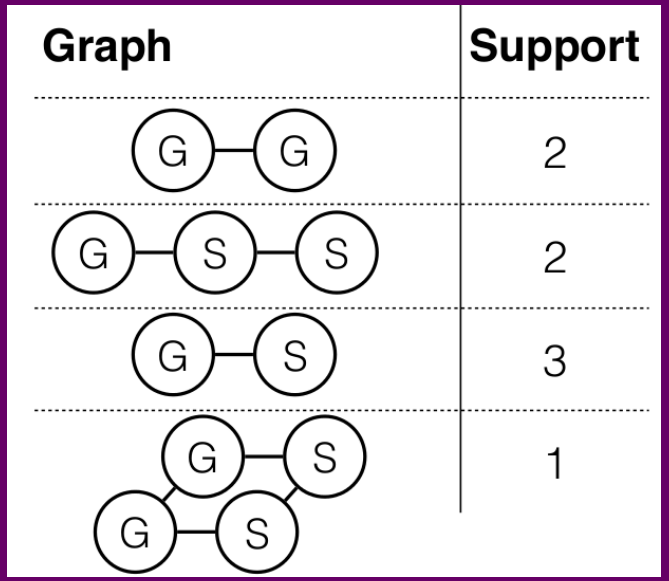
- Each sample is a graph (adjacency matrix, canonical label, adjacency list...)
- Graphs can be independent (e.g. the sitting-graph of the auditorium every week) or can be extracted from a larger graph (e.g. up to 2-friend distance from a person on a social network)



Support constraints

- A graph is frequent if it is supported by a minimum number of samples (minimum support)
- A graph is supported by a sample if is one of its subgraphs (or one of its isomorphisms is)
  - Even if the graph appears multiple times in the same sample it only counts one towards its support





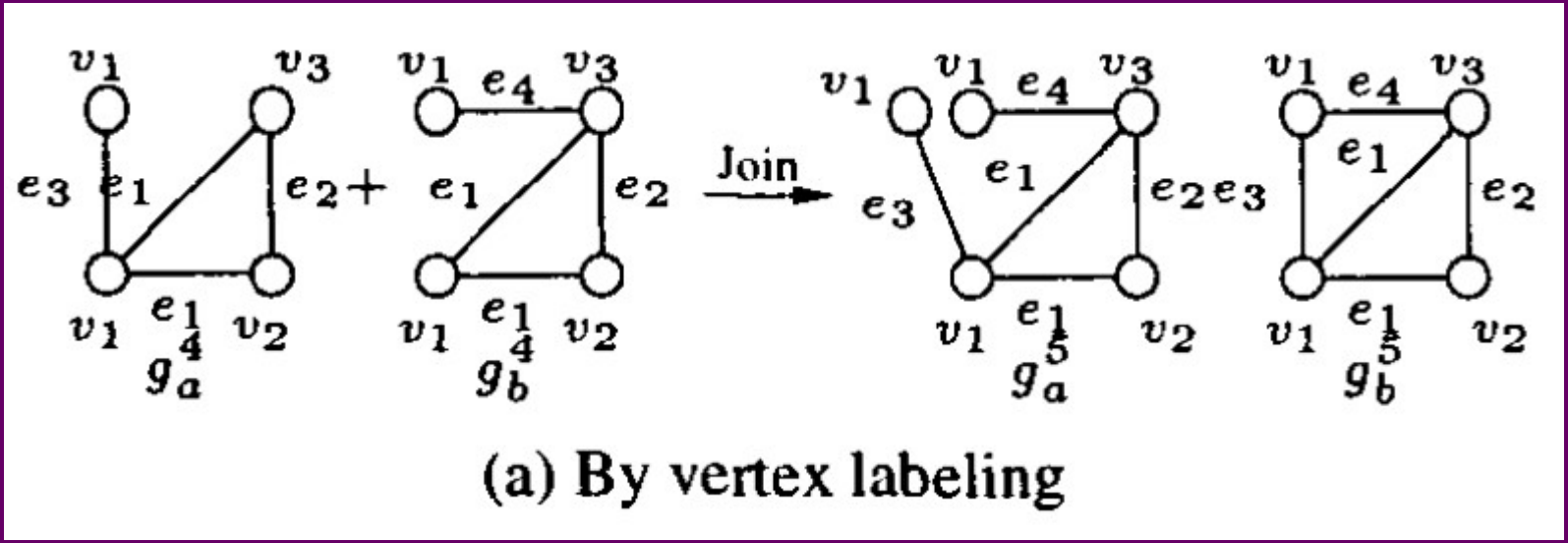
FSD algorithm

- Generate candidate set of length k+1 (graphs with k+edges) from frequent graphs of length k
  - Step 1: generation
  - Step 2: pruning
- Count the support of the graphs in the candidate set
- Drop graphs that are supported by a number of samples below the minimum threshold

Candidate generation

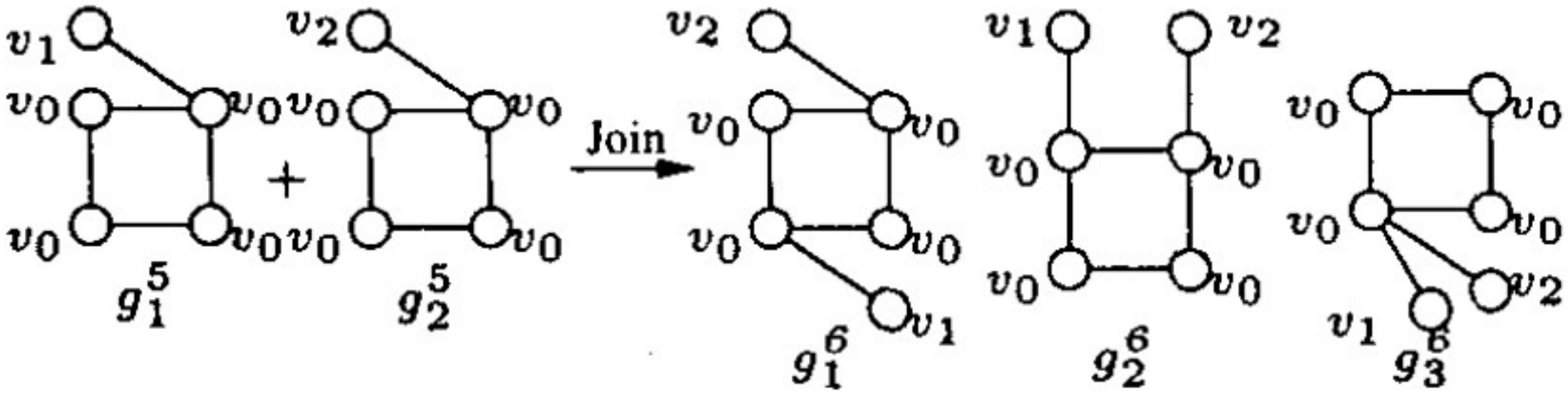
- Step 1: self-join the set of frequent graphs of length k
  - We join one graph G_1 with another G_2 if and only if removing one edge from G_1 leads to a subgraph contained in G_2 (the resulting subgraph is a core of G_1 and G_2)
    - Two graphs can share more than one core
- For every automorphism of the core, we create a candidate by adding the edge removed from G_1 to G_2 (using the vertex mapping specified by the automorphism)

Candidate generation



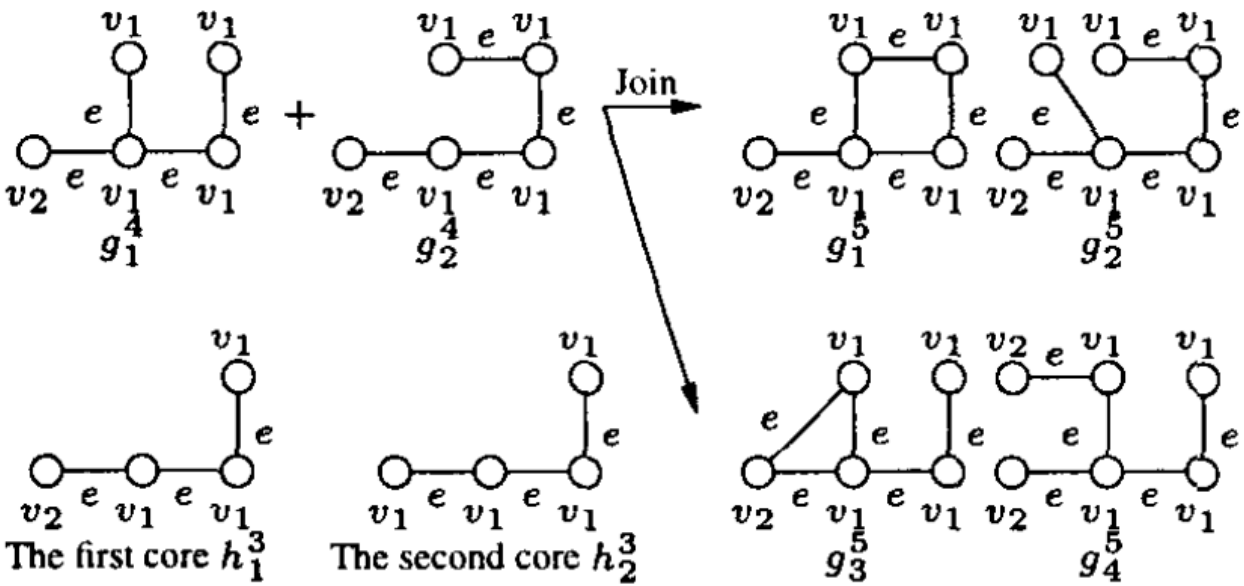
- If both graphs have one more vertex than the core with the same label, add two candidates
  - In this case: V_1

Candidate generation



(b) By multiple automorphisms of a single core

Candidate generation



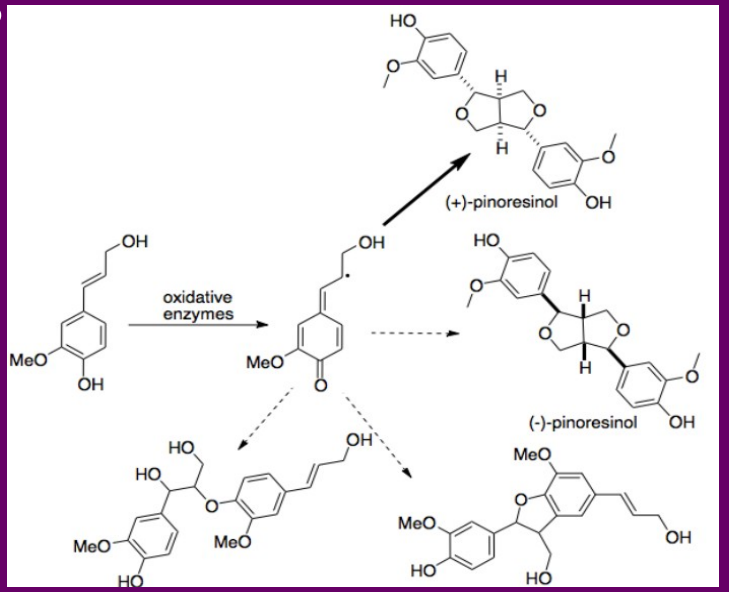
(c) By multiple cores

Candidate generation and Support count

- Step 2: prune the candidate list of (k+1)-graphs before counting support
  - A graph can only be frequent if all its subgraphs are frequent
- Counting support
  - Simple: Scan through database and count support for each subgraph
  - Optimization idea
    - We keep a list of the samples that support every frequent graph (vertical data format)
    - To evaluate the count support of a new candidate, we only evaluate those samples that support all of its frequent subgraphs (i.e. the intersection of the sample lists of each of its frequent subgraphs)

Graph Mining Applications

- Data understanding
- Graph mining has been used in bio-informatics (e.g. patterns in proteins and other organic compounds)
- Used also for social network analysis



Summary

- Most data mining algorithms are modular, and can be adjusted to solve different problems
- GSP: Apriori + sequences
  - Support count: time and order constraints
  - Candidate generation: contiguous sequences
- FSM: Apriori + graphs
  - Support count: subgraph isomorphism
  - Candidate generation: core of two graphs

Today's Lab

- Catch up lab with optional exercises
- We all meet in 4A58 to see if we can fit in
  - If not, we also use 4A54

Thanks for listening!