DATA MINING

SEQUENCE MINING AND GRAPH MINING

OVERVIEW

- Apriori refresher
- Frequent-sequence mining
- Graphs in data mining
- Introduction to graph mining

APRIORI

APRIORI IN A NUTSHELL

- Problem: we cannot test the frequency of all possible itemsets
- Needed: algorithm that reduces the number of candidates!
- But: we cannot miss interesting candidates in the process
- **Apriori property**: if an itemset is not frequent, its supersets won’t be either

APRIORI IN A NUTSHELL

Generate k-candidates using (only) frequent (k–1)-itemsets

- **Join** if only last element is different
- **Prune** if any k-subset is not in the list of frequent k-itemsets

Test frequency of candidates

FLASHBACK: APRIORI—STEP-BY-STEP

TID	Items
10	A, C, D
20	B, C
30	A, B, C, E
40	B, E

Let us apply Apriori to this data set, containing a list of items for each transaction ID.

FLASHBACK: APRIORI—STEP-BY-STEP

1. **Scan** for frequent 1-itemsets

TID	Items
10	A, C, D
20	B, C
30	A, B, C, E
40	B, E

C_1	Support
{A}	2
{B}	3
{C}	3
{D}	**1**
{E}	2

FLASHBACK: APRIORI—STEP-BY-STEP

2. **Discard** non frequent candidates: we have L_1

C_1	Support
{A}	2
{B}	3
{C}	3
{D}	1
{E}	2

L_1
{A}
{B}
{C}
{E}

FLASHBACK: APRIORI—STEP-BY-STEP

3. **Join** step to produce k = 2 candidates

L_1
{A}
{B}
{C}
{E}

→

C_2
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

FLASHBACK: APRIORI—STEP-BY-STEP

4. **Pruning** (trivial for k = 2)

C_2
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

C_2	Subset	In L_1?
{A, B}	{A}	yes
	{B}	yes
{A, C}	{A}	yes
	{C}	yes
{A, E}	{A}	yes
	{E}	yes
{B, C}	{B}	yes
	{C}	yes
{B, E}	{B}	yes
	{E}	yes
{C, E}	{C}	yes
	{E}	yes

FLASHBACK: APRIORI—STEP-BY-STEP

5. **Scan** and **discard** non-frequent candidates: we have L_2

C_2	Support
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	2
{C, E}	1

L_2	Support
{A, C}	2
{B, C}	2
{B, E}	2

FLASHBACK: APRIORI—STEP-BY-STEP

6. **Join** step to produce k = 3 candidates

L_2	Support
{A, C}	2
{B, C}	2
{B, E}	2

→

C_3
{B, C, E}

FLASHBACK: APRIORI—STEP-BY-STEP

7. **Prune**

C_3
{B, C, E}

C_3	Subset	In L_2?
{B, C, E}	{B, C}	yes
	{B, E}	yes
	{C, E}	no

FLASHBACK: APRIORI—STEP-BY-STEP

8. All C_3 candidates have been rejected. **End** and **return** L_1, L_2

TID	Items
10	A, C, D
20	B, C
30	A, B, C, E
40	B, E

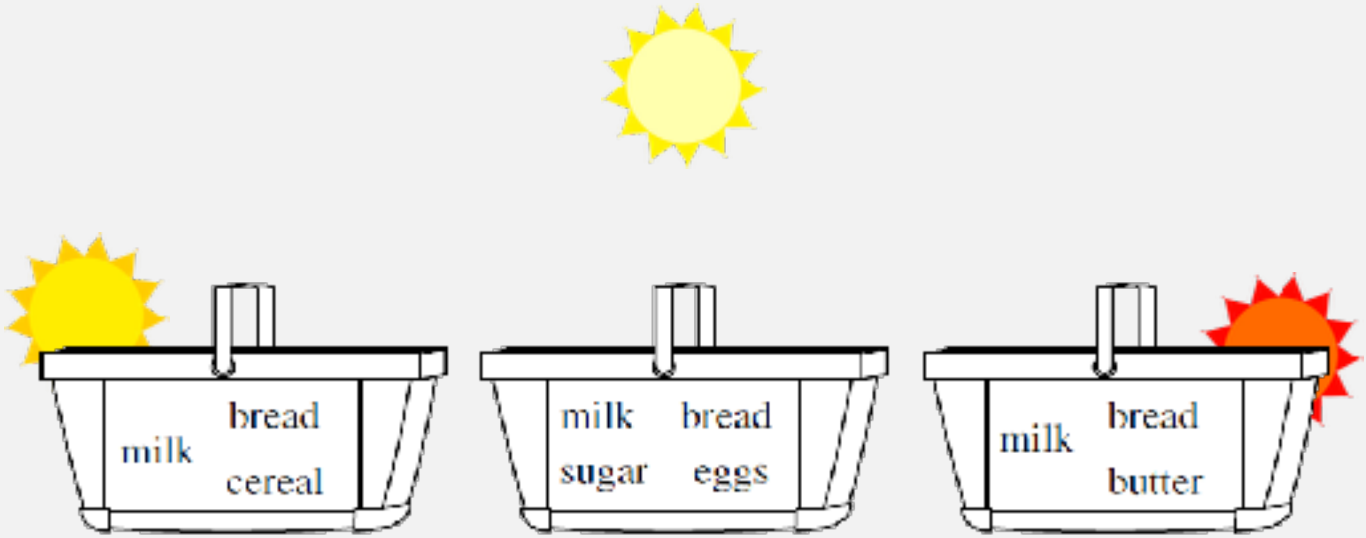
L_2
{A, C}
{B, C}
{B, E}

L_1
{A}
{B}
{C}
{E}

FREQUENT-SEQUENCE MINING

SEQUENCE

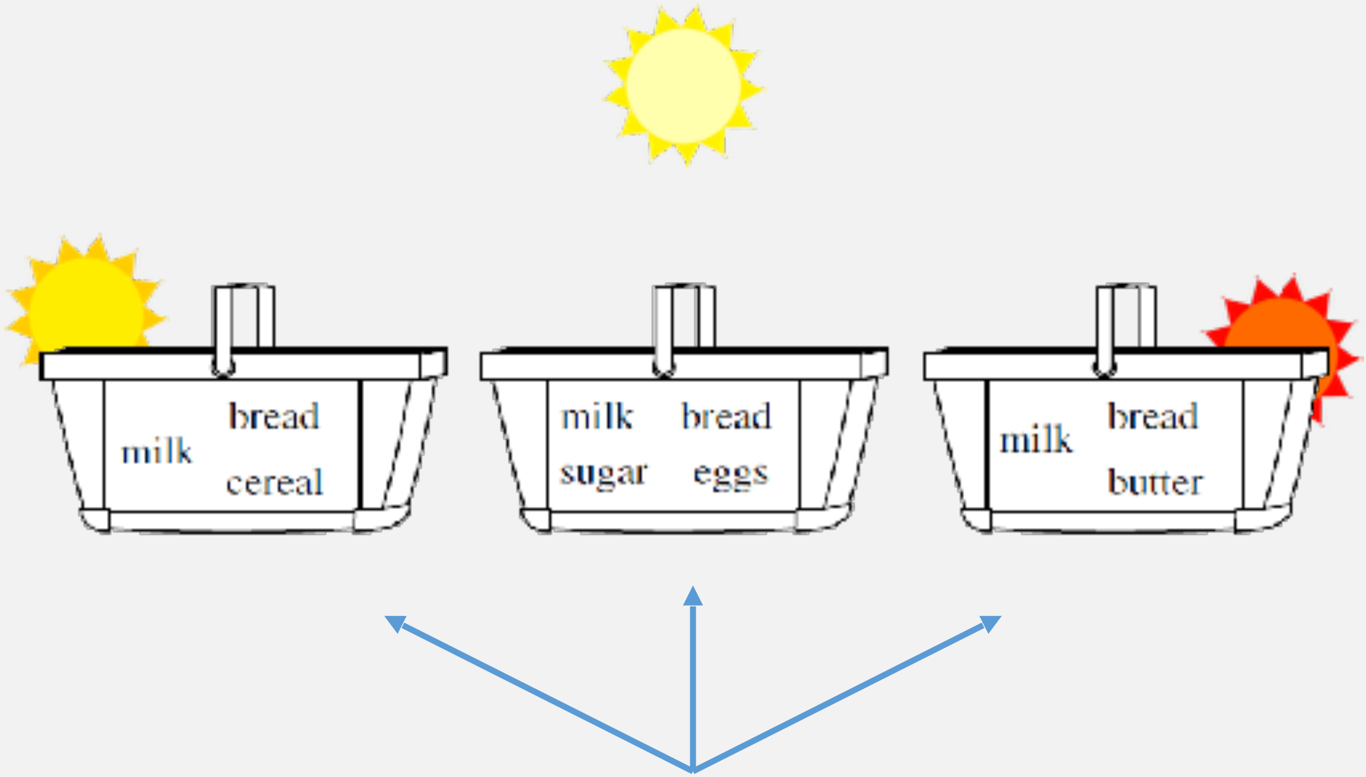
$$S = \langle e_0, e_1, \ldots, e_n \rangle$$



- Ordered list of events
- Events are also called elements
- E.g., in customer purchase each event is an itemset

SEQUENCE

$$S = \langle e_0, e_1, \ldots, e_n \rangle$$



The items in an itemset are considered **simultaneous**

SEQUENCE

$$S = \langle e_0, e_1, \ldots, e_n \rangle$$

- Itemsets in a sequence are written with parentheses: $(x_1, x_2, \ldots, x_q)$
- An item can only happen once in an itemset, but may appear in different events!
- Sequence length: number of items (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

SUB/SUPERSEQUENCE

S_1 is a subsequence of S_2 if:

- Every itemset in S_1 has a superset in S_2
- The order of the itemsets in S_1 is the same as the order of their supersets in S_2

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

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

SUB/SUPERSEQUENCE

S_1 is a subsequence of S_2 if:

- Every itemset in S_1 has a superset in S_2
- The order of the itemsets in S_1 is the same as the order of their supersets in S_2

⟨**(**bread, tomato**)** **(**apple, newspaper**)** **(**bread**)** **(**apple**)**⟩ **6-sequence**

⟨**(**bread**)** **(**newspaper**)** **(**apple**)**⟩ **3-subsequence**

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩
2. ⟨**(**newspaper**)** **(**tomato**)**⟩
3. ⟨**(**newspaper, tomato**)**⟩
4. ⟨**(**tomato**)** **(**apple**)**⟩
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩ **Yes**
2. ⟨**(**newspaper**)** **(**tomato**)**⟩
3. ⟨**(**newspaper, tomato**)**⟩
4. ⟨**(**tomato**)** **(**apple**)**⟩
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩ **Yes**
2. ⟨**(**newspaper**)** **(**tomato**)**⟩
3. ⟨**(**newspaper, tomato**)**⟩
4. ⟨**(**tomato**)** **(**apple**)**⟩
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩ — Yes
2. ⟨**(**newspaper**)** **(**tomato**)**⟩ — **No** (wrong order)
3. ⟨**(**newspaper, tomato**)**⟩
4. ⟨**(**tomato**)** **(**apple**)**⟩
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩ — **Yes**
2. ⟨**(**newspaper**)** **(**tomato**)**⟩ — **No** (wrong order)
3. ⟨**(**newspaper, tomato**)**⟩
4. ⟨**(**tomato**)** **(**apple**)**⟩
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩ — **Yes**
2. ⟨**(**newspaper**)** **(**tomato**)**⟩ — **No** (wrong order)
3. ⟨**(**newspaper, tomato**)**⟩ — **No** (missing super-itemset)
4. ⟨**(**tomato**)** **(**apple**)**⟩
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨(tomato) (newspaper)⟩ — **Yes**
2. ⟨(newspaper) (tomato)⟩ — **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ — **No** (missing super-itemset)
4. ⟨(tomato) (apple)⟩
5. ⟨(apple) (apple)⟩
6. ⟨(apple, apple)⟩
7. ⟨(newspaper) (apple)⟩
8. ⟨(bread) (tomato)⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨(tomato) (newspaper)⟩ — **Yes**
2. ⟨(newspaper) (tomato)⟩ — **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ — **No** (missing super-itemset)
4. ⟨(tomato) (apple)⟩ — **Yes**
5. ⟨(apple) (apple)⟩
6. ⟨(apple, apple)⟩
7. ⟨(newspaper) (apple)⟩
8. ⟨(bread) (tomato)⟩

Exercise: Which is a valid subsequence of …?

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

1. ⟨**(**tomato**)** **(**newspaper**)**⟩ — **Yes**
2. ⟨**(**newspaper**)** **(**tomato**)**⟩ — **No** (wrong order)
3. ⟨**(**newspaper, tomato**)**⟩ — **No** (missing super-itemset)
4. ⟨**(**tomato**)** **(**apple**)**⟩ — **Yes**
5. ⟨**(**apple**)** **(**apple**)**⟩
6. ⟨**(**apple, apple**)**⟩
7. ⟨**(**newspaper**)** **(**apple**)**⟩
8. ⟨**(**bread**)** **(**tomato**)**⟩

Exercise: Which is a valid subsequence of …?

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

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

Exercise: Which is a valid subsequence of …?

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

1. ⟨(tomato) (newspaper)⟩ — **Yes**
2. ⟨(newspaper) (tomato)⟩ — **No** (wrong order)
3. ⟨(newspaper, tomato)⟩ — **No** (missing super-itemset)
4. ⟨(tomato) (apple)⟩ — **Yes**
5. ⟨(apple) (apple)⟩ — **Yes**
6. ⟨(apple, apple)⟩
7. ⟨(newspaper) (apple)⟩
8. ⟨(bread) (tomato)⟩

Exercise: Which is a valid subsequence of …?

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

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

Exercise: Which is a valid subsequence of …?

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

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

Exercise: Which is a valid subsequence of …?

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

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

Exercise: Which is a valid subsequence of …?

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

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

Exercise: Which is a valid subsequence of …?

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

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

DATABASE

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)

- Each element: data sequence
- **Data sequence**: ordered list of itemsets

  May or may not have timestamps

DATABASE

Two examples

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	(Atrium)

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)

CONSTRAINTS—FREQUENT SEQUENCE

- Is supported by a minimum number of data sequences: **minimum support**
- If a sequence is a **subsequence of a data sequence**, it is **supported** by it. Ignore time, not order
- A sequence can only be supported **only once** by each data sequence

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)

CONSTRAINTS—FREQUENT SEQUENCE

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)	
(apple, bread)	
(bread)	
(apple)	

CONSTRAINTS—FREQUENT SEQUENCE

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)	
(bread)	
(apple)	

CONSTRAINTS—FREQUENT SEQUENCE

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)	
(apple)	

CONSTRAINTS—FREQUENT SEQUENCE

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)	

CONSTRAINTS—FREQUENT SEQUENCE

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

CONSTRAINTS—FREQUENT SEQUENCE

Only counts once!

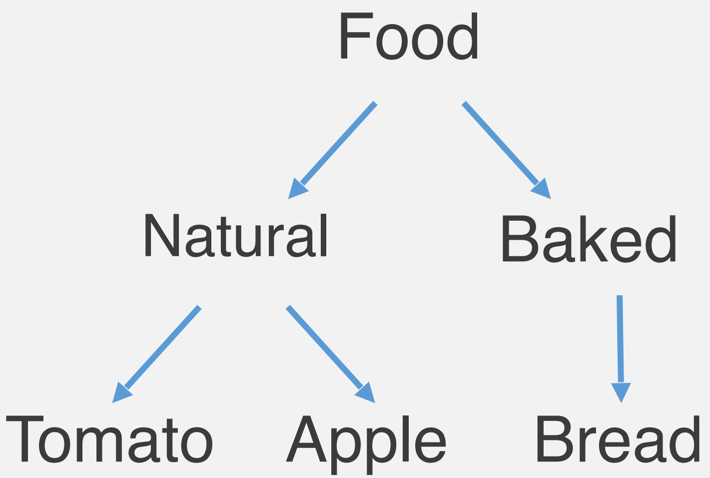
ds 1	02/02/2015	15:02	(bread, tomato)
	02/02/2015	15:17	(**apple**, newspaper)
	07/02/2015	09:20	(**apple**)
ds 2	03/02/2015	16:04	(bread)
	07/02/2015	18:11	(newspaper)
	10/02/2015	00:01	(apple)
ds 3	01/02/2015	19:45	(apple, bread)
ds 4	01/02/2015	19:50	(apple)

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

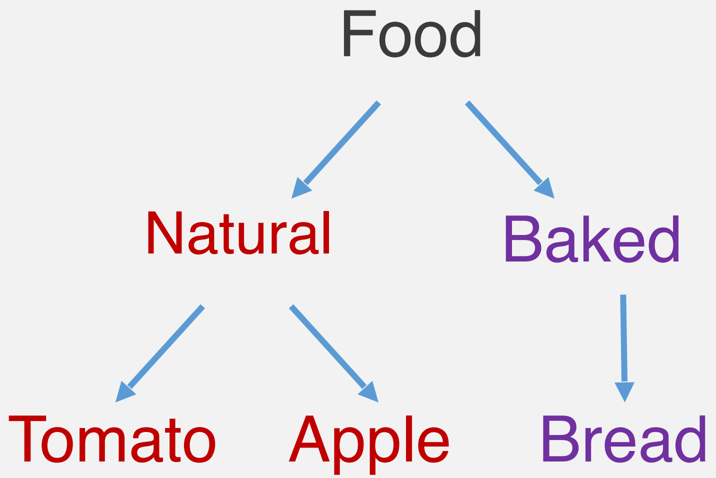
CONSTRAINTS—TAXONOMIES

When counting support an item is equivalent to all its ancestors

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)



CONSTRAINTS—TAXONOMIES



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**
(food) (newspaper)	2
(baked) (newspaper)	2
(natural, newspaper)	1
(food)	4
(natural)	4

CONSTRAINTS—SLIDING WINDOW

- Close itemsets may be considered simultaneous
- A super-itemset may be distributed across itemsets!

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	(bread)

Window size = 30 min

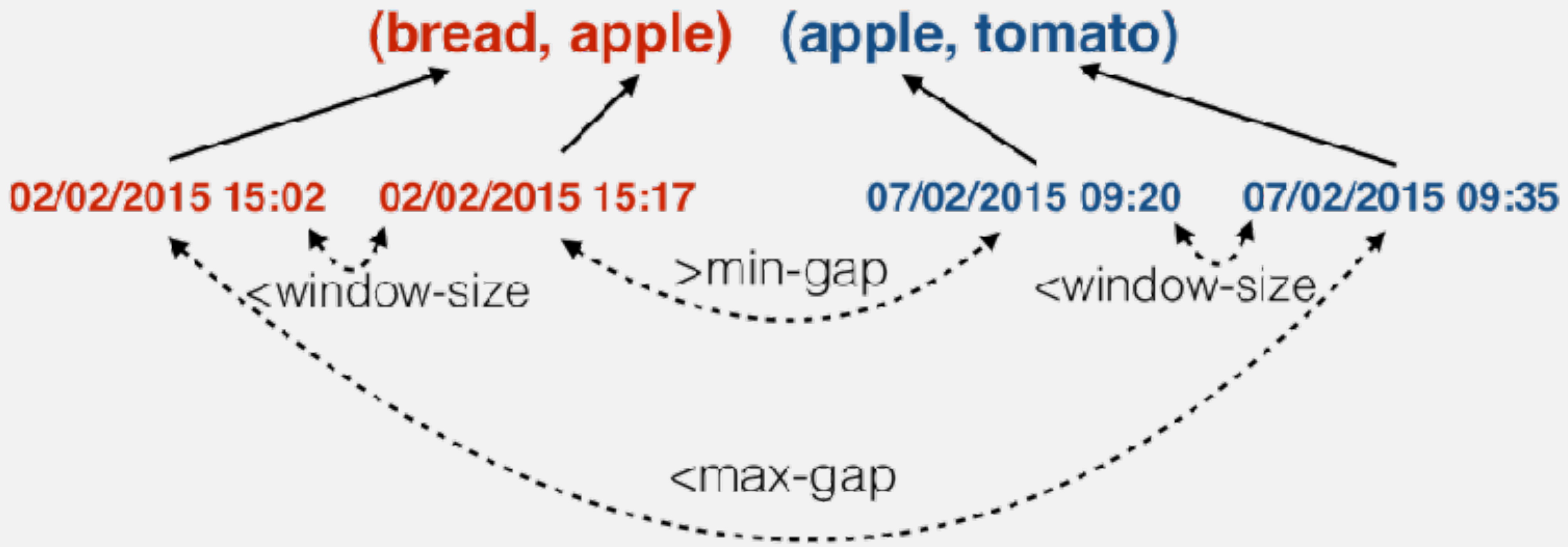
Sequence	**Support**
(bread) (apple)	2
(apple, bread)	2
(apple, tomato)	1
(bread)	3
(apple)	4

CONSTRAINTS—GAPS

- **Window size**: itemsets closer than this may be considered simultaneous
- **Min-gap**: minimum time between last item in one itemset and the first in the next
- **Max-gap**: maximum time between the first item in one itemset and the last in the next

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 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)



CONSTRAINTS

- **Anti-monotonic**: if a sequence does not satisfy it, neither do its supersequences

  E.g., Apriori property, “an itemset is only frequent if its subsets are”

- **Monotonic**: if a sequence satisfies it, so do its supersequences

  E.g., duration more than 10 min (supersequences are longer!)

- **Succint**: we can enumerate all and only those sequences that are guaranteed to satisfy the constraint

  E.g., price lower than 5€

GSP ALGORITHM

GSP: GENERALISED SEQUENTIAL PATTERNS

Extension of the Apriori algorithm to sequences!

1. Generate (k+1)-candidates using frequent k-sequences
    - Generate (join)
    - Prune
2. Count support of candidates
3. Drop non-frequent candidates

GSP—CANDIDATE GENERATION

- Join sequences S_1 and S_2 if removing the first item of S_1 and the last of S_2 produces the same sequence
  - This is a contiguous subsequence
  - **Items** need to be **sorted**! (E.g., lexicographical.)
- Add last item of S_2 to S_1
  - If last item is a separate itemset, append as new itemset. Otherwise add to the last itemset of
  - For k = 2 generate 1 and 2 itemsets (E.g., ⟨**(**A**)** **(**B**)**⟩ and ⟨**(**AB**)**⟩)

GSP—CANDIDATE GENERATION

S_1 : ⟨(bread, tomato) (apple)⟩

S_2 : ⟨(tomato) (apple, newspaper)⟩

S_1 – first: ⟨(tomato) (apple)⟩ = S_2 – last: ⟨(tomato) (apple)⟩

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

GSP—CANDIDATE GENERATION

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

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

S_1 – first: ⟨(tomato) (apple, newspaper)⟩ = S_2 – last: ⟨(tomato) (apple, newspaper)⟩

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

GSP—CANDIDATE PRUNING

- A candidate **may** only be frequent if all its contiguous subsequences are
- Generation of contiguous k-subsequences from (k+1)-sequence
  - Drop item from any itemset with more than one item
  - Dropping from first and last itemsets was used in join!

GSP—CANDIDATE PRUNING

5-candidate: ⟨**(**bread, tomato**)** **(**apple, newspaper**)** **(**apple**)**⟩

4-contiguous subsequences:

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

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

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

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

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

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

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

BEYOND GSP

- GSP requires many very **costly scans** for support count
- Improvements possible. E.g., hash-tree technique
- Alternative algorithms

  SPADE: vertical format

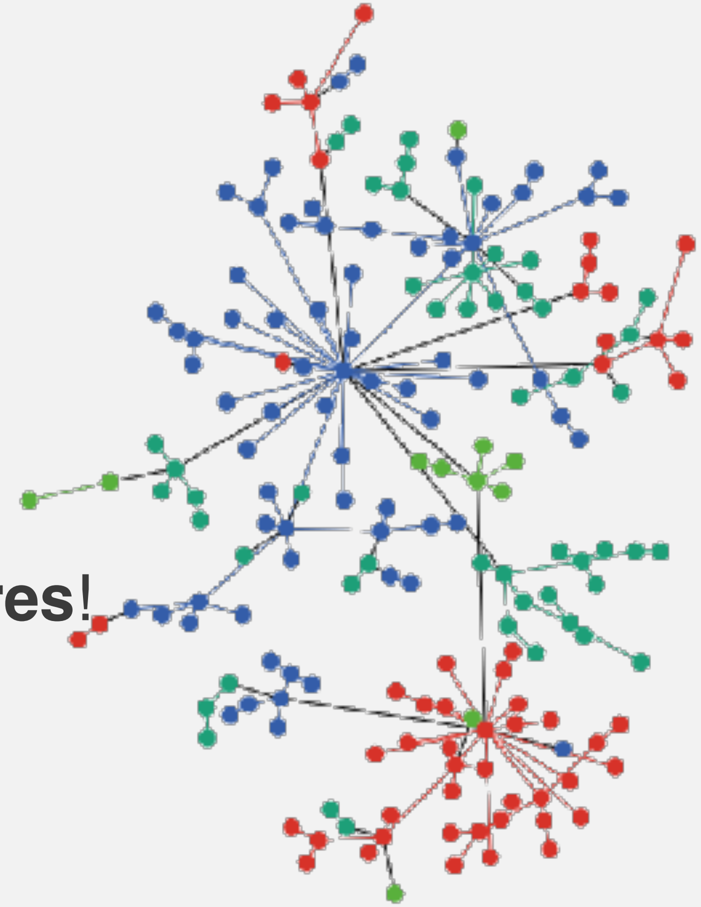
  PrefixSpan: *Pattern-growth* method, avoids candidate generation!

WHY SEQUENTIAL PATTERNS?

- Useful to understand data
- It is possible to extract association rules as with itemsets!

GRAPH MINING

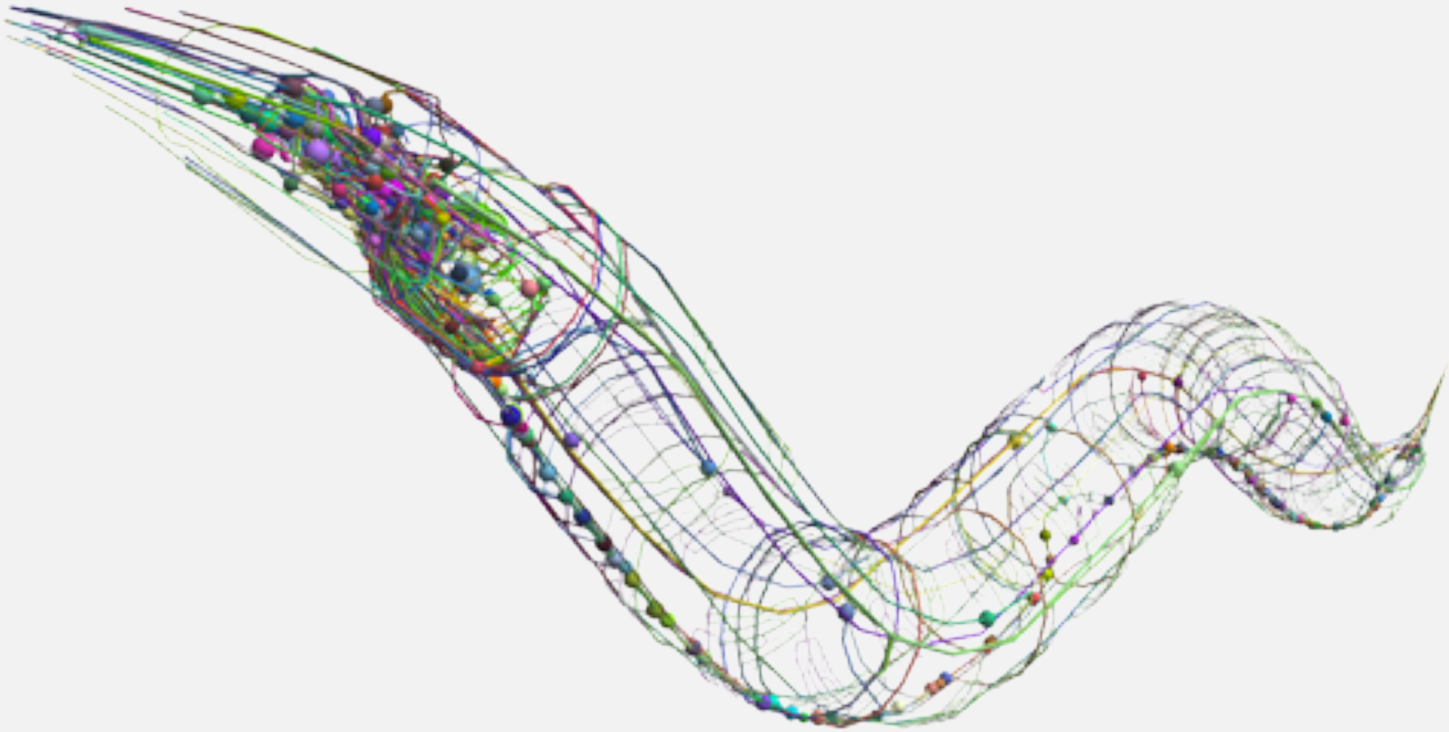
WHY GRAPHS?



Graphs are great for more **complex data structures**!

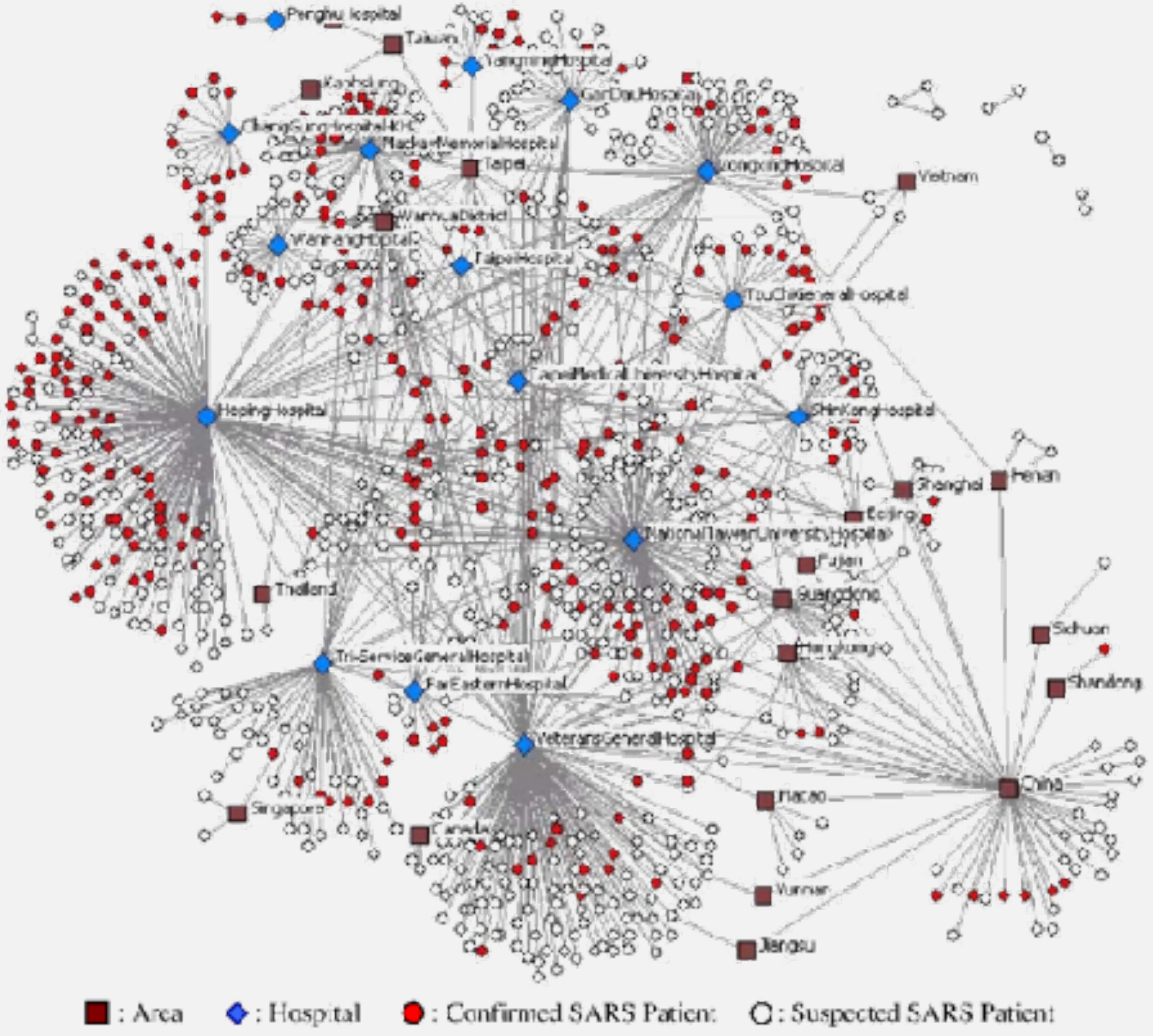
- Social networks
- Computational biology
- Long “etc.”

WHY GRAPHS?



The OpenWorm Project

WHY GRAPHS?

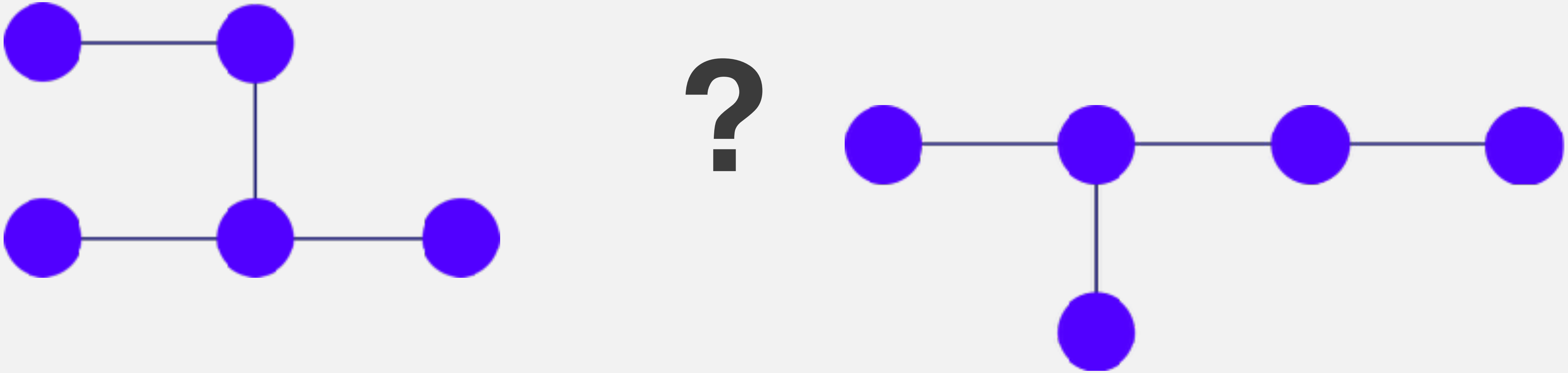


DOI: 10.1007/978-3-540-72608-1_3

FREQUENT GRAPH MINING

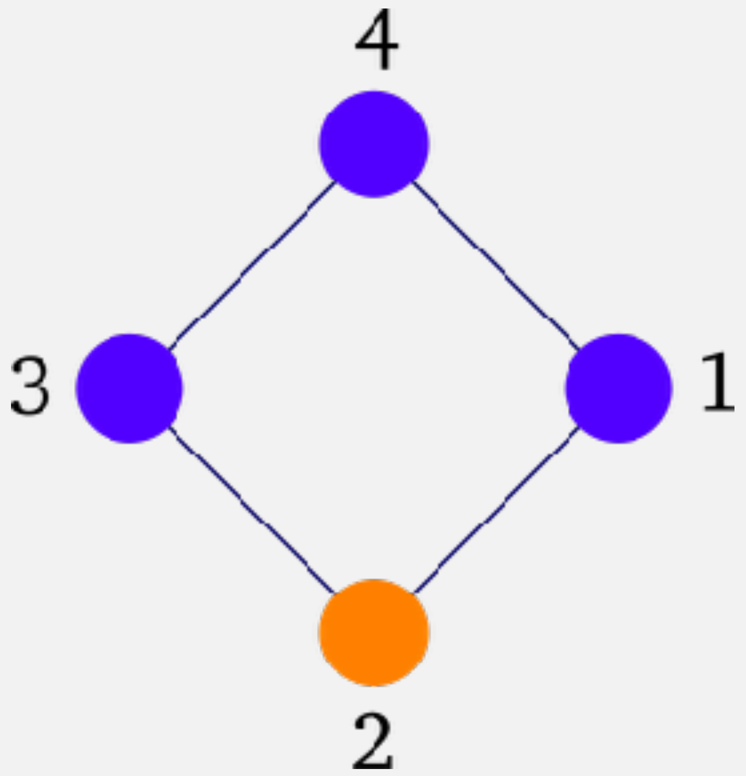
- Adaptation of Apriori ideas
- Key question: how do you compare two graphs?

GRAPH COMPARISON

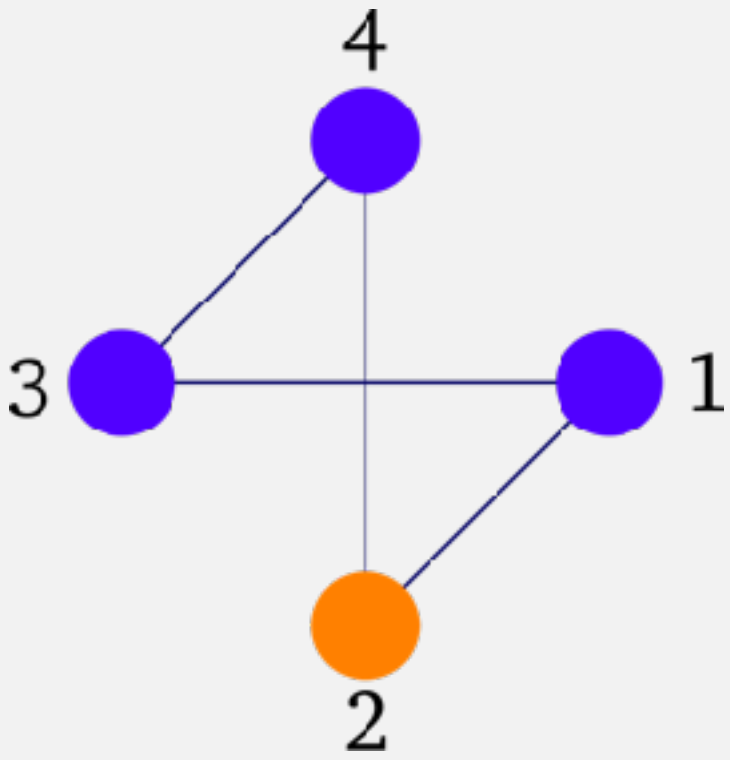


Problem: **ISOMORPHISM** (“equal shape”)

GRAPH ISOMORPHISM

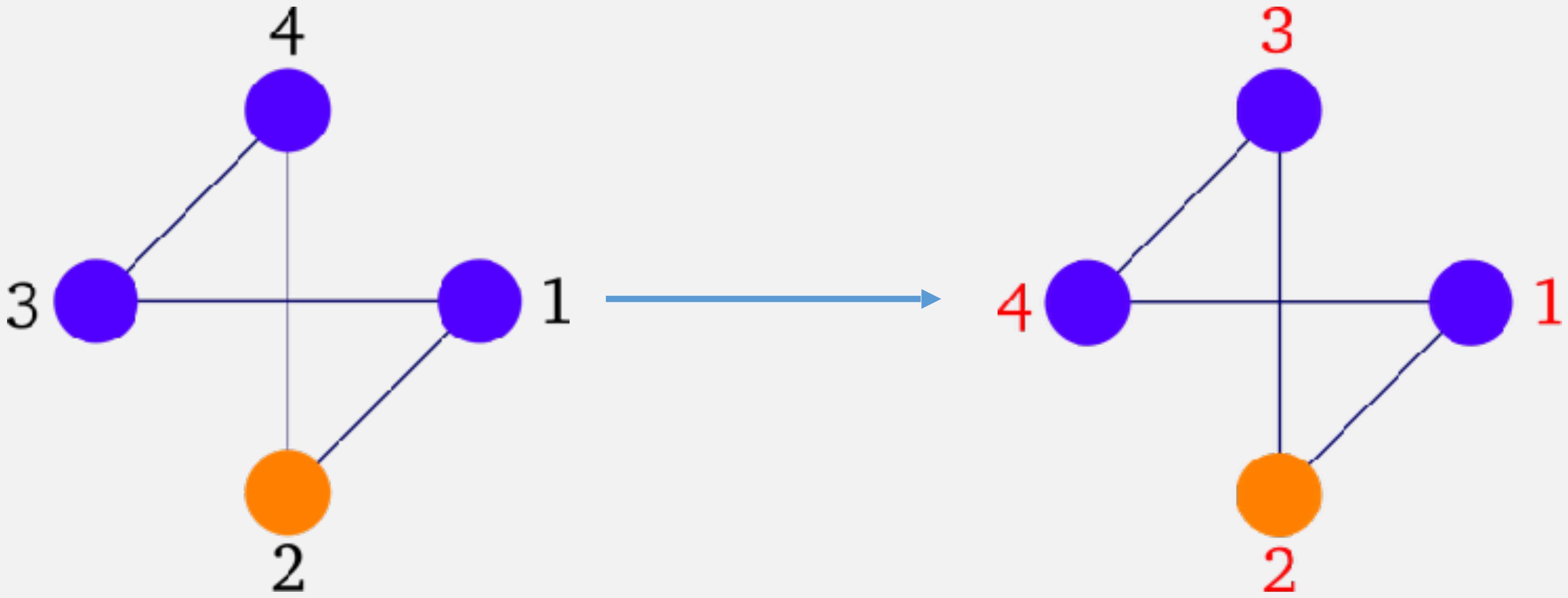


?



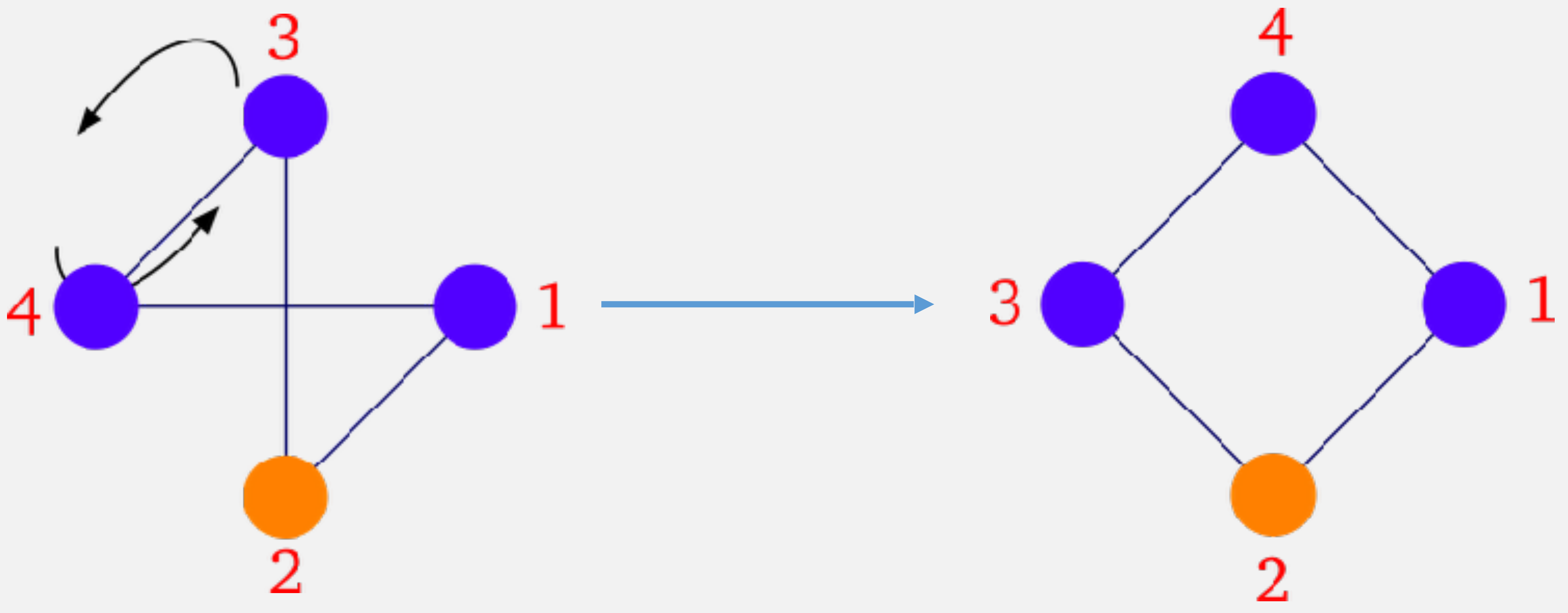
Node identifiers are arbitrary; are these graphs the same?

GRAPH ISOMORPHISM



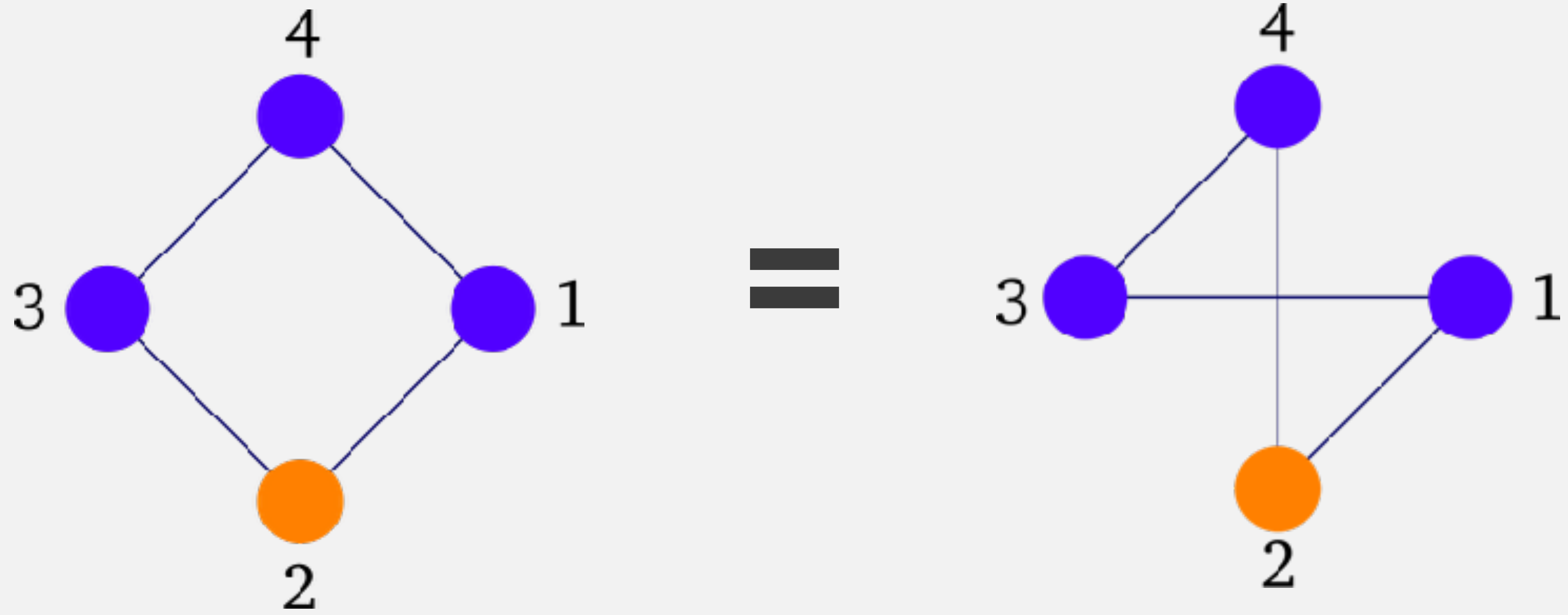
Node identifiers are arbitrary!

GRAPH ISOMORPHISM

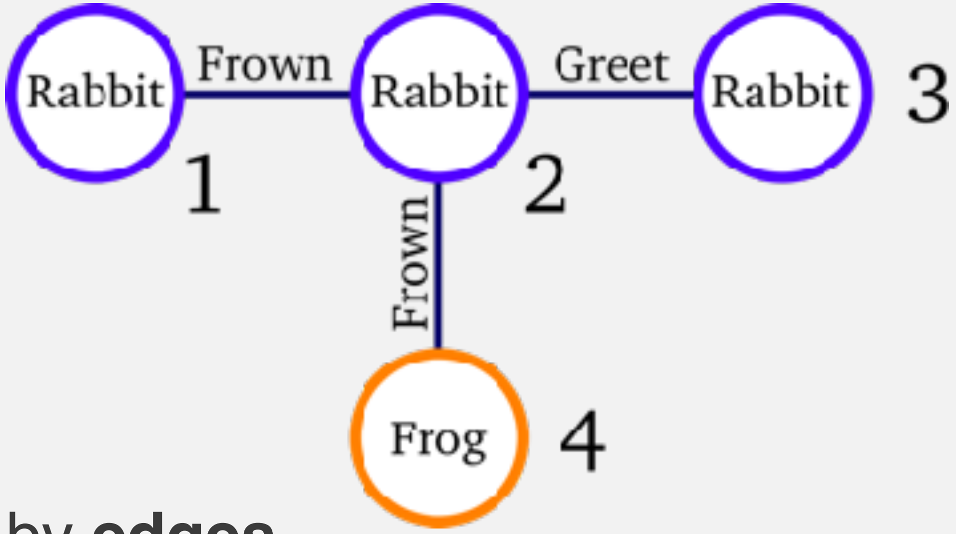


Position of nodes is not fixed! (Not in this case at least.)

GRAPH ISOMORPHISM



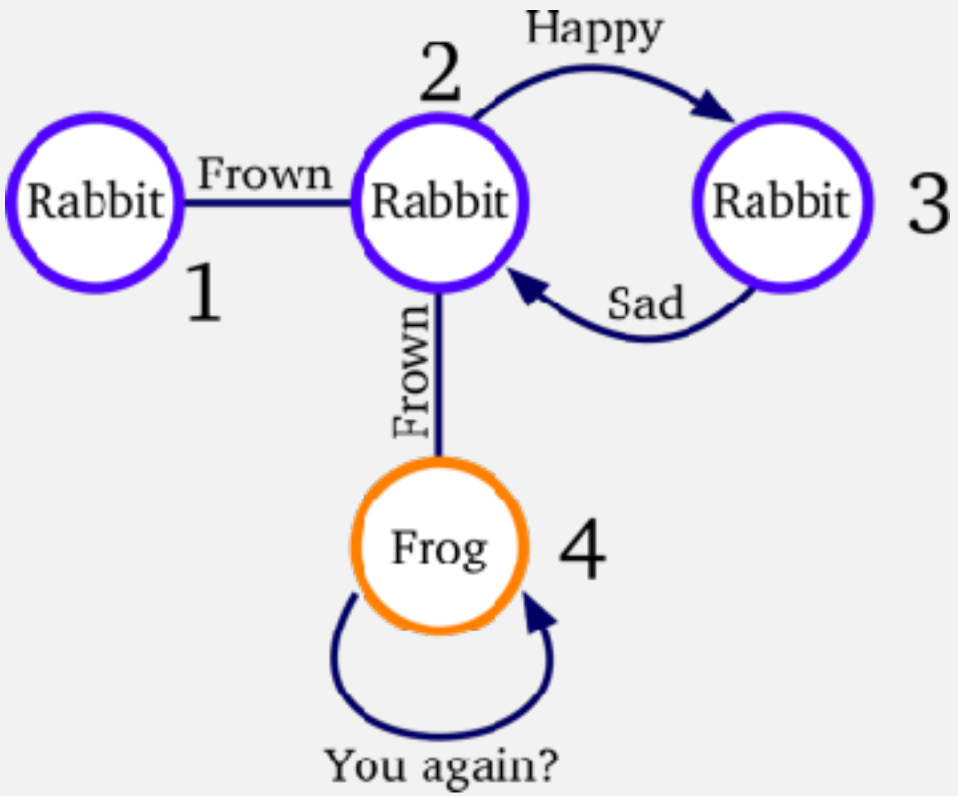
GRAPH REPRESENTATION



- **Graph**: set of **vertices** (nodes) connected by **edges**
- Vertices and edges have non-unique **labels**
- Vertices have unique but arbitrary **identifiers**

  Edges are defined by their label and the nodes they connect

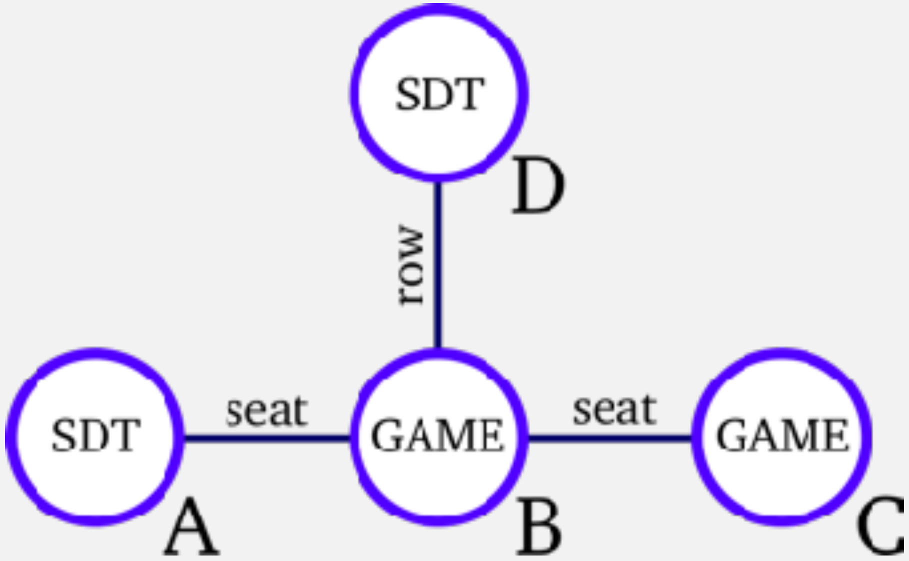
GRAPH REPRESENTATION



Note: We will work talk about simple, undirected graphs

GRAPH REPRESENTATION

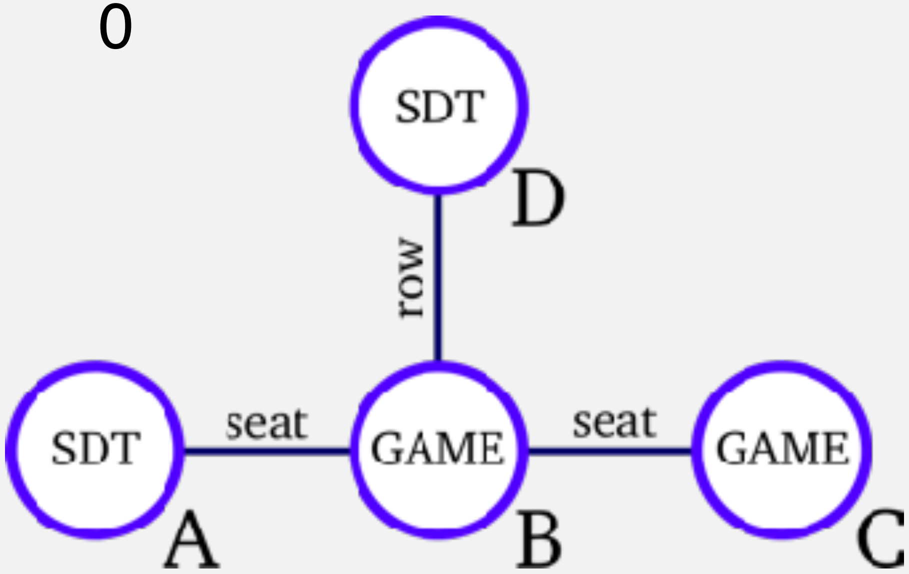
- The first step towards comparison is unique representation of isomorphisms
- Let's begin by representing graphs as a matrix



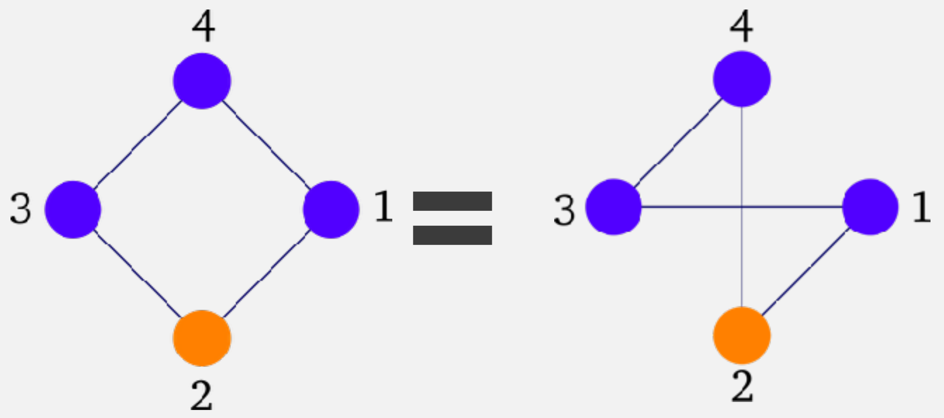
ADJACENCY MATRIX

	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

- Columns and rows for vertices
- Cells represent the edge between the row and column nodes
- Zero if no edge



CANONICAL LABEL

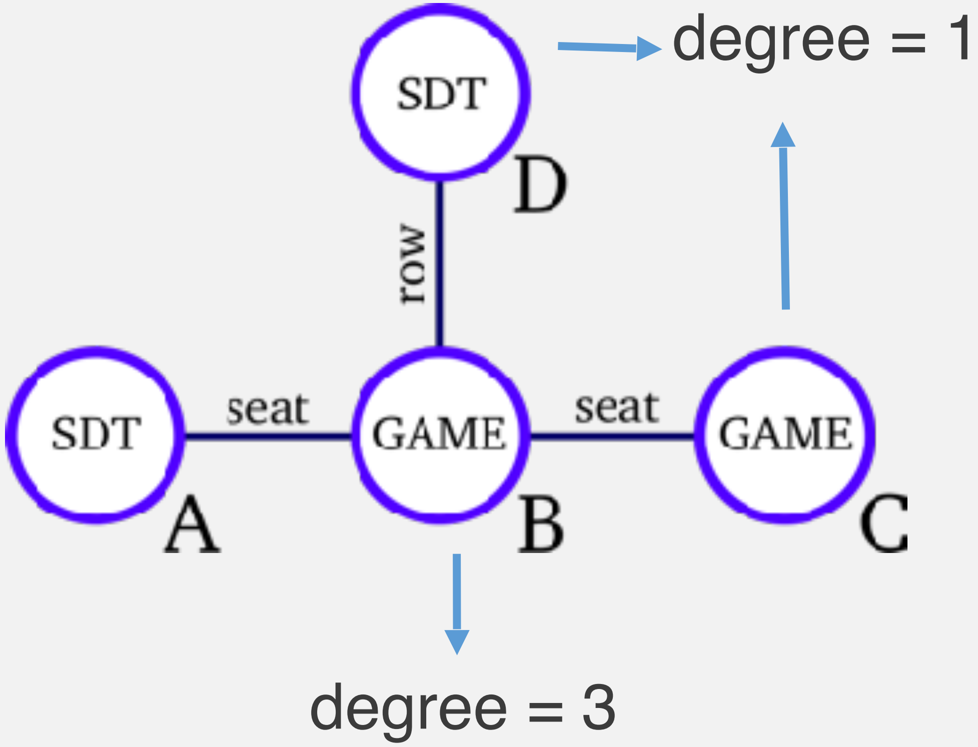


- Unique code that represents a graph and all its isomorphic graphs
- Flattening (processing) adjacency matrix
    1. Sort vertices by **degree** (number of edges)
    2. Sort vertices (of same degree) by label
    3. Compare edge permutations

CANONICAL LABEL

1. Sort vertices by **degree** (number of edges)

	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



CANONICAL LABEL

1. Sort vertices **by degree** (number of edges)

	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

CANONICAL LABEL

2. Sort vertices (of same degree) **by label**

	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

CANONICAL LABEL

3. Compare edge **permutations**

Permutation 1

	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

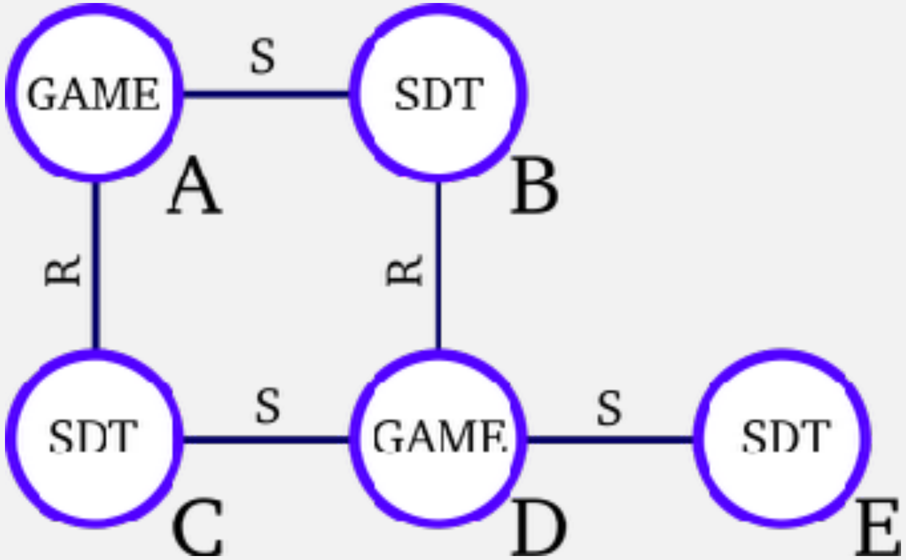
0; 00; seat, **seat**, row

Permutation 2

	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

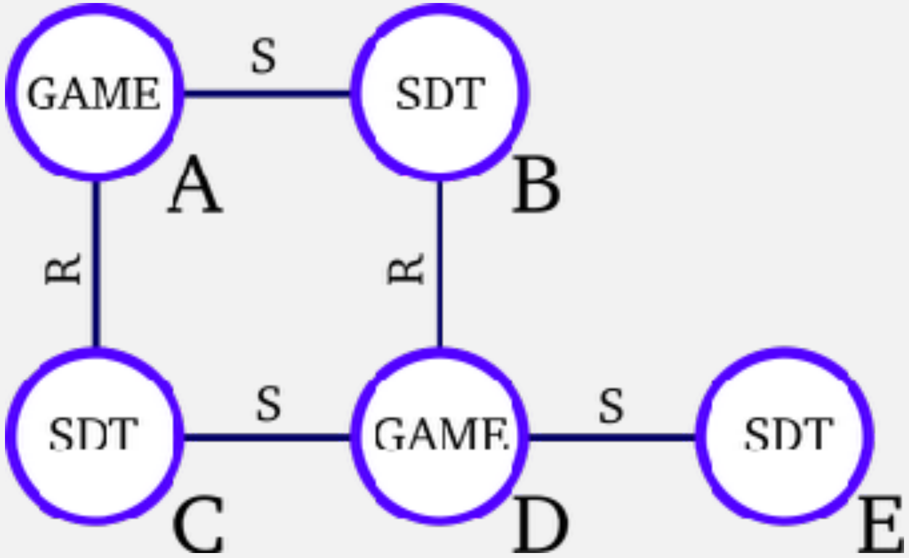
0; 00; seat, **row**, seat

CANONICAL LABEL—STEP BY STEP EXAMPLE



CANONICAL LABEL—STEP BY STEP EXAMPLE

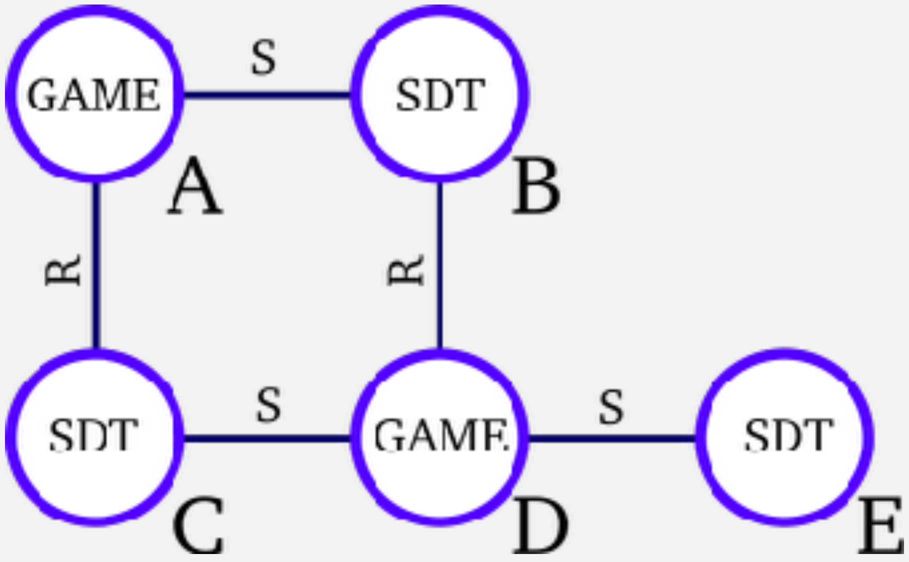
- We start with the adjacency matrix



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

CANONICAL LABEL—STEP BY STEP EXAMPLE

1. Sort vertices by **degree** (number of edges)



Node	Degree
A	2
B	2
C	2
D	3
E	1

STEP BY STEP EXAMPLE

1. Sort vertices by **degree** (number of edges)

Node	Degree
A	2
B	2
C	2
D	3
E	1

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

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

CANONICAL LABEL—STEP BY STEP EXAMPLE

2. Sort vertices (of same degree) by label

A = GAME, B = SDT, C = SDT

⇒ A before B, C

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

CANONICAL LABEL—STEP BY STEP EXAMPLE

3. Compare permutations

For each permutation we generate a **code** reading **below the diagonal**

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

CANONICAL LABEL—STEP BY STEP EXAMPLE

3. Compare permutations

For each permutation we generate a **code** reading **below the diagonal**

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

Label: 0

CANONICAL LABEL—STEP BY STEP EXAMPLE

3. Compare permutations

For each permutation we generate a **code** reading **below the diagonal**

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

Label: 0; 0S

CANONICAL LABEL—STEP BY STEP EXAMPLE

3. Compare permutations

For each permutation we generate a **code** reading **below the diagonal**

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

Label: 0; 0S; 0R0

CANONICAL LABEL—STEP BY STEP EXAMPLE

3. Compare permutations

For each permutation we generate a **code** reading **below the diagonal**

Label: **0; 0S; 0R0; S0RS**

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

CANONICAL LABEL—STEP BY STEP EXAMPLE

3. Compare permutations

- **Rules 1 and 2 have fixed the position of E, A and D**

  A = GAME > B = C = SDT

- Compare (E A **B C** D) and (E A **C B** D)

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

CANONICAL LABEL—STEP BY STEP EXAMPLE

Permutation (E A **B** **C** D)

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

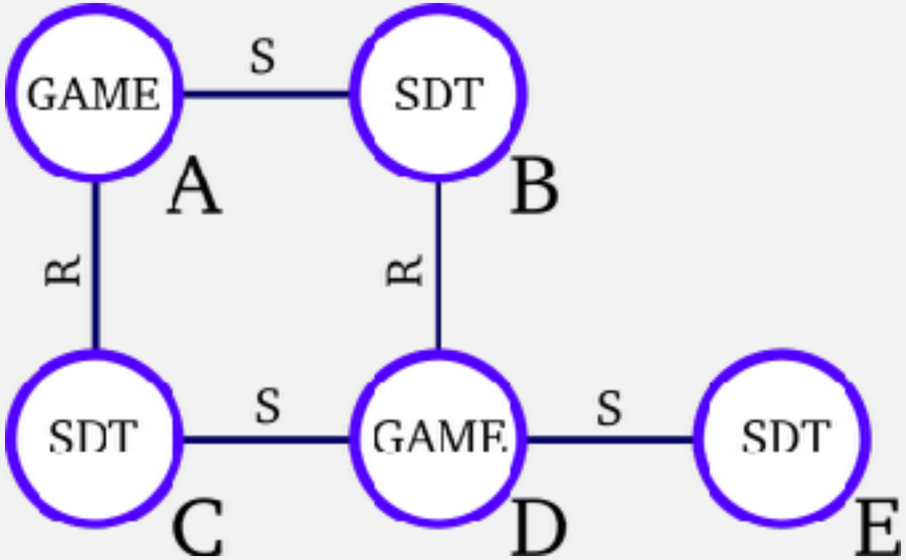
Label: 0; 0**S**; 0R0; S0RS

Permutation (E A **C** **B** D)

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

Label: 0; 0**R**; 0S0; S0SR

CANONICAL LABEL—STEP BY STEP EXAMPLE



Canonical label: 0; 0**R**; 0S0; S0SR

SUBGRAPHS

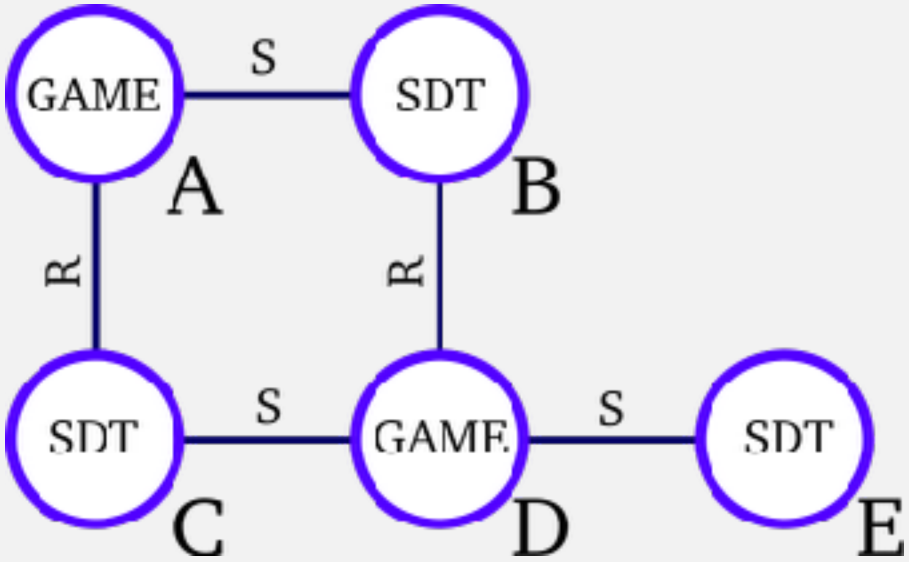
- G_1 is a subgraph of G_2 if we can find an **isomorphic graph of G_1 within G_2**
- If G_1 and G_2 are the same the operation is an **automorphism**
  (and returns all the isomorphisms of the graph!)
- A subgraph with k edges is called a k-subgraph
  (but some methods count nodes!)

SUBGRAPHS

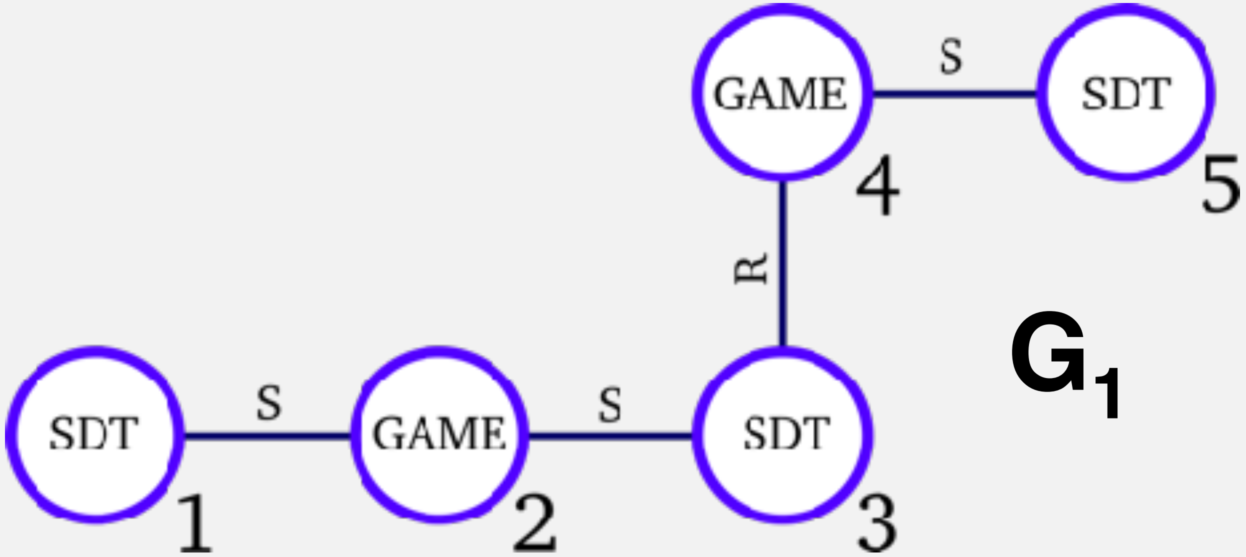
Testing if a graph is a subgraph of another is a **costly operation**!

SUBGRAPHS

Is G_1 a subgraph of G_2?



G_2

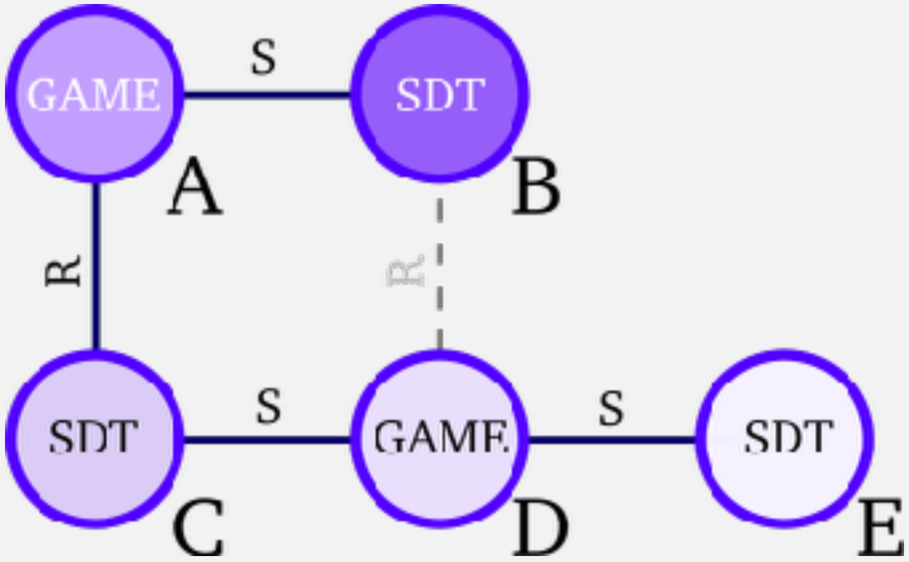


G_1

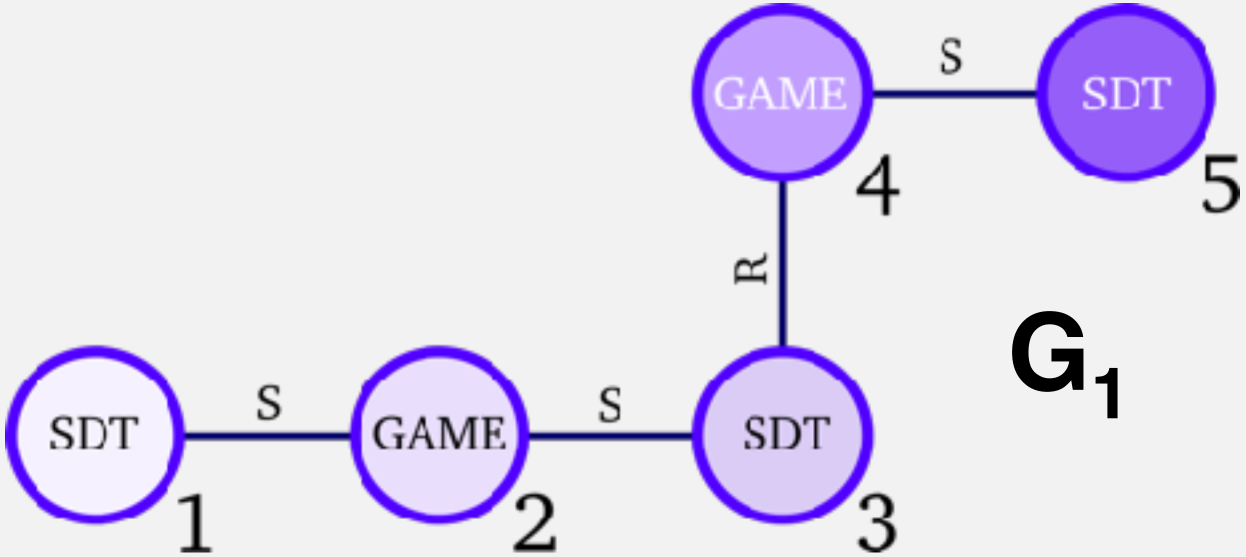
SUBGRAPHS

Is G_1 a subgraph of G_2?

yes!



G_2



G_1

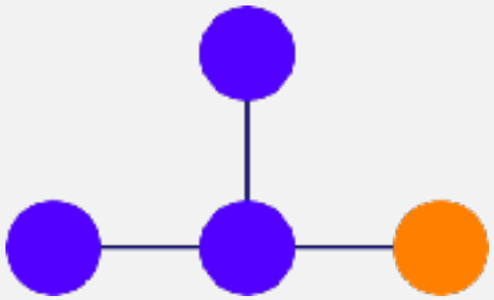
APRIORI FOR GRAPHS

APRIORI FOR GRAPHS

- Adapting Apriori for graphs is not trivial
- (k+1)-candidates: increase one edge or one vertex?
- Adding two graphs does not create unique results

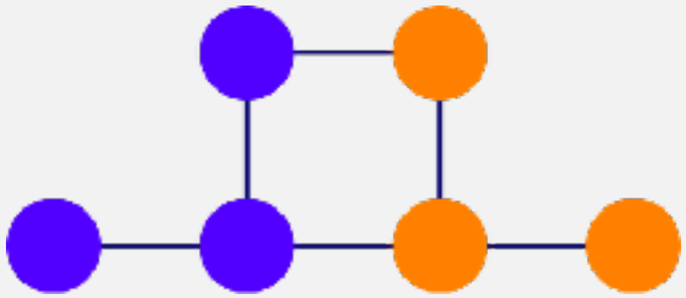
DATASET

S 1



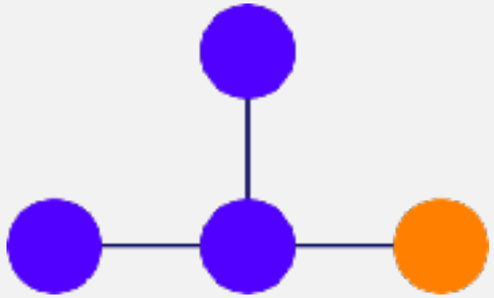
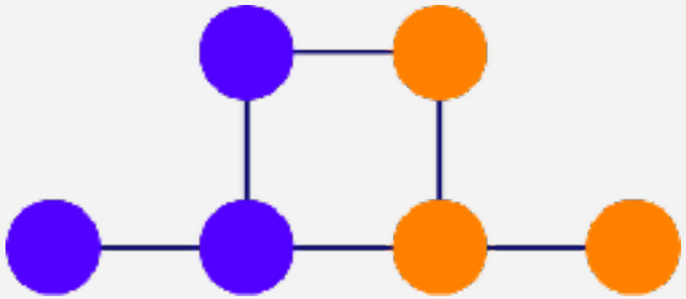
S 2

S 3



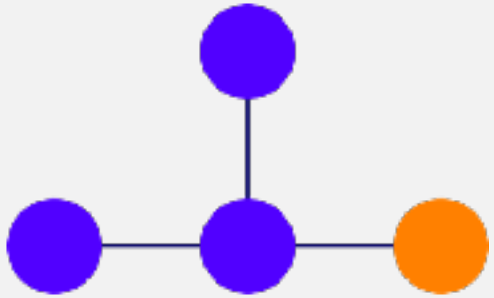
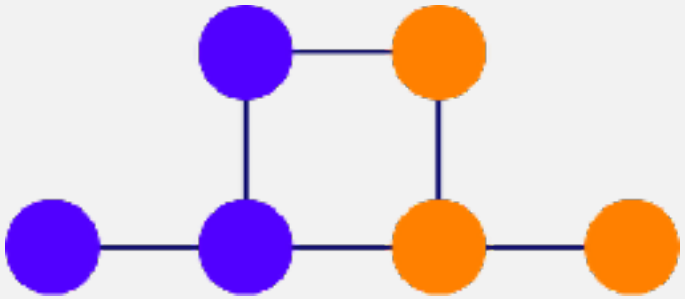
- Each sample is a graph
- Independent samples (social network of different users) or from a larger graph (distance-2 social networks starting at different nodes)

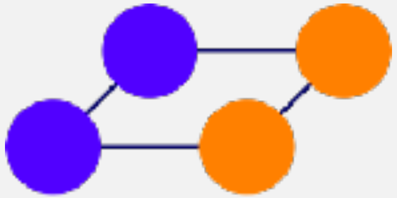
SUPPORT COUNT

S 1	
S 2	
S 3	

- A graph is supported by a sample if it contains one of its subgraphs
- A sample supports at most once each graph

SUPPORT COUNT

S 1	
S 2	
S 3	

Graph	**Support**
	**2**
	**2**
	**3**
	**1**

FSG ALGORITHM

1. Generate (k+1)-candidates adding a node
   AGM adds one vertex instead!
   - Join
   - Prune
2. Count support of candidates
3. Drop non-frequent candidate graphs

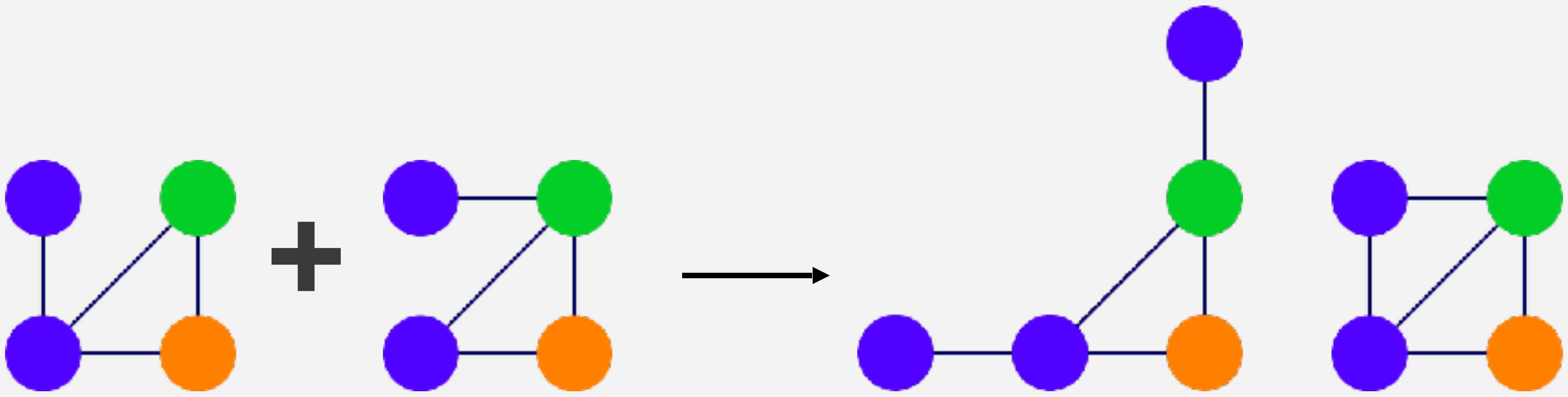
FSG—JOIN

- We add G_1 and G_2 if removing an edge from G_1 results in a subgraph in G_2
- This subgraph is called a core. G_1 and G_2 may share more than one!
- We add the removed edge from G_1 to G_2

FSG—JOIN

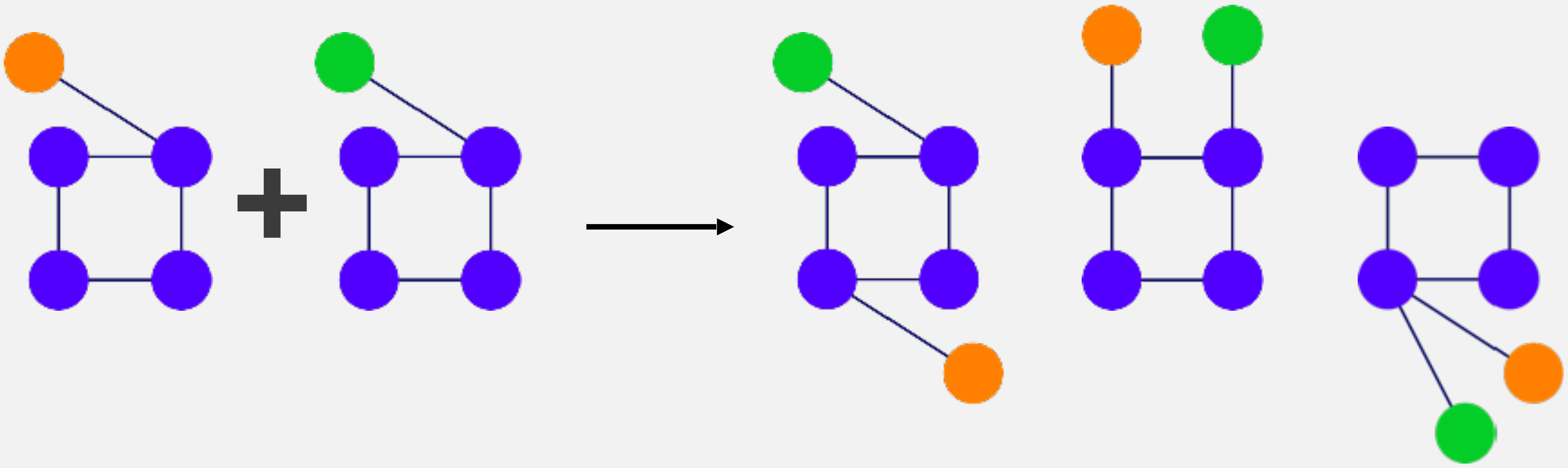
Both candidates have one more vertex than the core

If these vertices have the same label, we add two candidates



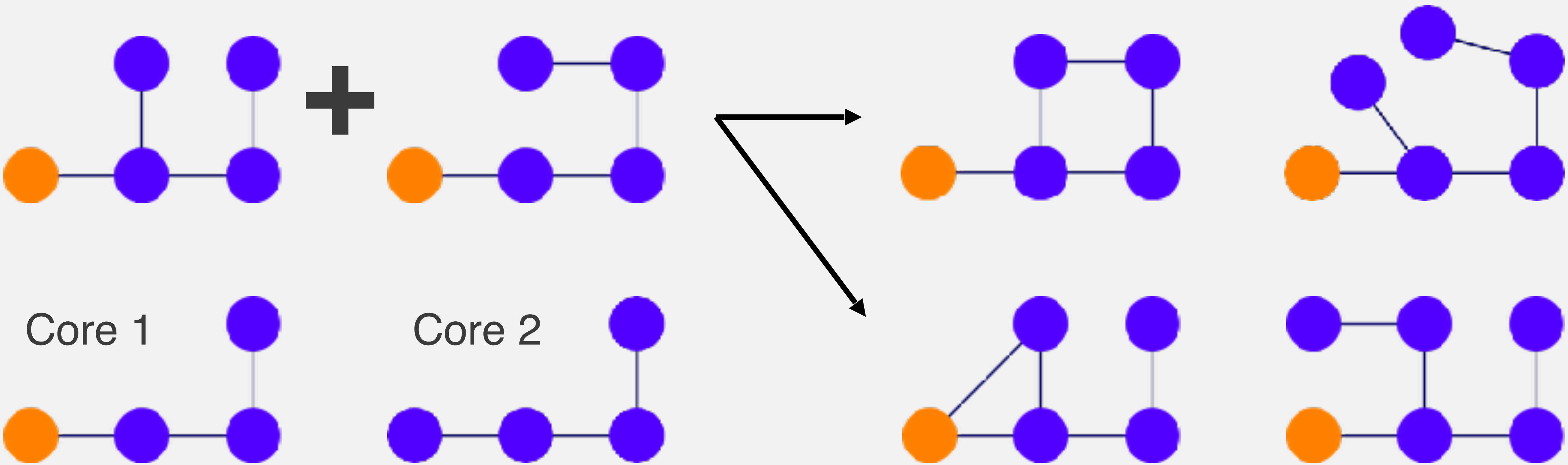
FSG—JOIN

We have to consider all automorphisms (versions) of the core!



FSG—JOIN

We have to consider all cores!



FSG—PRUNE

- A priori property: a graph may only be frequent if all its subgraphs are!
- After pruning we need to scan to count the support of the candidates
- As always, there are possible alternatives, such as vertical data format

APRIORI—SUMMARY

Most data mining algorithms may be adapted to different problems!

- GSP: Apriory + sequences
    - Support count: time and order constraints
    - Candidate generation: contiguous sequences
- FSM: Apriori + graphs
    - Support count: isomorphisms
    - Candidate generation: core of two graphs

THANKS FOR LISTENING!