Large-Scale Graph Mining

Vincent Leroy

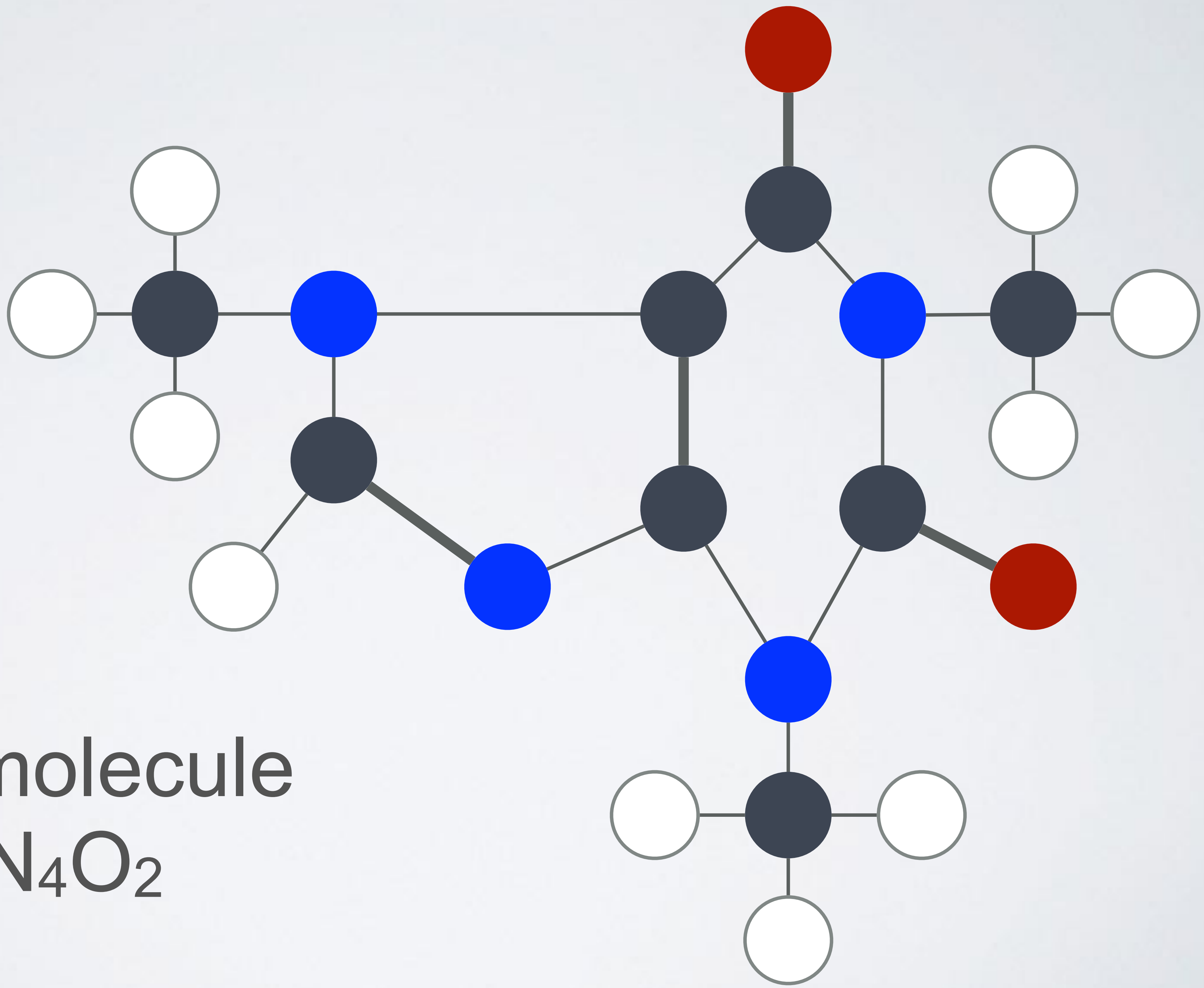
FREQUENT SUBGRAPH MINING (FSM)

- Graphs represent complex data
  - Chemical compounds, proteins
  - Social networks
  - Knowledge bases (ontologies)
- Frequent Subgraph Mining
  - Discover regularities in the structure of a graph
    - Properties and interactions (citations graph, organization structure)
    - Privacy (social networks)
    - Link prediction (recommender systems, linked data)

10M entities, 120M facts

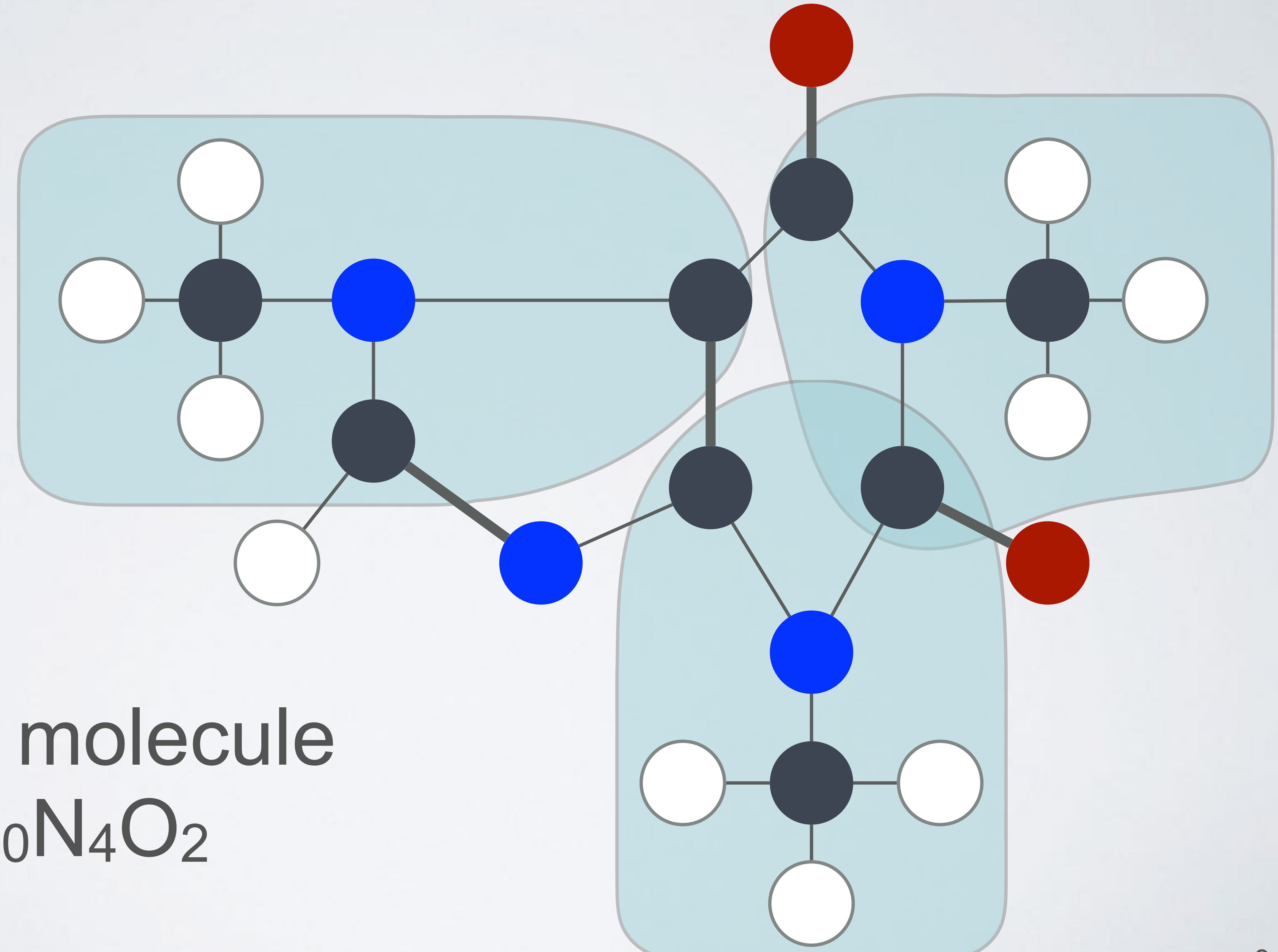
570M entities, 18B facts (2012)

FSM: CHEMISTRY



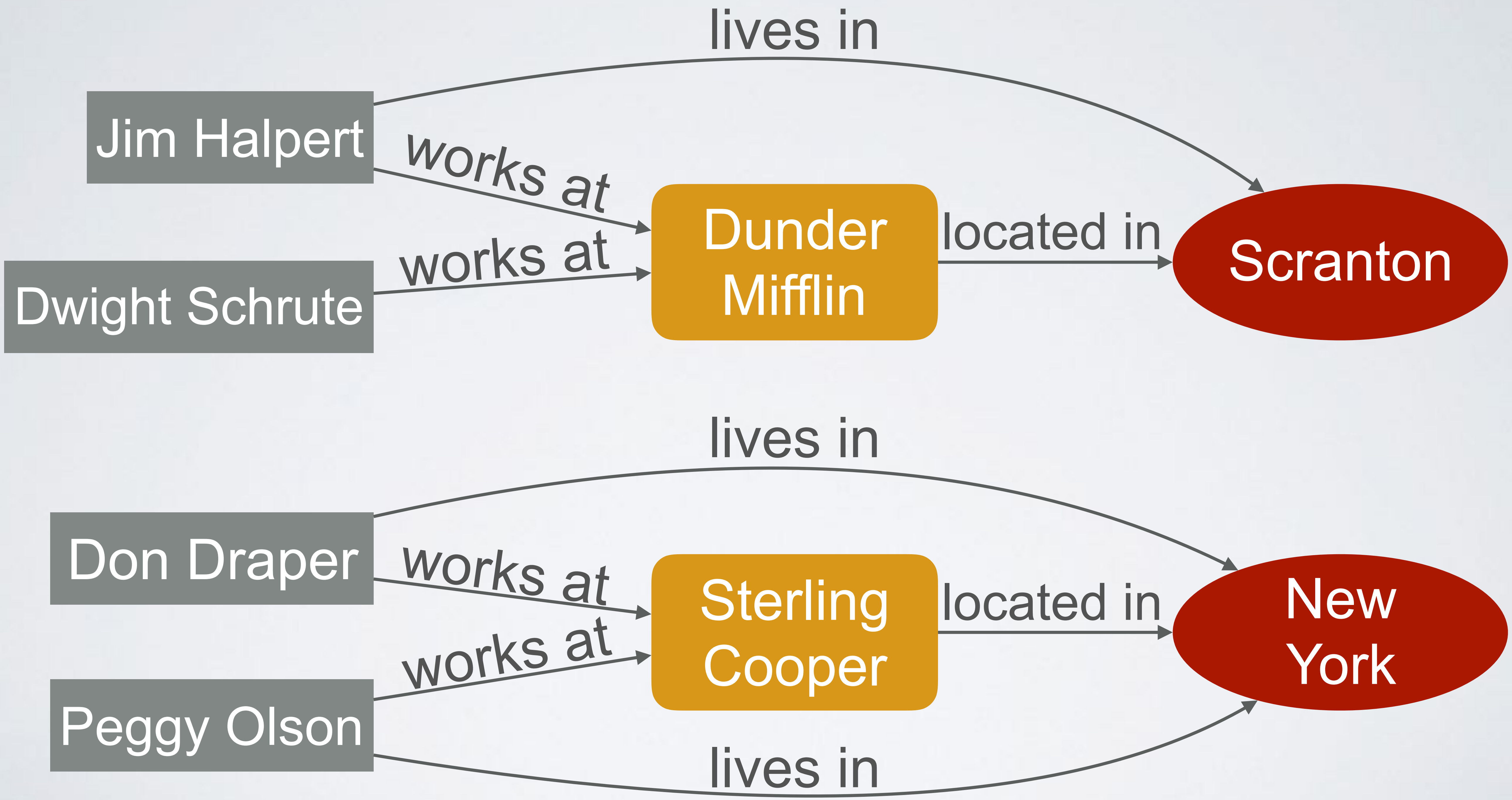
Caffeine molecule
$C_8H_{10}N_4O_2$

FSM: CHEMISTRY

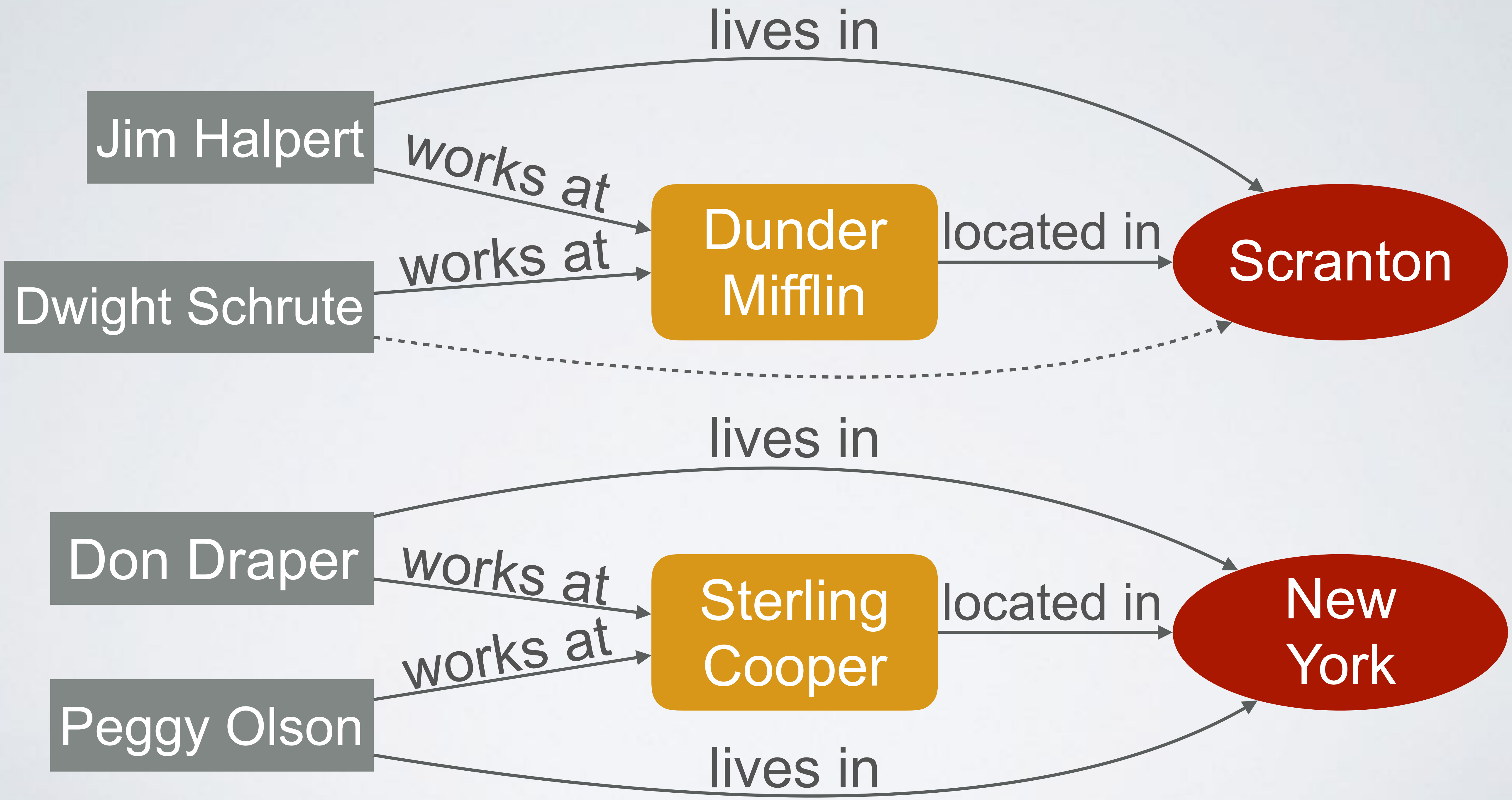


Caffeine molecule
$C_8H_{10}N_4O_2$

FSM: KNOWLEDGE BASE

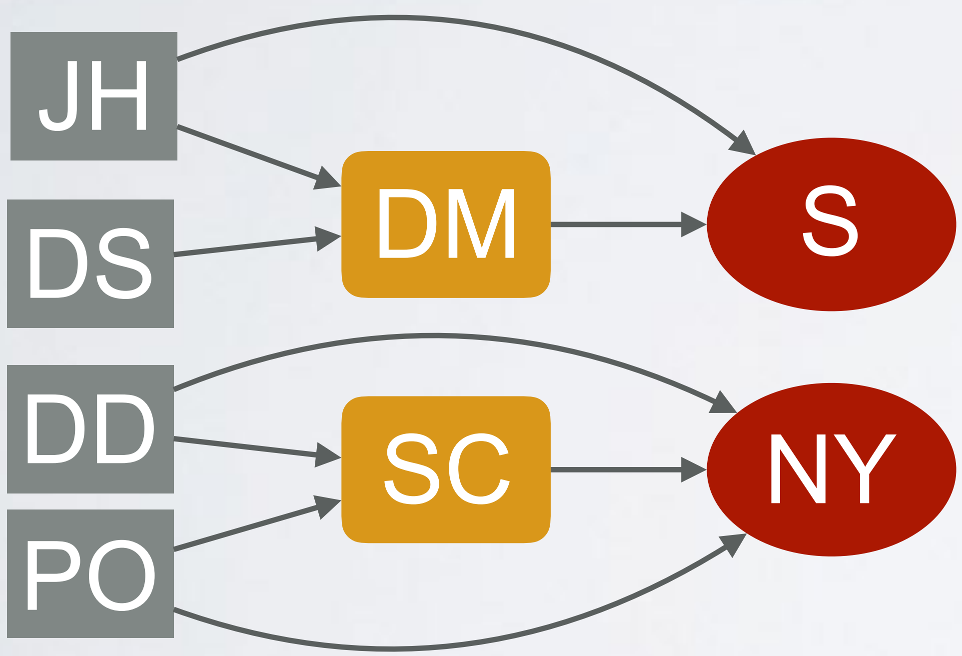


FSM: KNOWLEDGE BASE

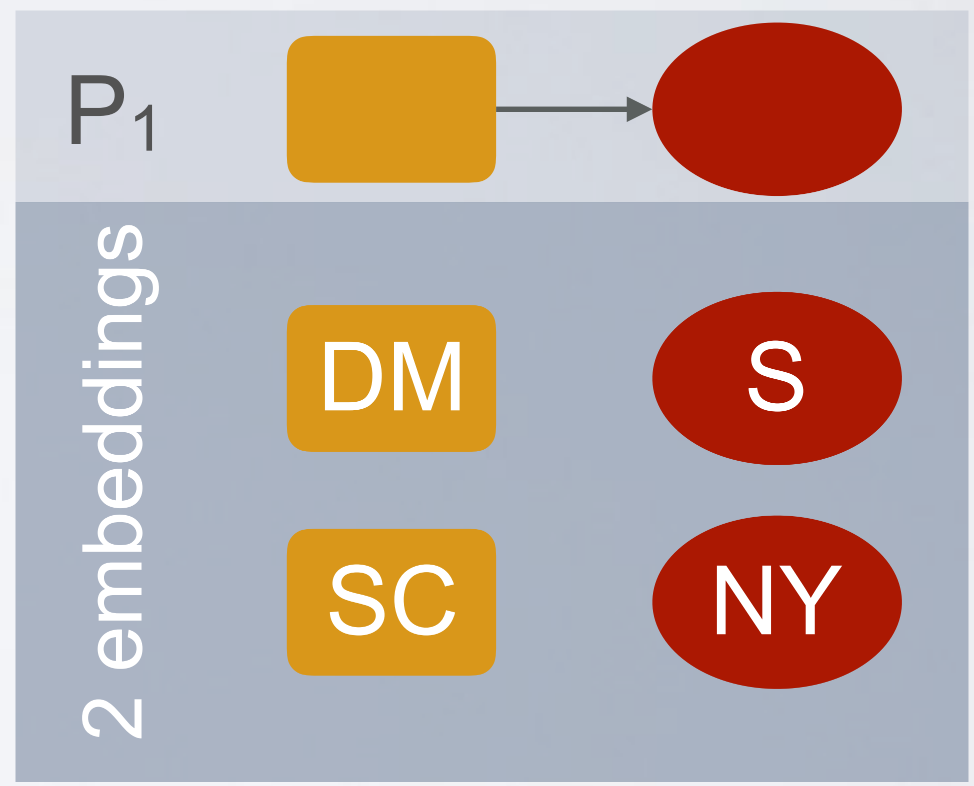


FSM: DEFINITION

- Find all frequent (*support* ≥ ε) subgraphs



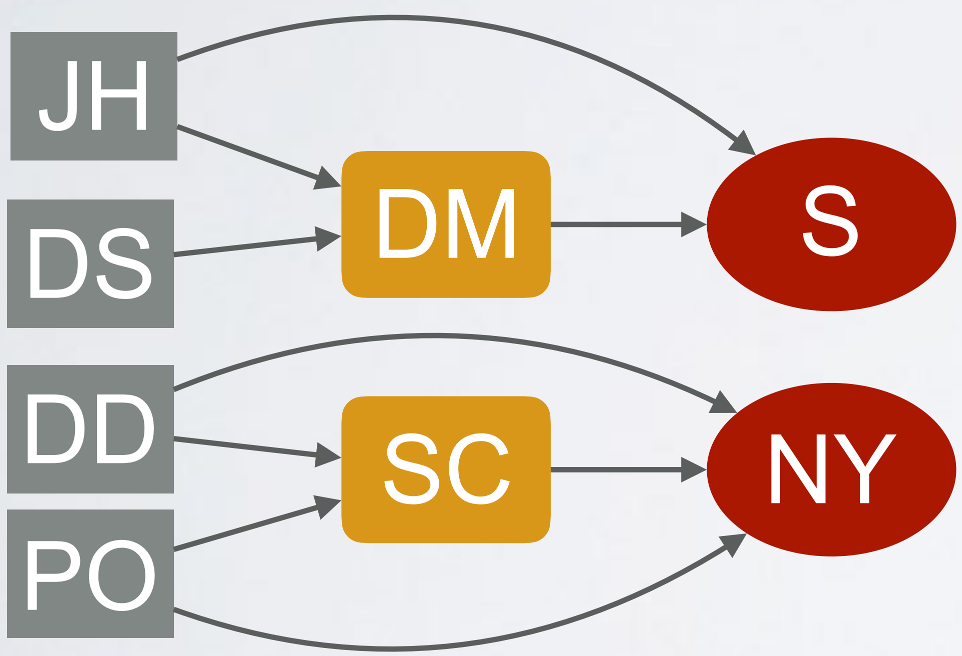
Input Graph

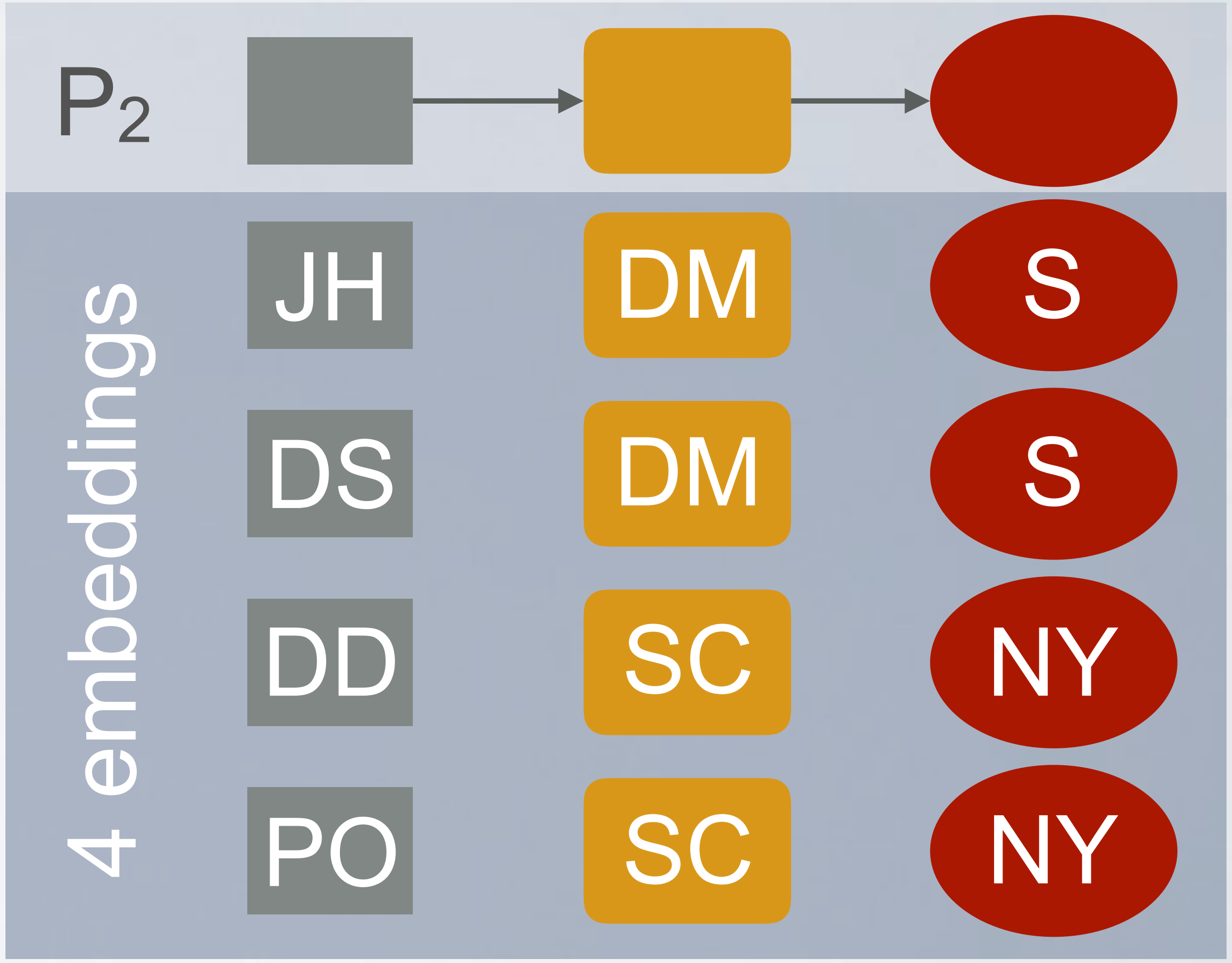


Support(P_1) = 2

SUPPORT DEFINITION

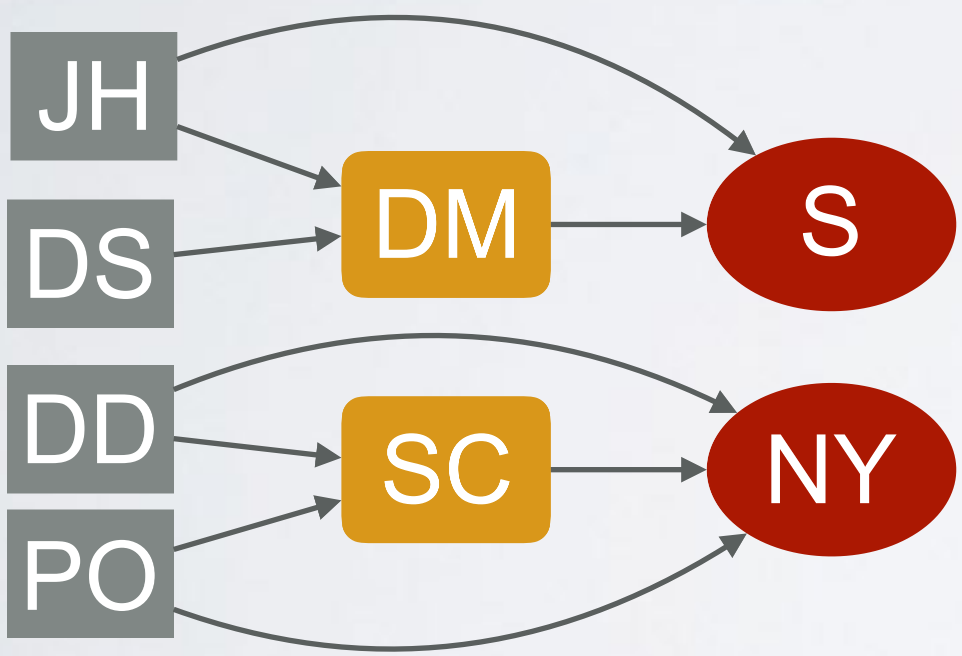
- Find all frequent (*support* ≥ ε) subgraphs

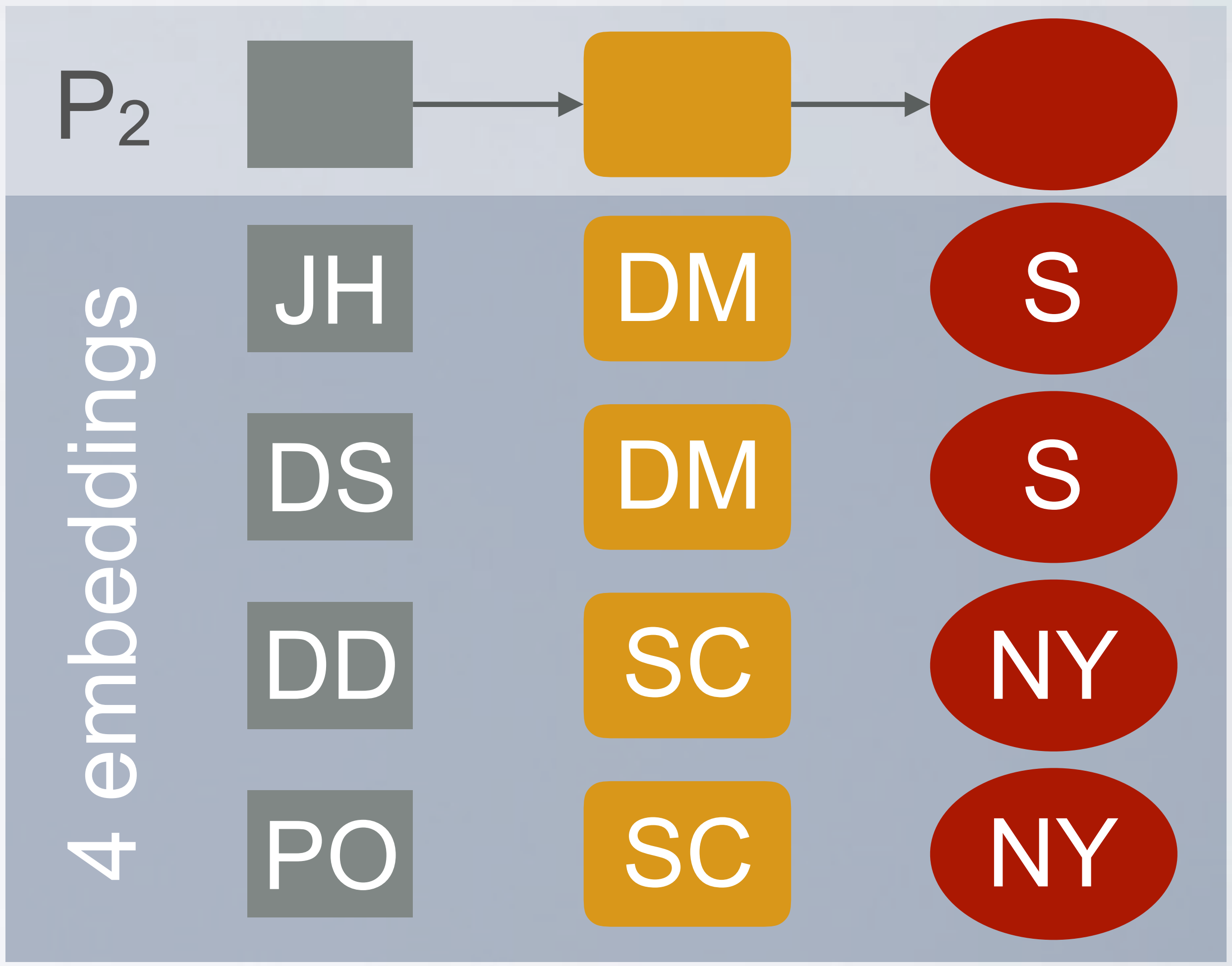




SUPPORT DEFINITION

- Find all frequent (*support* ≥ ε) subgraphs



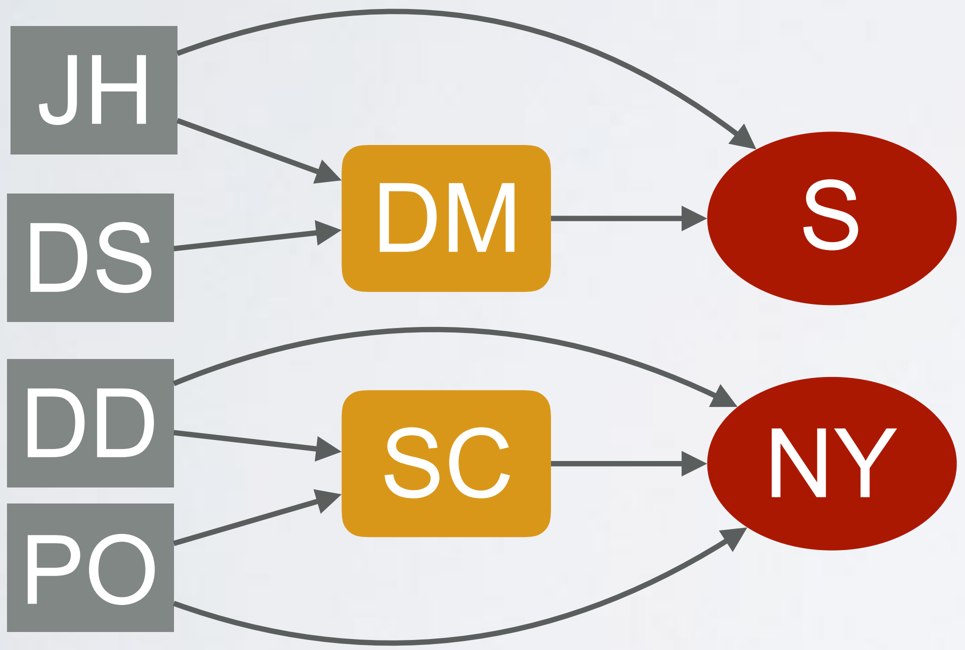


Minimum Image Support: min(#mappings)

Anti-monotony

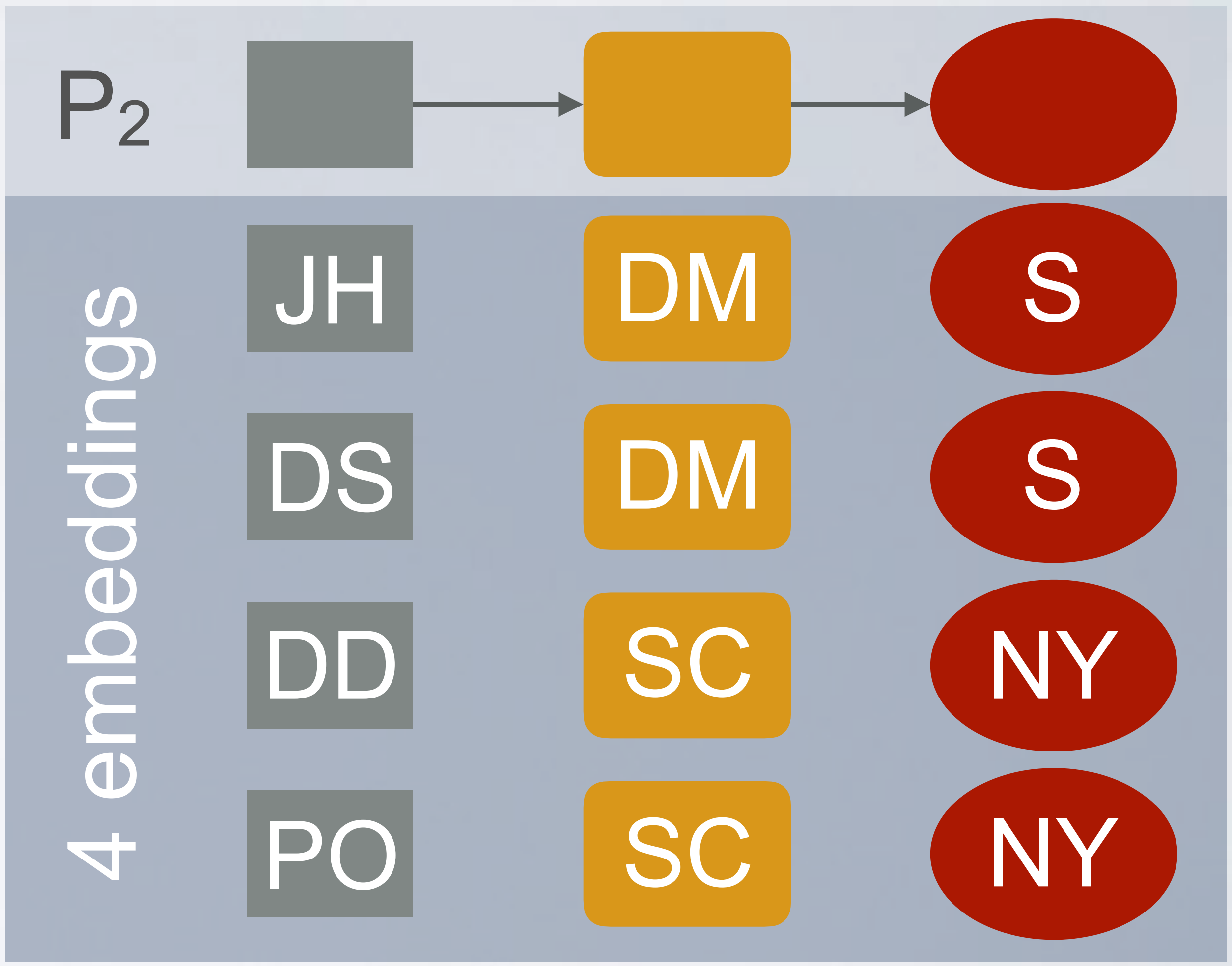
SUPPORT DEFINITION

- Find all frequent (*support* ≥ ε) subgraphs



Minimum Image Support: min(#mappings)

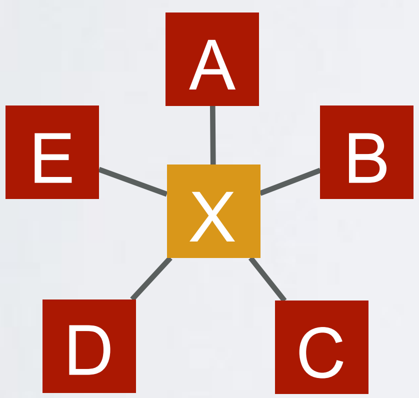
Anti-monotony



$\text{Support}(P_2) = 2$

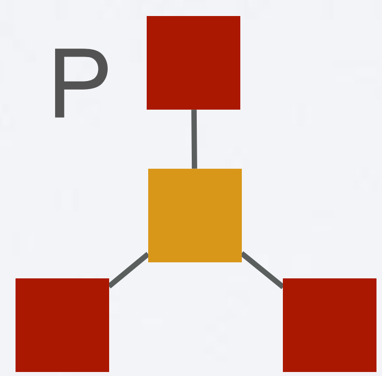
CHALLENGE

- Computing the support requires keeping track of embeddings



Input graph

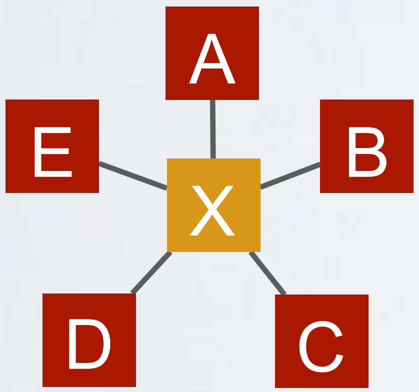
P



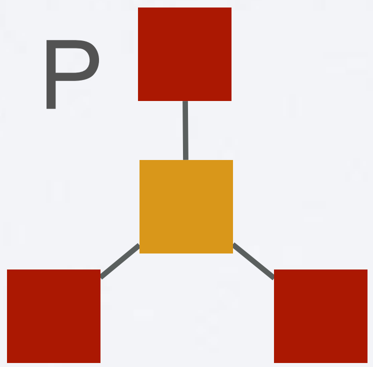
Pattern identified

CHALLENGE

- Computing the support requires keeping track of embeddings
  - Up to *factorial*(V) embeddings for a single pattern due to symmetry (ex: 10! > 3M)
  - Mining larger patterns and dealing with high-degree vertices is costly



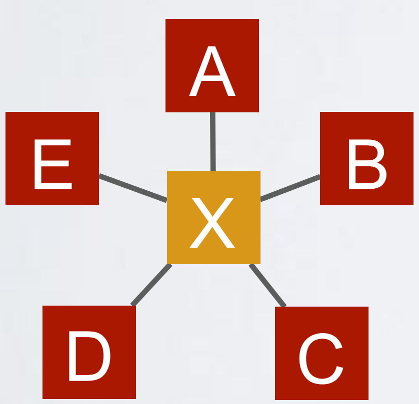
Input graph



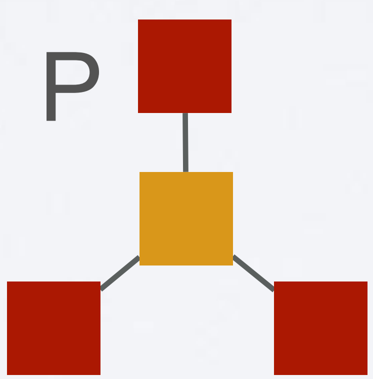
Pattern identified

CHALLENGE

- Computing the support requires keeping track of embeddings
  - Up to *factorial*(V) embeddings for a single pattern due to symmetry (ex: 10! > 3M)
  - Mining larger patterns and dealing with high-degree vertices is costly



Input graph



Pattern identified

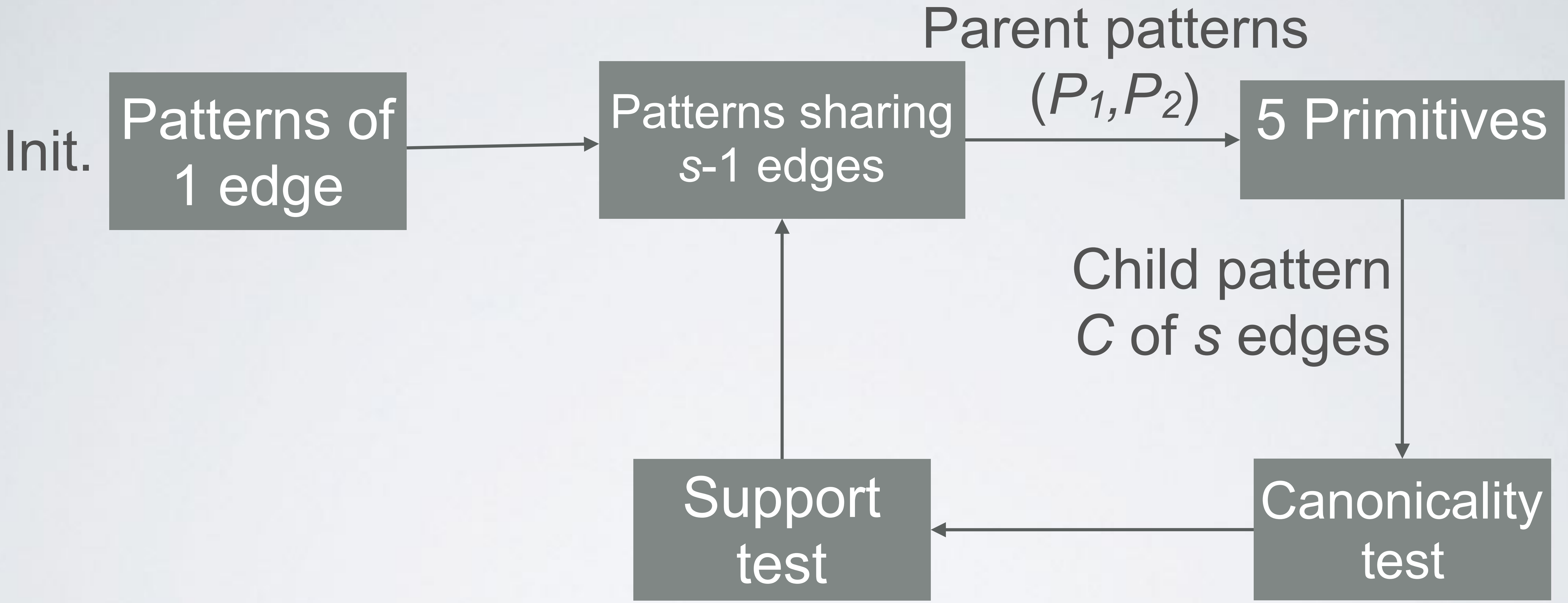
STATE OF THE ART

- *Arabesque* [SOSP 2015]
  Use more resources: parallel and distributed computation
- *ScaleMine* [SC 2016]
  Simplify the problem: compute a minimal set of embeddings to reach the support threshold ε
  - Lose accurate information on support which is important for many applications

CONTRIBUTIONS

- Address the core algorithmic and data structure problem of FSM with a new algorithm: SAMi
    - Define 5 primitive operations to recursively enumerate patterns
    - Propose a compressed representation of embeddings that circumvents the cost of enumerating embeddings

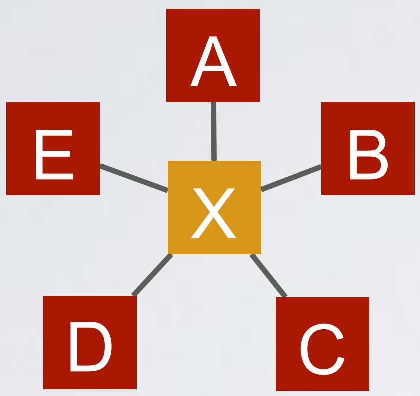
OVERVIEW OF SAMi



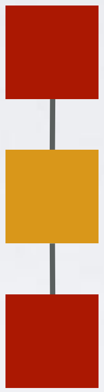
EMBEDDINGS REPRESENTATION

INTUITION

G



P



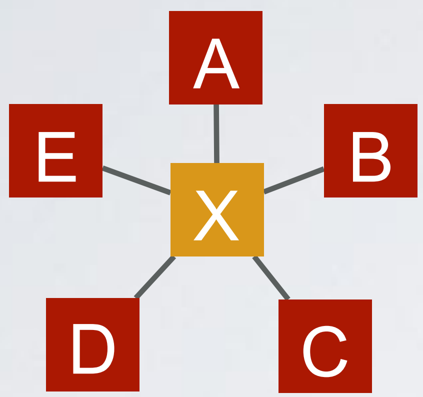
A	X	B
A	X	C
A	X	D
A	X	E
B	X	A

…

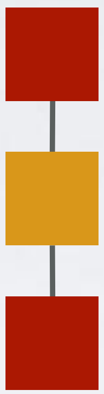
There must be a more compact way to express this

INTUITION

G



P



A	X	B
A	X	C
A	X	D
A	X	E
B	X	A

…

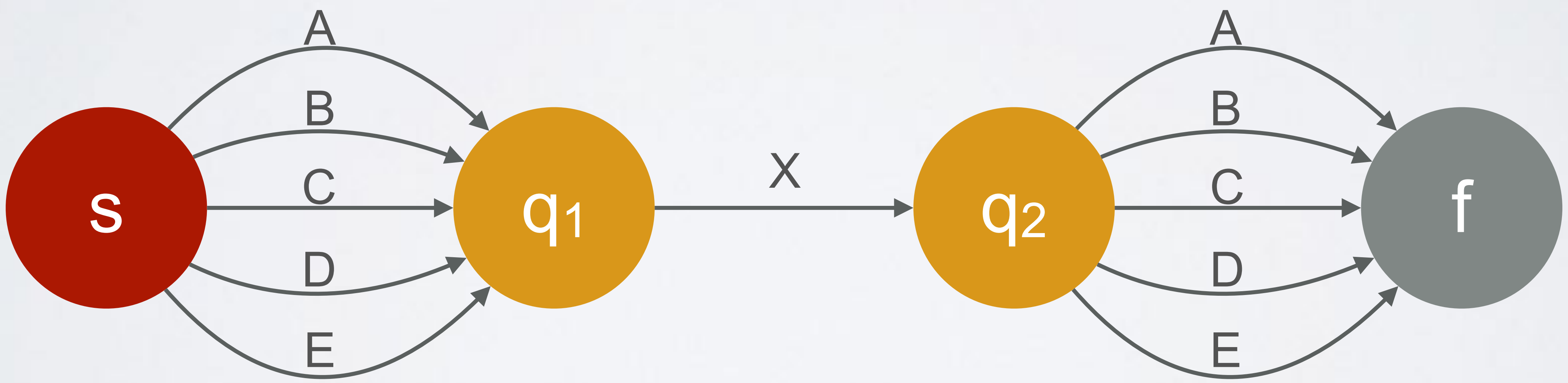
There must be a more compact way to express this

One of A B C D E Then X Then one of A B C D E

*No duplicates

AUTOMATON REPRESENTATION OF EMBEDDINGS

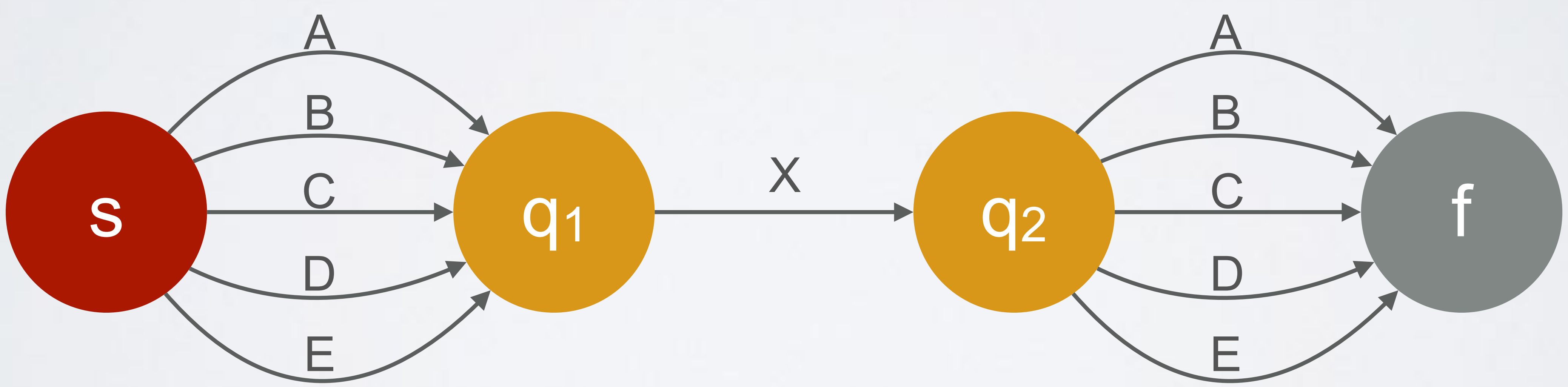
- Deterministic finite automaton
  - Alphabet: all vertex identifiers from the input graph
  - Words accepted: embeddings of the pattern*



*No duplicates

AUTOMATON REPRESENTATION OF EMBEDDINGS

- Deterministic finite automaton
  - Alphabet: all vertex identifiers from the input graph
  - Words accepted: embeddings of the pattern*



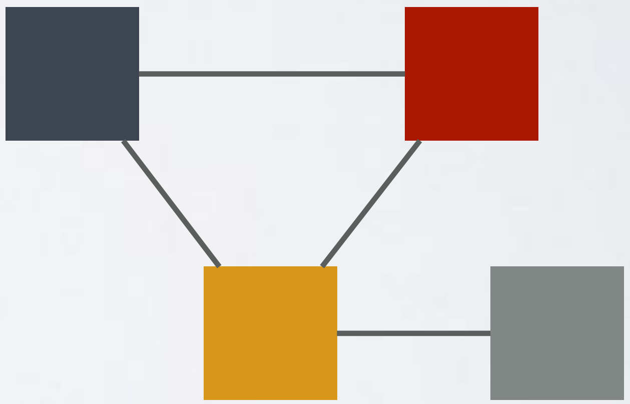
*No duplicates

We save memory, can we do more?

PATTERN GENERATION

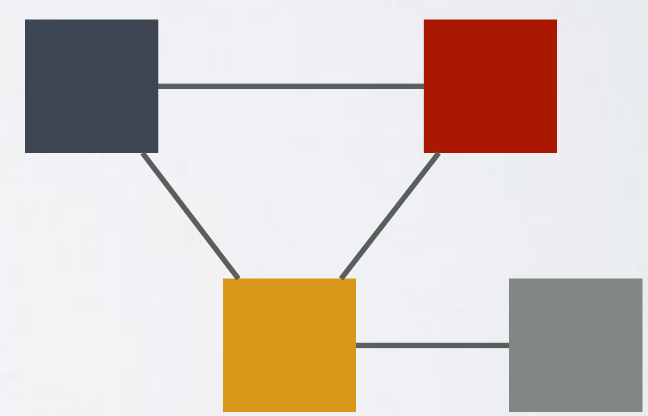
PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



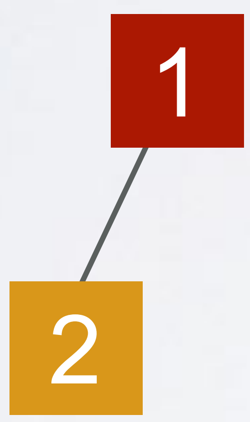
PATTERN REPRESENTATION

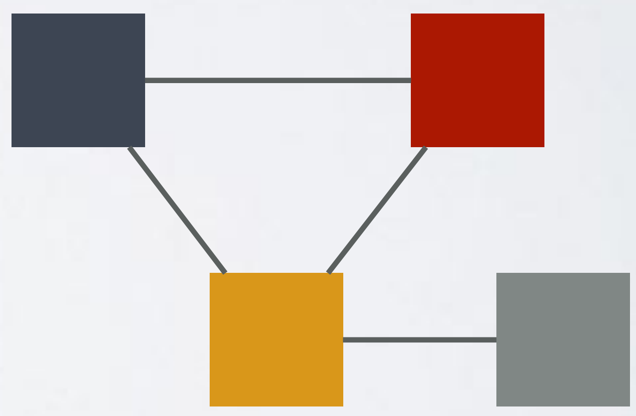
- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



PATTERN REPRESENTATION

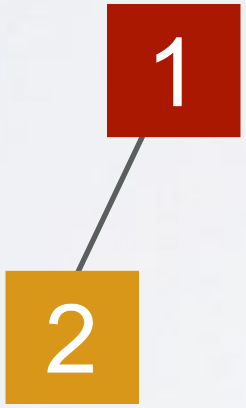
- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



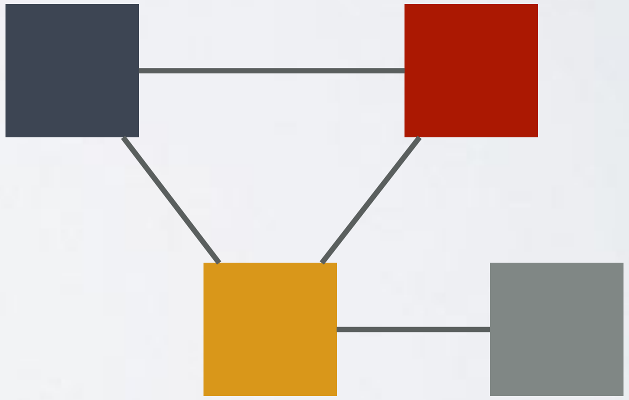


PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



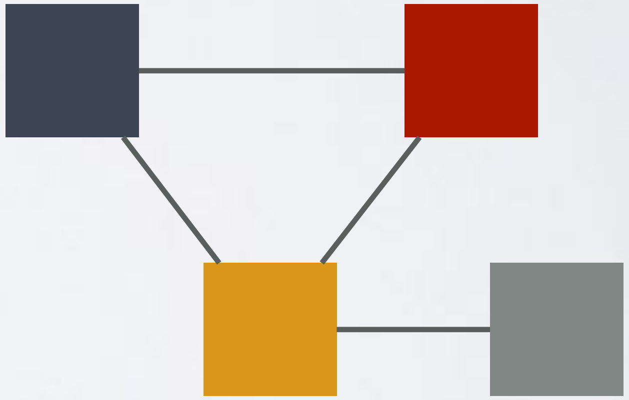
(1,2)



PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable

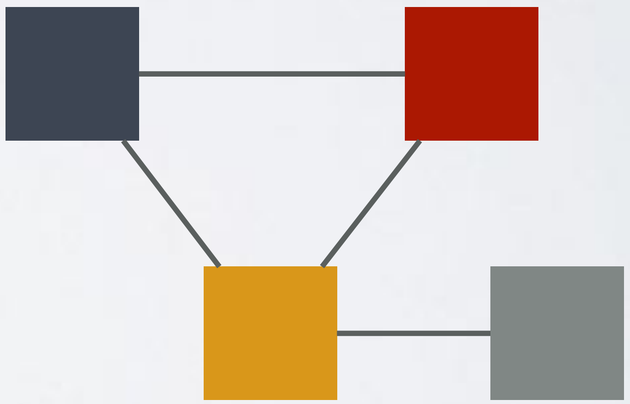
(1,2)



PATTERN REPRESENTATION

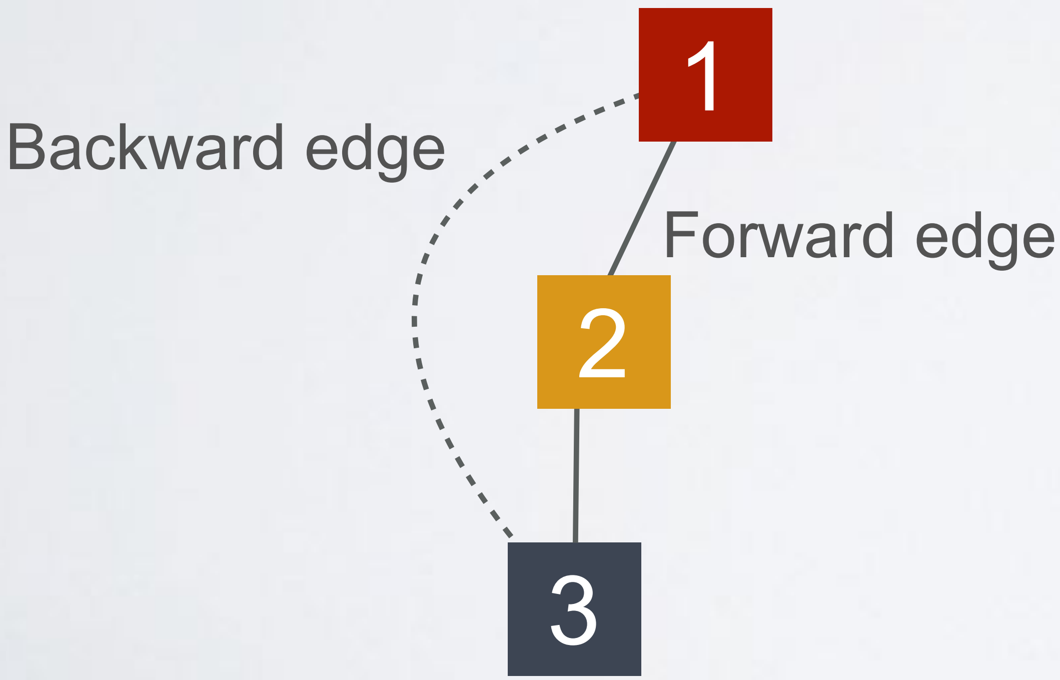
- Patterns generated recursively by adding edges
    - Graph structure represented using *DFS codes* (gSpan, 2002)
    - Different codes can describe the same graph
    - Examples on unlabeled undirected graphs, but generalizable

(1,2) ,(2,3)

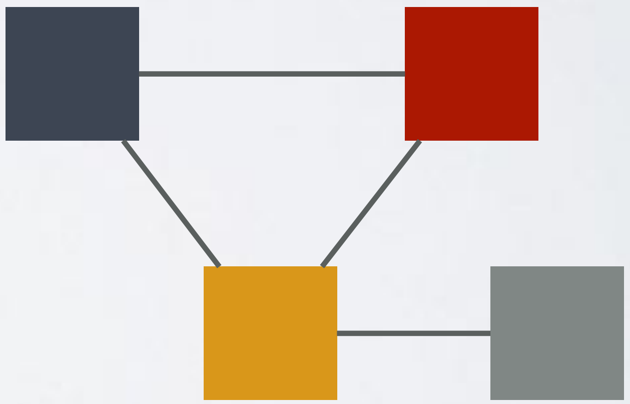


PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable

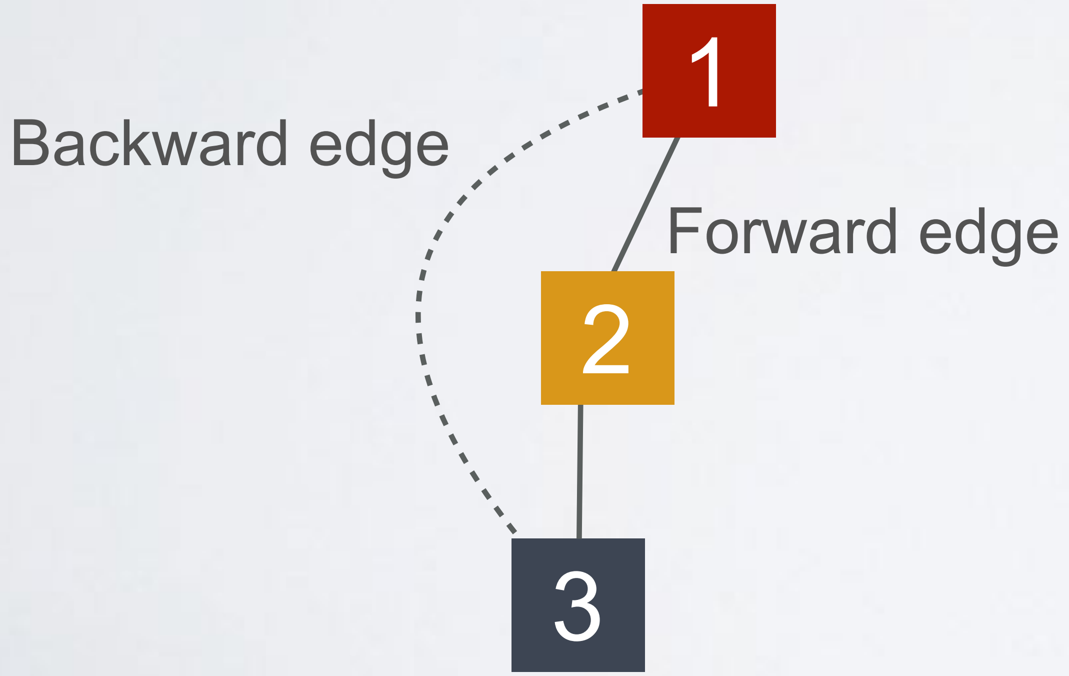


(1,2) ,(2,3)

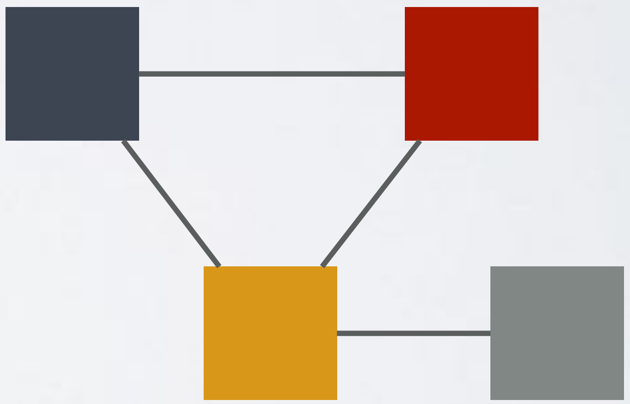


PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable

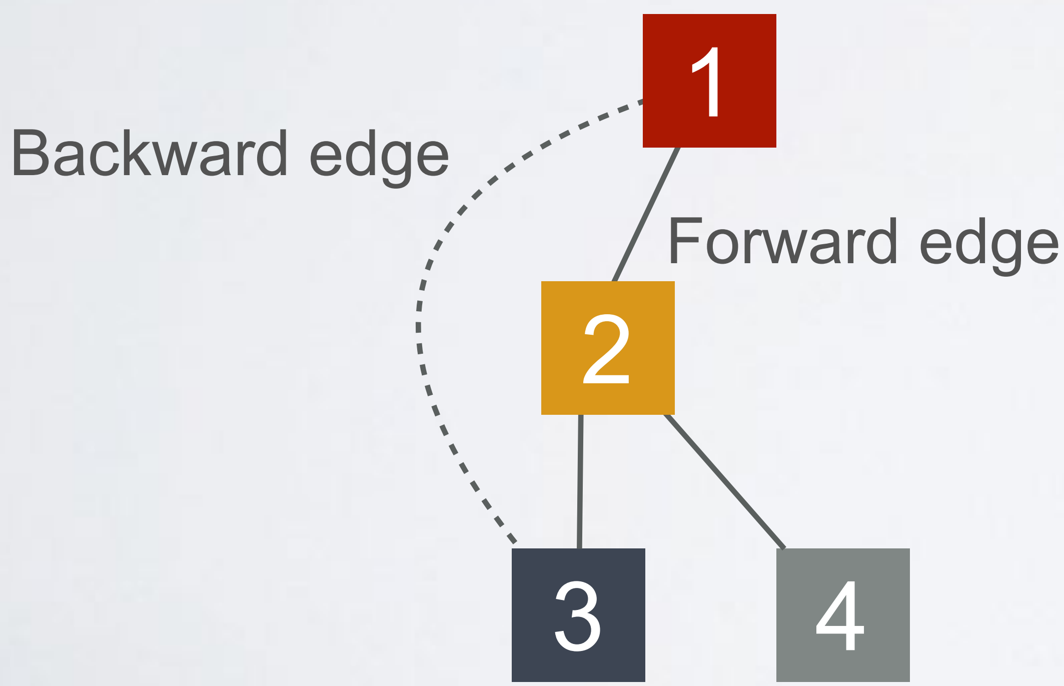


(1,2),(2,3),(3,1)

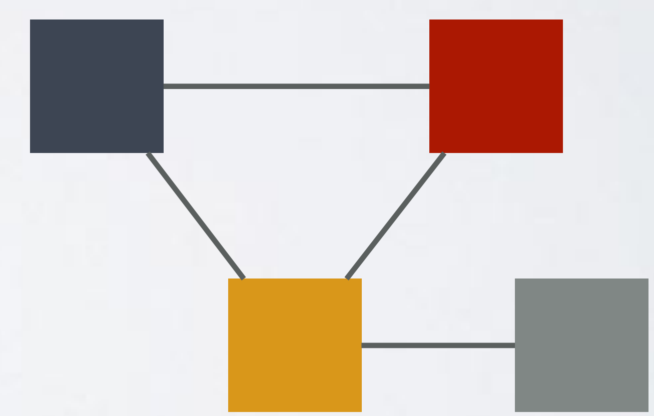


PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable

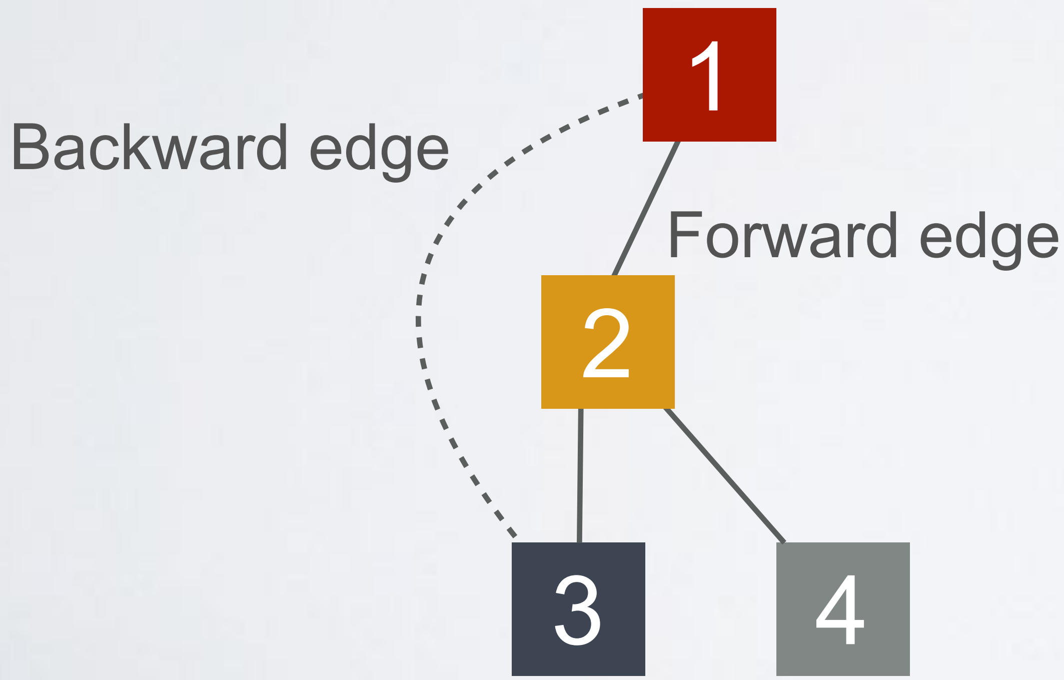


(1,2),(2,3),(3,1)

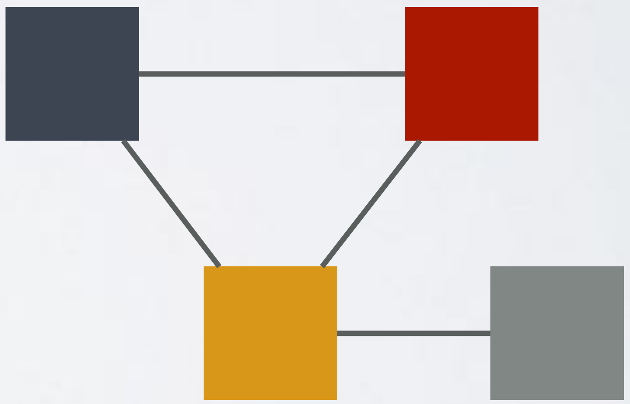


PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable

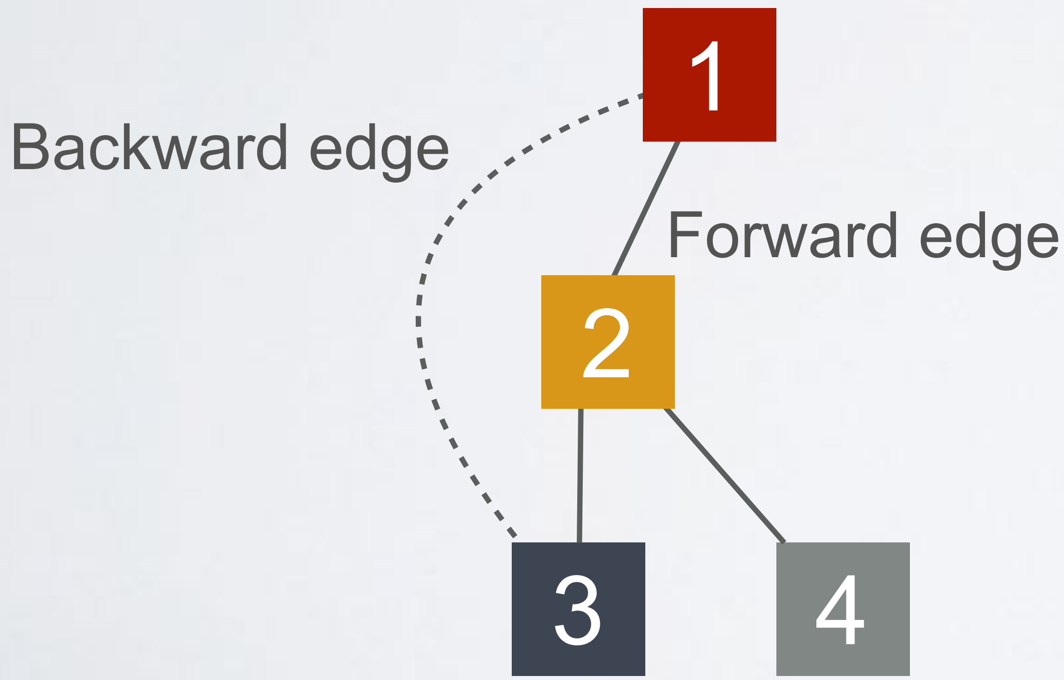


(1,2),(2,3),(3,1),(2,4)



PATTERN REPRESENTATION

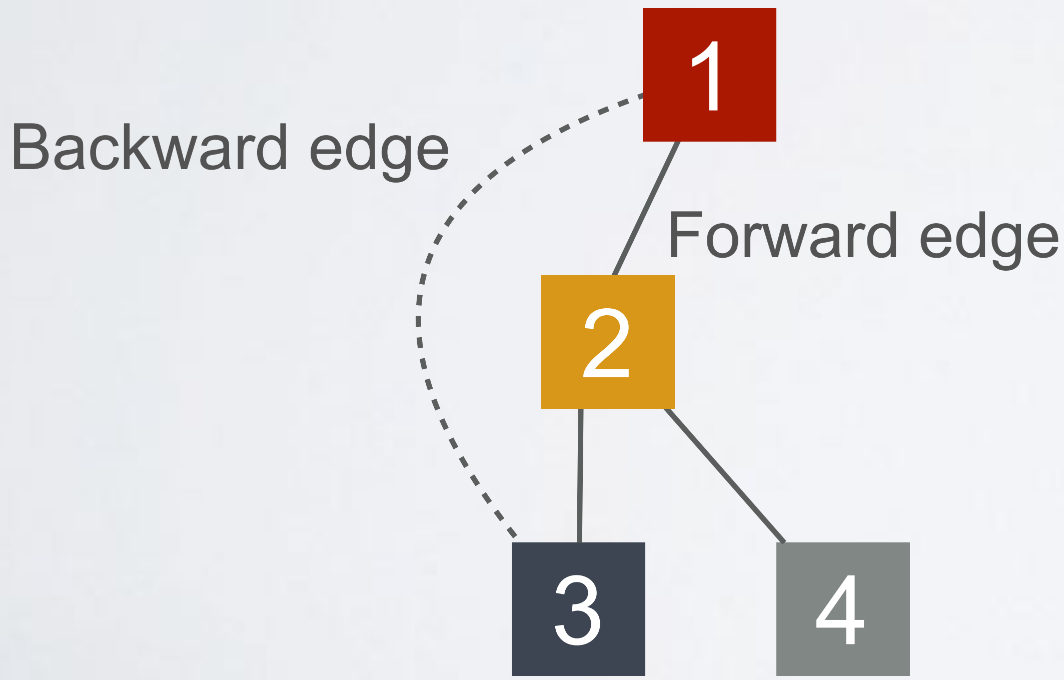
- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



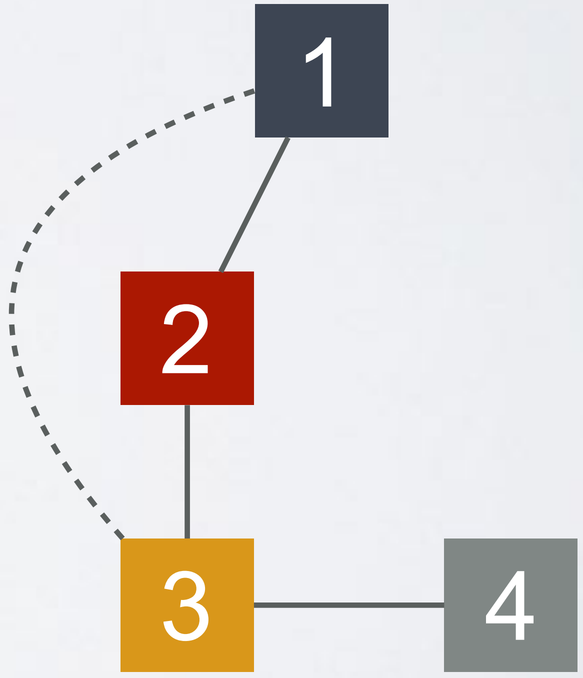
(1,2),(2,3),(3,1),(2,4)

PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



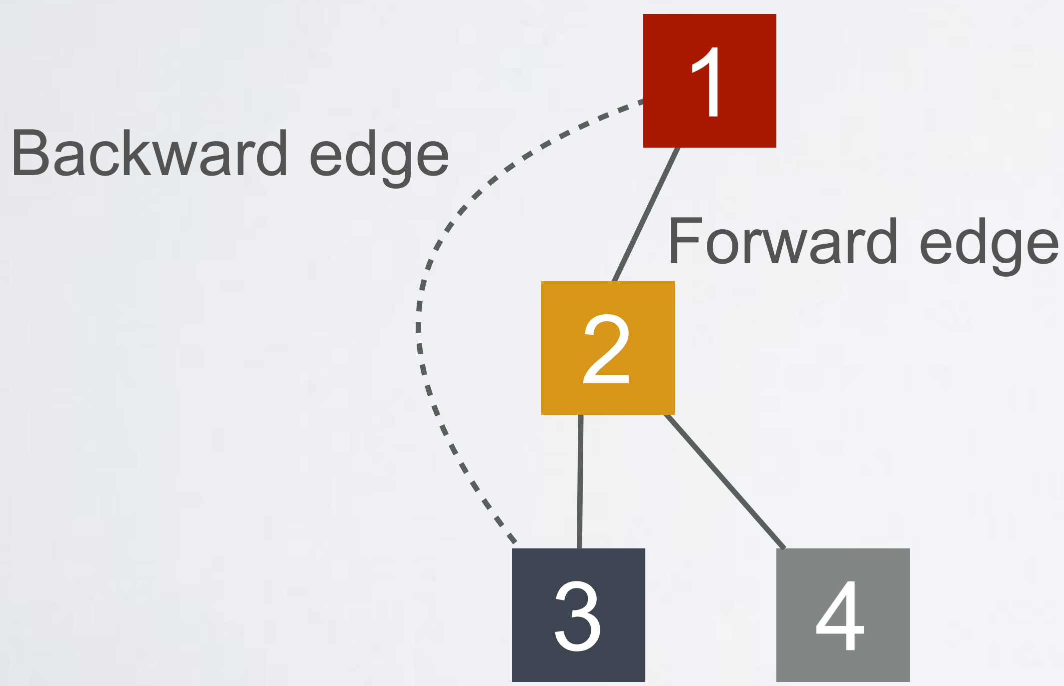
(1,2) ,(2,3) ,(3,1) ,(2,4)



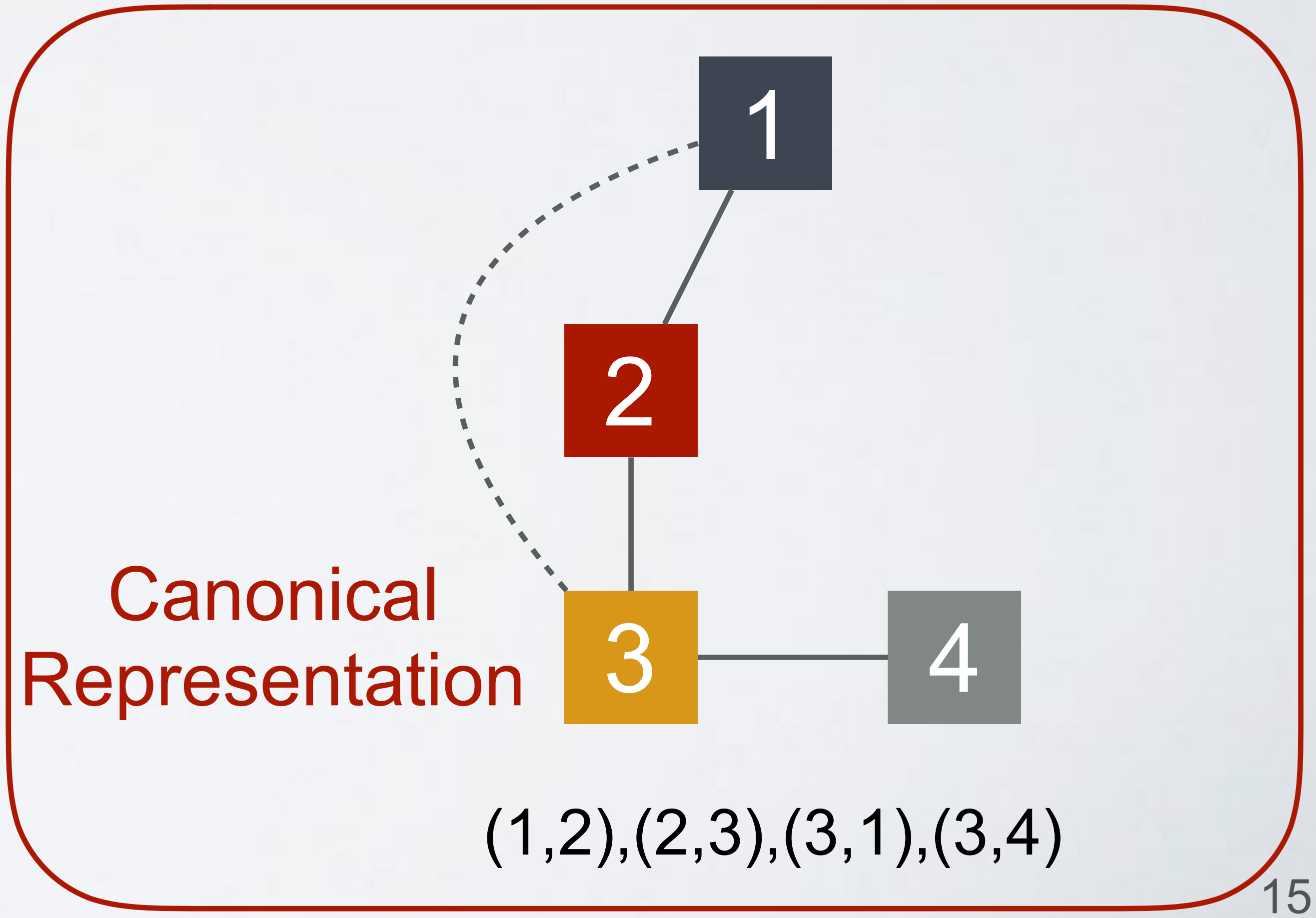
(1,2),(2,3),(3,1),(3,4)

PATTERN REPRESENTATION

- Patterns generated recursively by adding edges
  - Graph structure represented using *DFS codes* (gSpan, 2002)
  - Different codes can describe the same graph
  - Examples on unlabeled undirected graphs, but generalizable



(1,2) ,(2,3) ,(3,1) ,(2,4)



RECURSIVE GENERATION

- Canonical child pattern of *s* edges obtained from 2 parents of *s-1* edges

P_1 $e_1 \ldots e_{s-2}, e_{s-1}$

P_2 $e_1 \ldots e_{s-2}, e_{s'}$

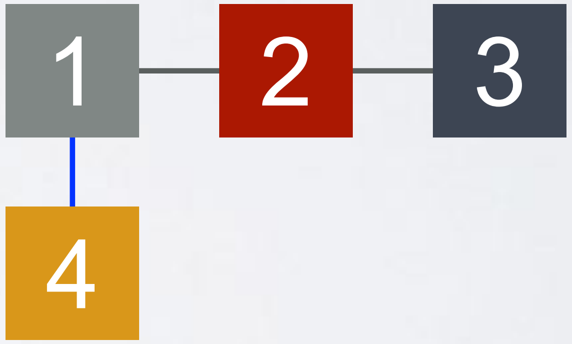
C $e_1 \ldots e_{s-2}, e_{s-1}, e_s$

RECURSIVE GENERATION

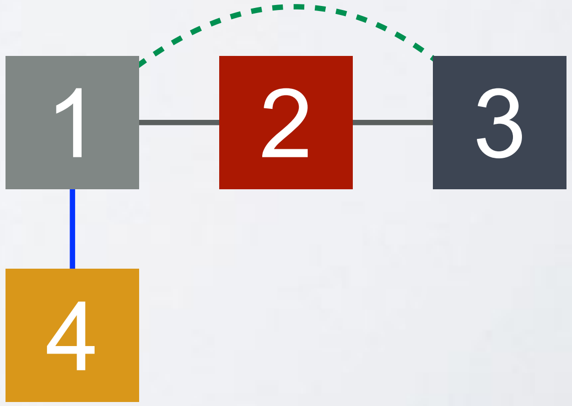
- Canonical child pattern of *s* edges obtained from 2 parents of *s-1* edges

P_1 $e_1 \ldots e_{s-2}, e_{s-1}$

P_2 $e_1 \ldots e_{s-2}, e_{s'}$



C $e_1 \ldots e_{s-2}, e_{s-1}, e_s$

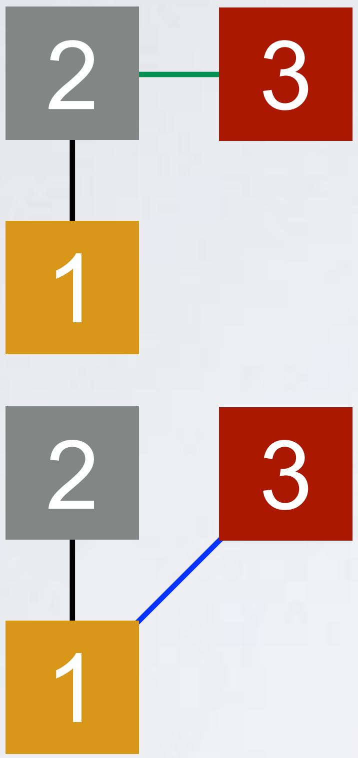


GENERATION PRIMITIVES

		e_{s-1}	
		Backward	Forward
e_s	Backward	BB-merge	FB-merge
	Forward	BF-merge	FF-merge Extension

Completeness: no canonical frequent pattern is missed

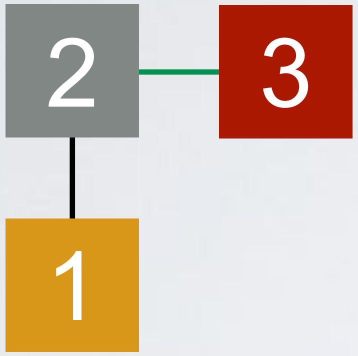
FB-MERGE



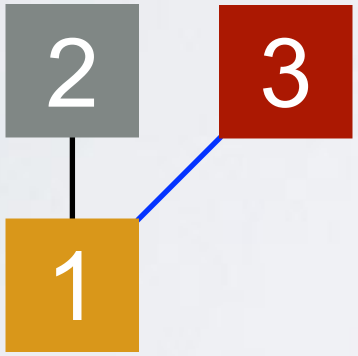
P_1 (1,2),(2,3)

P_2 (1,2),(1,3)

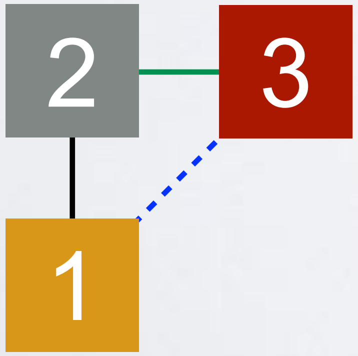
FB-MERGE

P_1 

(1,2),(2,3)

P_2 

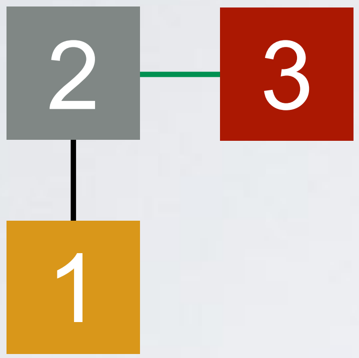
(1,2),(1,3)

C 

(1,2),(2,3),(3,1)

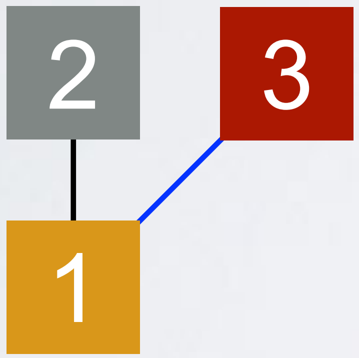
FB-MERGE

P_1



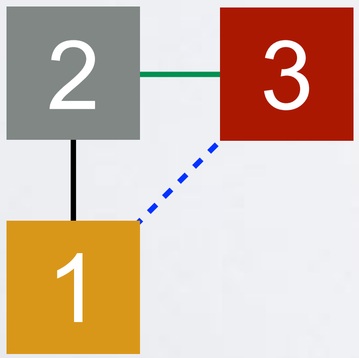
(1,2),(2,3)

P_2



(1,2),(1,3)

C



(1,2),(2,3),(3,1)

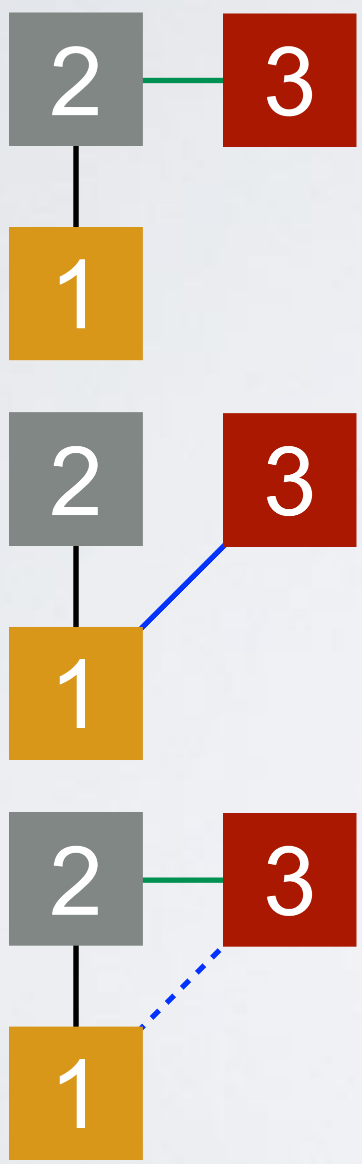
$e_s = \text{swap}(e_s')$

FB-MERGE

P_1

P_2

C



(1,2),(2,3)

(1,2),(1,3)

(1,2),(2,3),(3,1)

$e_s = \text{swap}(e_s')$

(v_1,v_2,v_3) embedding of P_1 and P_2

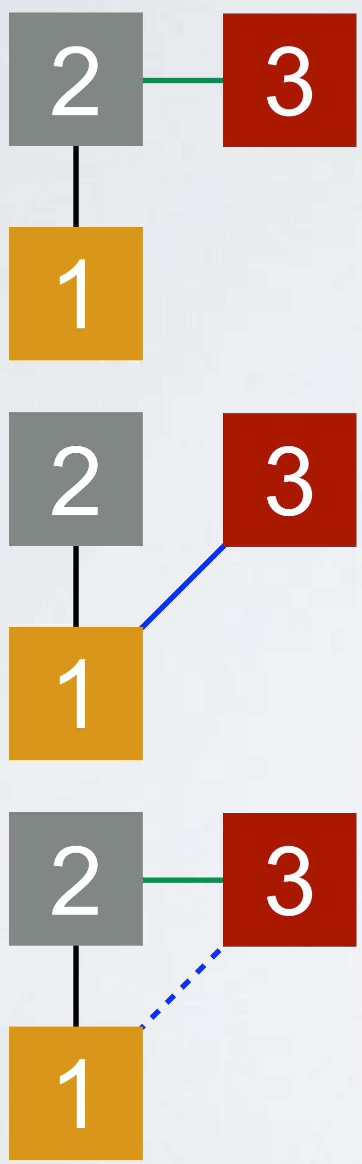
(v_1,v_2,v_3) embedding of C

FB-MERGE

P_1 (1,2),(2,3)

P_2 (1,2),(1,3)

C (1,2),(2,3),(3,1)



$e_s = \mathrm{swap}(e_s')$

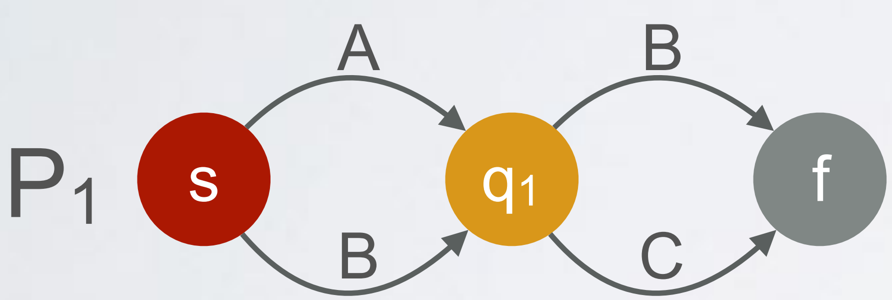
(v_1,v_2,v_3) embedding of P_1 and P_2

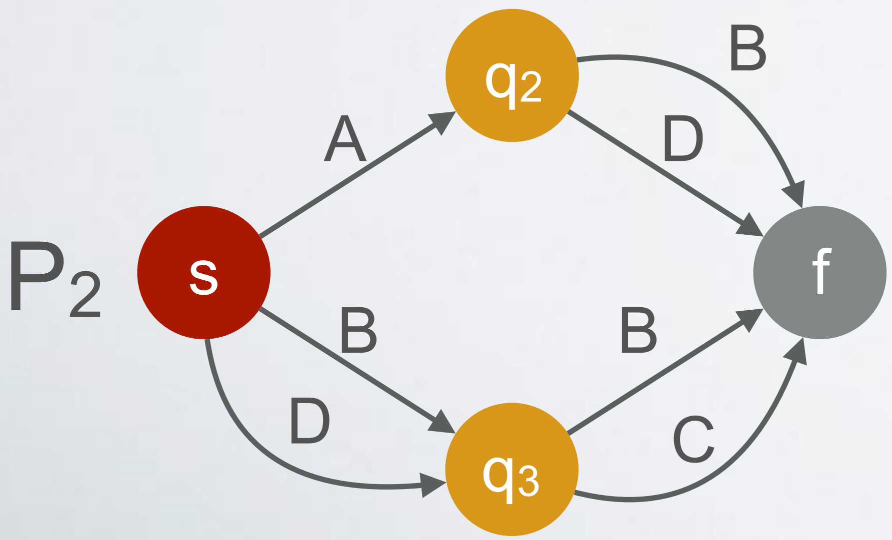
(v_1,v_2,v_3) embedding of C

Intersection of embeddings

FB-MERGE ON AUTOMATA OF EMBEDDINGS

- FB-merge: intersections of embeddings
  - Generate an automaton that accepts $W_{P_1} \cap W_{P_2}$: product of automata $O(\#states^2)$

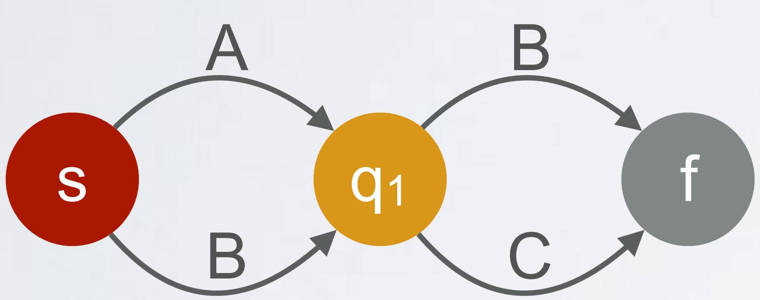




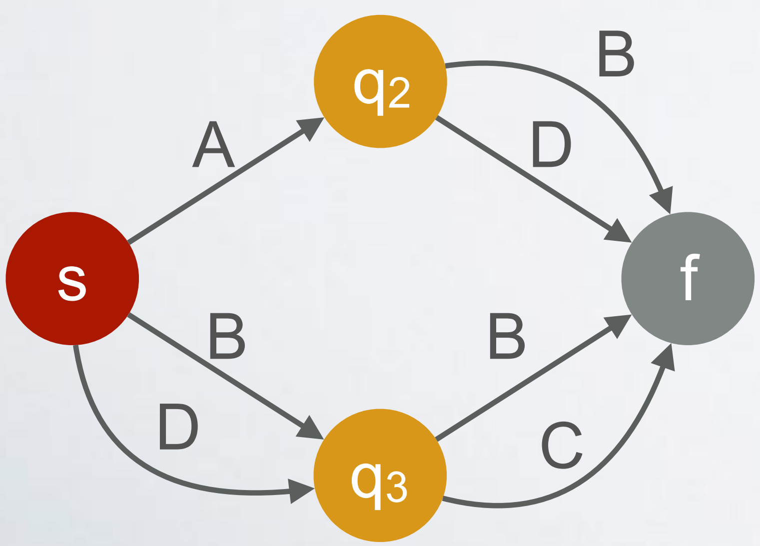
FB-MERGE ON AUTOMATA OF EMBEDDINGS

- FB-merge: intersections of embeddings
  - Generate an automaton that accepts $W_{P_1} \cap W_{P_2}$: product of automata $O(\#states^2)$

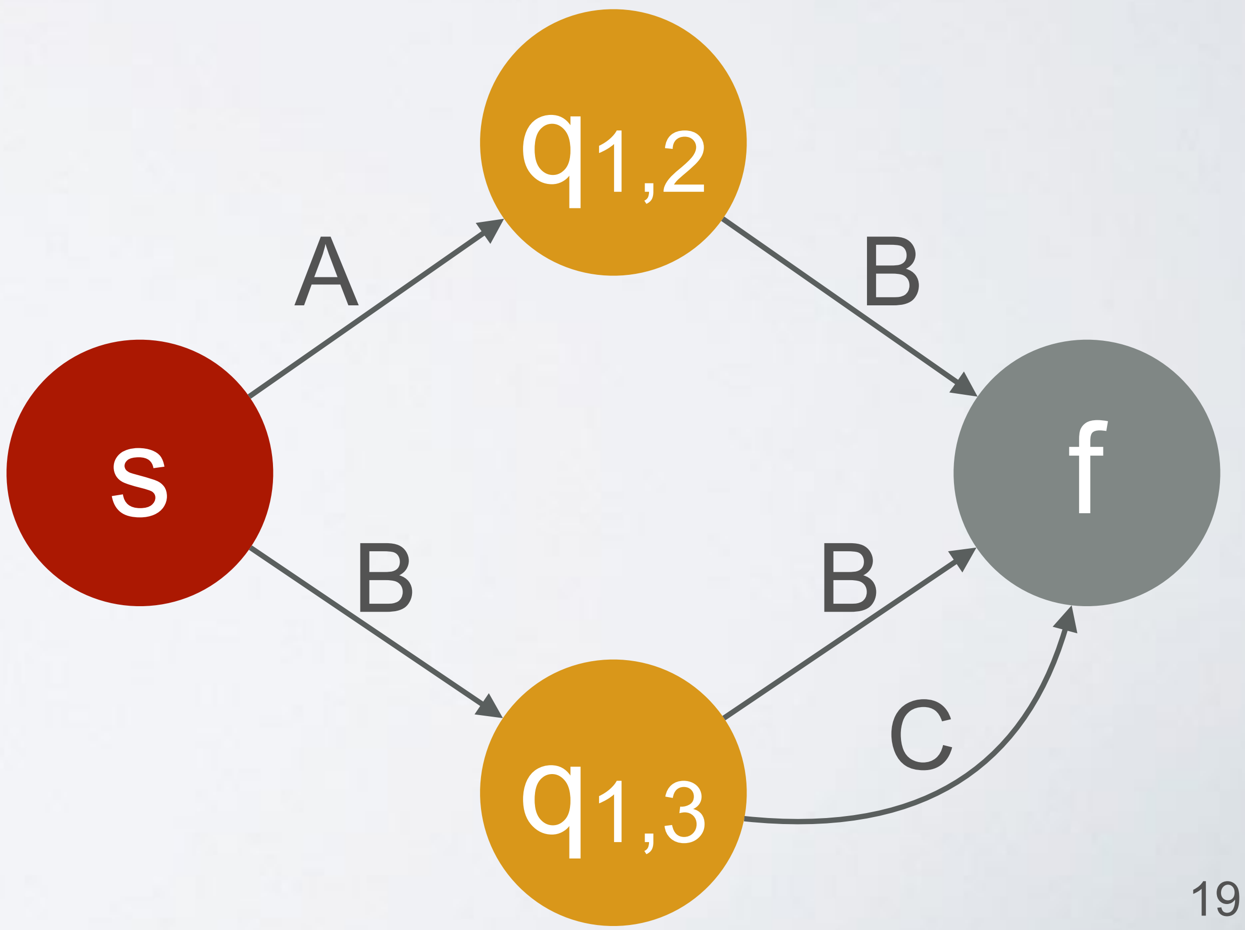
P_1



P_2

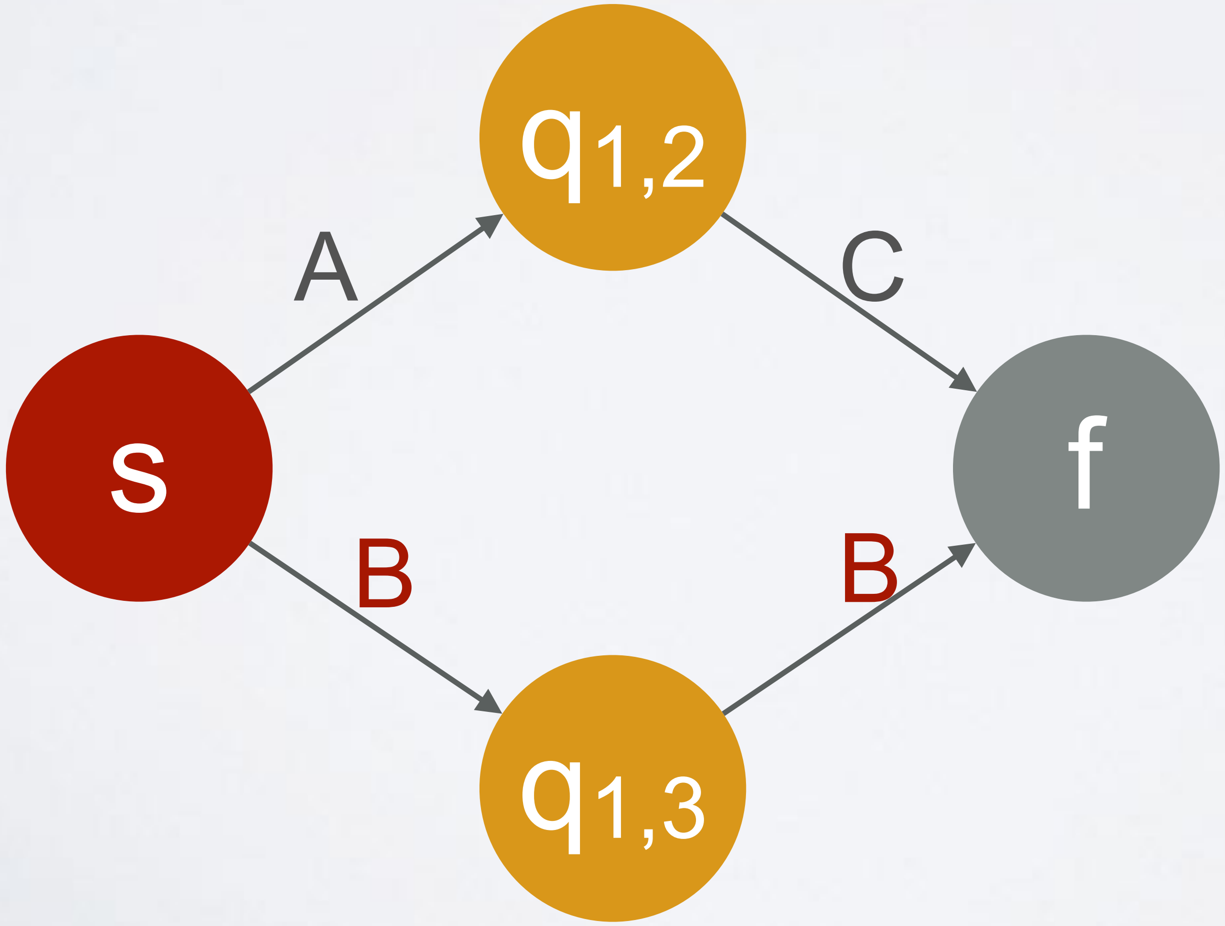


C



AUTOMATA VALIDATION

- Support computed directly from automata
  - Mappings of vertex *i* are the labels of transitions at level *i*
  - *No duplicates* rule, check that each transition has at least a valid path



PRIMITIVES ON AUTOMATA: GENERALIZATION

- Each of the 5 primitives can be performed directly on automata
  - O(#embeddings) becomes O(#automaton states^2)
  - Compact automata lead to huge gains
    - Minimization: Revuz's algorithm

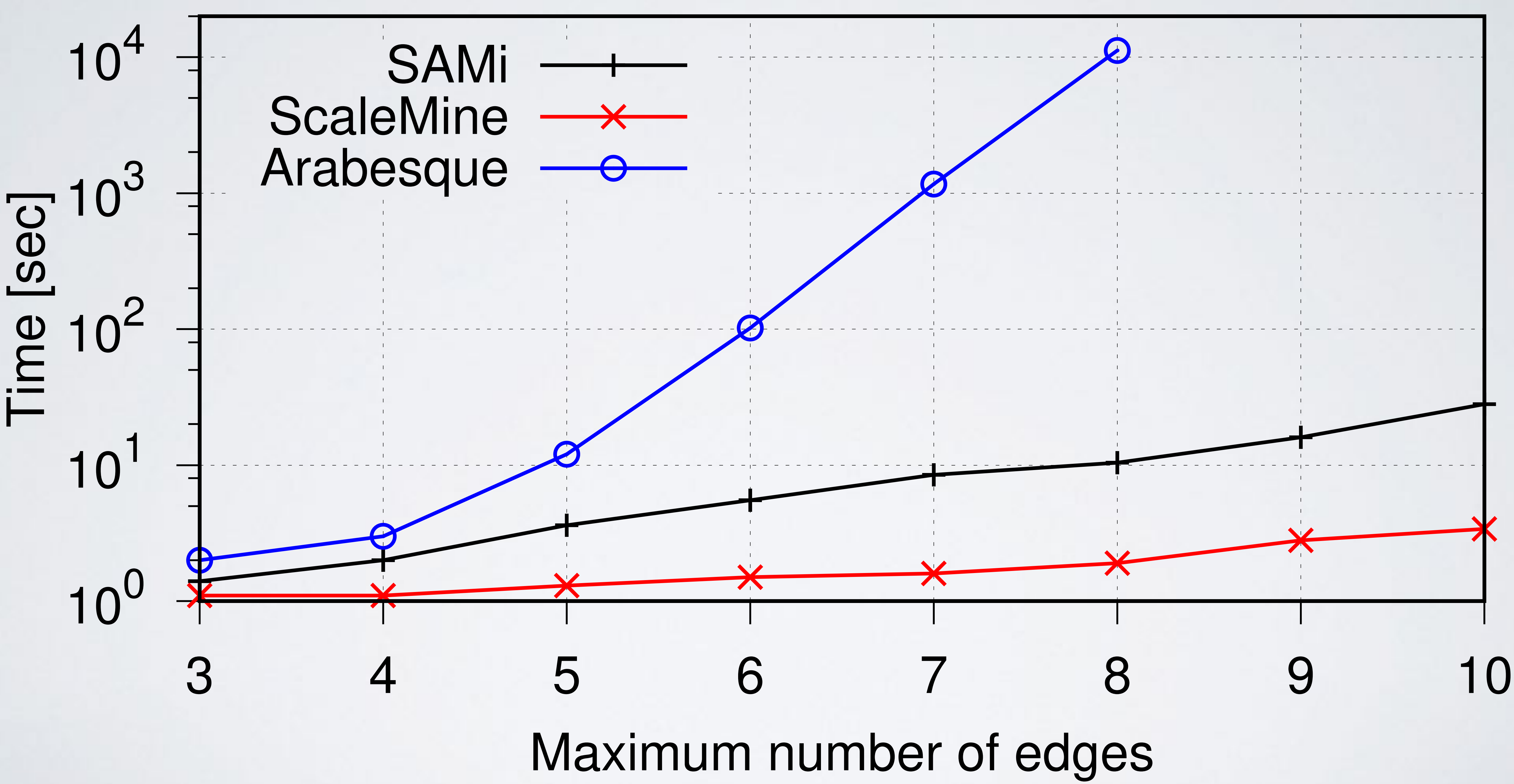
SAMi is complete: all frequent patterns are generated in their canonical representation

EXPERIMENTS

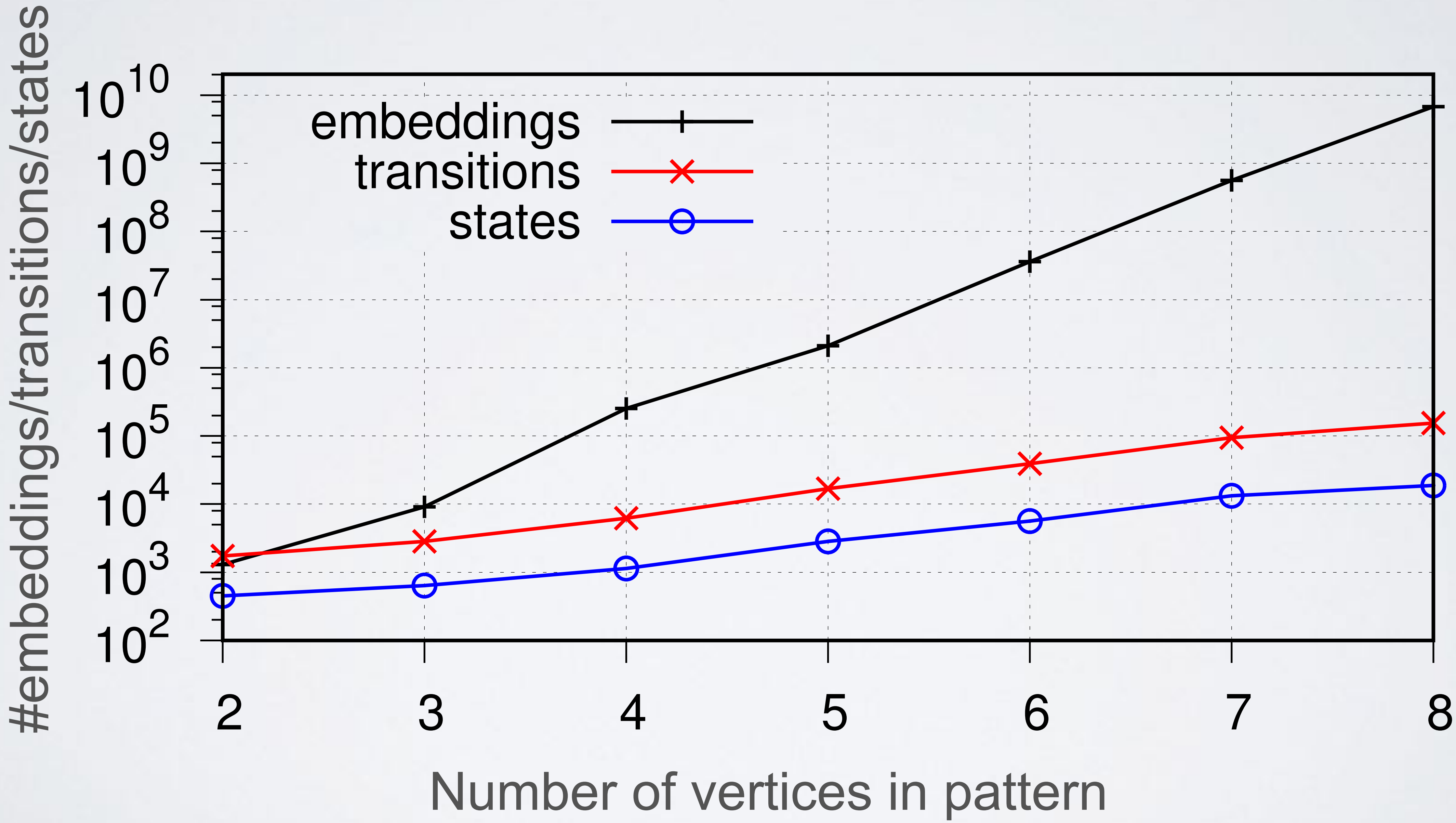
SETUP

- Datasets
  - Citeseer: 3k vertices, 5k edges
  - Patents: 2M vertices, 13M edges
  - Yago: 2M vertices, 4M edges
- Parameters
  - Pattern complexity (#edges)
  - Support threshold (ε)
- Measures
  - Mining time
  - #embeddings / automata size

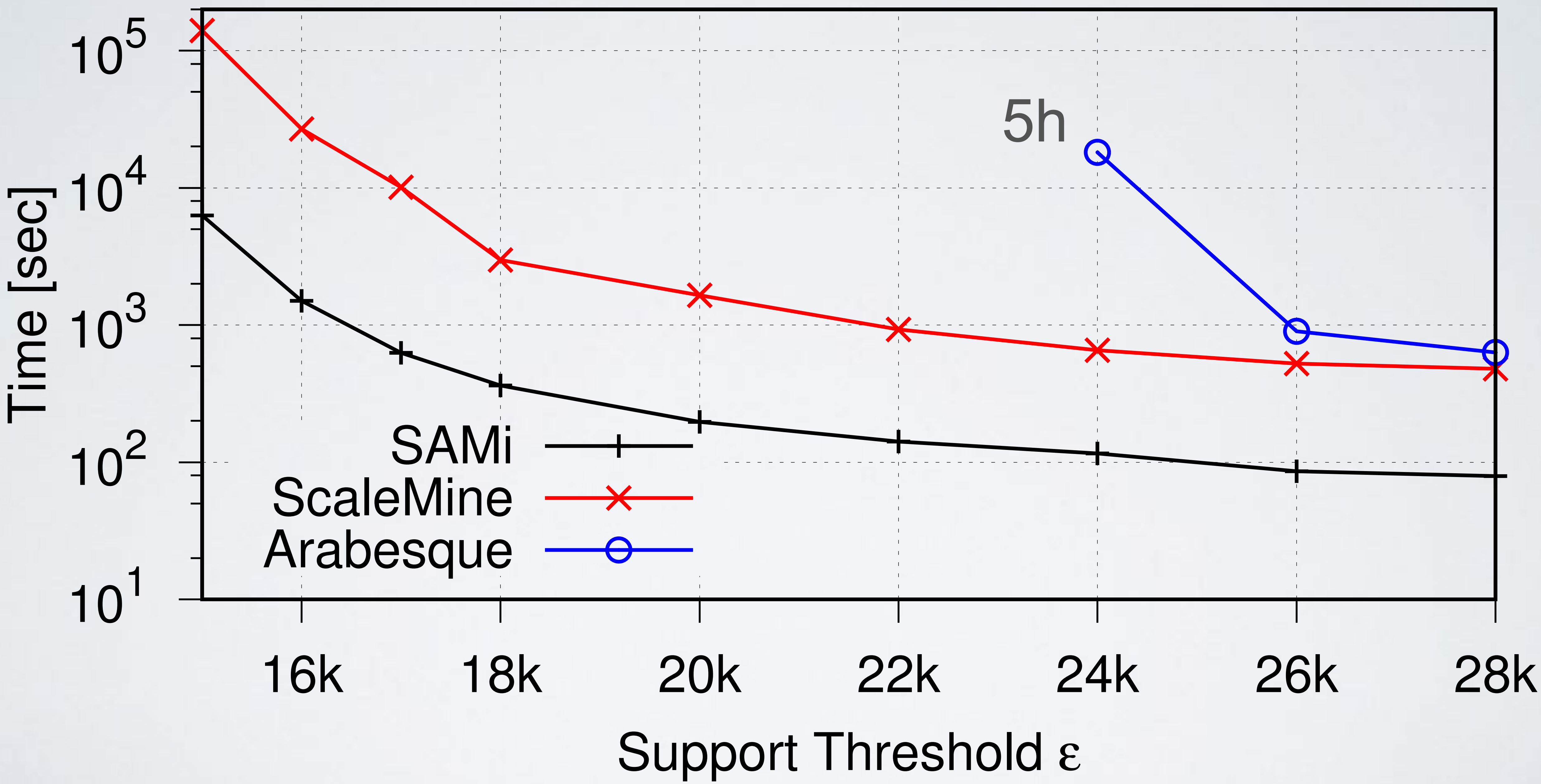
PERFORMANCE: CITESEER



EMBEDDINGS COMPRESSION



PERFORMANCE: PATENTS



PERFORMANCE: YAGO

	Max. #edges = 3	4	5
$\varepsilon = 2$	0:02:47	1:14:57	3:44:11
$\varepsilon = 10$	0:02:28	1:14:49	2:52:21
$\varepsilon = 100$	0:02:26	1:14:26	2:35:28
$\varepsilon = 1000$	0:02:07	1:11:19	2:13:05

Ontological Pathfinding [SIGMOD16]
AMIE+ [VLDBJ]

Max #edges=3

GRAPH MINING: CONCLUSION

- Addresses the fundamental problems of FSM
  - Pattern generation process (5 primitives)
  - Compressed representation of embeddings
- Three orders of magnitude faster than state of the art
  - Opens new possibilities: knowledge graph mining
  - Qualitative evaluation of mining outcome