Graph and Web Mining - Motivation, Applications and Algorithms - Chapter 2

Prof. Ehud Gudes
Department of Computer Science
Ben-Gurion University, Israel

Outline

- Basic concepts of Data mining and Association rules
  - Apriori algorithm
- Motivation for Graph mining
- Applications of Graph Mining
- Mining Frequent Subgraphs - Transactions
  - BFS/Apriori Approach (FSG and others)
  - DFS Approach (gSpan and others)
  - Diagonal Approach
  - Greedy Approach
- Mining Frequent Subgraphs – Single graph
  - The support issue
  - The Path-based algorithm
  - Constraint-based mining
  - Other algorithms

Single Graph Setting

Most existing algorithms use a Transaction setting approach.
That is, if a pattern appears in a transaction even multiple times it is counted as 1! (FSG, gSPAN)

What if the entire database is a single graph?
This is called: the single transaction setting

We need a different Support definition!

Problem Statement (single graph setting)

- ***Input:*** (*D, minSup*)
  - A single graph D (e.g. the Web or DBLP or an XML file)
  - Minimum support *minSup*
- ***Output:*** (All frequent subgraphs).
  - A subgraph is frequent if the number of its "occurrences" in D is above an admissible support measure
- *Definition of an admissible support measure?*

single graph setting - Motivation

- ***Often the input is a single large graph***
- ***Examples:***
  - The web or portions of it
  - A social network (e.g. a network of users communicating by email at BGU)
  - A large XML database such as: DBLP or Movies database
- *Mining such large graph databases is very useful!*

The Path Algorithm (Vanetik, Gudes, Shimony ICDM 2002, TKDE 2006)

Goal: Find all frequent connected subgraphs of a database graph.

Basic approach: Apriori or BFS

But the basic building block is a Path not an Edge!
This works since any graph can be decomposed to a set of disjoint paths

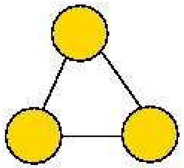
Result: faster convergence of the algorithm

Support issue (DAMI J. Sept 2006)

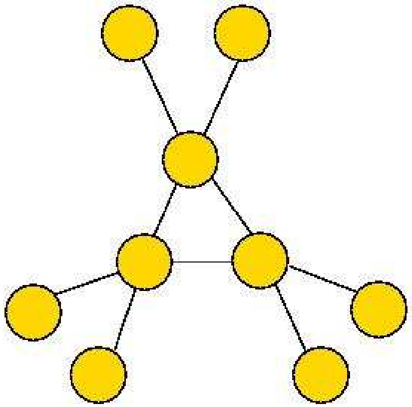
Definition a support measure **S** is <u>admissible</u> if for any pattern **P** and any sub-pattern **Q** $\subset$ **P** => **S(Q)≥S(P).**

Problem: the number of appearances of the graph pattern in the database graph is not an admissible support measure!

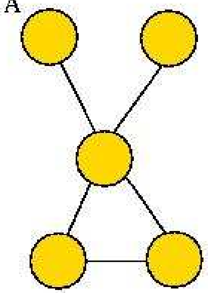
B



Database graph



A



Graph **A** appears 3 times in the database graph, while graph **B** $\subset$ **A** appears only once!.

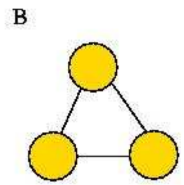
The Instance graph

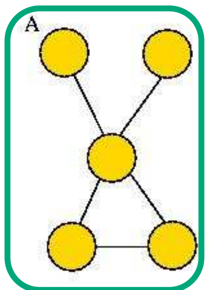
Definition

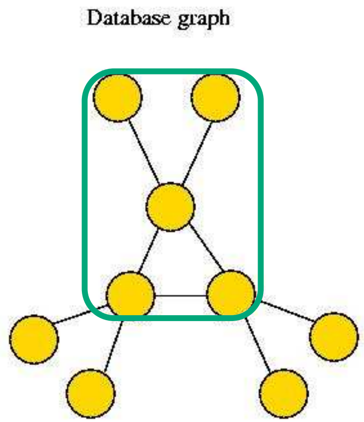
An *instance graph* **I(P)** of pattern **P** in database graph **G** is a graph

$$\mathbf{G = (V,E)} \text{ where } \mathbf{V = \{g \subset G,\ g \approx P\}} \text{ and}$$
$$\mathbf{E = \{(g,h),\ g,h \in V} \text{ and } \mathbf{E(g) \cap E(h) \neq \emptyset\}}.$$

That is: it's a graph where each node represents a different occurrence of the pattern in the database graph, and an edge between two nodes indicates that there is at least one edge overlap between the corresponding occurences







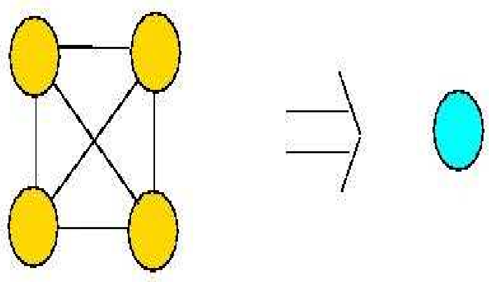
Support issue

<u>Operations on instance graph:</u>

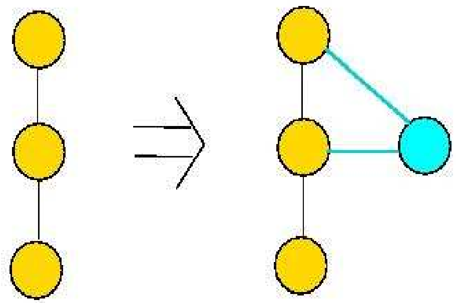
- *clique contraction*
  replacing a clique C by a single node c such that only the nodes that were adjacent to each node of C may become adjacent to c
- *node expansion*
  replacing an existing node v by a new subgraph whose nodes may or may not be adjacent to the nodes adjacent to v
- *node addition*
  adding a new node to the graph and arbitrary edges between the new node and the old ones
- *edge removal*

Example of operations on the instance graph

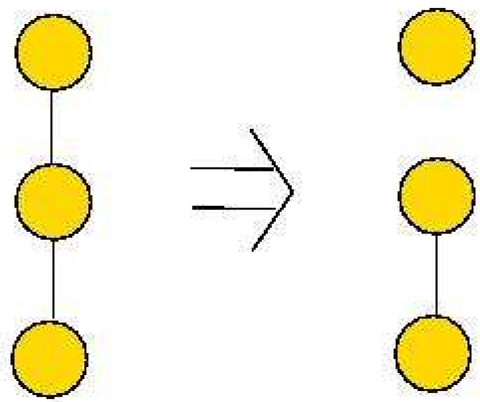
- **clique contraction**



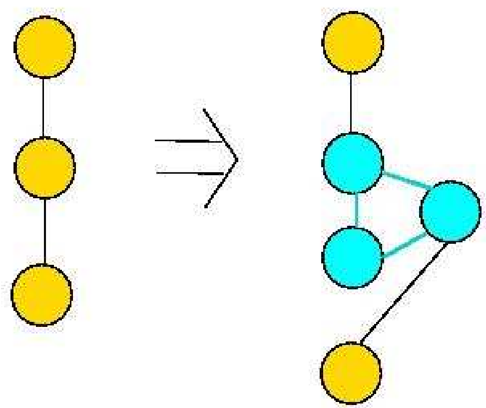
- **vertex addition**



- **edge removal**



- **vertex expansion**



The main result

Theorem

A support measure **S** is an admissible support measure **if and only if** it is non-decreasing on the instance graph **I(P)** of every pattern **P** under clique contraction, node expansion, node addition and edge removal.

See KDD journal (DAMI) Sept. 2006

Example of support measure - MIS

Admissible support measure - MIS: easy to show that's its admissible

$$\text{MIS} = \frac{\text{Maximum independent set size of instance graph}}{\text{Number of edges in the database graph}}$$

Motivation for MIS measure:

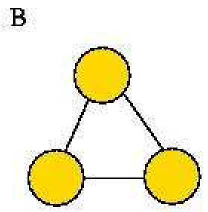
We are interested in *typical* structure, i.e. structures created by many users that are somewhat independent.

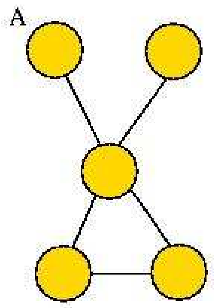
A single complex structure that has many references is less interesting for us.

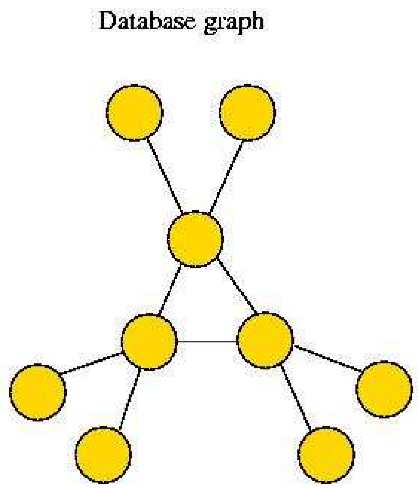
Most common support measures are covered by this definition, including the standard support measure for transaction databases.

The MIS Measure - Example

Definition a support measure **S** is <u>admissible</u> if for any pattern **P** and any sub-pattern **Q** $\subset$ **P** **=>** **S(Q)≥S(P).**







Instance graph for A is also a triangle, therefore MIS(A) = 1
Instance graph for B is one node, therefore MIS(B) = 1
MIS(B) < = MIS(A) Admissible!

The MIS Measure

- The only measure found so far to be admissible (challenge – find others!)
- Was also adopted by Kuramochi- **Finding Frequent Patterns in a Large Sparse Graph** [SDM2004]**)**
- Since finding the MIS is an NP-complete problem, Kuramochi suggests several approximating measures which are easier to compute.

Path-based mining algorithm (Vanetik, Gudes, Shimony)

- The algorithm uses paths (instead of edges) as basic building blocks for pattern construction. Larger building blocks may make the search more efficient
- It starts with one-path graphs and combines them into 2-, 3- etc. path graphs.
- The combination technique does not use graph operations and is easy to implement.

Path number

Definition. *Path number* **p(G)** of a graph is the minimal number of edge-disjoint paths that cover all edges in the graph.

A collection of **p(G)** paths that cover all edges is called a *minimal path cover*.

A graph G is *Eulerian* if it can be covered by a single cyclic path (i.e P(G) = 1), Otherwise:

For an undirected graph **G=(V,E),**

$$p(G) = \frac{\left| v \in V \middle| d(v) \;\; is\ odd \right|}{2}$$

For a directed graph **G=(V,E),**

$$p(G) = \frac{\sum_{v \in V} \left| d^+(v) - d^-(v) \right|}{2}$$

Path Facts

Definition Graph **G'=G\P** where **P** is an edge-disjoint path denotes the graph obtained by removing from **G** all edges of path **P**

Claim 1: Let **P** be any path from minimal path cover of a connected graph **G**. Then **p(G\P)=p(G)-1**.

Claim 2: In any path cover of a connected graph **G** there are at least two paths
$\mathbf{P_1,P_2}$ such that $\mathbf{G\backslash P_1}$ and $\mathbf{G\backslash P_2}$ are connected.

We also define an *order* $\leq_P$ on paths in order to represent a path decomposition of a graph in a unique way.
We only store decompositions that are *minimal* with respect to this Order (denoted by P-minimal).

Order on paths

Let P be a path in an undirected (or directed) graph G.
Path degree $pd(v)$ (**path indegree** $pd^{+}(v)$ and **path outdegree** $pd^{-}(v)$) of a node v in P is a number of edges in P adjacent to v (the number of ingoing and outgoing edges for v in P).

A **representative tuple RT(v)** of vertex v in path P is a tuple
$<label(v), pd(v)>$ ($<label(v), pd^{+}(v), pd^{-}(v)>$ if P is directed).

These tuples can be compared lexicographically.

Order on paths (cont.)

Path descriptor $D(P)$ of path P is the ordered set $\{RT(v) \mid v \in V(P)\}$ where representative tuples are arranged in non-decreasing order with respect to the natural lexicographical order on them.

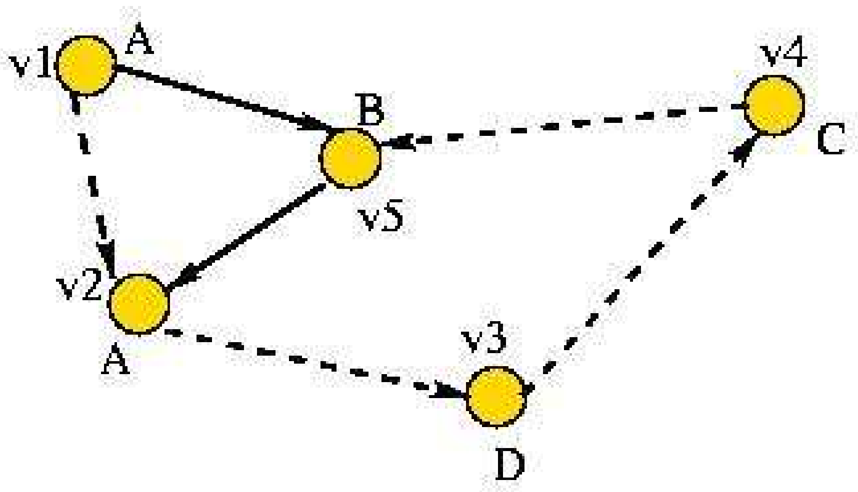
Definition. Let P,Q be paths. Then $P \leq_p Q$ iff $D(P) \leq D(Q)$.

This relation allows us to compare path covers as follows.

Let X and Y be path covers of G, sorted in non-decreasing order according to $\leq_p$ order. Then $X \leq_P Y$ if X is lexicographically smaller than or equal to Y.

Order on paths : example

We have paths $P_1 = v1,v2,v3,v4,v5$ and $P_2 = v1,v5,v2$



D **(*P*$_1$)** = **<A,0,1>,<A,1,1>,<B,1,0>,<C,1,1>,<D,1,1>**
D **(*P*$_2$)** = **<A,0,1>,<A,1,0>,<B,1,1>** ***(order is Lexicographic not by vertex order)***

Therefore, $P_2 \leq_P P_1$.

The Three phases of the mining algorithm

- Phase #1 finds all frequent graph patterns with path number 1

- Phase #2 finds all frequent graph patterns with path number 2 by "joining" pairs of patterns found in phase #1

- Phase #3 finds all frequent graph patterns with path number n≥3 by "joining" pairs of patterns with path number (n-1) (and apply Apriori pruning).

The main problem: how candidates are "joined"?

How to store patterns: the composition relation

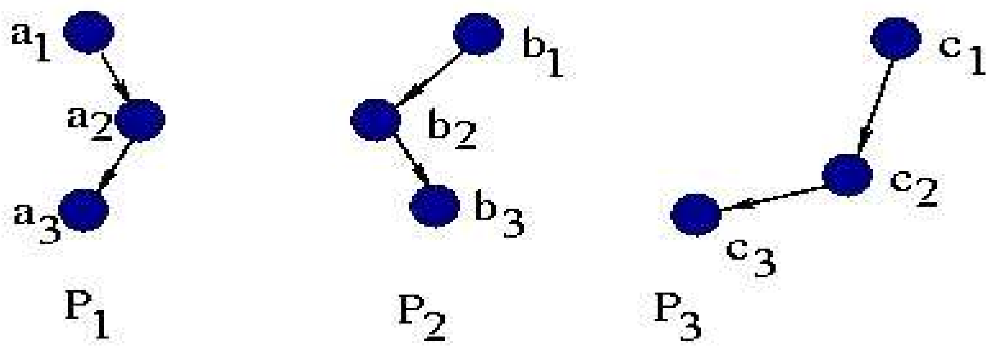
A **composition relation** $C(P_1,\ldots,P_n)$ (or C) on paths $P_1,\ldots,P_n$ of graph G is a table with nodes of G as rows and paths as columns such that $C[i,j] \neq \perp$ iff i-th node of G is also a node of path P_j.

$C(P_1,P_2,P_3)$:

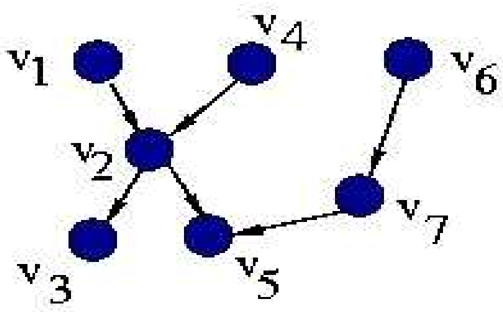
Node	P_1	P_2	P_3
v1	a1	⊥	⊥
v2	a2	b2	⊥
v3	a3	⊥	⊥
v4	⊥	b1	⊥
v5	⊥	b3	c3
v6	⊥	⊥	c1
v7	⊥	⊥	c2

Restoring patterns: graph composition

By treating table rows as graph nodes and defining edges *(i,j)* whenever two nodes of a path P_k, appearing in rows i and j, have an edge between them, we can construct a graph corresponding to composition relation $C(P_1,\ldots,P_n)$. A **graph composition** of $C(P_1,\ldots,P_n)$ is denoted by $\Omega(C)$.



$\Omega(C(P_1,P_2,P_3))$:



The Uniqueness property

- By using the lexicographic order on Paths and defining the Composition Relation as representing the Minimal order of Paths we get a unique representation for every graph.

- This is similar to canonical labeling and needed to eliminate duplicates and assure the completeness of the Algorithm

How to remove a path: subtraction

Subtraction of a path P_i from a composition relation C, $C \backslash P_3$, consists of:

a) eliminating the i-th column from the table;

b) removal of all rows containing only *null* values.

$C(P_1,P_2,P_3) \backslash P_3$:

Node	P_1	P_2
v1	a1	$\perp$
v2	a2	b2
v3	a3	$\perp$
v4	$\perp$	b1
v5	$\perp$	b3

Subtracting of (several) paths from C is also called a **projection** of C onto the remaining paths $P_1,\ldots,P_n$, denoted by $C|_{\{1,\ldots,n\}}$.

How to combine graphs

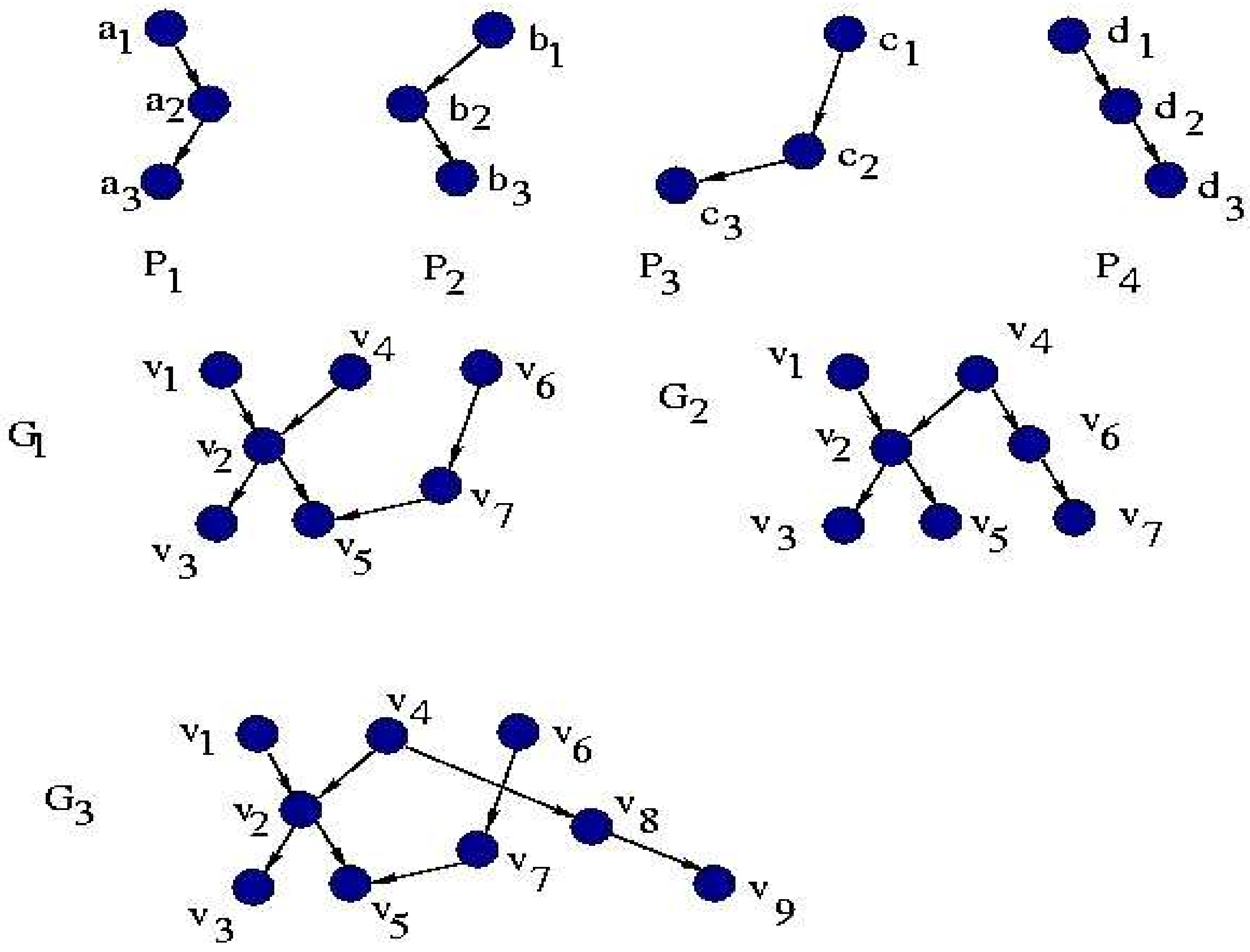
A **bijective sum** $BS(C_1, C_2, I1, I2)$ of composition relations C_1 and C_2, where $I1, I2$ are sets of indices and $C_1|_{I1} = C_2|_{I2}$, is a composition relation obtained by adding all columns of C_2 corresponding to paths that are *not in* C_1, to the table of C_1.

Bijective sum: example

Bijective sum of C_1 and C_2 on common paths P_1 and P_2.

C_1				C_2				C3				
	P_1	P_2	P_3		P_1	P_2	P_3		P_1	P_2	P_3	P_4
v1	a1			v1	a1			v1	a1			
v2	a2	b2		v2	a2	b2		v2	a2	b2		
v3	a3			v3	a3			v3	a3			
v4		b1		v4		b1	d1	v4		b1		d1
v5		b3	c3	v5		b3		v4		b3	c3	
v6			c1	v6			d2	v6			c1	
v7			c2	v7			d3	v7			c2	
								v8				d2
								v9				d3

Bijective sum: an example



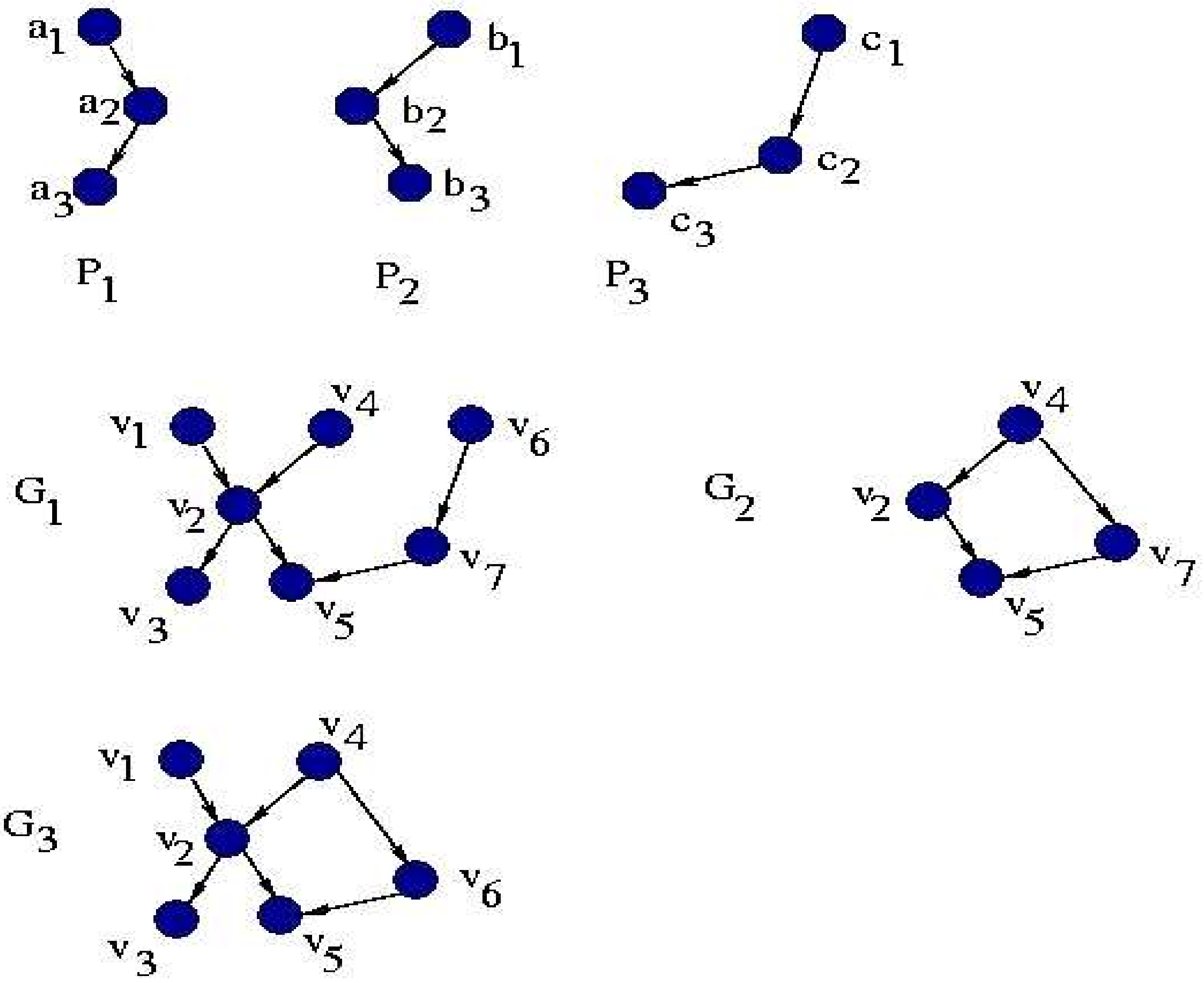
How to join more paths: splice

A **splice** $\oplus_{i,j}$ of two composition relations $C_1(P_1,\ldots,P_n)$ and $C_2(P_i,P_j)$, is a composition relation that turns every node common to P_i and P_j in C_2, into the node common to P_i and P_j in C_1 as well.

$C_3 = C_1(P_1,P_2,P_3) \oplus_{2,3} C_2(P_2,P_3)$:
Note: splice does not increase k, C_3 stays As 3-path

C_1				C_2			C_3			
	P_1	P_2	P_3		P_2	P_3		P_1	P_2	P_3
v1	a1			v2	b1	c1	v1	a1		
v2	a2	b2		v4	b2		v2	a2	b2	
v3	a3			v5	b3	c3	v3	a3		
v4		b1		v7		c2	v4		b1	c1
v5		b3	c3				v5		b3	c3
v6			c1				v6			c2
v7			c2							

Splice: an example



Phases of the path-based mining algorithm

- Phase #1 finds all frequent graph patterns with path number 1

- Phase #2 finds all frequent graph patterns with path number 2 by "joining" pairs of patterns found in phase #1

- Phase #3 finds all frequent graph patterns with path number $n \geq 3$ by "joining" pairs of patterns with path number (n-1).

Algorithm: Phase 1 – finding frequent 1-paths

1. Find all frequent edges and add them to L_1. Set $k \leftarrow 2$.
2. Set $C_k \leftarrow \emptyset$, $L_k \leftarrow \emptyset$.
3. For every path $P \in L_{k-1}$ and every edge $e=(v,u) \in L_1$ do:
   a. Let X be all nodes of P if P is cyclic and
      all unbalanced nodes of P if P is non-cyclic.
   b. For every $x \in X$ such that $x \approx v$
      add $Q=(V(P) \cup \{u\}, E(P) \cup \{x,u)\})$ to C_k if $p(Q)=1$.
   c. For every $x \in X$ such that $x \approx u$
      add $Q=(V(P) \cup \{v\}, E(P) \cup \{(v, x)\})$ to C_k if $p(Q)=1$.
   d. For every $x,y \in X$ such that $x \approx v$, $x \approx u$
      and $(x,y) \notin E(P)$,
      add $Q=(V(P), E(P) \cup \{(x, y)\})$ to C_k if $p(Q)=1$.
4. Compute frequency of all paths from C_k and
   add the frequent ones to L_k.
5. If $L_k \leftarrow \emptyset$, stop. Otherwise, set $k \leftarrow k+1$ and go to step 2.

Phase #1 – overview

Definition A node ***v*** in graph ***G*** is *balanced* if degree of ***v*** is even (for undirected graphs). A node is *unbalanced* if it is not balanced.

Phase #1 constructs candidate paths by adding one edge at a time.

If the path is *cyclic* (i.e is a (not necessarily simple) cycle, we can add edge anywhere (providing the labels match):

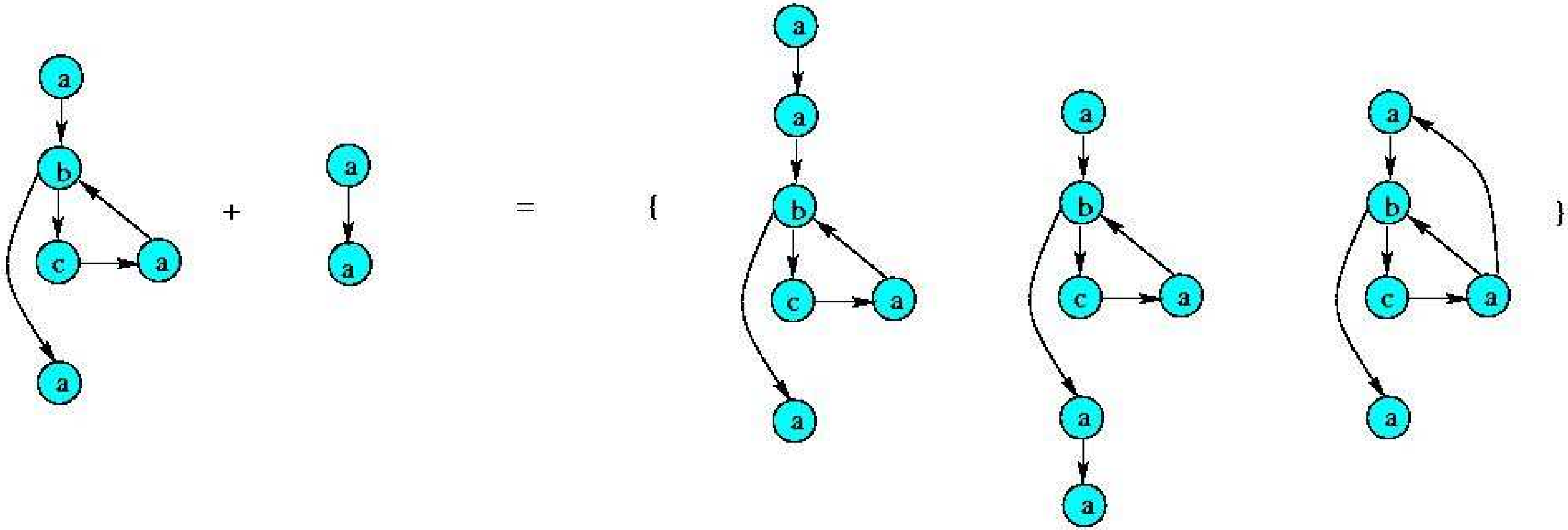
1. between two existing nodes,
2. between existing and new node.

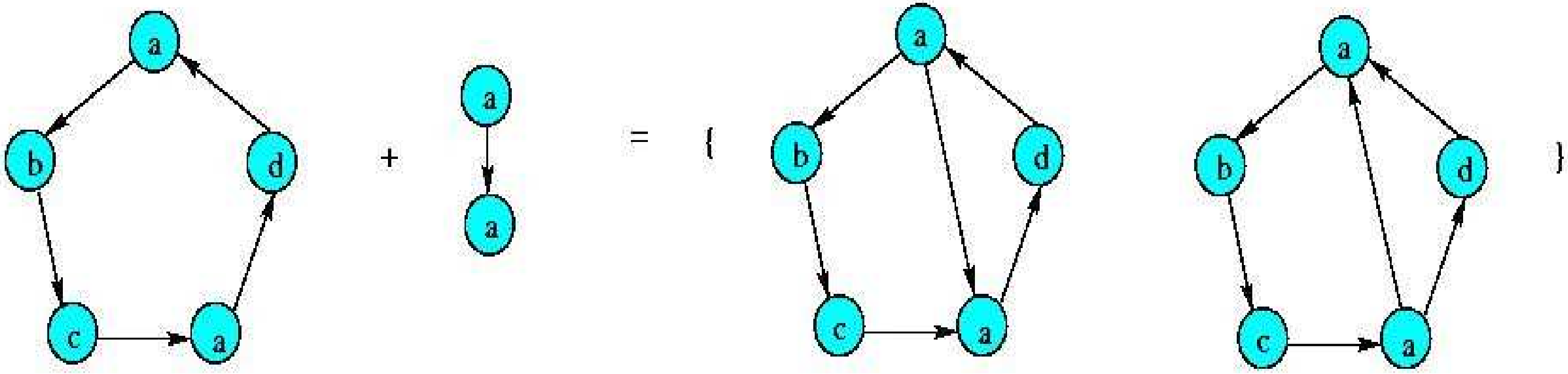
If the path is *not cyclic*, we can add edge between pair of nodes one of which is *unbalanced*:

1. between two existing unbalanced nodes,
2. between existing unbalanced and existing balanced node,
3. between existing unbalanced node and a new node.

Now each candidate is checked for its support, only the frequent ones will be extended in next iteration

Phase #1 – Example





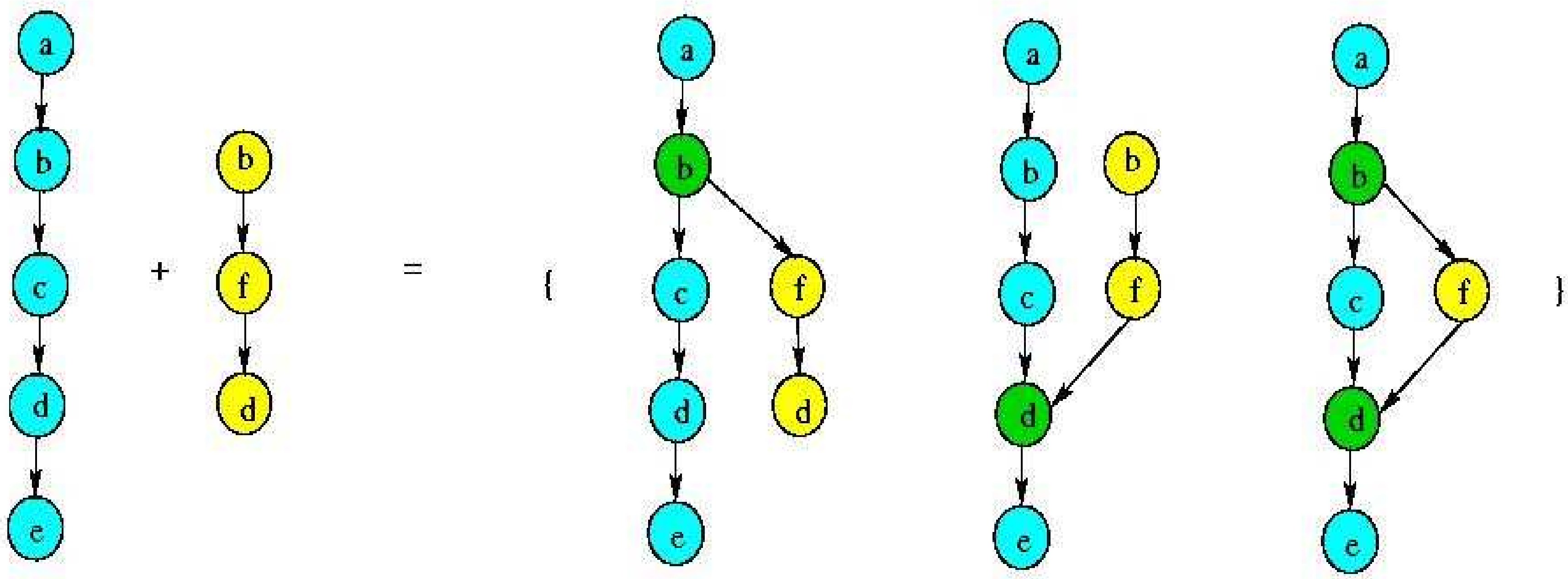
Phase #2 – 2-Path generation

Algorithm: Phase #2

1. Let L_1 be the set of all frequent paths.
   Set $C_2 \leftarrow \varnothing$, $L_2 \leftarrow \varnothing$.
2. For every pair $P_1, P_2 \in L_1$ and every possible label-preserving composition relation C on P_1 and P_2 do:
   a. If $p(\Omega(< P_1, P_2, C >))=2$, add $< P_1, P_2, C>$ to C_2.
3. Remove all tuples producing non P-minimal graphs from C_2.
4. For every $t \in C_2$ if $\Omega(t)$ is frequent, add it to L_2.

Phase #2 - Example

Join of two paths produced three graphs with path number 2



Exercise: show the composition relations

Phase #3 – overview

Phase #3: *Input* = frequent graphs with path number ***k***
Output = frequent graphs with path number ***(k+1)***

The main step:

1. find a common ***(k-1)***-subgraph of two ***k***-graphs,

/* Note, this is quite easy because only needed is finding (k-1) equal paths in the corresponding composition relations */

2. if found, join these graphs into ***(k+1)***-graph using ***bijective sum*** operation,

Additional step:

3. for bijective sum ***G*** of two graphs and two paths ***P*** and ***Q*** in which these graphs differ, find all frequent combinations of ***P*** and ***Q*** in $\boldsymbol{L_2}$, and join them with G using ***splice*** operation.
(Note the 'Splice' doesn't increase the size of the candidate graph, i.e. its still (k+1))

Phase #3 – Graphs with $p(G) \geq 3$

Algorithm: Phase #3

1. Let L_2 be the set of all frequent path pairs. Set $k \leftarrow 3$.
2. Set $C_k \leftarrow \emptyset$, $L_k \leftarrow \emptyset$.
3. For every $t_1, t_2 \in L_{k-1}$ such that $t_1 = <P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_j, \ldots, P_k, C_1>$ and $t_2 = <P_1, \ldots, P_j, \ldots, P_{j-1}, P_{j+1}, \ldots, P_k, C_2>$ do:
   a. Let $C = BS(C_1, C_2, (k)\text{-}i\text{-}j, (k)\text{-}i\text{-}j)$.
   b. Add $t = < P_1, \ldots, P_k, C >$ to C_k (if $p(\Omega(t)) = k$).
   c. For every $t_3 = < P_i, P_j, C_3 > \in L_2$,
      add $t = < P_1, \ldots, P_k, C \oplus_{i,j} C_3 >$ to C_k (if $p(\Omega(t)) = k$).
4. Remove all non P-minimal tuples from C_k.
5. Add every $t \in C_k$, where $\Omega(t)$ is frequent, to L_k.
6. If $L_k = \emptyset$, stop. Otherwise, set $k \leftarrow k+1$ and go to step 2.

Main theorem: Algorithm is Sound and Complete
i.e. it finds all and only frequent sub-graphs!

Proof outline:

Theorem 1 All frequent graphs with path number 1 are produced by phase 1 of the algorithm.

How to prove: For every path *P* and unbalanced vertex *v* of *P* there exists a vertex *u* such that $(u,v) \in E(P)$ and $P \backslash (u,v)$ is a path.

Theorem 2 All frequent graphs with path number 2 are produced by phase 2 of the algorithm.

How to prove: Each graph *G* with $p(G)=2$ can be expressed as a label-preserving composition relation on two paths from its any $\leq_P$ minimal path decomposition.

Proof outline (cont.)

Theorem 3 All frequent graphs with $p(G)>2$ are produced by phase 3 of the algorithm.

Main steps: 1. There exists two paths P,Q in minimal path decomposition of G such that $G\backslash P$ and $G\backslash Q$ are connected.

2. $G\backslash P$ and $G\backslash Q$ are also frequent and were found (by induction)
3. If P and Q are disjoint in G, using BS operation on their composition relations will produce G.
4. Otherwise, BS and $\oplus$ operations combined will produce G.

Complexity

Exponential – as the ***number*** of frequent patterns can be exponential on the size of the database (like any Apriori alg.)

Difficult tasks: (NP hard)

1. Support computation that consists of:
   a. Finding all instances of a frequent pattern in the database. (sub-graph isomorphism)
   b. Computing MIS (maximum independent set size) of an instance graph.

Relatively easy tasks:

1. Candidate set generation:
   polynomial on the size of frequent set from previous iteration,
2. Elimination of isomorphic candidate patterns:
   graph isomorphism computation is at worst exponential on the size of a pattern, not the database.

Complexity (cont.)

Why is mining in real-life databases easier ?

- real databases tend to be sparse rather than dense,
- real databases tend to have large number of different labels.

Impact on algorithm's complexity:

- the number of database subgraphs isomorphic to a given graph pattern is not exponential,
- the size of instance graph is not exponential,
- instance graphs tend to be very sparse, which makes the task of finding MIS much easier.

Additional improvements:

- approximate techniques can be used for MIS computation as user usually does not care for the *exact* support value. ([Kuramochi2004])

Experiment overview

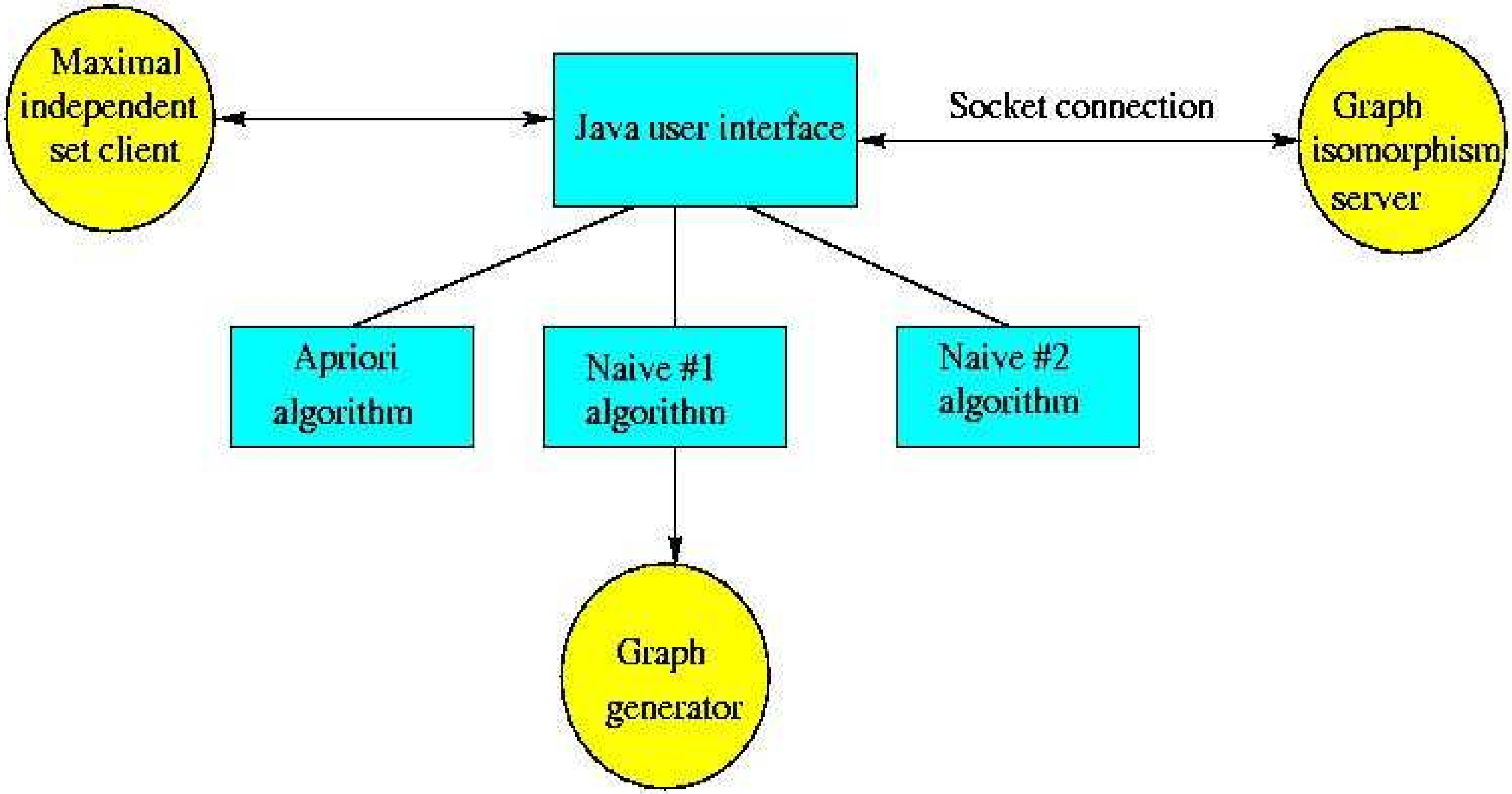
Goals of our experiments are:

- To compare our algorithm with naïve algorithms:
  - Naive1 – produce *all* graphs and compute their support (B. D. McKay "*Isomorph-free exhaustive generation*", J. of Algorithms vol. 26, 1998)
  - Naive2 – at each iteration, add *edge* to frequent graphs from previous iteration
  - FSG – without some optimizations
- To study algorithm's behavior on various graph topologies:
  - cliques
  - trees
  - sparse graphs vs dense graphs

- To study the effect of following parameters on the number of frequent patterns found:
  - size of the database
  - number of different labels

- Test algorithm on both synthetic and real-life databases

Experiments setting



Experimental results on synthetic data: trees

Notation: S – support; N – nodes; L – labels; E – edges
FP – frequent patterns
C – candidate patterns; I – isomorphism checks;
SC – support calculations; ALG - algorithm in use.

#	N	L	S	FP	ALG	C	I	SC
1	40	4	7%	15	Naive2	100	24	92
					Path	52	47	52
2	50	4	7%	16	Naive2	110	41	102
					Path	45	45	42
3	50	6	3%	37	Naive2	470	82	458
					Path	202	239	205
4	50	8	3%	27	Naive2	306	62	290
					Path	119	91	111
5	60	4	5%	15	Naive2	100	24	92
					Path	52	47	52
6	60	6	5%	44	Naive2	728	203	716
					Path	175	868	276
7	60	8	5%	14	Naive2	103	18	87
					Path	41	29	26

Experimental results on synthetic data: sparse graphs

#	N	E	L	S	FP	ALG	C	I	SC
1	40	50	4	7%	14	Naive2	60	33	52
						Path	49	55	42
2	40	50	6	5%	17	Naive2	84	48	76
						Path	59	70	54
3	50	60	6	5%	28	Naive2	355	74	343
						Path	117	185	143
4	60	80	4	4%	16	Naive2	101	31	93
						Path	56	58	56
5	60	80	6	3%	27	Naive2	265	86	253
						Pathi	120	102	110
6	70	90	8	3%	27	Naive2	252	77	236
						Path	126	98	110
7	80	100	8	3%	32	Naive2	403	74	387
						Path	149	127	141

Subsets of Movie database used in experiments

Data set #	Nodes	Edges	Labels
1	12656	13878	112
2	8337	9416	25
3	7027	7851	22
4	4730	4813	90
5	2757	2794	76
6	1293	1292	91

Experimental results on subsets of Movie database

Data set	#1	#2	#3	#4	#5	#6	Data set	#1	#2	#3	#4	#5	#6
Support	Number of frequent patterns						Support	Number of frequent patterns					
90%	3	3	3	3	2	4	20%	8	6	5	9	6	46
80%	3	3	3	3	2	5	10%	15	12	10	11	7	79
70%	4	3	3	4	2	5	9%	16	12	10	11	7	79
60%	4	3	3	4	2	9	8%	16	12	10	12	8	84
50%	5	3	3	5	2	21	7%	18	12	10	12	9	84
40%	6	4	4	5	2	32	6%	21	13	11	12	10	86
30%	7	5	4	8	5	34	5%	22	14	11	16	11	86

Comparison

Our algorithm vs naive ones

- ✓ Naive1 algorithm does not work on graphs with ≥ 10 nodes
- ✓ Our algorithm produces less candidate patterns and therefore performs less support computations than Naive2 algorithm.

Trees vs sparse graphs

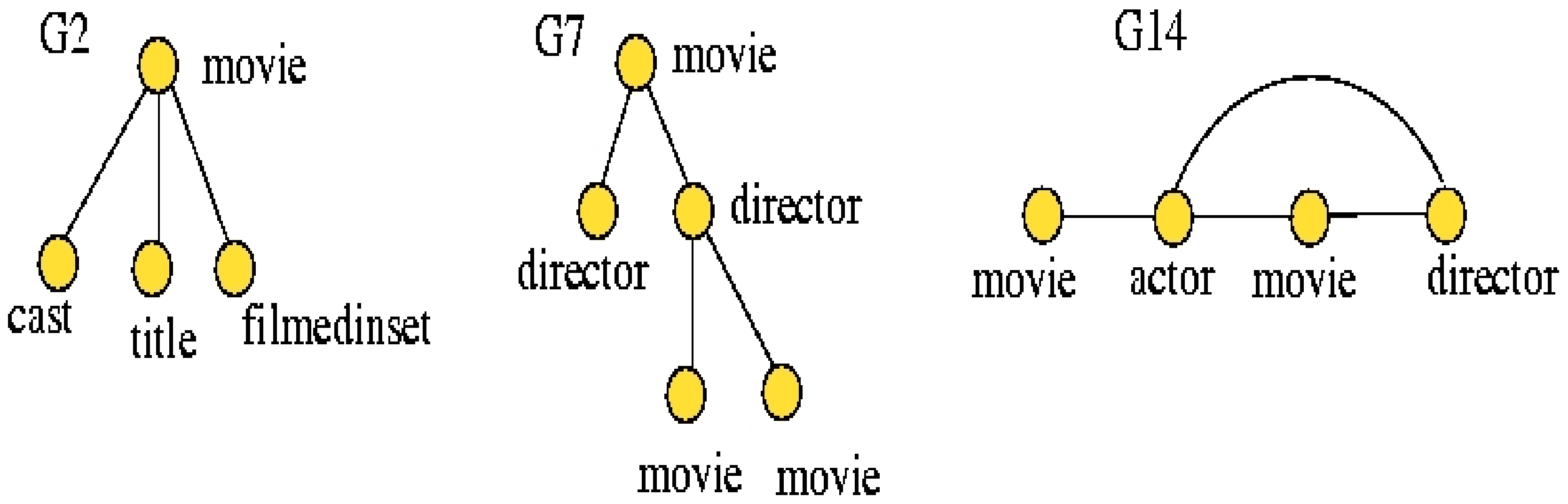
- ✓ Support computation is easier for trees
- ✓ Less candidate patterns are generated for trees

Synthetic vs real-life data

- ✓ Synthetic graphs are not very regular. When increasing number of labels, the chance of finding non-trivial frequent graph patterns decreases drastically.

  Large real-life graph databases are highly regular and contain complex frequent graph patterns.

Pattern examples in Movies database



Further Evaluation

- ✓A full comparison with FSG (not so simple because of the different support definitions) – appeared in TKDE Nov 2006

- ✓Path algorithm was better for single graph setting and
- ✓comparable for transaction setting

Conclusions

- ✓ An Apriori-like algorithm for mining graph patterns that uses edge-disjoint paths as building blocks has been constructed.

- ✓ A problem of defining support measure for semi-structured data was addressed.

- ✓ An experimental analysis of the algorithm was conducted.

Papers in ICDM2002 and ICDE2004 and journal papers in TKDE2006 and DMKD2006

Future work

- Usage of building blocks other than edge disjoint paths, such as trees.

- Using Apriori-TID technique at the advanced stages of the search.

- Treat patterns that have high degree of resemblance, such as bisimilar patterns, as representatives of their equivalence classes and generate representatives of each class instead of the full search.

- Find additional examples of admissible support measures.

- Take into account topological properties of a database graph
- while computing support.

Additional Approaches for Single Graph Setting

- BFS Approach
  - hSiGram
- DFS Approach
  - vSiGram
- Both use approximations of the MIS measure

M. Kuramochi and G. Karypis

Finding Frequent Patterns in a Large Sparse Graph

In Proc. Of SIAM 2004.

Partially labeled patterns in semi-structured data – Vanetik et. Al. - ICDE2004

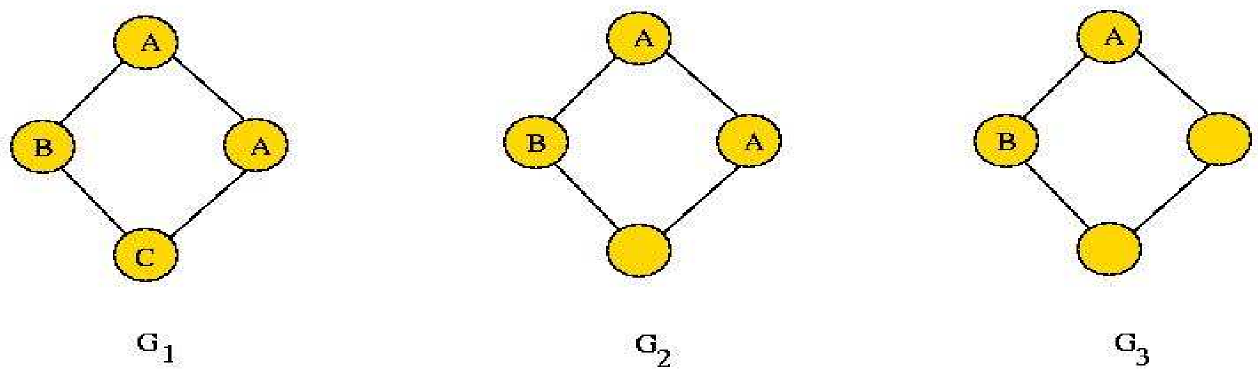
Partially labeled patterns

A graph pattern *G=(V,E)* is partially labeled if exists $v \in V$ without a label (denoted by *label(v)=?*). Otherwise, a graph pattern is called fully labeled.

Pattern *G* is weaker than pattern *H*, denoted by $G \leq_w H$, if

1. ***G* is isomorphic to *H*,**
2. **all nodes that have a label in *G* have the same label in *H*,**
3. **there exist node(s) that have a label in *H* but do not have a label in *G*.**

Example. Here, $G_3 \leq_w G_2 \leq_w G_1$



Partially labeled patterns

The algorithm.

The algorithm adapts the Path algorithm to find only the strongest and maximal frequent partially labeled graphs – see paper for details

Outline

- Basic concepts of Data mining and Association rules
  - Apriori algorithm
- Motivation for Graph mining
- Applications of Graph Mining
- Mining Frequent Subgraphs - Transactions
  - BFS/Apriori Approach (FSG and others)
  - DFS Approach (gSpan and others)
  - Diagonal Approach
  - Greedy Approach
- Mining Frequent Subgraphs – Single graph
  - The support issue
  - The Path-based algorithm
  - Constraint-based and other algorithms

Graph Pattern Explosion Problem

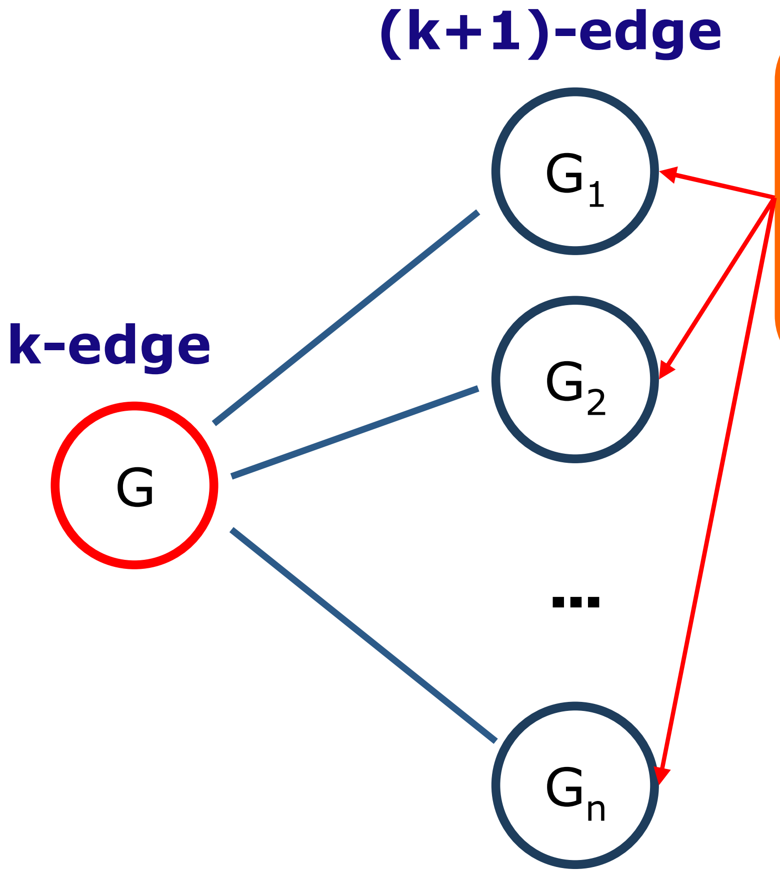
- If a graph is frequent, all of its subgraphs are frequent ─ **the Apriori property**
- An **n**-edge frequent graph may have 2^n subgraphs
- Among **422** chemical compounds which are confirmed to be active in an AIDS antiviral screen dataset, there are **1,000,000** if the minimum support is 5%

Closed Frequent Graphs

- Motivation: Handling graph pattern explosion problem
- Closed frequent graph
  - A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G
- If some of G’s subgraphs have the same support, it is unnecessary to output these subgraphs (nonclosed graphs)
- Note close item-sets algorithms (e.g. GenMax and MaxMiner)

CLOSEGRAPH (Yan & Han, KDD'03)

A Pattern-Growth Approach

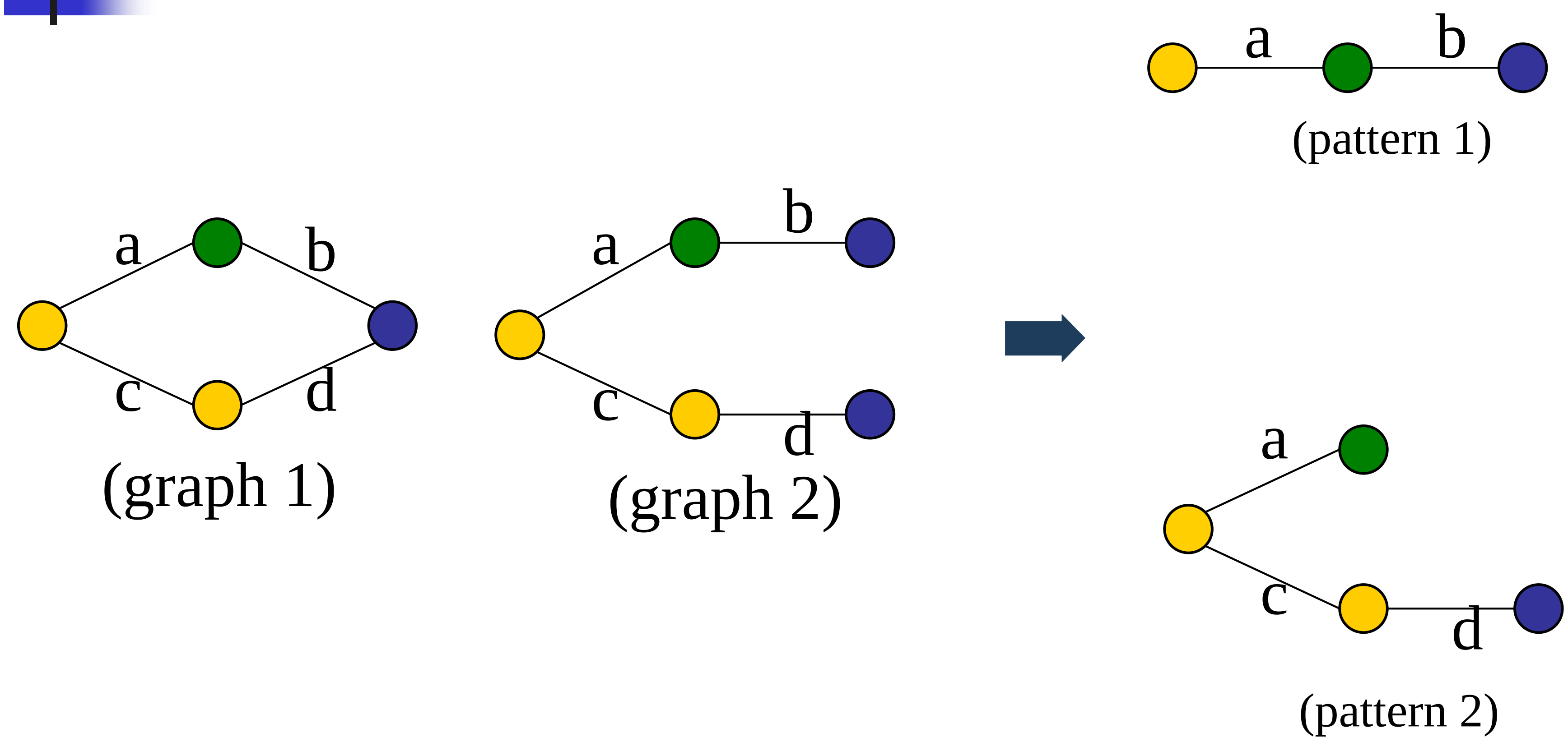


At what condition, can we stop searching their children i.e., early termination?

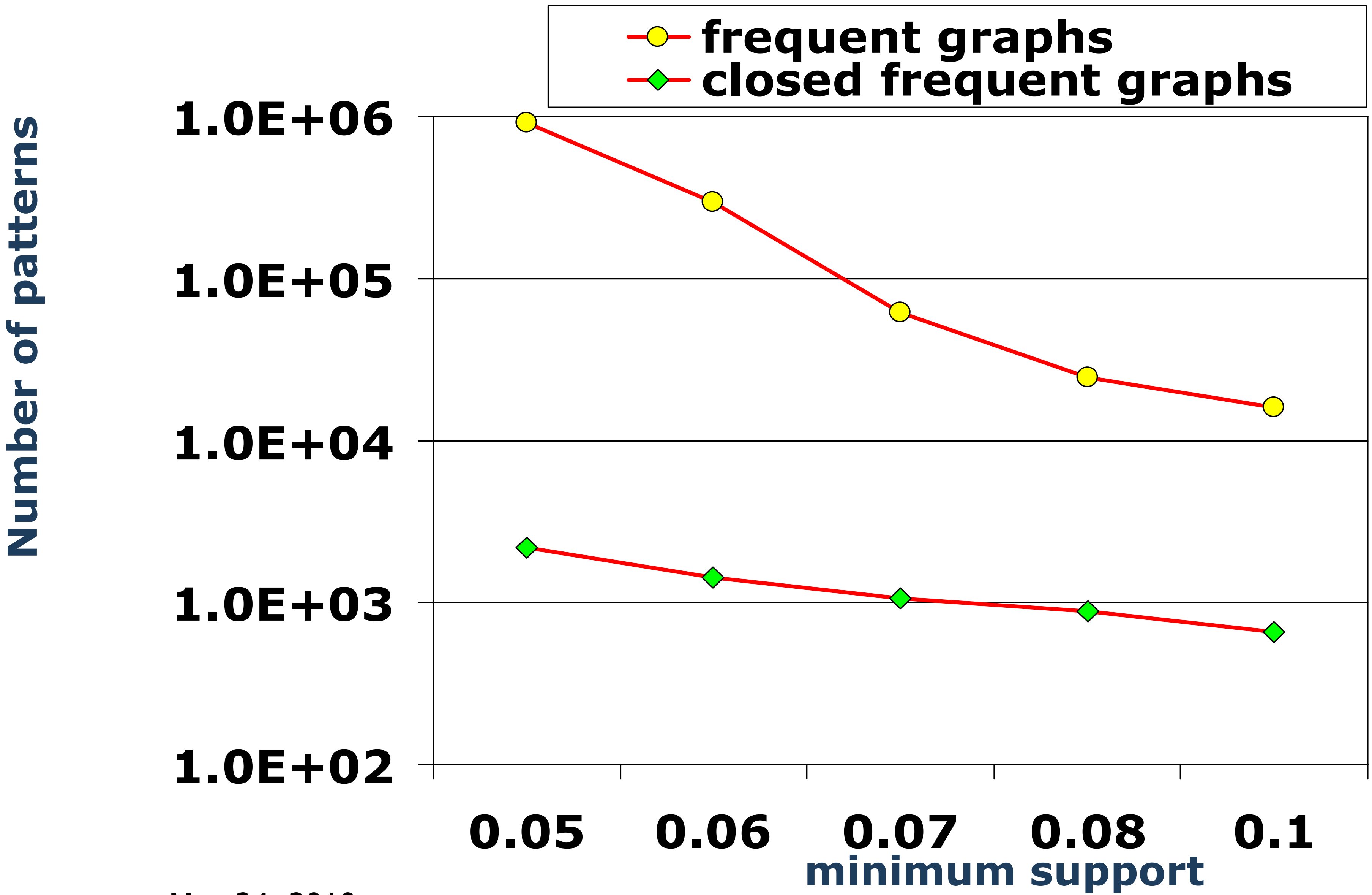
If G and G' are frequent, G is a subgraph of G'. If **in any part of the graph in the dataset where G occurs, G' also occurs**, then we need not grow G, since none of G's children will be closed except those of G'.

(See figure 4 in paper)

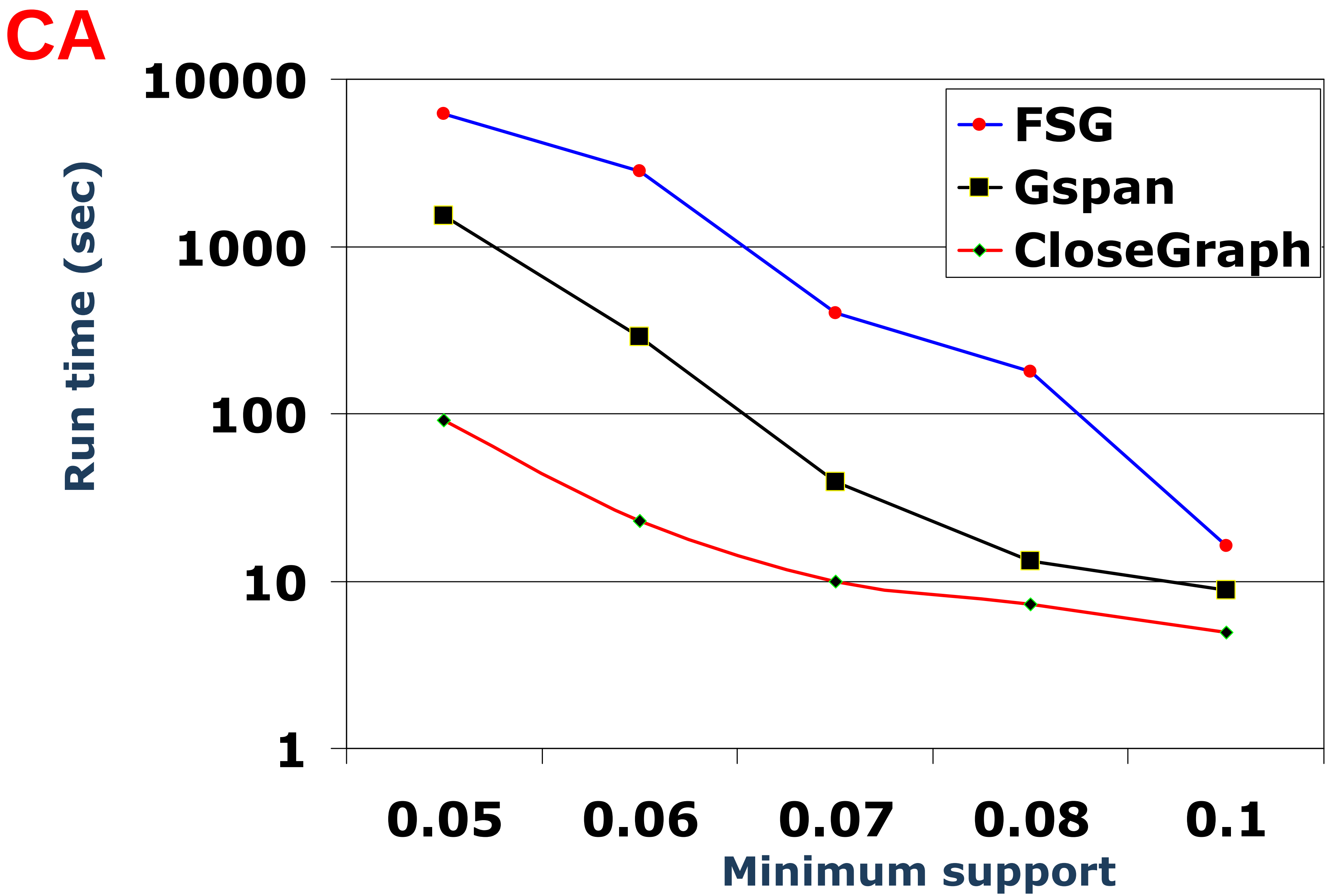
Handling Tricky Exception Cases



Number of Patterns: Frequent vs. Closed



Runtime: Frequent vs. Closed



Constraint-Based Graph Pattern Mining

- **F. Zhu, X. Yan, J. Han, and P. S. Yu, "gPrune: A Constraint Pushing Framework for Graph Pattern Mining", PAKDD'07**
- There are often various kinds of constraints specified for mining graph pattern P, e.g.,
  - max_degree(P) ≥ 10
  - diameter(P) ≥ δ
- Most constraints can be pushed deeply into the mining process, thus greatly reduces search space
- Constraints can be classified into different categories
  - Different categories require different pushing strategies

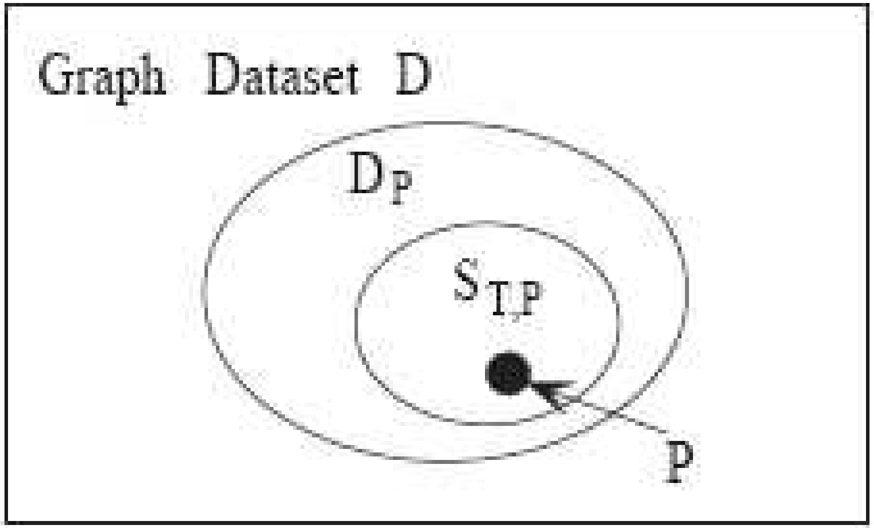
Pattern Pruning vs. Data Pruning

- Pattern Pruning

  Pruning a pattern saves the mining associated with all the patterns that grow out of this pattern, which is D_P

- Data Pruning

  Data pruning considers both the pattern P and a graph G $\in D_P$, and data pruning saves a portion of D_P



***D_P* is the data search space of a pattern *P*. *$S_{T,P}$* is the portion of *D_P* that can be pruned by data pruning.**

Pruning Properties Overview

- Pruning property: A property of the constraint that helps prune either the pattern search space or the data search space.
- Pruning Pattern Search Space
  - Strong P-antimonotonicity
  - Weak P-antimonotoniciy
- Pruning Data Search Space
  - Pattern-separable D-antimonotonicity
  - Pattern-inseparable D-antimonotonicity

Pruning Pattern Search Space

- Strong P-antimonotonicity
  - A constraint C is ***strong P-antimonotone*** if a pattern violates C, all of its super-patterns do so too
  - E.g., C: "The pattern is acyclic"
- Weak P-antimonotoniciy
  - A constraint C is ***weak P-antimonotone*** if a graph P (with at least k vertices) satisfies C, there is at least one subgraph of P with one vertex less that satisfies C
  - E.g., C: "The density ratio of pattern P ≥ 0.1", i.e.,

$$\frac{|E(P)|}{|V(P)|(|V(P)|-1)/2} \geq 0.1$$

A densely connected graph can always be grown from a smaller densely connected graph with one vertex less

Pruning Data Space (I): Pattern-Separable D-Antimonotonicity

- Pattern-separable D-antimonotonicity

  A constraint C is *pattern-separable D-antimonotone* if a graph G cannot make P satisfy C, then G cannot make any of P's super-patterns satisfy C

  - C: "the number of edges ≥ 10", or "the pattern contains a benzol ring".
- Use this property: *recursive data reduction*
  - A graph is pruned from the data search space for pattern P if G cannot satisfy this C

The gprune algorithm

Algorithm 1 PatternGrowth

$S \leftarrow \{P\}; F \leftarrow F \bigcup \{P\}; S_t \leftarrow \emptyset$
while $S \neq \emptyset$;
  $Q \leftarrow pop(S)$;
  for each graph $G \in D_Q$
    Augment Q and save new patterns in S_t;
    Check pattern pruning on each $P \in S_t$;
  for each augmented pattern $Q' \in S_t$
    Construct support data space $D_{Q'}$ for Q';
    Check data pruning on $D_{Q'}$;
  $F \leftarrow F \bigcup S_t$;
  $S \leftarrow S \bigcup S_t$;
return F;

Graph Constraints: A General Picture

Constraint	strong P-antimonotone	weak P-antimonotone	pattern-separable D-antimonotone	pattern-inseparable D-antimonotone
$Min\_Degree(G) \geq \delta$	No	No	No	Yes
$Min\_Degree(G) \leq \delta$	No	Yes	No	Yes
$Max\_Degree(G) \geq \delta$	No	No	Yes	Yes
$Max\_Degree(G) \leq \delta$	Yes	Yes	No	Yes
$Density\_Ratio(G) \geq \delta$	No	Yes	No	Yes
$Density\_Ratio(G) \leq \delta$	No	Yes	No	Yes
$Density(G) \geq \delta$	No	No	No	Yes
$Density(G) \leq \delta$	No	Yes	No	Yes
$Size(G) \geq \delta$	No	Yes	Yes	Yes
$Size(G) \leq \delta$	Yes	Yes	No	Yes
$Diameter(G) \geq \delta$	No	Yes	No	Yes
$Diameter(G) \leq \delta$	No	No	No	Yes
$EdgeConnectivity(G) \geq \delta$	No	No	No	Yes
$EdgeConnectivity(G) \leq \delta$	No	Yes	No	Yes
G contains P (e.g., P is a benzol ring)	No	Yes	Yes	Yes
G does not contain P (e.g., P is a benzol ring)	Yes	Yes	No	Yes

Important References

[1] X. Yan and J. Han, "*gSpan: Graph-Based Substructure Pattern Mining*", ICDM'02

[2] [5] M. Kuramochi, G. Karypis, "*An Efficient Algorithm for Discovering Frequent Subgraphs*" IEEE TKDE, September 2004 (vol. 16 no. 9)

[3] N. Vanetik, E.Gudes, and S. E. Shimony, *Computing Frequent Graph Patterns from Semistructured Data*, Proceedings of the 2002 IEEE ICDM'02 and TKDE 2006

[4] Kuramochi et. al- Finding Frequent Patterns in a Large Sparse Graph [SDM2004])

[5] X. Yan and J. Han, "*CloseGraph: Mining Closed Frequent Graph Patterns*", KDD'03

[6] F. Zhu, X. Yan, J. Han, and P. S. Yu, "gPrune: A Constraint Pushing Framework for Graph Pattern Mining", PAKDD'07

[7] Wang et. Al. Scalable mining of large Disk-based graph databases, KDD 2004