BITS Pilani
Hyderabad Campus

BITS Pilani

Dr. Manik Gupta
Assistant Professor
Department of CSIS

innovate achieve lead

BITS Pilani
Hyderabad Campus

Data Mining (CS F415)
Lecture 15 – Subgraph Mining

Thursday, 20th February 2020

Research Project Mid-Semester Demos

Please fill in your preferences for the Demo next week:

https://doodle.com/poll/i6ebv5nfxasg3emf

Few important things (PLEASE FOLLOW THEM!):

- Please be on time else your assessment will not be carried out. All the team members need to be present.
- Please do not contact us regarding slot swaps. Coordinate amongst yourself and inform us via email.
- Please carry your own laptops and make sure the demo runs as well as report/slides are present and working.
- Create 5 slides (time limit 5 min!!)
  - Team work division
  - Project and data overview
  - Reasons for choosing the techniques used
  - Shortcomings
  - Next steps

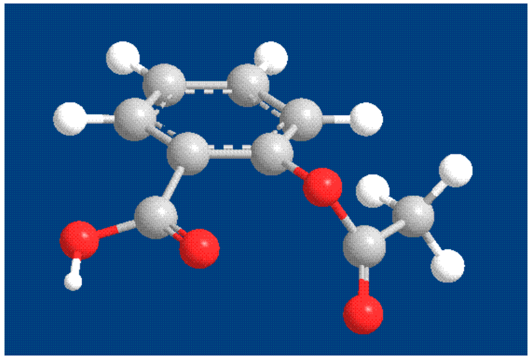
Today's Agenda

- Subgraph Mining

Modeling Data With Graphs…

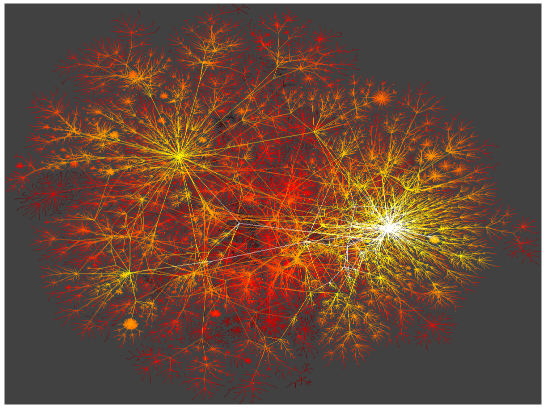
Going Beyond Transactions

Graphs are suitable for capturing arbitrary relations between the various elements.

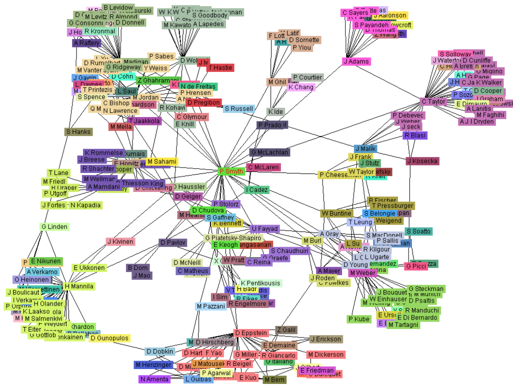


Aspirin

Yeast protein interaction network



Internet



Co-author network

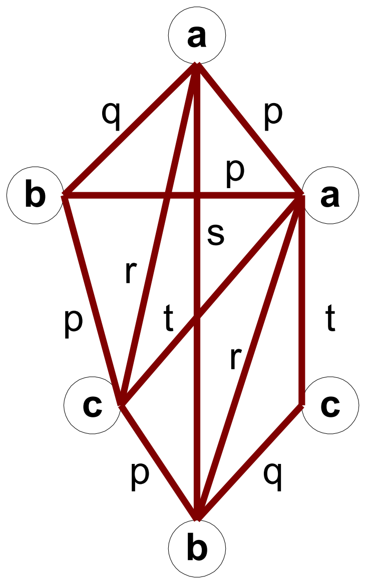
Frequent Subgraph Mining

- Extend association rule mining to find frequent subgraphs
- Useful for
  - Web Mining
  - Computational chemistry
  - Bioinformatics
  - Spatial data sets

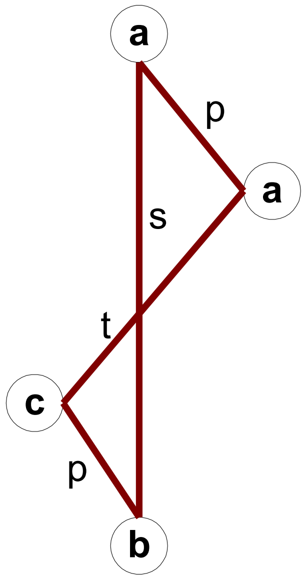
Application	Graphs used to analyze	Vertices	Edges
Web mining	Web browsing patterns	Web pages	Hyperlinks between pages
Computational chemistry	Structure of chemical compounds	Atoms	Bonds
Networking	Internet routing	Server computers	Interconnection between servers
Bioinformatics	Gene/protein interaction	Genes/proteins	Regulatory relations, physical binding

Graph Definitions

- A graph G = (V,E) is composed of vertices (nodes) V and a set of edges E.
- In labeled graphs, vertices, edges or both can have labels describing them and differentiating them from each other
- Graph G’ =(V’,E’) is a subgraph of G = (V,E) if V’ is a subset of V and E’ is a subset of E



(a) Labeled Graph

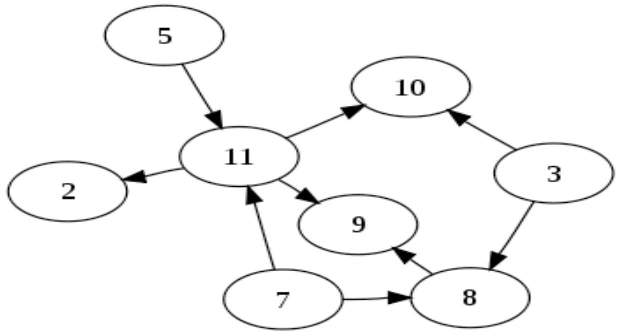


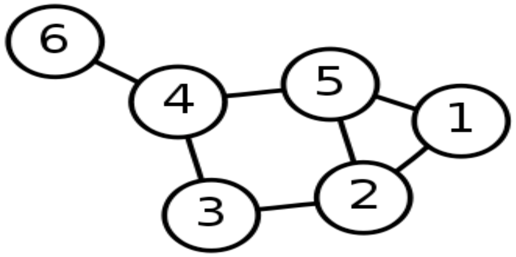
(b) Subgraph

- Graph contains 6 vertices and 11 edges .
- The subgraph contains only 4 of the 6 vertices and 4 of the 11 edges in the original graph.

Directed and Undirected Graphs

- Graph is *directed* if the edges are *oriented* (denoted by arrowhead), i.e. edge (u,v) is different object from edge (v,u)
- Graph is undirected if edges have no orientation (u,v) and (v,u) denote the same object
- A graph is connected if there exists a path between every pair of vertices in the graph.
- We concentrate on undirected, connected graphs

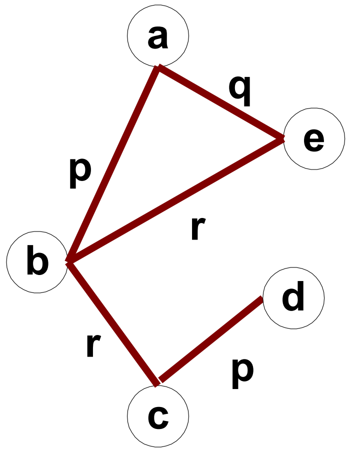




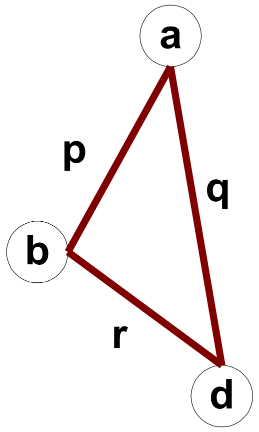
Representing Graphs as Transactions

- One approach to mine subgraphs efficiently is to transform the graph dataset into a transaction database
  - Each combination of edge label with corresponding vertex labels is defined as an item
  - The width of the transaction is the number of edges in the graph

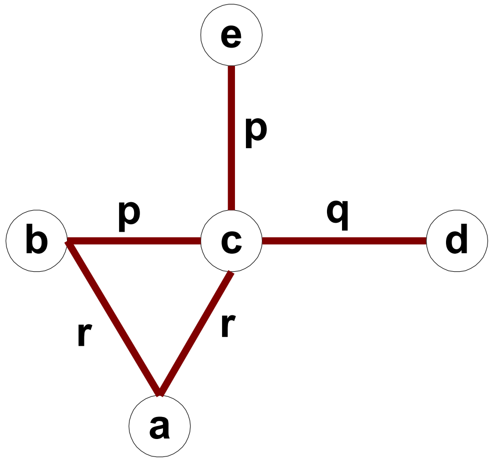
Representing Graphs as Transactions



G1



G2

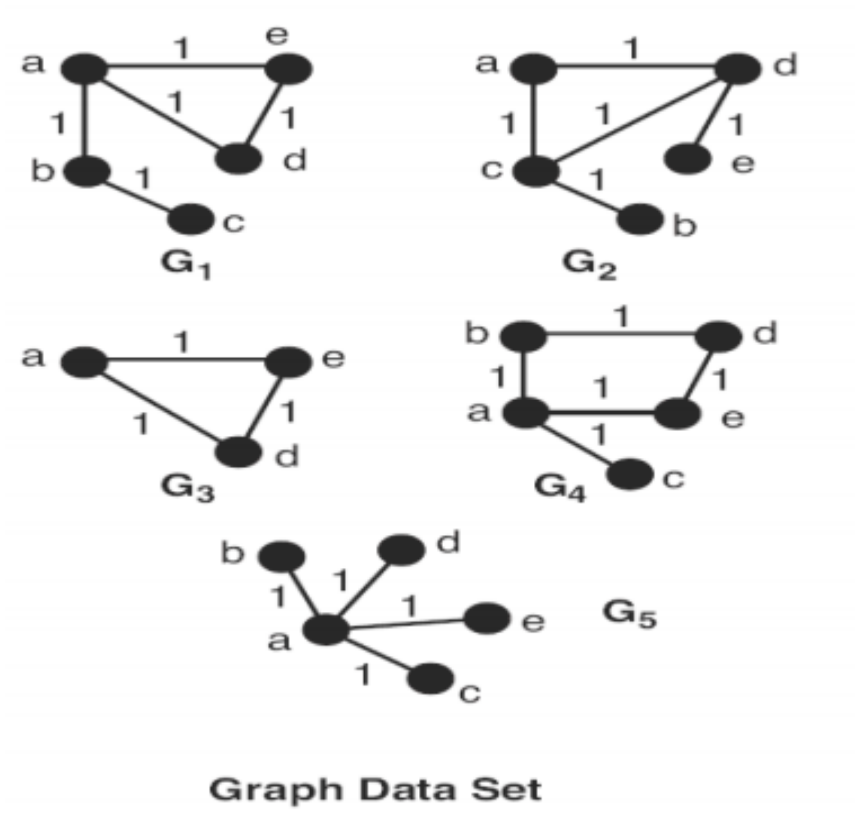


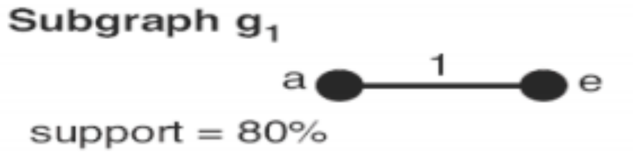
G3

	(a,b,p)	(a,b,q)	(a,b,r)	(b,c,p)	(b,c,q)	(b,c,r)	…	(d,e,r)
G1	1	0	0	0	0	1	…	0
G2	1	0	0	0	0	0	…	0
G3	0	0	1	1	0	0	…	0
G3	…	…	…	…	…	…	…	…

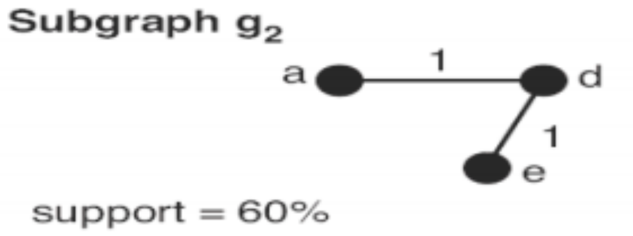
Support of a subgraph

Given a collection of graphs **G**, the support of subgraph g is defined as the fraction of all graphs that contain g as its subgraph

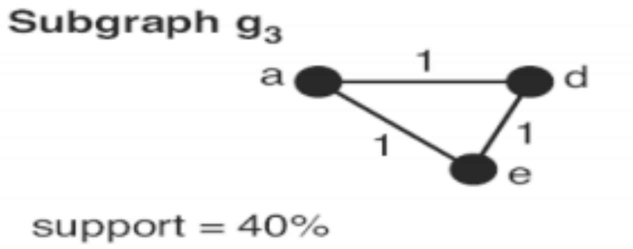




g1 is a subgraph of G1, G3, G4, and G5.



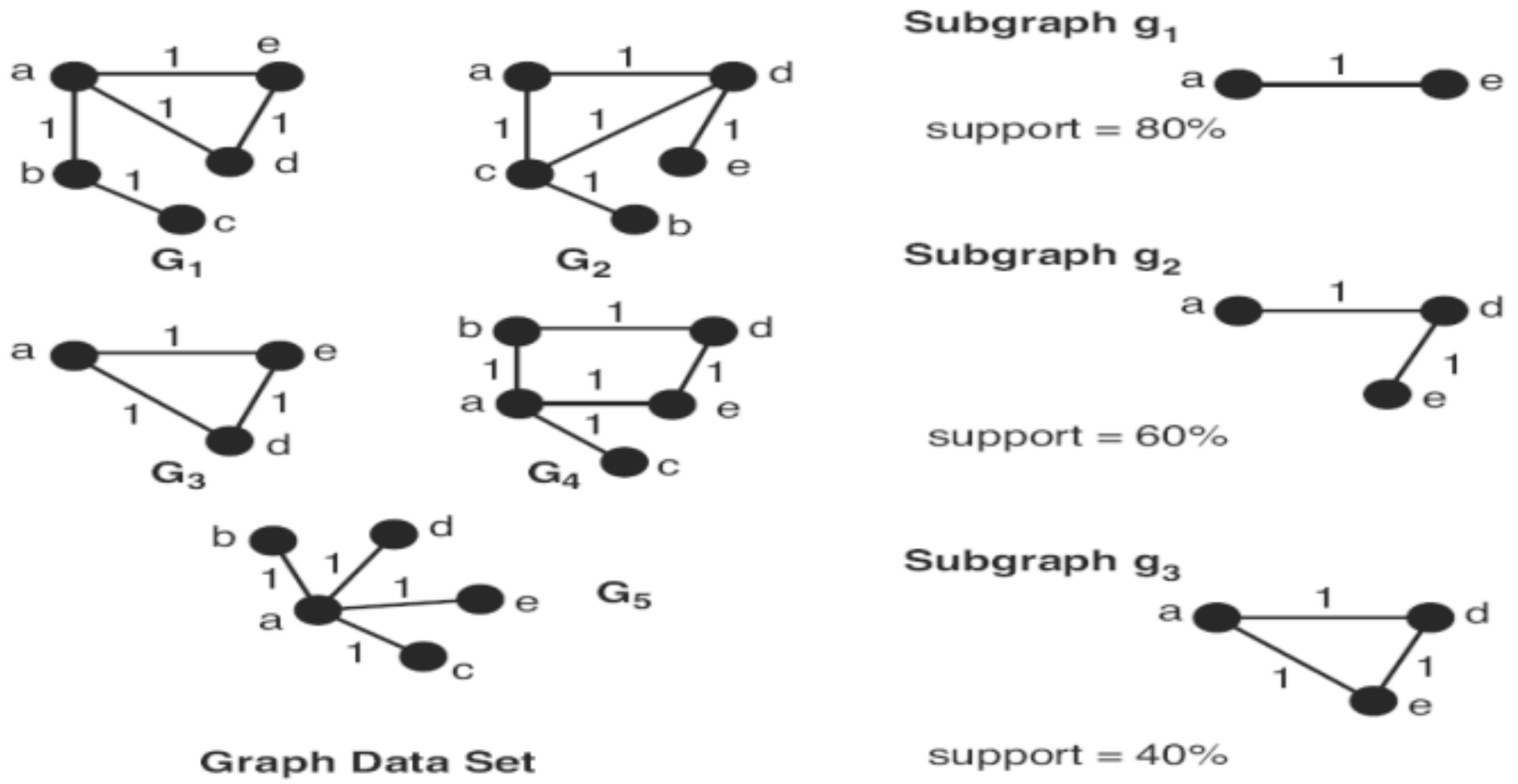
g2 is a subgraph of G1, G2, and G3



g3 is a subgraph of G1 and G3.

Frequent subgraph mining-Definition

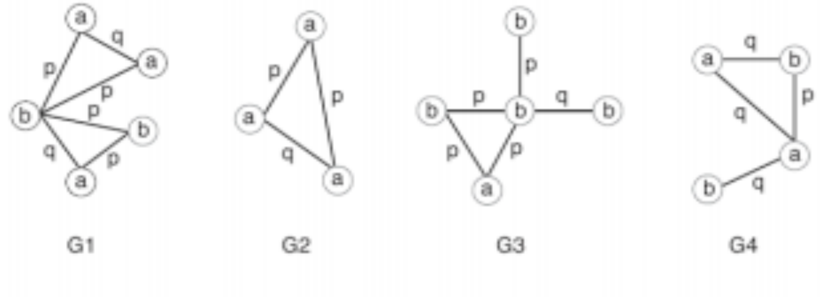
- Given a set of graphs **G** and a support threshold *minsup*, the goal is to find all subgraphs g with support s(g) at least *minsup*



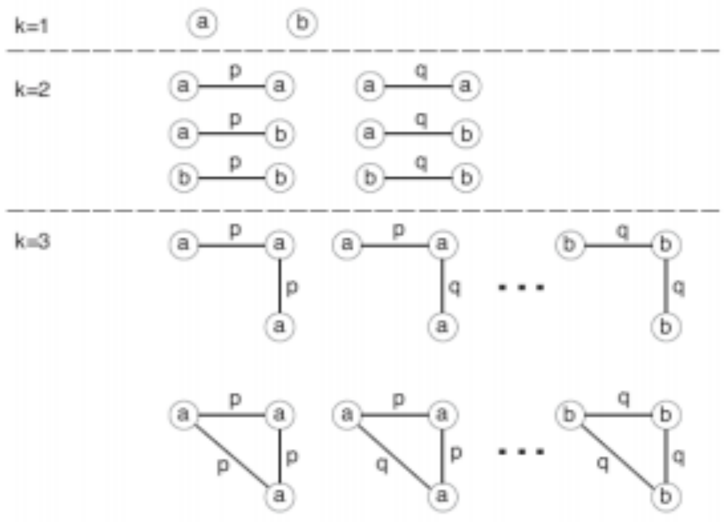
Brute-force method

vertex labels are chosen from the set {a, b}
edge labels are chosen from the set {p, q}

- Generate all connected subgraphs, count the supports, and prune
- Problem: exponential number of subgraphs
- Considerably higher number of subgraphs than itemsets, given the same items (=node labels)
  - An item can appear only once in an itemset but many times in a subgraph
  - Given a fixed set of nodes, edges can be organized and labeled in many ways to create a set of different subgraphs



(a) Example of a graph data set.



(b) List of connected subgraphs.

Apriori-like approach for frequent subgraph mining

Try to follow the usual Apriori scheme:

1. Find frequent 1-subgraphs
2. Repeat until no more frequent subgraphs are found-
   - Candidate generation

     Use frequent (*k-1*)-subgraphs to generate candidate *k*-subgraph
   - Candidate pruning

     Prune candidate subgraphs that contain infrequent (*k-1*)-subgraphs
   - Support counting

     Count the support of each remaining candidate
   - Eliminate candidate *k*-subgraphs that are infrequent

Candidate generation

- Goal: Merge a pair of k-1-subgraphs to create a k-subgraph
- First need to define k – size of the subgraph
  - Number of vertices: merge two subgraphs that have k-1 vertices
  - Number of edges: merge two subgraphs that have k-1 edges
- How to avoid generating the same subgraph many times
  - Require that the k-1 subgraphs share a common k-2 subgraph, called a **core**

Candidate Generation Methods

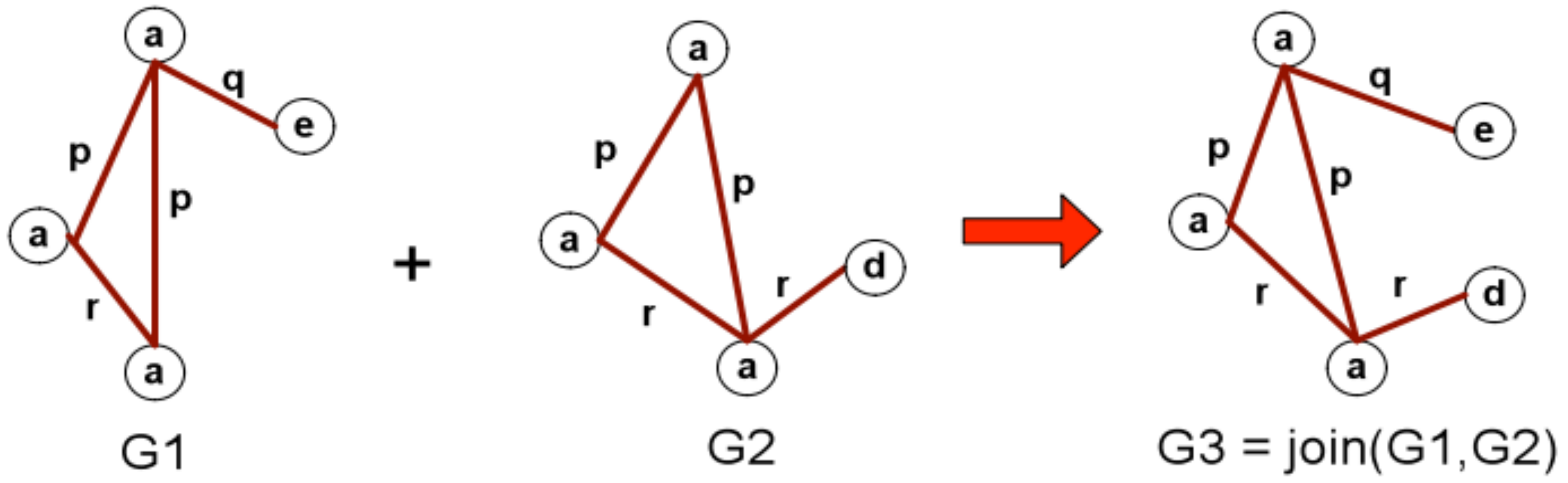
- Approach of iteratively expanding a subgraph by adding an extra vertex is known as vertex growing.
- Approach of adding an extra edge to the existing subgraphs is known as edge growing.

Candidate Generation: How is it different from itemset mining

- In Apriori:
  - Merging two frequent *k*-itemsets will produce a candidate (*k* +*1*)-itemset
- In frequent subgraph mining
  - Merging two frequent *k*-subgraphs will in general produce more than one candidate (*k*+*1*)-subgraph

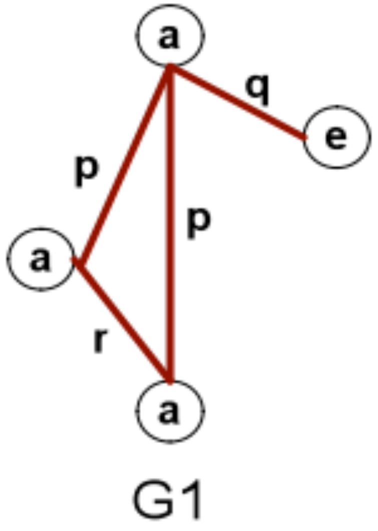
Candidate generation via vertex growing

- Generate a candidate by merging two subgraphs (G1, G2) that have a common core (subgraph of k-2 vertices) plus 1 extra vertex each
- A set of candidates will be generated that differ by one edge (d,e) and its label: G3 below is the candidate without the edge (d,e)



Adjacency matrix representation

- The vertex-growing approach can be viewed in terms of combining adjacency matrices of the subgraphs
- In our adjacency matrix representation
  - Rows and columns correspond to nodes
    - non-zero cells along a row (column) correspond to neighbors
  - Cells correspond to edges
    - cell contains edge label (or zero if no edge)



$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix}$$

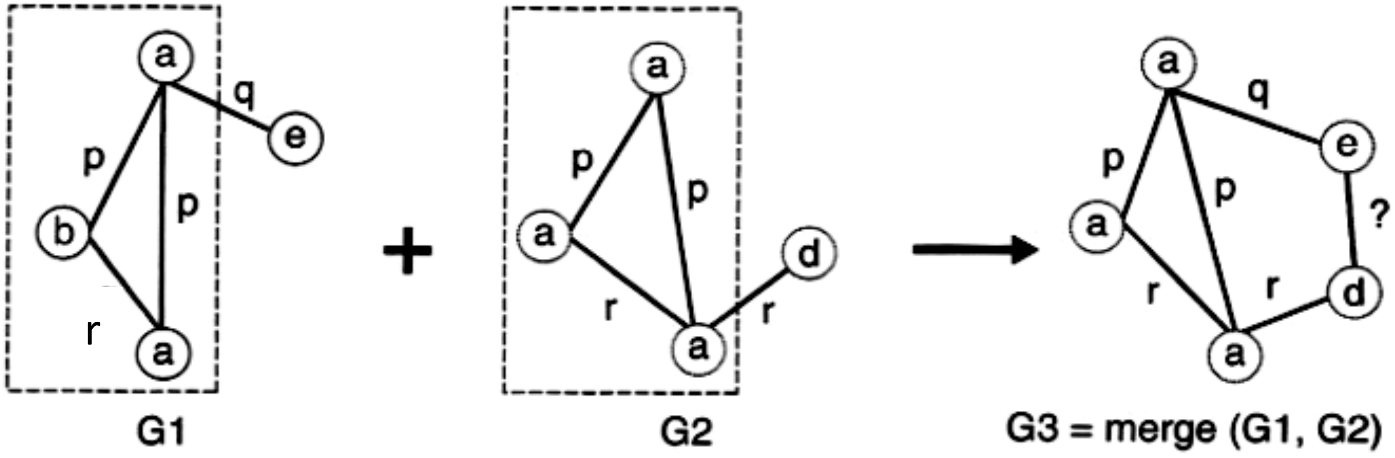
Candidate generation via vertex growing

- Adjacency Matrix Representation
  - M(i, j) in the matrix contains
    - either the label of the edge connecting between the vertices v_i and v_j
    - or zero, if there is no edge between them.
- The vertex-growing approach can be viewed as the process of generating a $k \times k$ adjacency matrix by combining a pair of $(k - 1) \times (k - 1)$ adjacency matrices.

Subgraph Merging Procedure via Vertex Growing

- An adjacency matrix $M^{(1)}$ is merged with another matrix $M^{(2)}$ if the **submatrices obtained by removing the last row and last column of $M^{(1)}$ and $M^{(2)}$ are identical to each other.**
- The resulting matrix is the **matrix $M^{(1)}$, appended with the last row and last column of matrix $M^{(2)}$.**
- The remaining entries of the new matrix are either **zero** or **replaced by all valid edge labels** connecting the pair of vertices.

Multiplicity of Candidates in vertex growing

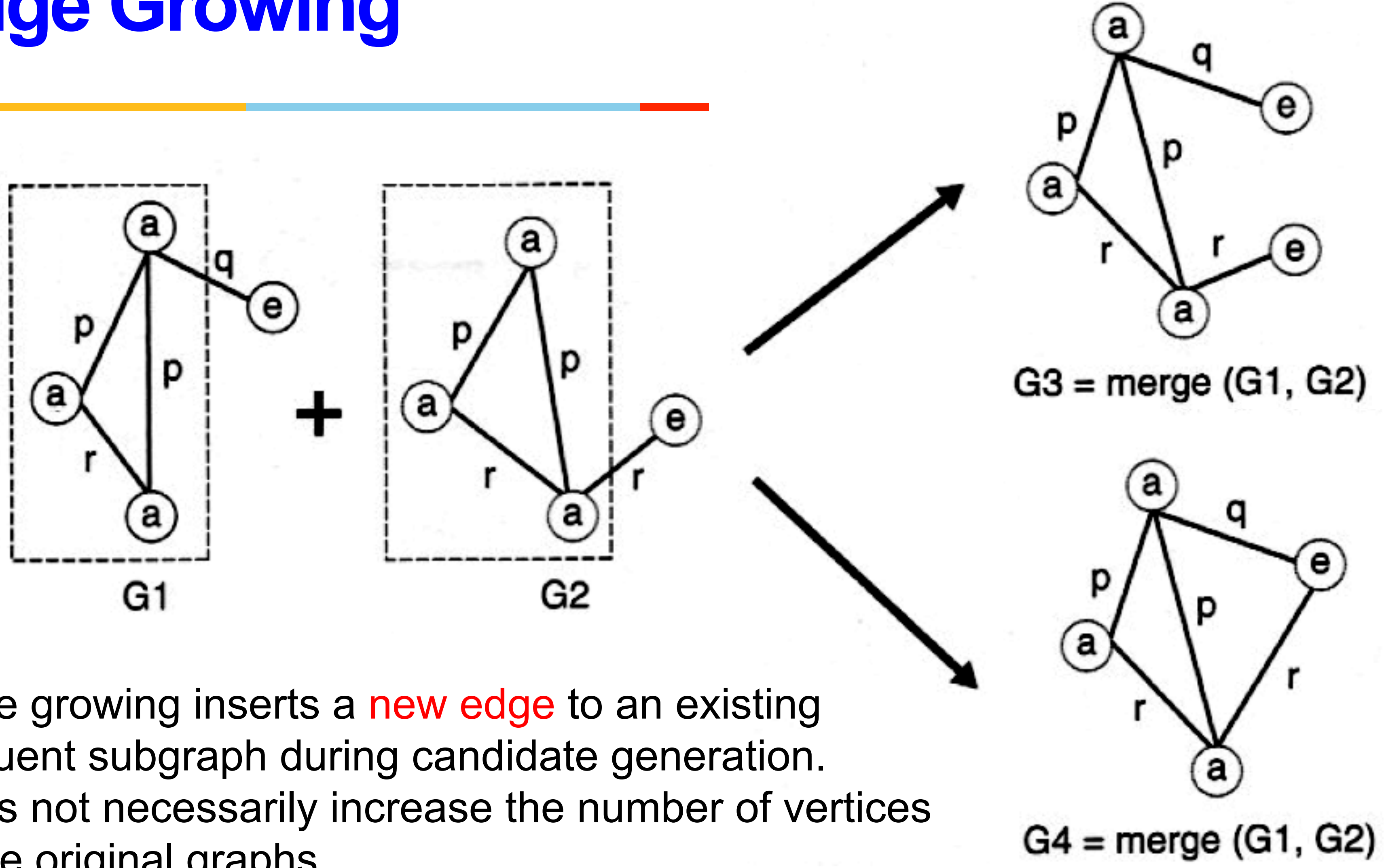


$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix} \quad M_{G2} = \begin{pmatrix} 0 & p & p & 0 \\ p & 0 & r & 0 \\ p & r & 0 & r \\ 0 & 0 & r & 0 \end{pmatrix} \quad M_{G3} = \begin{pmatrix} 0 & p & p & q & 0 \\ p & 0 & r & 0 & 0 \\ p & r & 0 & 0 & r \\ q & 0 & 0 & 0 & ? \\ 0 & 0 & r & ? & 0 \end{pmatrix}$$

- The number of edges in G3 depends on whether the vertices d and e are connected.
- Since the edge label is unknown, we need to consider all possible edge labels for (d, e), thus increasing the number of candidate subgraphs substantially.

The resulting matrix is the first matrix, appended with the last row and last column of the second matrix. The remaining entries of the new matrix are either zero or replaced by all valid edge labels connecting the pair of vertices.

Edge Growing



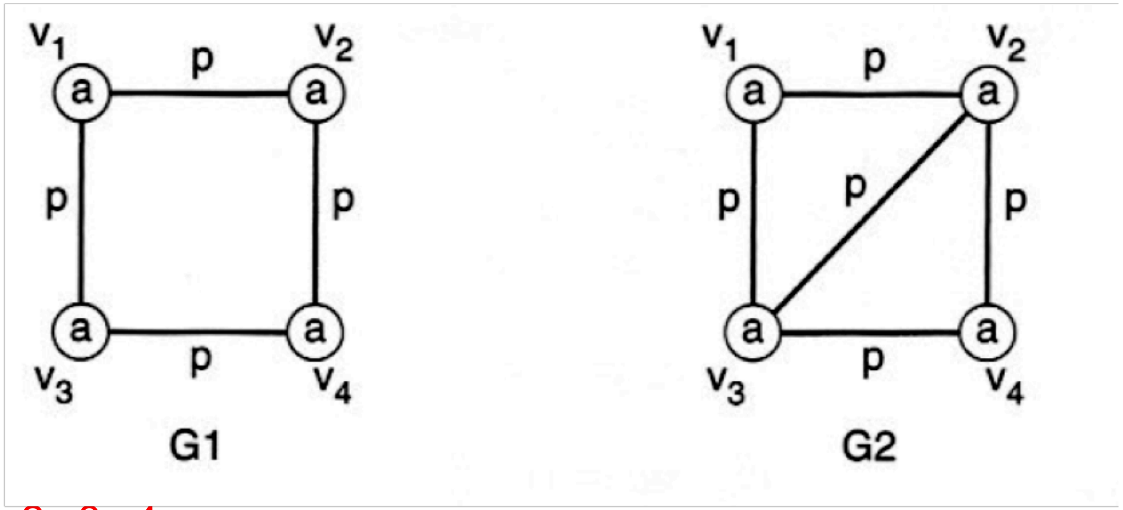
- Edge growing inserts a new edge to an existing frequent subgraph during candidate generation.
- Does not necessarily increase the number of vertices in the original graphs.
- Criterion for merging is **topological equivalence of the core**

Subgraph Merging Procedure via Edge Growing

- A frequent subgraph $g^{(1)}$ is merged with another frequent subgraph $g^{(2)}$ only if the **subgraph obtained by removing an edge from $g^{(1)}$ is topologically equivalent to the subgraph obtained by removing an edge from $g^{(2)}$**
- After merging, the resulting candidate is the **subgraph $g^{(1)}$ appended with the extra edge from $g^{(2)}$**

Topological equivalence

- Two vertices are topologically equivalent if they have:
  - The same label and
  - The same number and label of edges incident to them.



v1,v2,v3,v4 are topologically equivalent

v1,v4 are topologically equivalent
v2,v3 are topologically equivalent

- If a new edge is attached to any one of the four vertices, the resulting graph will look the same.
- v1 is not topologically equivalent to v2 because the number of edges incident on the vertices is different. Therefore, attaching a new edge to v1 results in a different graph than attaching the same edge to v2.

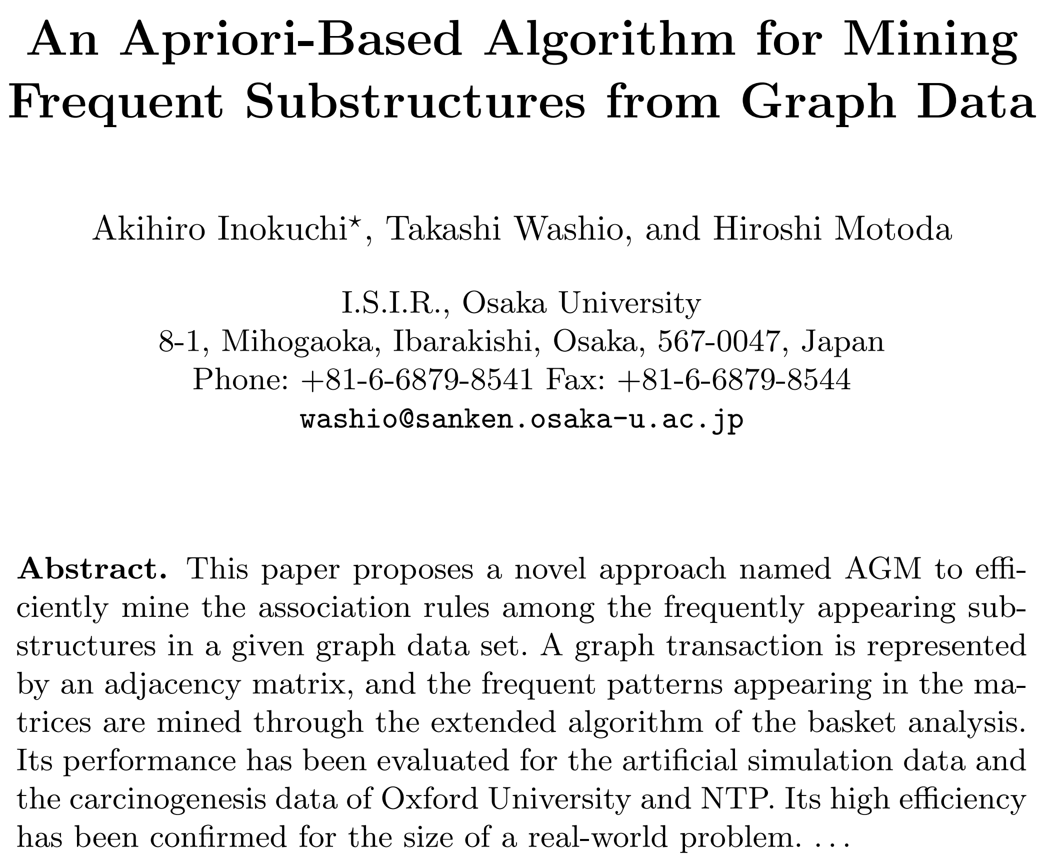
Multiplicity of candidates in Edge growing

- Notion of topologically equivalent vertices can lead to multiple candidate subgraphs being generated during edge growing.
- Although the edge-growing procedure can produce multiple candidate subgraphs, the number of candidate subgraphs tends to be smaller than those produced by the vertex-growing strategy.

Candidate pruning

- Given a candidate k-subgraph, we need to check whether all the k-1-subgraphs are frequent
- Approach:
  - Successively remove one edge from the k-subgraph
  - If the resulting k-1 subgraph is connected check whether it is frequent
  - If not, remove the k-subgraph from the candidates
- Checking whether a k-1-subgraph is contained in the list of frequent k-1-subgraphs is not easy
- Requires solving graph isomorphism problem, i.e. checking whether two graphs are topoplogically equivalent

Vertex growth paper



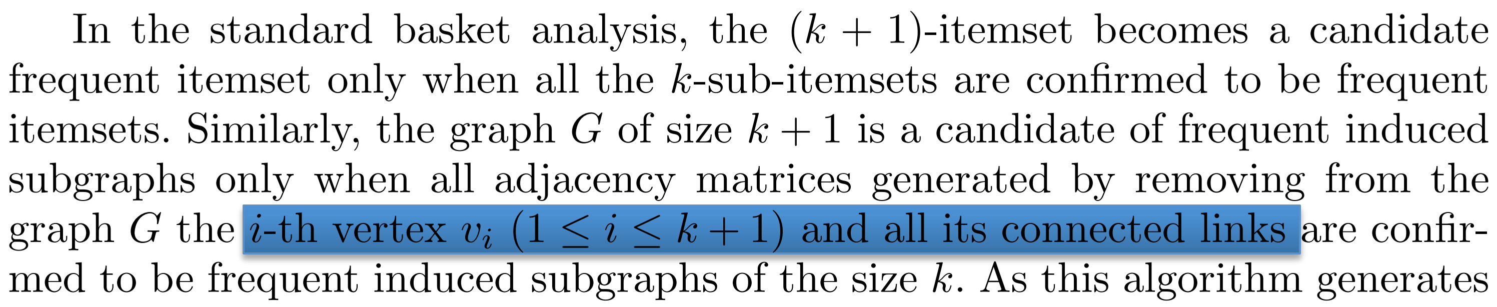
An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data

Akihiro Inokuchi*, Takashi Washio, and Hiroshi Motoda

I.S.I.R., Osaka University
8-1, Mihogaoka, Ibarakishi, Osaka, 567-0047, Japan
Phone: +81-6-6879-8541 Fax: +81-6-6879-8544
`washio@sanken.osaka-u.ac.jp`

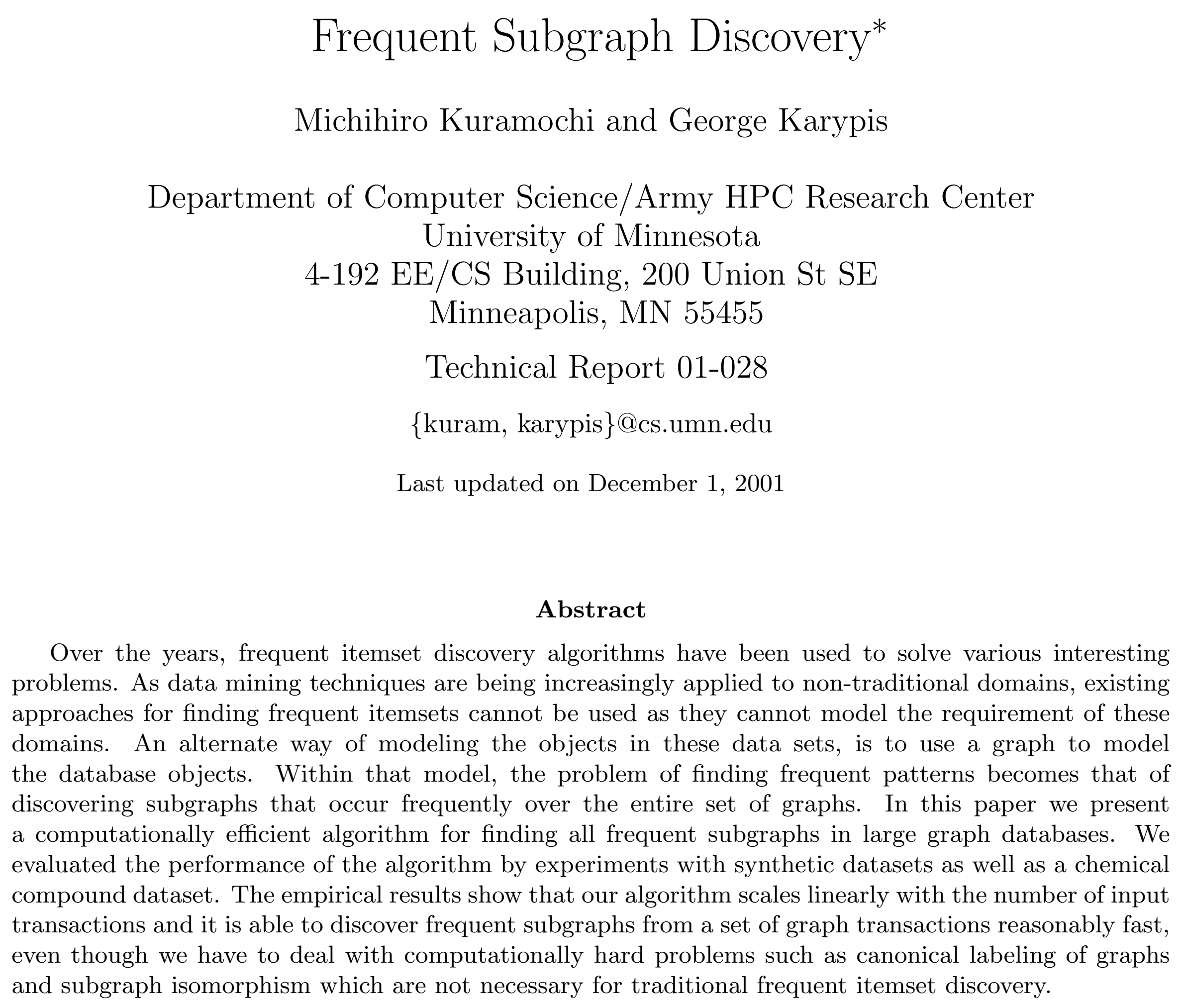
Abstract. This paper proposes a novel approach named AGM to efficiently mine the association rules among the frequently appearing substructures in a given graph data set. A graph transaction is represented by an adjacency matrix, and the frequent patterns appearing in the matrices are mined through the extended algorithm of the basket analysis. Its performance has been evaluated for the artificial simulation data and the carcinogenesis data of Oxford University and NTP. Its high efficiency has been confirmed for the size of a real-world problem. . . .

Extract from the paper…



In the standard basket analysis, the $(k+1)$-itemset becomes a candidate frequent itemset only when all the k-sub-itemsets are confirmed to be frequent itemsets. Similarly, the graph G of size $k+1$ is a candidate of frequent induced subgraphs only when all adjacency matrices generated by removing from the graph G the i-th vertex v_i $(1 \leq i \leq k+1)$ and all its connected links are confirmed to be frequent induced subgraphs of the size k. As this algorithm generates

Edge growth paper



Frequent Subgraph Discovery*

Michihiro Kuramochi and George Karypis

Department of Computer Science/Army HPC Research Center
University of Minnesota
4-192 EE/CS Building, 200 Union St SE
Minneapolis, MN 55455

Technical Report 01-028

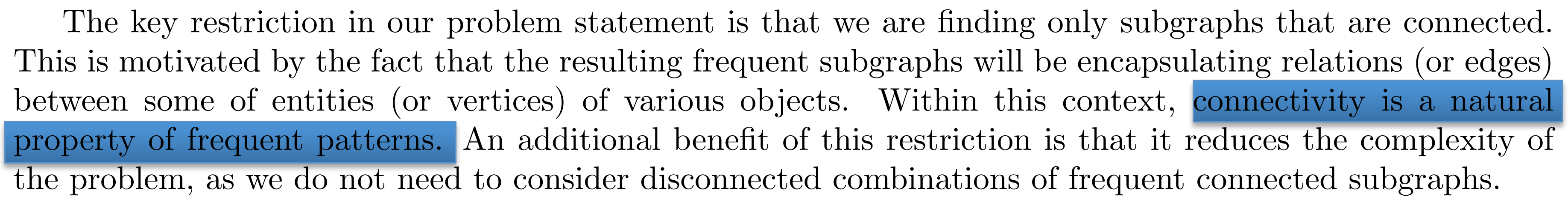
{kuram, karypis}@cs.umn.edu

Last updated on December 1, 2001

Abstract

Over the years, frequent itemset discovery algorithms have been used to solve various interesting problems. As data mining techniques are being increasingly applied to non-traditional domains, existing approaches for finding frequent itemsets cannot be used as they cannot model the requirement of these domains. An alternate way of modeling the objects in these data sets, is to use a graph to model the database objects. Within that model, the problem of finding frequent patterns becomes that of discovering subgraphs that occur frequently over the entire set of graphs. In this paper we present a computationally efficient algorithm for finding all frequent subgraphs in large graph databases. We evaluated the performance of the algorithm by experiments with synthetic datasets as well as a chemical compound dataset. The empirical results show that our algorithm scales linearly with the number of input transactions and it is able to discover frequent subgraphs from a set of graph transactions reasonably fast, even though we have to deal with computationally hard problems such as canonical labeling of graphs and subgraph isomorphism which are not necessary for traditional frequent itemset discovery.

Extract from paper…



The key restriction in our problem statement is that we are finding only subgraphs that are connected. This is motivated by the fact that the resulting frequent subgraphs will be encapsulating relations (or edges) between some of entities (or vertices) of various objects. Within this context, connectivity is a natural property of frequent patterns. An additional benefit of this restriction is that it reduces the complexity of the problem, as we do not need to consider disconnected combinations of frequent connected subgraphs.

Graph Isomorphism

- Graph isomorphism
  - Determining if two graphs are topologically equivalent is the graph isomorphism problem
- Needed for the candidate generation step, to determine whether a candidate has been generated and candidate pruning step, to check whether its (*k-1*)-subgraphs are frequent

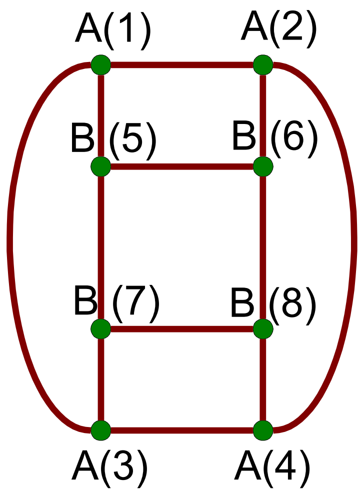
Graph Isomorphism

- **Use canonical labeling** to handle isomorphism
  - **Map each graph into an ordered string representation** (known as its code) such that two isomorphic graphs will be mapped to the same canonical encoding
- A canonical label has the property that **if two graphs are isomorphic, then their codes must be the same**.
- This property allows to test for graph isomorphism by comparing the canonical labels of the graphs.

Creation of canonical labels

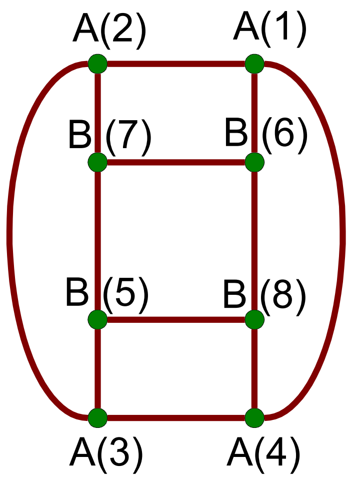
- The first step toward constructing the canonical label of a graph is to **find an adjacency matrix representation for the graph**
- The second step is to **determine the string representation for each adjacency matrix**.
  - Since the adjacency matrix is symmetric, it is sufficient to construct the string representation based on the upper triangular part of the matrix

String representation of Adjacency Matrices



	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	0	1	1	0	1	0	0	0
A(2)	1	0	0	1	0	1	0	0
A(3)	1	0	0	1	0	0	1	0
A(4)	0	1	1	0	0	0	0	1
B(5)	1	0	0	0	0	1	1	0
B(6)	0	1	0	0	1	0	0	1
B(7)	0	0	1	0	1	0	0	1
B(8)	0	0	0	1	0	1	1	0

Code =1 10 011 1000 01001 001010 0001011



	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	0	1	0	1	0	1	0	0
A(2)	1	0	1	0	0	0	1	0
A(3)	0	1	0	1	1	0	0	0
A(4)	1	0	1	0	0	0	0	1
B(5)	0	0	1	0	0	0	1	1
B(6)	1	0	0	0	0	0	1	1
B(7)	0	1	0	0	1	1	0	0
B(8)	0	0	0	1	1	1	0	0

Code =1 01 101 0010 10000 010011 0001110

Support counting

- Support counting is a potentially costly operation because all the candidate subgraphs contained in each graph must be determined.
- More efficient is to
  - Maintain a list of graph IDs (‘TID sets’) associated with each frequent k-1 subgraph
  - Intersect the lists of graph IDs of the k-1 subgraphs that are merged together to generate the new candidate k subgraph
  - Compute the subgraph isomorphism on the graphs in the intersected list to determine whether they contain a particular candidate subgraph.

Thanks!

Next Lecture:

- Introduction to Clustering

Readings:

- Chapter 7 - Tan & Kumar