SEQUENCE AND GRAPH MINING

SEQUENCE MINING

Motivation

- In many data mining tasks, the order and timing of events contains important information.

- Frequent itemsets only capture the co-occurrences.
  - No order between the items, order of transactions not considered

Motivation

- An online shopping company would like to extract patterns about web pages visited in each session as an attempt to predict customer behavior
- Data collected:

<{Homepage} {Electronics} {Cameras and Camcorders} {Digital cameras} {Shopping Cart} {Return to Shopping}>

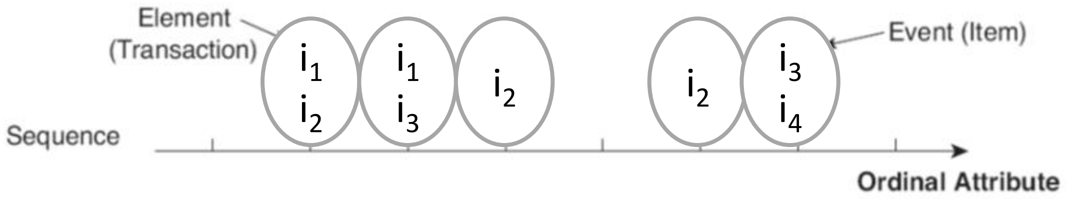
<{Homepage} {Books} {Programming Algorithms} {Modeling and Simulation} >

Temporal information is not captured by <session-Id, items> model

Sequential Pattern Mining

- Goal: discover sequential patterns in a sequence data set
- Sequence:
  - An ordered list of elements
  - Each element is a collection of one or more events

$$s = \langle e_1\ e_2\ e_3\ \ldots\ e_n \rangle$$
$$e_j = \{i_1, i_2, i_3, \ldots, i_k\}$$



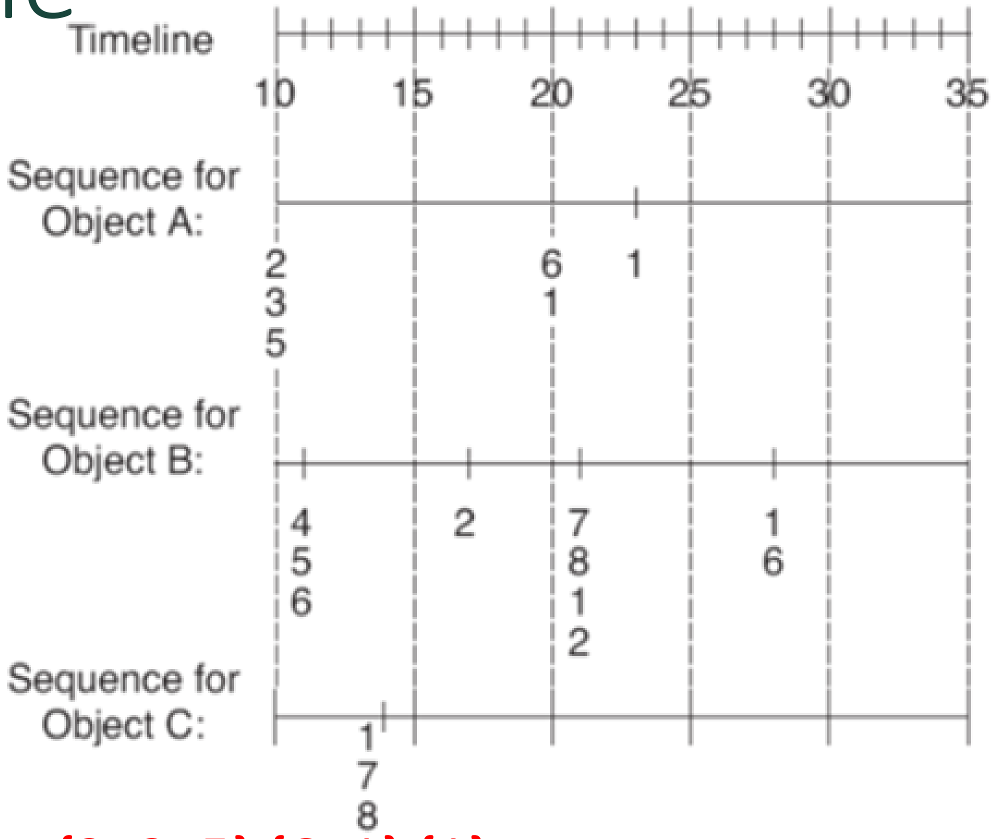
- Length of a sequence: number of elements in it
- k-sequence: contains k events

$$s = \langle \{1, 2\}\ \{3, 4\}\ \{5\}\ \{6, 7, 8\} \rangle$$
s is an 8-sequence of length 4

Sequence Data Example

Sequence Database:

Object	Timestamp	Events
A	10	2, 3, 5
A	20	6, 1
A	23	1
B	11	4, 5, 6
B	17	2
B	21	7, 8, 1, 2
B	28	1, 6
C	14	1, 8, 7

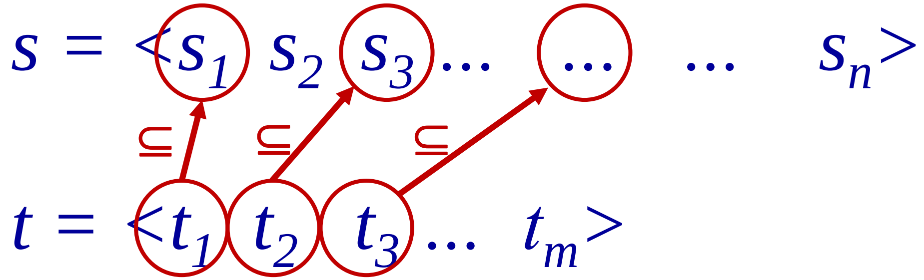


Sort all events associated with a given object in increasing order of their timestamp

<{2, 3, 5} {6, 1} {1}>
<{4, 5, 6} {2} {7, 8, 1, 2} {1, 6}>
<{1, 8, 7}>

Subsequence

- A sequence t is a subsequence of a sequence s if each ordered element in t is a subset of an ordered element of s



- t is a subsequence of s if there exists integers $1 \leq j_1 < j_2 < \ldots < j_m \leq n$ such that $t_1 \subseteq s_{j1}, t_2 \subseteq s_{j2}, \ldots, t_m \subseteq s_{jm}$

Sequence s	Sequence t	Is t a subsequence of s?
<{2, 4} {3, 5, 6} {8}>	<{2} {3, 6} {8}>	Yes
<{2, 4} {3, 5, 6} {8}>	<{2} {8}>	Yes
<{1, 2} {3, 4}>	<{1} {2}>	No

Pattern Discovery

- Task: Given a sequence data set D and a user-specified minimum support *minsup*, the goal is to find all sequences with support >= *minsup*

Object	Timestamp	Events
A	1	1, 2, 4
A	2	2, 3
A	3	5
B	1	1, 2
B	2	2, 3, 4
C	1	1, 2
C	2	2, 3, 4
C	3	2, 4, 5
D	1	2
D	2	3, 4
D	3	4, 5
E	1	1, 3
E	2	2, 4, 5

Minsup = 50%

Examples of Sequential Patterns:

<{1,2}>	s=60%
<{2,3}>	s=60%
<{2,4}>	s=80%
<{3} {5}>	s=80%
<{1} {2}>	s=80%
<{2} {2}>	s=60%
<{1} {2,3}>	s=60%
<{2} {2,3}>	s=60%
<{1,2} {2,3}>	s=60%

Pattern Discovery

- Computationally challenging because there are exponentially many subsequences of a given sequence
- Brute force:

1-sequences	$<i_1>\ <i_2>\ \ldots\ <i_n>$
2-sequences	$<\{i_1, i_2\}>\quad <\{i_1, i_3\}> \ldots \quad <\{i_{n-1}, i_n\}> \ldots$ $<\{i_1\}, \{i_2\}>\quad <\{i_1\}, \{i_3\}> \ldots \quad <\{i_{n-1}\}, \{i_n\}>$
3-sequences	$<\{i_1, i_2, i_3\}>\quad <\{i_1, i_2, i_4\}> \ldots \quad <\{i_1, i_2\}, \{i_1\} \ldots >$ $<\{i_1\}, \{i_1, i_2\}> \ldots <\{i_1\}, \{i_1\}, \{i_3\}> \ldots$

- Number of candidate subsequences is substantially larger than number of candidate itemsets
  - An item can appear at most once in an itemset but an event can appear multiple times in a sequence
  - Order matters in sequences but not in itemsets

Apriori Principle

- Any data sequence that contains a k-sequence also contains all its (k-1)-subsequences => Apriori principle holds
- Apriori-like algorithm for generating frequent data sequences

1. *Generate frequent 1-sequences*
2. *Repeat:*
    1. *Merge pairs of frequent (k-1)-sequences to generate candidate k-sequences*
    2. *Prune candidates whose (k-1)-subsequences are infrequent*
    3. *Make a pass over the data set to count the supports of the remaining candidates*
    4. *Construct F_k as subset of sequences in step 3 satisfying min support*

Candidate Set Generation

Merge two k-sequences s_1 and s_2 if the subsequences obtained by:

- dropping the first event of s_1
- dropping the last event of s_2

are identical

s_1: <{1} {2 3} {4}>   drop first event: <{2 3} {4}>
s_2: <{2 3} {4 5}>   drop last event: <{2 3} {4}>

s_1 and s_2 can be merged to generate a candidate 5-sequence

Candidate Set Generation

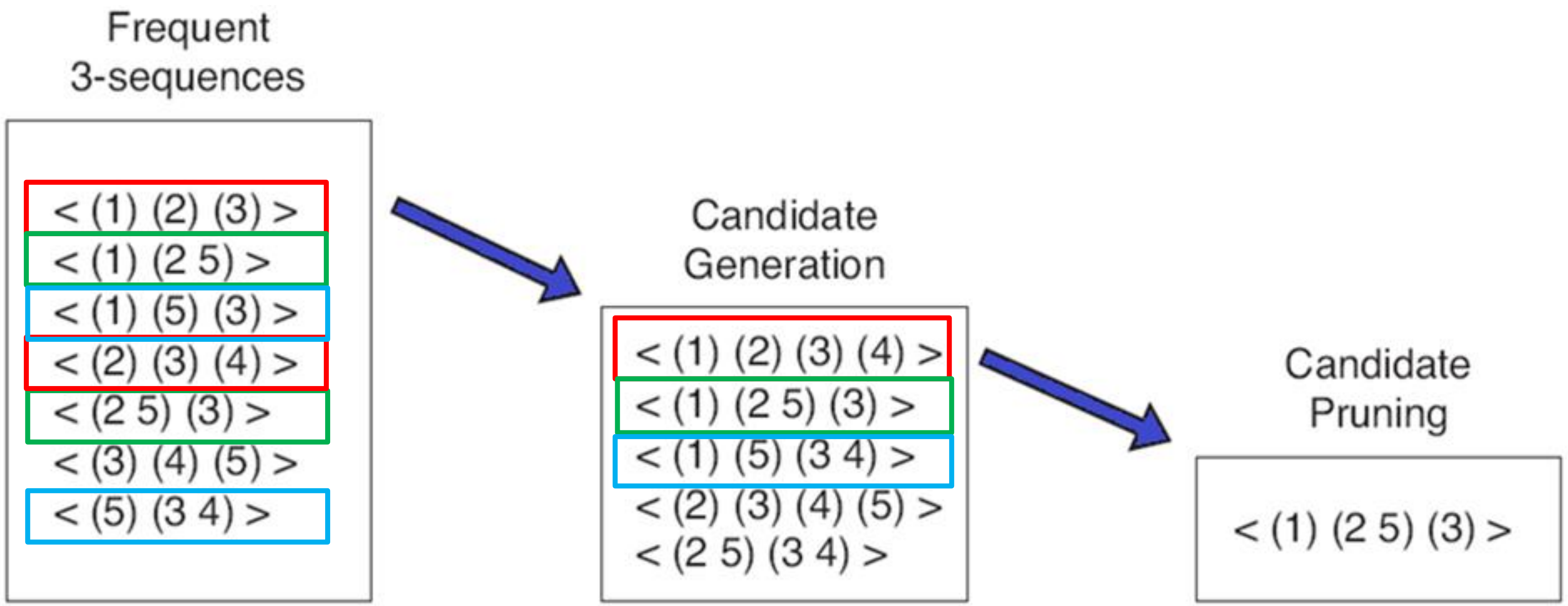
- If the last element of s_2 has more than one event, append the last event from the last element of s_2 to the last element of s_1

  s_1: <{1} {2 3} {4}>
  s_2: <{2 3} {4 5}>
  Result: <{1} {2 3} {4 5}>

- If the last element of s_2 has only one event, append the last element of s_2 to the end of s_1 as a separate element

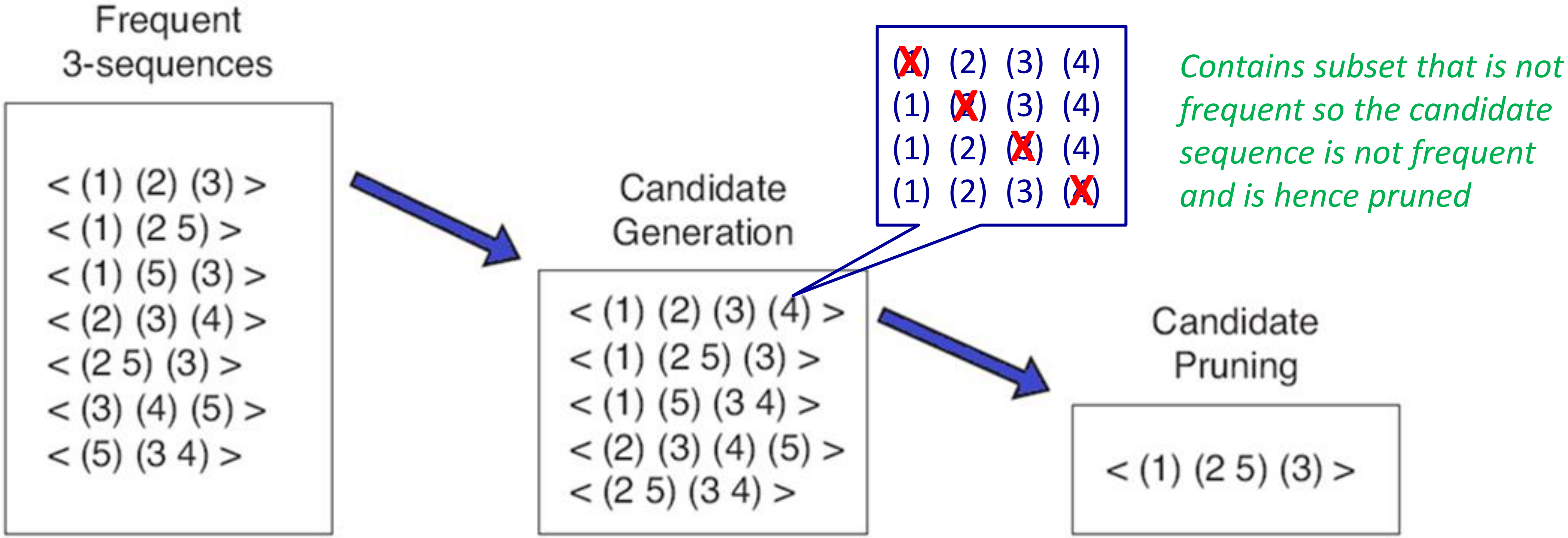
  s_1: <{1} {2 3} {4}>
  s_2: <{2 3} {4} {5}>
  Result: <{1} {2 3} {4} {5}>

Candidate Pruning and Support Count



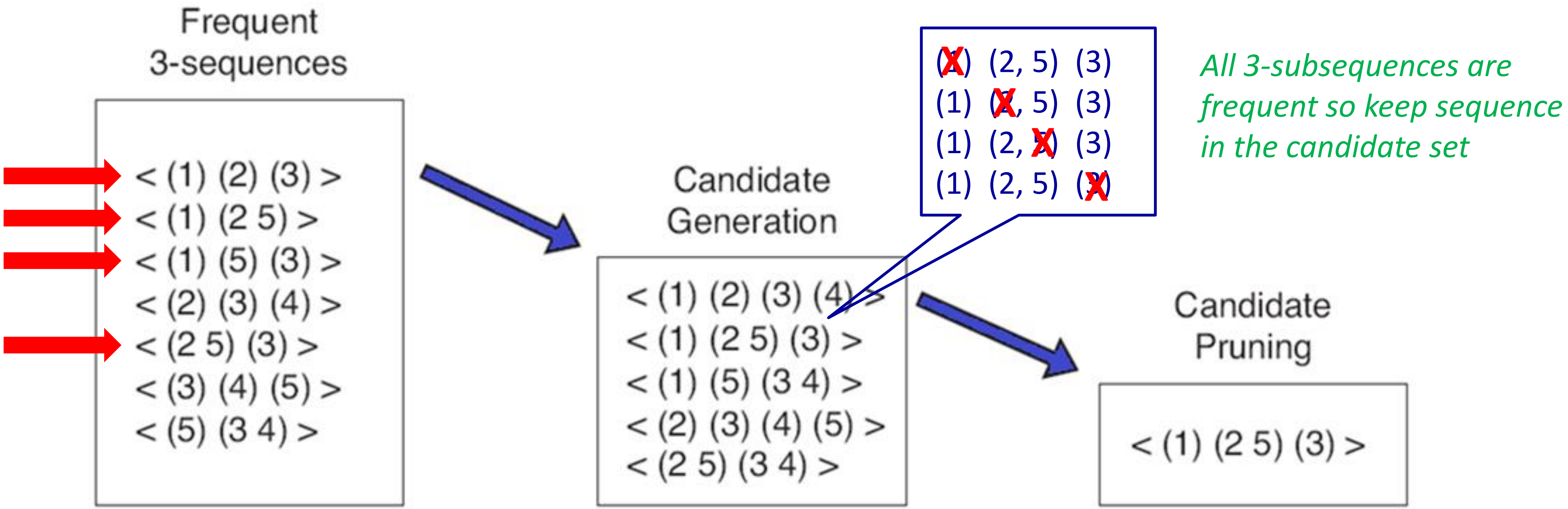
Candidate Pruning and Support Count

Prune a candidate k-sequence if at least one of its (k-1)-sequences is not frequent



Candidate Pruning and Support Count

Prune a candidate k-sequence if at least one its (k-1)-sequences is not frequent



Time constraints

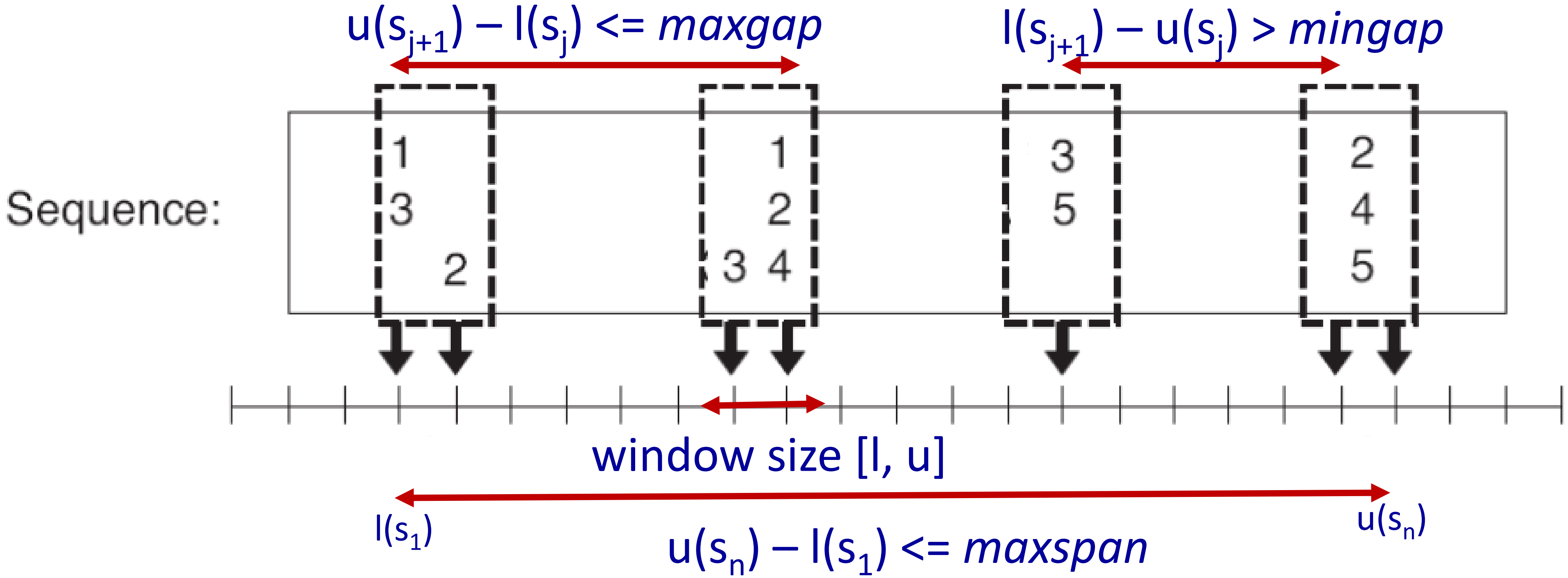
- In some applications, relative timing of the transactions is crucial to define the pattern
- Credit card fraud:

  The fraudulent user would do the purchases in short time interval to make maximum use of the card before it is closed.

- We impose some timing constraints to mine such patterns. Some of the timing constraints that can be imposed on a pattern.
- Approach: modify candidate pruning to directly prune candidates that violate time constraints

Time constraints

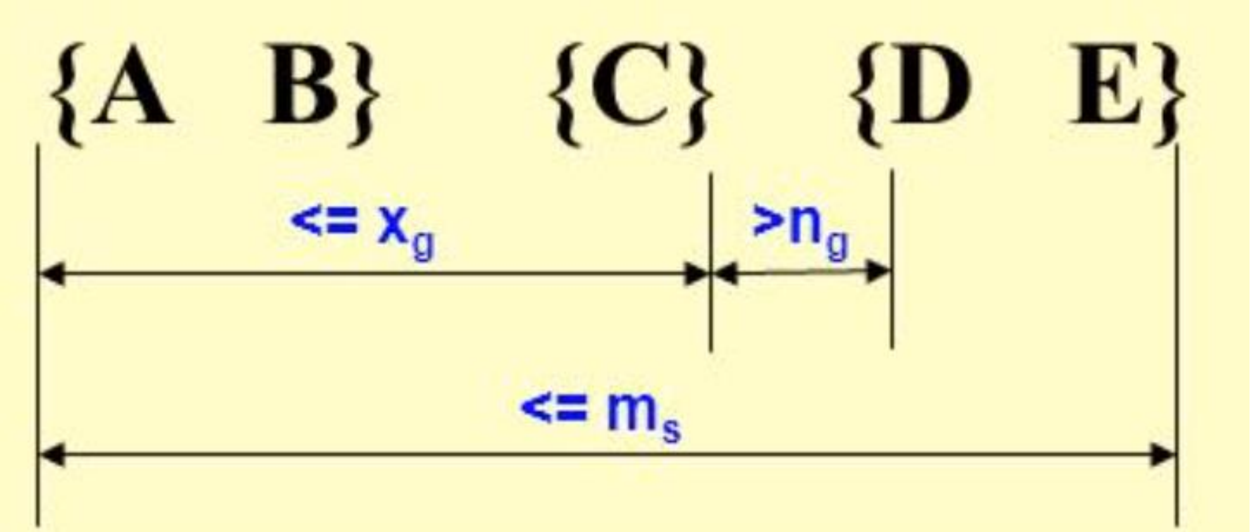
Each sequential pattern is associated with a time window [l,u]. L is the earliest occurrence of an event. U is the latest occurrence of an event.



17

Time constraints

- We consider three kinds of constraints:
  - max-span constraint (m_s): maximum allowed time between the latest and the earliest occurrence of events in the entire sequence.
  - max-gap constraint (x_g): maximum length of a gap between two consecutive element.
  - min-gap constraint (n_g): minimum length of a gap between two consecutive element.



Time constraints – Example 1

- Each itemset is tagged by the time of purchase:
- Constraints: maxgap = 3 mingap = 1 maxspan = 3
- Consider data sequence *S* and sequential pattern *T*:

S: <{Juice}^{1} {Eggs, Chips}^{2} {Chips}^{3} {Coke}^{4} {Cheese, Bread}^{5} {Water}^{6}>

T: <{Eggs} {Cheese}>

Gap = 5 – 2 = 3 Span = 5 – 2 = 3

So:

Gap <= maxgap

Gap > mingap

Span <= maxspan

All three constraints are satisfied

Time constraints – Example 2

- Each itemset is tagged by the time of purchase:
- Constraints: maxgap = 3 mingap = 1 maxspan = 3
- Consider data sequence *S* and sequential pattern *T*:

S: <{Juice}^{1} {Eggs, Chips}^{2} {Chips}^{3} {Coke}^{4} {Cheese, Bread}^{5} {Water}^{6}>

T: <{Juice}{Cheese,Bread}>

Gap = 5 – 1 = 4 Span = 5 – 1 = 4

So:

Gap <= maxgap? FALSE

Gap > mingap? TRUE

Span <= maxspan? FALSE

Time constraints – Example 3

- Each itemset is tagged by the time of purchase:
- Constraints: maxgap = 3 mingap = 1 maxspan = 3
- Consider data sequence *S* and sequential pattern *T*:

S: <{Juice}^{1} {Eggs, Chips}^{2} {Chips}^{3} {Coke}^{4} {Cheese, Bread}^{5} {Water}^{6}>

T: <{Juice} {Eggs} {Water}>

Gap = 2 – 1 = 1
Mingap not satisfied
Maxgap satisfied

Gap = 6 – 2 = 4
Mingap satisfied
Maxgap not satisfied

Span = 6 – 1 = 5
Maxspan not satisified

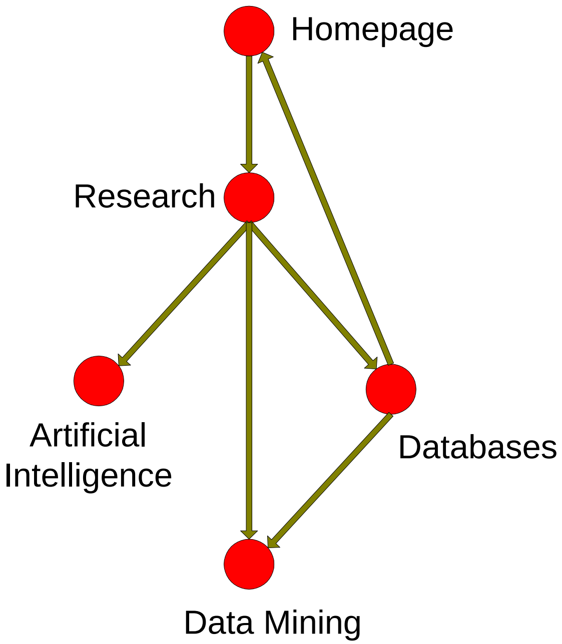
Mingap and maxgap are not satisfied by every pair of consecutive elements. So they are not satisfied by the pattern

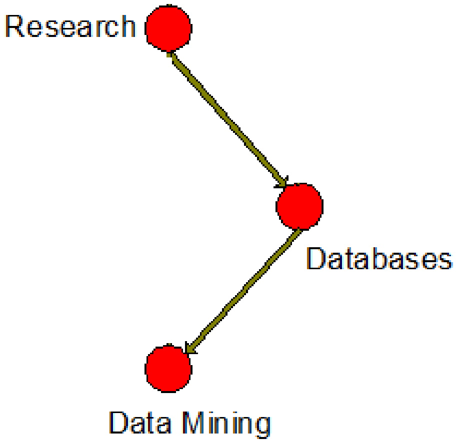
SUBGRAPH MINING

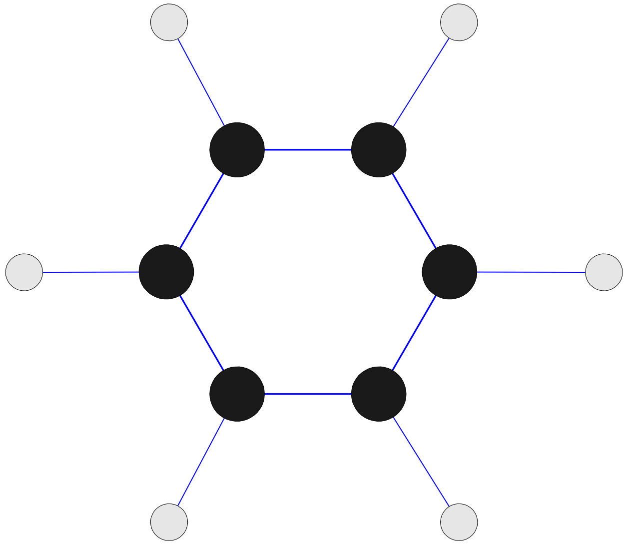
Subgraph patterns

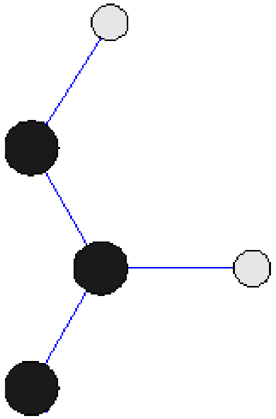
Application	Graphs	Vertices	Edges
Web mining	Web browsing history	Web pages	Hyperlink between pages
Computational Chemistry	Structure of chemical compounds	Atoms or ions	Bond between atoms or ions
Network Computing	Computer networks	Computers and Servers	Interconnection between machines
Semantic Web	Collection of XML documents	XML Elements	Parent-child relationship
Bioinformatics	Protein structures	Amino acids	Contact residue

Subgraph patterns - Example





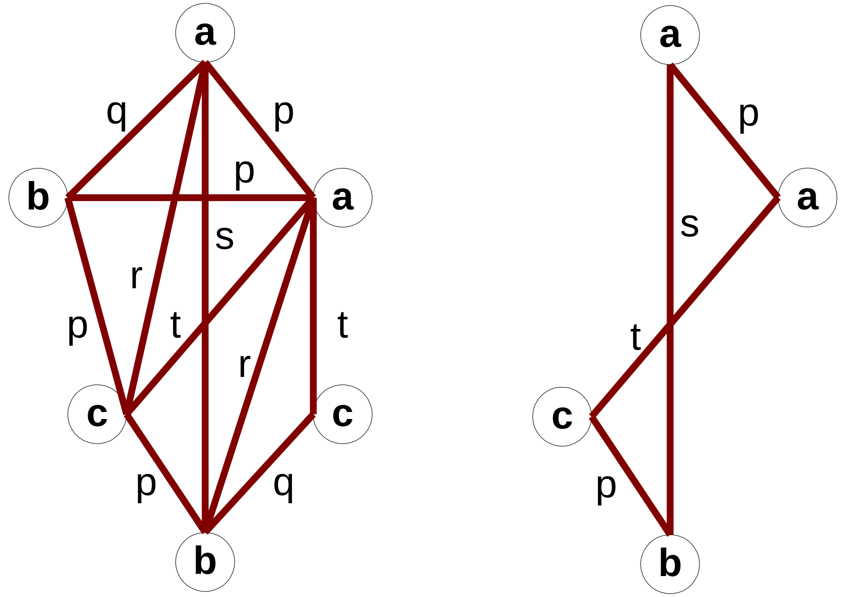




Definitions

- A graph G = (V, E) is composed of:
  - A set of vertices (or nodes) V
  - A set of edges E
  - A label for each edge e, l(e), if the graph is labeled

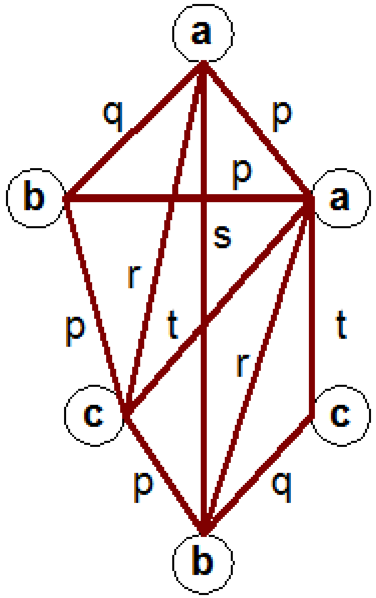
- Subgraph: a graph G' =(V', E') is a subgraph of another graph G = (V, E) if:
  - V' is a subset of V
  - E' is a subset of E



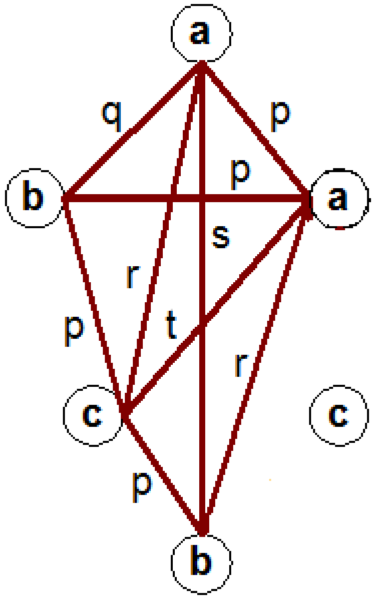
(a) Labeled Graph (b) Subgraph

Definitions

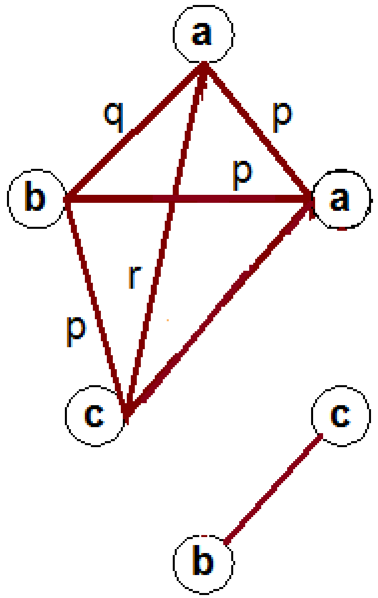
- Connected graph: if there is a path between every pair of vertices
  - “Fully connected”



Connected



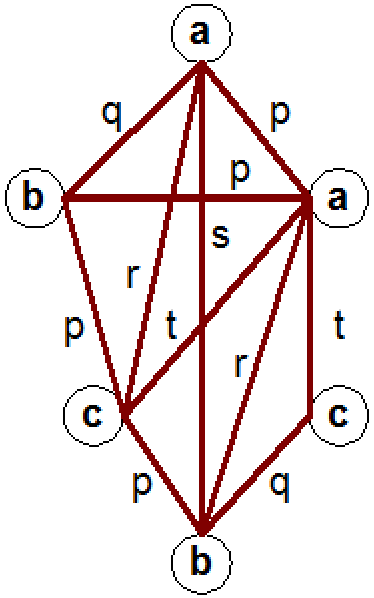
Disconnected



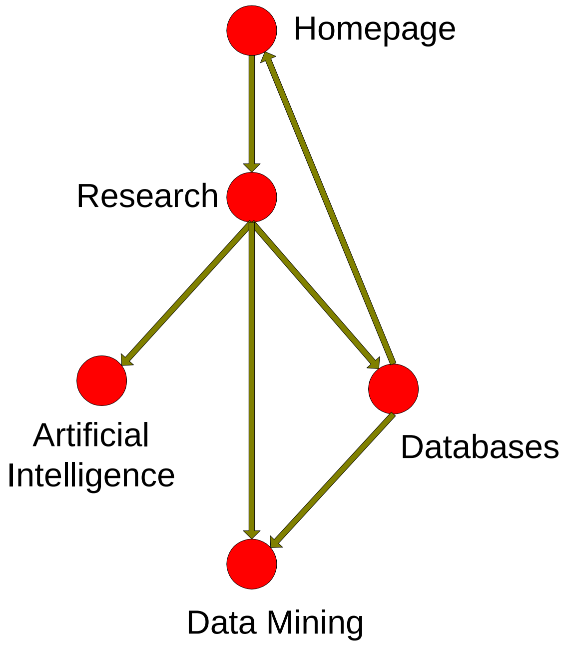
Disconnected

Definitions

- Undirected graph: if it contains undirected edges: (v_i, v_j) is the same as (v_j, v_i)



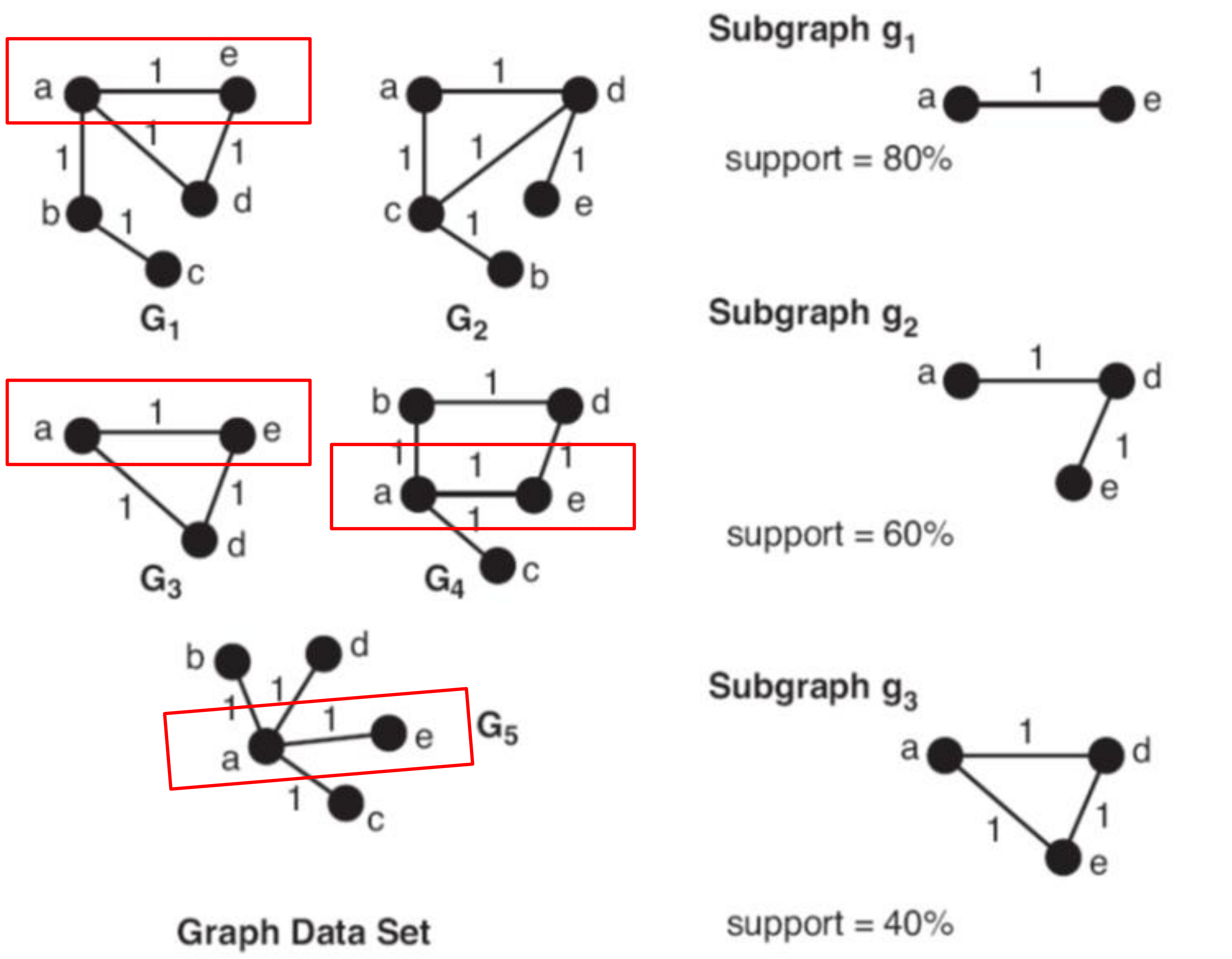
Undirected



Directed

Support

- Given a collection of graphs C_g, the support for subgraph g is the fraction of all graphs in C_g that contain g as its subgraph

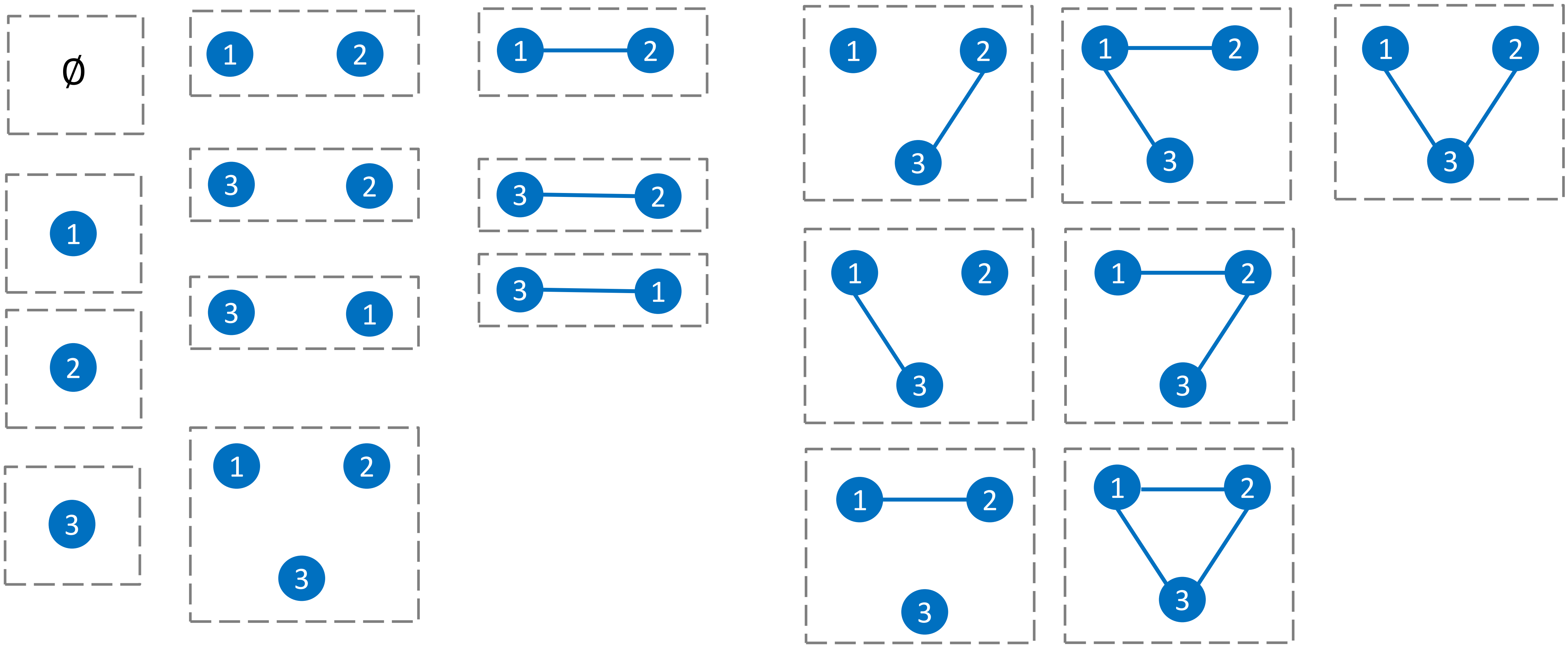


G1
G2
G3
G4
G5
Graph Data Set
Subgraph g1
support = 80%
Subgraph g2
support = 60%
Subgraph g3
support = 40%

G1, G2, G3

G1, G3

Mining Frequent Subgraphs



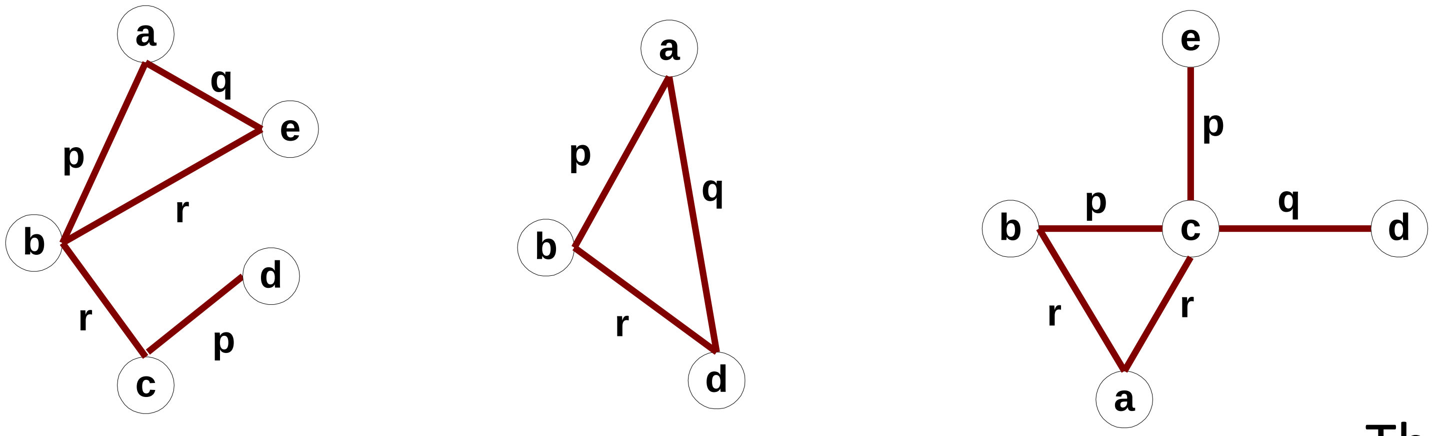
Approach 1

- Transform graphs and subgraphs into transaction format

- Each combination of vertex label – edge label – vertex label is defined as an item

Approach 1

- Transform graphs and subgraphs into transaction format



G1 G2 G3

The width of the transaction?

The number of edges in the graph

	(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	…	…	…	…	…	…	…	…

Approach 1

- Transform graphs and subgraphs into transaction format

Problem: Multiple edges will be mapped into one item if they have the same label combination

Valid transformation only if every edge in the graph has a unique combination of label and vertices

one-to-one mapping

Approach 2 – Apriori like

- The apriori algorithm still holds because a k-graph is frequent only if all of its (k-1) graphs are frequent.

What is a k-graph? k vertices or k edges

Vertex growing

Edge growing

- You start by a small size graph and generate candidates by adding a vertex/edge.
- Candidate generation in graphs is complex

Approach 2 – Apriori like

Candidate Generation: merge pairs of (k-1)-subgraphs to obtain candidate k-subgraph

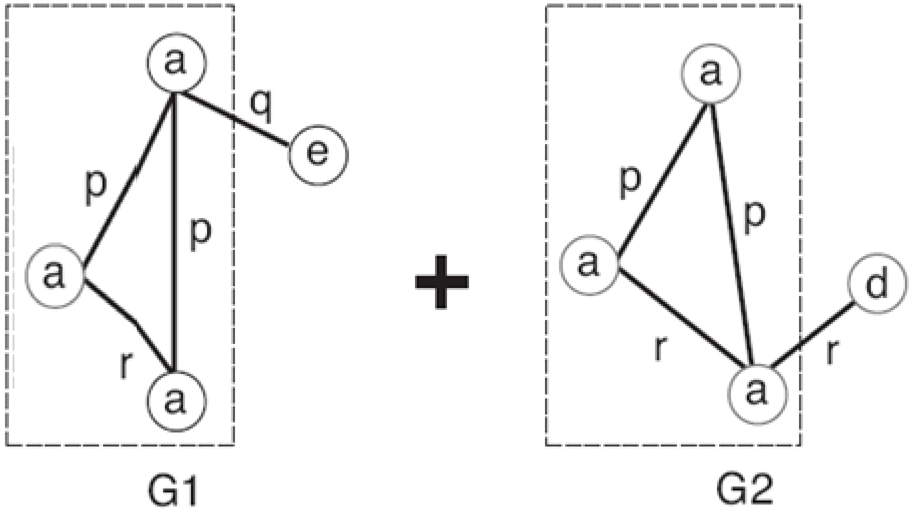
Candidate Pruning: discard all candidate k-subgraphs contains infrequent (k-1)-subgraphs

Support Counting: count number of subgraphs containing each candidate

Candidate Elimination: discard all candidates not satisfying minsupport

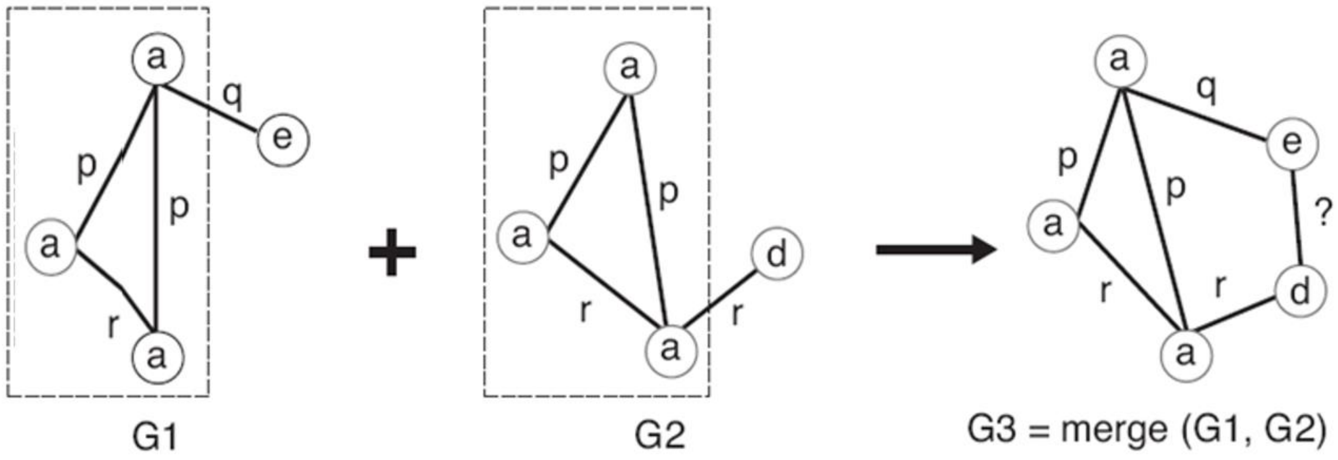
Approach 2 – Apriori like

- Candidate Generation:
- These candidates are generated by joining two similar but slightly different frequent subgraphs.
- A pair of frequent (k-1)-subgraphs are merged to form a candidate k-subgraph, if they share a common (k-2)-subgraph (i.e., core).
- Vertex based generation: Size (# vertices)
- Edge based generation: Size (# edges)



Vertex Growing (AGM approach)

- In the vertex growing approach, increases the substructure by one vertex at each step.
- At each step we will merge two similar but slightly different subgraphs that differs by one vertices.
- The new candidate will have the core structure and the additional two vertices.

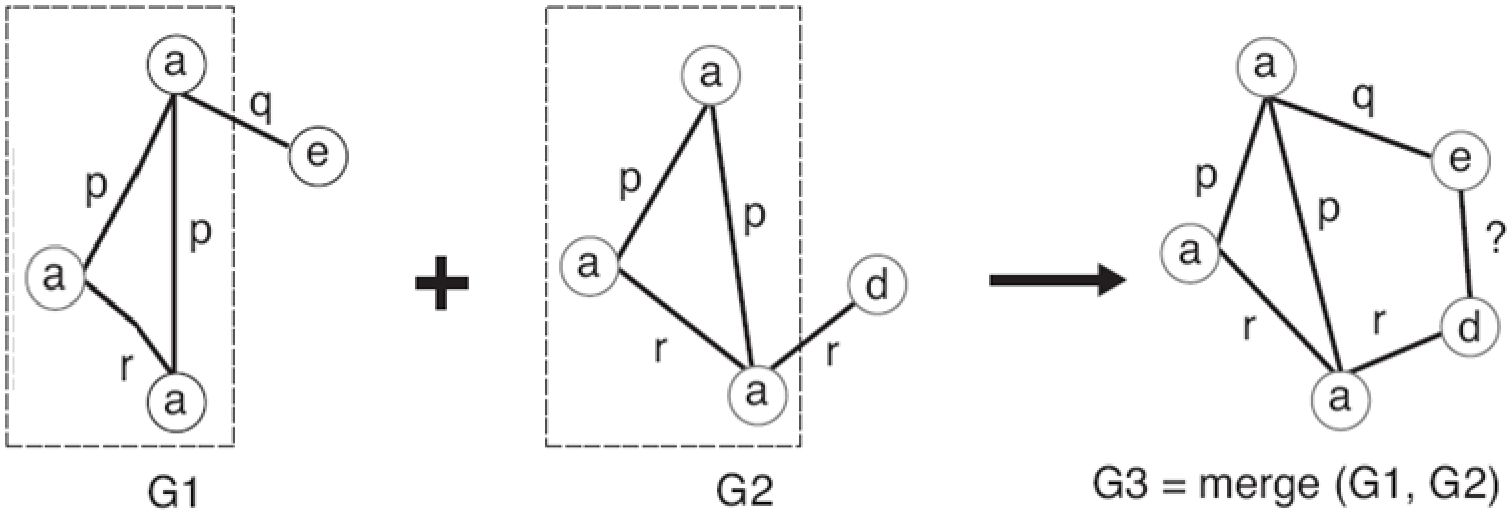


Vertex Growing (AGM approach)

Adjacency matrix:
-Rows and columns correspond to nodes;

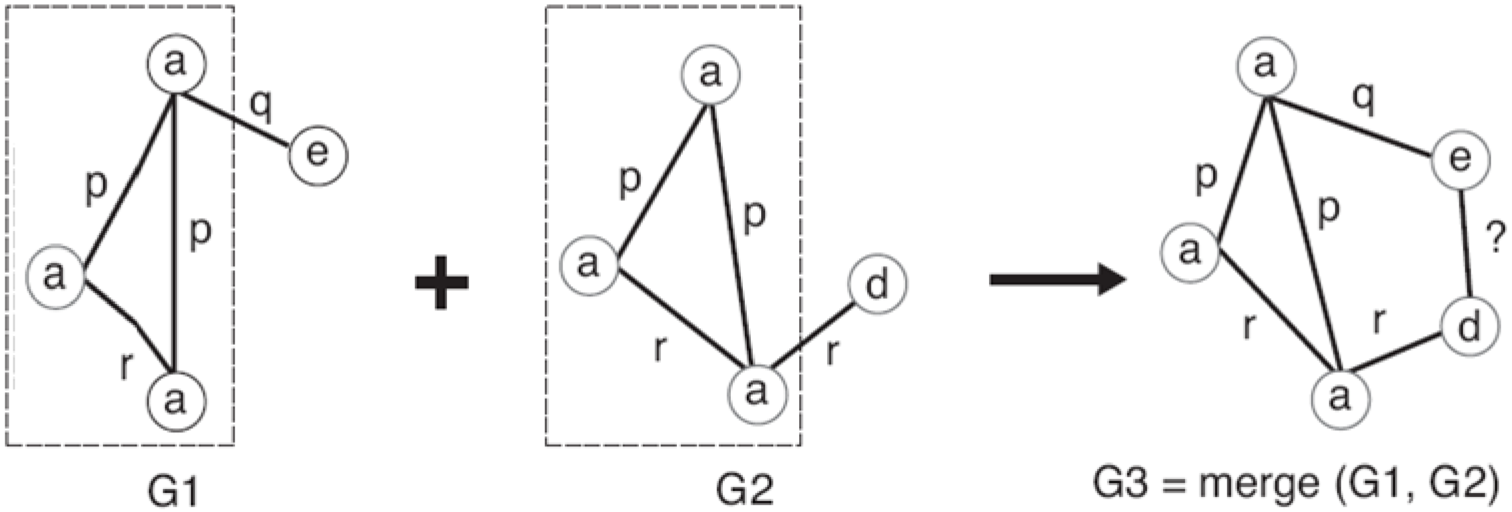
- non-zero cells along a row (column) correspond to neighbors

- Cells correspond to edges
- cell contains edge label (or zero if no edge) .



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

Vertex Growing (AGM approach)



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

-Vertex growing takes two adjacency matrices that differ in the last row, and creates an **augmented matrix** by adding the last row and last column of the second matrix to the first matrix.