Data Mining:

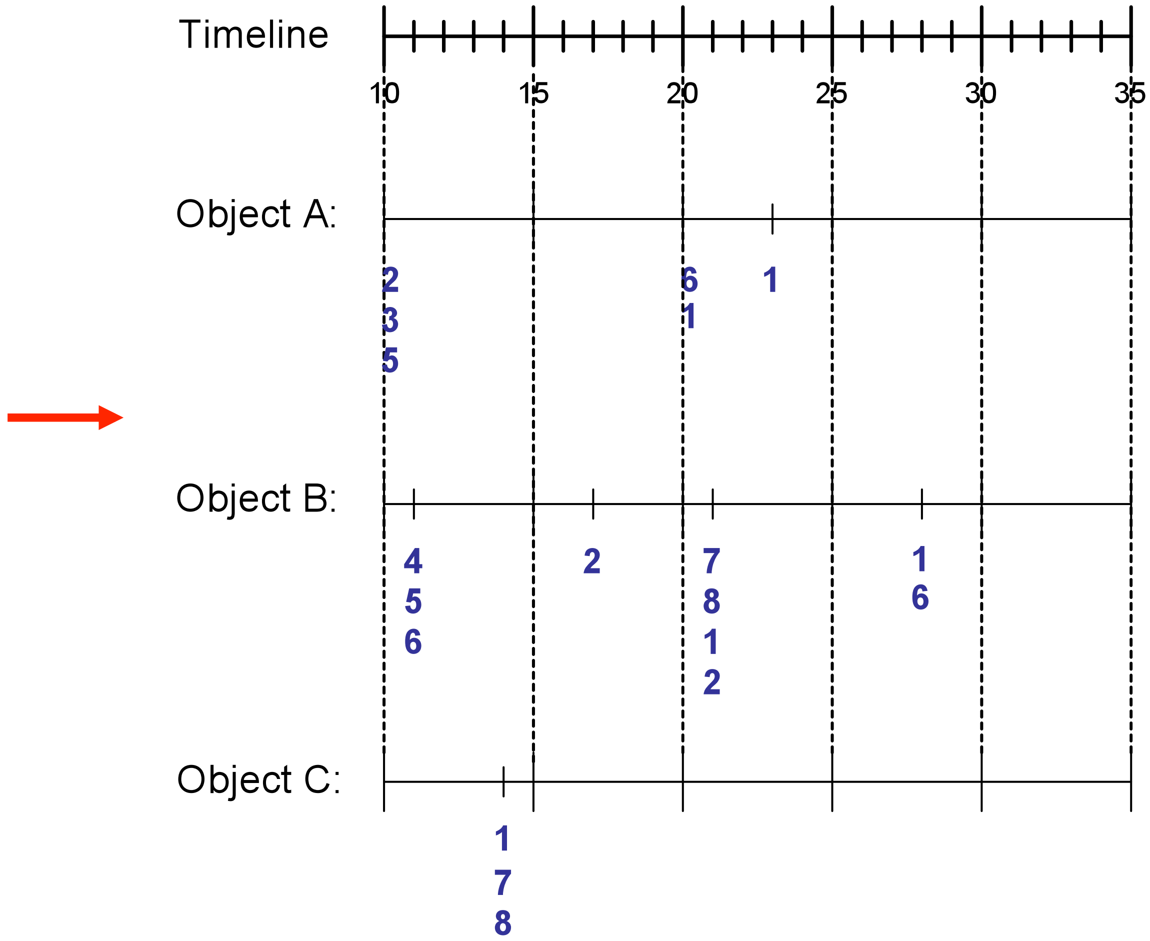
2. Assoziationsanalyse

C) Non-Standard Data

Sequence Data

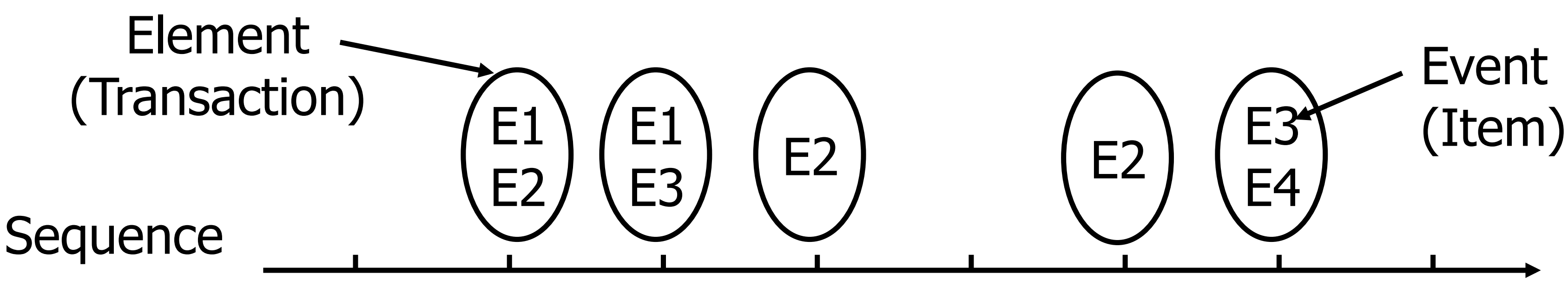
Sequence Database:

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



Examples of Sequence Data

Sequence Database	Sequence	Element (Transaction)	Event (Item)
Customer	Purchase history of a given customer	A set of items bought by a customer at time t	Books, diary products, CDs, etc
Web Data	Browsing activity of a particular Web visitor	A collection of files viewed by a Web visitor after a single mouse click	Home page, index page, contact info, etc
Event data	History of events generated by a given sensor	Events triggered by a sensor at time t	Types of alarms generated by sensors
Genome sequences	DNA sequence of a particular species	An element of the DNA sequence	Bases A,T,G,C



Formal Definition of a Sequence

- A sequence is an ordered list of elements (transactions).

$$s = < e_1\ e_2\ e_3\ \ldots >$$

  - Each element contains a collection of events (items)

$$e_i = \{i_1,\ i_2,\ \ldots,\ i_k\}$$

  - Each element may be attributed by a specific time.

- Length of a sequence, $|s|$, is given by the number of elements of the sequence.

- A k-sequence is a sequence that contains k events (items).

Examples of Sequence

- Sequence of different transactions by a customer at an online store:

  < {digital camera, iPad} {memory card} {headphone, iPad cover} >

- Sequence of initiating events causing the nuclear accident at 3-mile Island:

  <   {clogged_resin} {outlet_valve_closure} {loss_of_feedwater}
  {condenser_polisher_outlet_valve_shut} {booster_pumps_trip}
  {main_waterpump_trips} {main_turbine_trips} {reactor_pressure_increases}>

- Sequence of books checked out by a borrower at a library:

  <{Fellowship_of_the_Ring} {Hobbit The_Two_Towers} {Return_of_the_King}>

- Sequence of courses taken by a student …

Sequence Data vs. Market Basket Data

Sequence Database:

Customer	Date	Items bought
A	10	2, 3, 5
A	20	1, 6
A	23	1
B	11	4, 5, 6
B	17	2
B	21	1, 2, 7, 8
B	28	1, 6
C	14	1, 7, 8

Market Basket Data:

Items
2, 3, 5
1,6
1
4,5,6
2
1,2,7,8
1,6
1,7,8

Actually, specific dates usually are ignored; only the order of transactions (1st, 2nd, 3rd, ... purchase) per customer is relevant.

Formal Definition of a Subsequence

- A sequence $<a_1 \, a_2 \ldots a_n>$ is contained in another sequence $<b_1 \, b_2 \ldots b_m>$ $(m \geq n)$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, …, $a_n \subseteq b_{in}$.

Data sequence	Subsequence	Contains?
< {2,4} {3,5,6} {8} >	< {2} {3,5} >	Yes
< {1,2} {3,4} >	< {1} {2} >	No
< {2,4} {2,4} {2,5} >	< {4} {2} >	Yes

- The support of a subsequence w is defined as the fraction of data sequences that contain w .
- A *sequential pattern* is a frequent subsequence, i.e., a subsequence whose support is ≥ *minsup*.

Sequential Pattern Mining

- Given:
  - a database of sequences
  - a user-specified minimum support threshold, *minsup*

- Task:
  - **Find all subsequences with support ≥ *minsup*.**

Sequential Pattern Mining: Challenge

- Given a sequence: <{a b} {c d e} {f} {d g h}>
  - Examples of subsequences:

    <{a} {c d} {f} {g} >, < {c d e} >, < {b} {d} >, etc.

- How many k-subsequences can maximally be extracted from a given n-sequence?

<{a b} {c d e} {f} {d g h}> n=9

k=4: Y _ _ Y Y _ _ _ Y

$$\underbrace{\text{Y \_ \_ Y Y \_ \_ \_ Y}}$$

<{a} {d e} {h}>

Answer :

$$\binom{n}{k} = \binom{9}{4} = 126$$

sum for arbitrary k: 2^n-1

Sequential Pattern Mining: Example

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

(relative timestamps!)

Minsup = 50%

Examples of Frequent Subsequences:

< {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%

Generating Sequential Patterns

- Given n events (lex. sorted): $i_1, i_2, i_3, \ldots, i_n$
- Candidate 1-subsequences:

  $<\{i_1\}>, <\{i_2\}>, <\{i_3\}>, \ldots, <\{i_n\}>$

- Candidate 2-subsequences:

  $<\{i_1, i_2\}>, <\{i_1, i_3\}>, \ldots, <\{i_1\} \{i_1\}>, <\{i_1\} \{i_2\}>, \ldots, <\{i_n\} \{i_n\}>$

- Candidate 3-subsequences:

  $<\{i_1, i_2, i_3\}>, <\{i_1, i_2, i_4\}>, \ldots, <\{i_1, i_2\} \{i_1\}>, <\{i_1, i_2\} \{i_2\}>, \ldots,$

  $<\{i_1\} \{i_1, i_2\}>, <\{i_1\} \{i_1, i_3\}>, \ldots, <\{i_1\} \{i_1\} \{i_1\}>, <\{i_1\} \{i_1\} \{i_2\}>, \ldots$

- Etc. This would be brute-force.
- But the Apriori principle holds for k-subsequences.

Generalized Sequential Pattern (GSP) Algorithm

- **Step 1**:
  – Make the first pass over the sequence database D to yield all the frequent 1-sequences
- **Step 2**:

  Repeat until no new frequent sequences are found:
  – **Candidate Generation**:
    - Merge pairs of frequent subsequences found in the (k-1)*th* pass to generate candidate sequences that contain k items
  – **Candidate Pruning**:
    - Prune candidate *k*-sequences that contain infrequent (k-1)-subsequences
  – **Support Counting**:
    - Make a new pass over the sequence database D to find the support for these candidate sequences
  – **Candidate Elimination**:
    - Eliminate candidate k-sequences whose actual support is less than *minsup*

Candidate Generation

- Base case (k=2):
  - Merging two frequent 1-sequences <{i_x}> and <{i_y}> will produce 1-2 candidate 2-sequences: <{i_x} {i_y}> and (if i_x<i_y) <{i_x i_y}>
- General case (k>2):
  - A frequent (k-1)-sequence w_1 is merged* with another frequent (k-1)-sequence w_2 to produce a candidate k-sequence if the subsequence obtained by removing the first event in w_1 is the same as the subsequence obtained by removing the last event in w_2
  - The resulting candidate after merging is given by the sequence w_1 extended with the last event of w_2.
    - If the last two events in w_2 belong to the same element, then the last event in w_2 becomes part of the last element
    - Otherwise, the last event in w_2 becomes a separate appended element

*) here noncommutative operation!

Candidate Generation Examples

- Merging the sequences
  w_1=<{1} {2 3} {4}> and w_2 =<{2 3} {4 5}>
  will produce the candidate sequence < {1} {2 3} {4 5}> because the last two events in w_2 (4 and 5) belong to the same element

- Merging the sequences
  w_1=<{1} {2 3} {4}> and w_2 =<{2 3} {4} {5}>
  will produce the candidate sequence < {1} {2 3} {4} {5}> because the last two events in w_2 (4 and 5) do not belong to the same element

- w_1=<{1 2 3}> and w_2 =<{2 3 4} > merge to < {1 2 3 4}>.

- We do not have to merge the sequences
  w_1 =<{1} {2 6} {4}> and w_2 =<{1} {2} {4 5}>
  to produce the candidate < {1} {2 6} {4 5}>,
  because if the latter is a viable candidate, then it can be obtained by merging w_1=<{1} {2 6} {4}> with < {2 6} {4 5}>

GSP Example

Frequent 3-sequences

< {1} {2} {3} >
< {1} {2 5} >
< {1} {5} {3} >
< {2} {3} {4} >
< {2 5} {3} >
< {3} {4} {5} >
< {5} {3 4} >

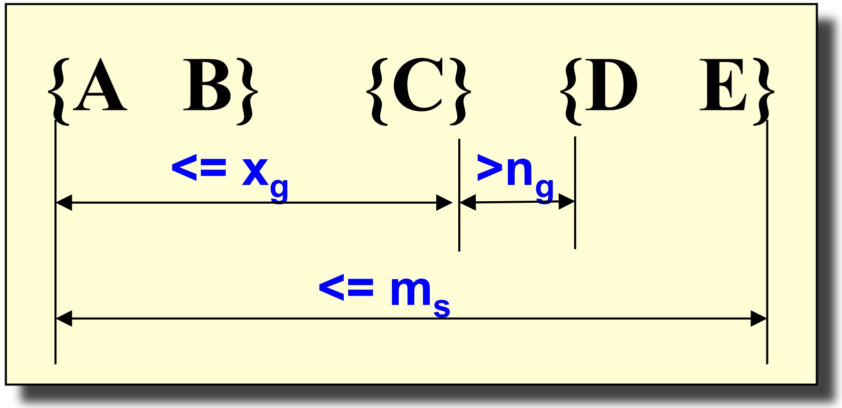
Candidate Generation

< {1} {2} {3} {4} >
< {1} {2 5} {3} >
< {1} {5} {3 4} >
< {2} {3} {4} {5} >
< {2 5} {3 4} >

Candidate Pruning

< {1} {2 5} {3} >

Timing Constraints for Subsequences



x_g: max-gap
n_g: min-gap
m_s: maximum span

These parameters induce constraints on time differences of adjacent or start-end events in subsequences (as indicated). Here, we assume elements of given data sequences to be timestamped by 1,2,3, …

Let x_g = 2, n_g = 0, m_s= 4.

Data sequence	Subsequence	Supports ? (Data sequence contains subsequence and subseq. satisfies constraints wrt data seq.) ?
< {2,4} {3,5,6} {4,7} {4,5} {8} >	< {6} {5} >	Yes
< {1} {2} {3} {4} {5}>	< {1} {4} >	No ($\cancel{x_g}$)
< {1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	Yes
< {1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	No ($\cancel{x_g}$, $\cancel{m_s}$)

Mining Sequential Patterns with Timing Constraints

- Approach 1:
  - Mine sequential patterns without timing constraints
  - Postprocess the discovered patterns

- Approach 2:
  - Modify GSP to directly prune candidates that violate timing constraints
  - But: Does Apriori principle still hold?

Apriori Principle for Sequence Data

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

Suppose:

x_g = 1 (max-gap)

n_g = 0 (min-gap)

m_s = 5 (maximum span)

minsup = 60%

<{2} {5}> support = 40%

but

<{2} {3} {5}> support = 60%

Problem exists because of max-gap constraint

No such problem if max-gap is infinite

<{2}{3}{5}> must not be pruned due to <{2}{5}>!

Contiguous Subsequences

- s is a contiguous* subsequence of

  $w = <e_1>< e_2>\ldots< e_k>$

  if any of the following conditions holds:

  1. s is obtained from w by deleting an item from either e_1 or e_k
  2. s is obtained from w by deleting an item from any element e_i that contains at least 2 items
  3. s is a contiguous subsequence of s', and s' is a contiguous subsequence of w (recursive definition)

- Examples: s = < {1} {2} >
  - is a contiguous subsequence of
    < {1} {2 3}>, < {1 2} {2} {3}>, and < {3 4} {1 2} {2 3} {4} >
  - is not a contiguous subsequence of
    < {1} {3} {2}> and < {1,2} {3} {2}>

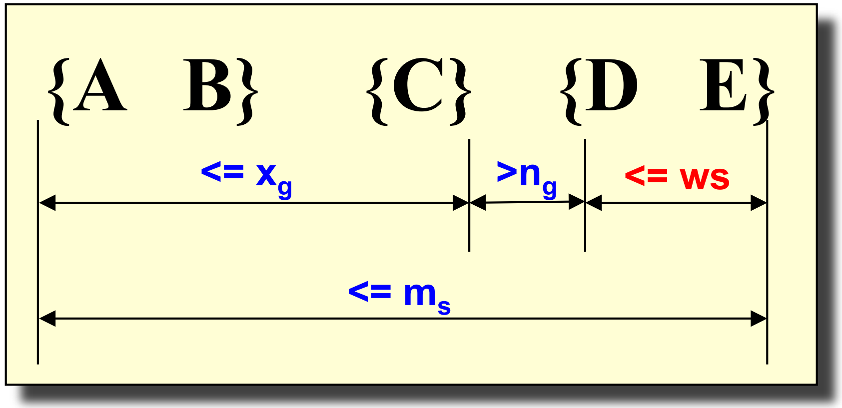
*) zusammenhängend

Modified Candidate Pruning Step

- Without maxgap constraint:
  - A candidate k-sequence is pruned if at least one of its (k-1)-subsequences is infrequent

- With maxgap constraint:
  - The following reduced Apriori principle still holds:
    If a k-sequence is frequent, then all of its **contiguous subsequences** (all gaps =1!) must be frequent.
  - Thus a candidate k-sequence is pruned if at least one of its contiguous (k-1)-subsequences is infrequent.
  - Already then support counting must be applied.

Timing Constraints (II)



x_g: max-gap

n_g: min-gap

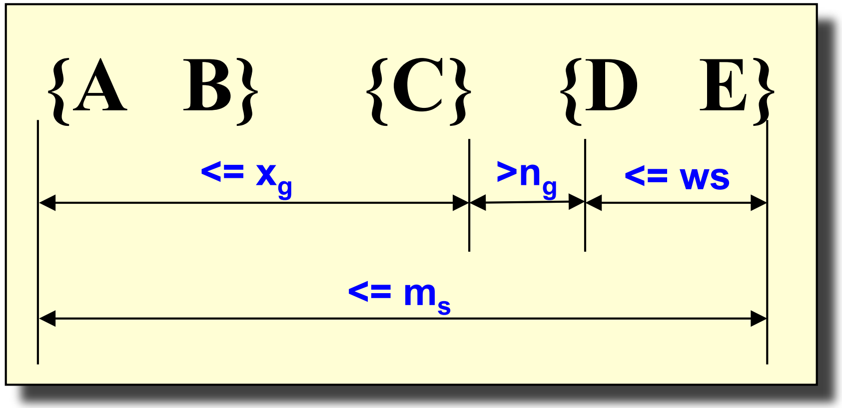
ws: window size
(within one element)

m_s: maximum span

x_g = 2, n_g = 0, ws = 1, m_s= 5

Data sequence	Subsequence	Supports?
< {2,4} {3,5,6} {4,7} {4,6} {8} >	< {3,4,5} >	Yes
< {1} {2} {3} {4} {5}>	< {1,2} {3,4} >	No (w_s)
< {1,2} {2,3} {3,4} {4,5}>	< {1,2} {3,4} >	Yes

Timing Constraints (II)



x_g: max-gap

n_g: min-gap

ws: window size

m_s: maximum span

x_g = 5, n_g = 0, ws = 1, m_s= 5

Data sequence	Subsequence	Supports?
< {DBS} {Statistics} {Data Mining} >	< {DBS,Statistics} {Data Mining} >	Yes
< {Statistics} {DBS} {Data Mining} >	< {DBS,Statistics} {Data Mining} >	Yes
< {Statistics} {X} {Y} {DBS} {Data Mining} >	< {DBS,Statistics} {Data Mining} >	No

Modified Support Counting Step

- Given a candidate pattern: <{a, c}>
  - All data sequences

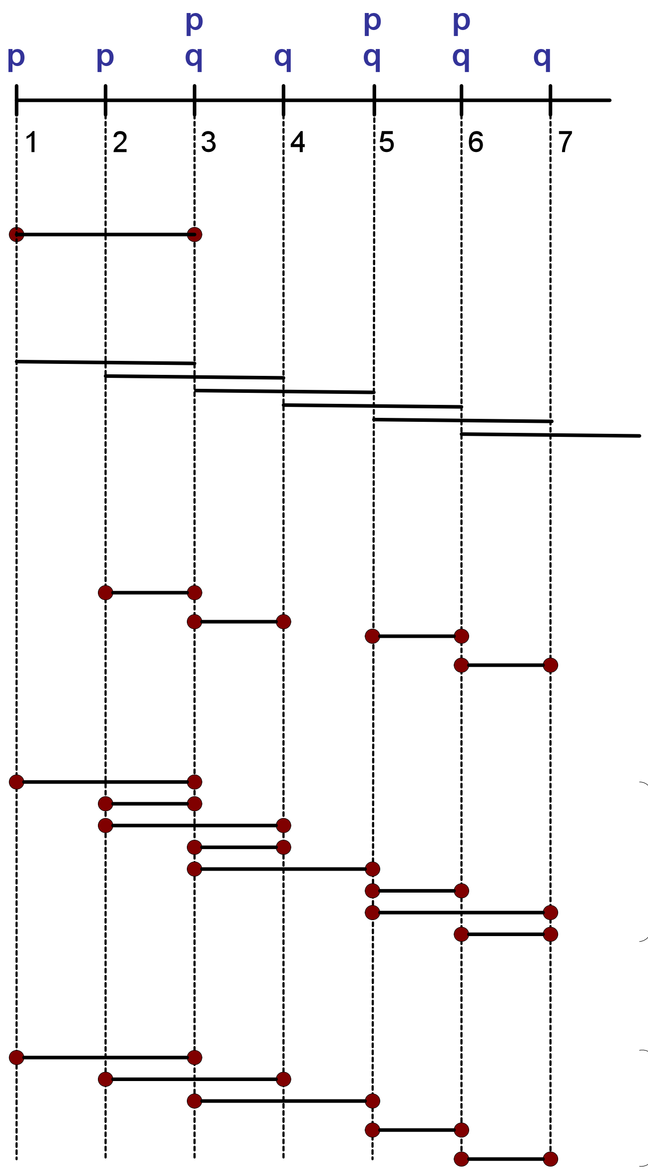
    <… {a c} … >,
    <… {a} … {c}…> (where time({c}) – time({a}) $\leq$ ws)
    <…{c} … {a} …> (where time({a}) – time({c}) $\leq$ ws)

    will contribute to the support count of the candidate pattern

- *Note!* Using min-gap and window size constraints, the original subsequence condition need not hold and need not be checked any more; actually, the original condition becomes a special case of constraint satisfaction:
  - **x_g / m_s** arbitrary
  - **ws=0** (only simultaneous events per element)
  - **n_g=0** (no nonpositive “gaps”, i.e. no order inversion nor simultaneity of elements)

Possible Support Counting Schemes

Object's Timeline



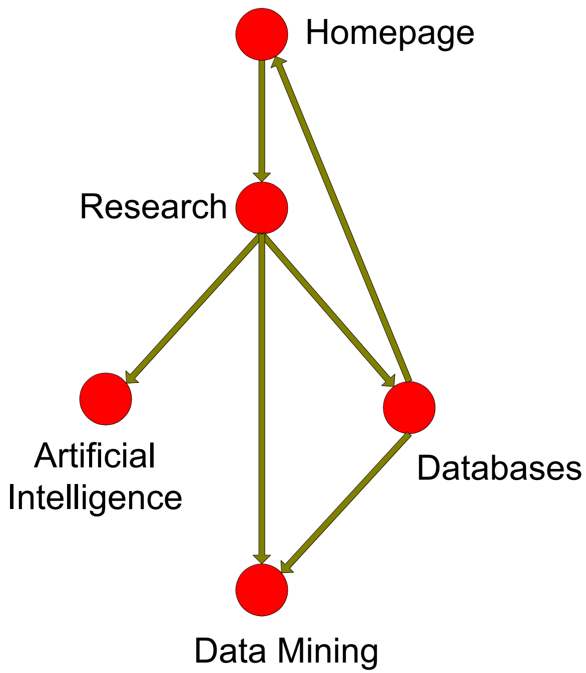
Sequence: (p) (q)

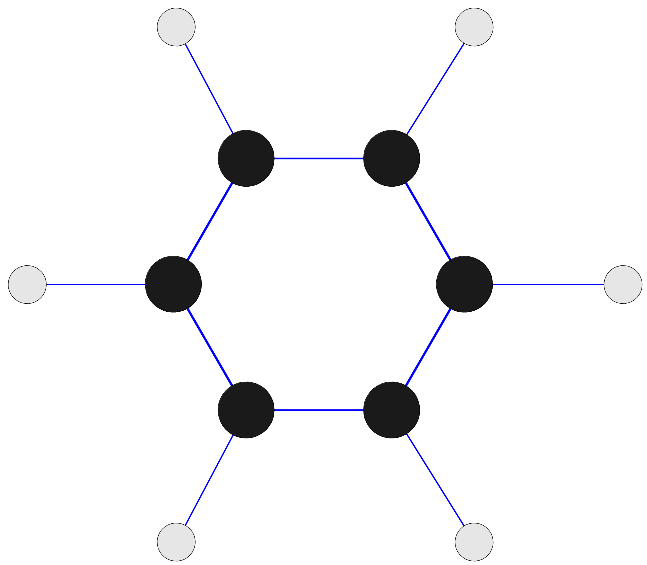
Method	Support Count	
COBJ	1	**One occurrence per object**
CWIN	6	**One occurrence per sliding window (of length maxspan)**
CMINWIN	4	**Number of minimal windows of occurrence**
CDIST_O	8	**Distinct occurrences with possible event-timestamp overlapping**
CDIST	5	**Distinct occurrences without event-timestamp overlapping**

Assume:
x_g = 2 (max-gap)
n_g = 0 (min-gap)
ws = 0 (window size)
m_s = 2 (maximum span)

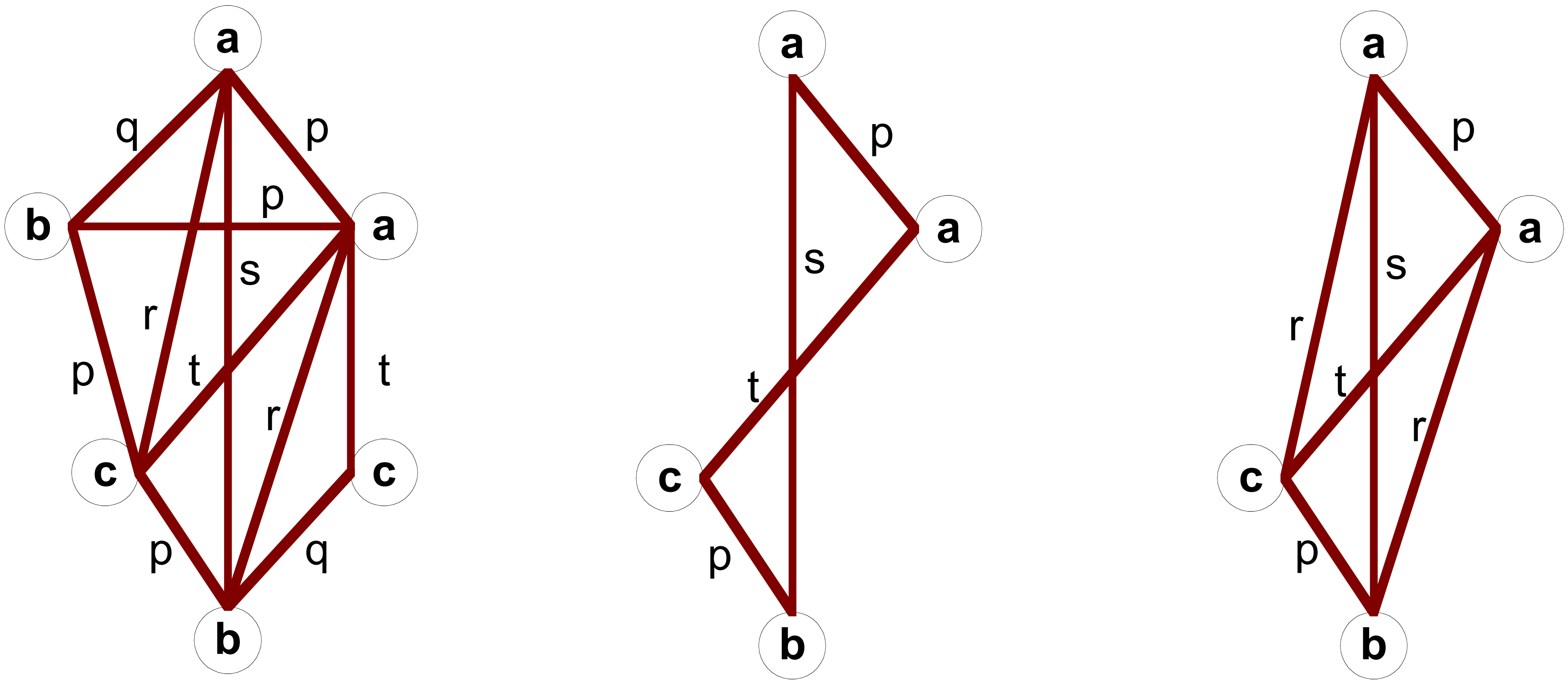
Frequent Subgraph Mining

- Extend association rule mining to finding
  frequent subgraphs
- Useful for web mining, semantic web mining (XML documents), computational chemistry, bioinformatics, spatial data sets, etc





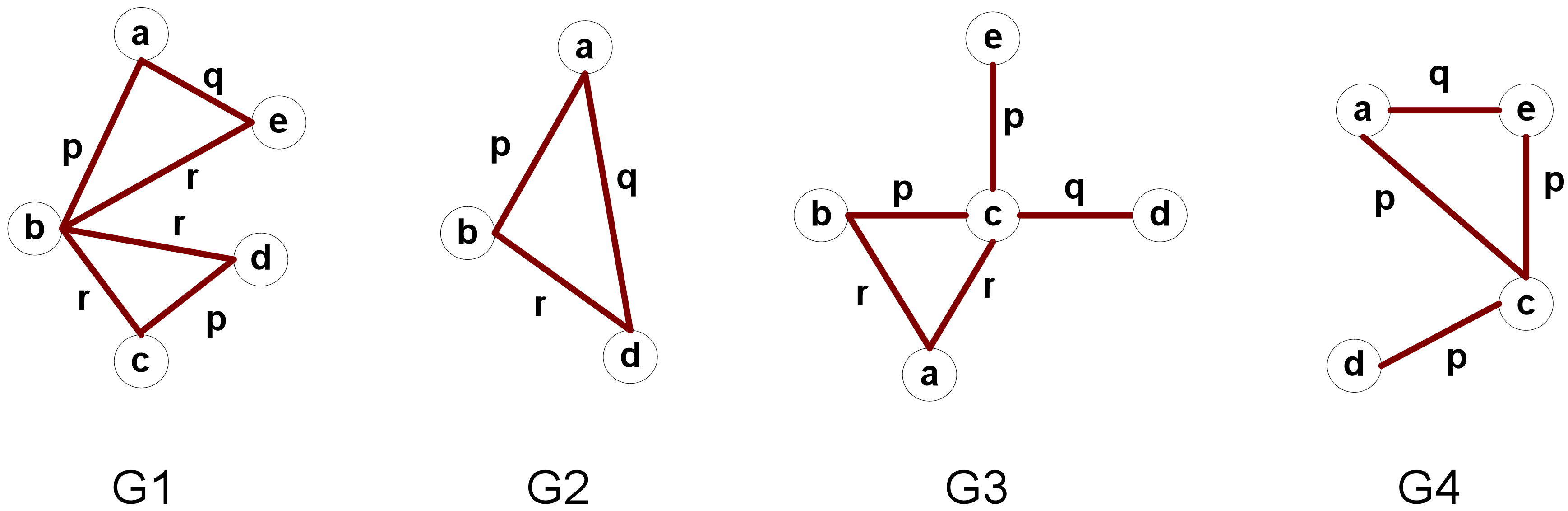
Graph Definitions



(a) Labeled Graph (b) Subgraph (c) Induced Subgraph

We focus on undirected, connected graphs.

Representing Graphs as Transactions



	(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
G4	0	0	0	0	0	0	…	0

A graph is considered as a set of edges represented by its vertex and edge labels. This works only, if these edge representations are unique.

Challenges

- Nodes may contain duplicate labels
- Support (and confidence)
  - How to define them?
- Additional constraints imposed by pattern structure
  - Support (and confidence) are not the only constraints
  - Assumption: frequent subgraphs must be ***connected***
- Apriori-like approach:
  - Use frequent k-subgraphs to generate frequent (k+1)-subgraphs
    - What is k?

Support

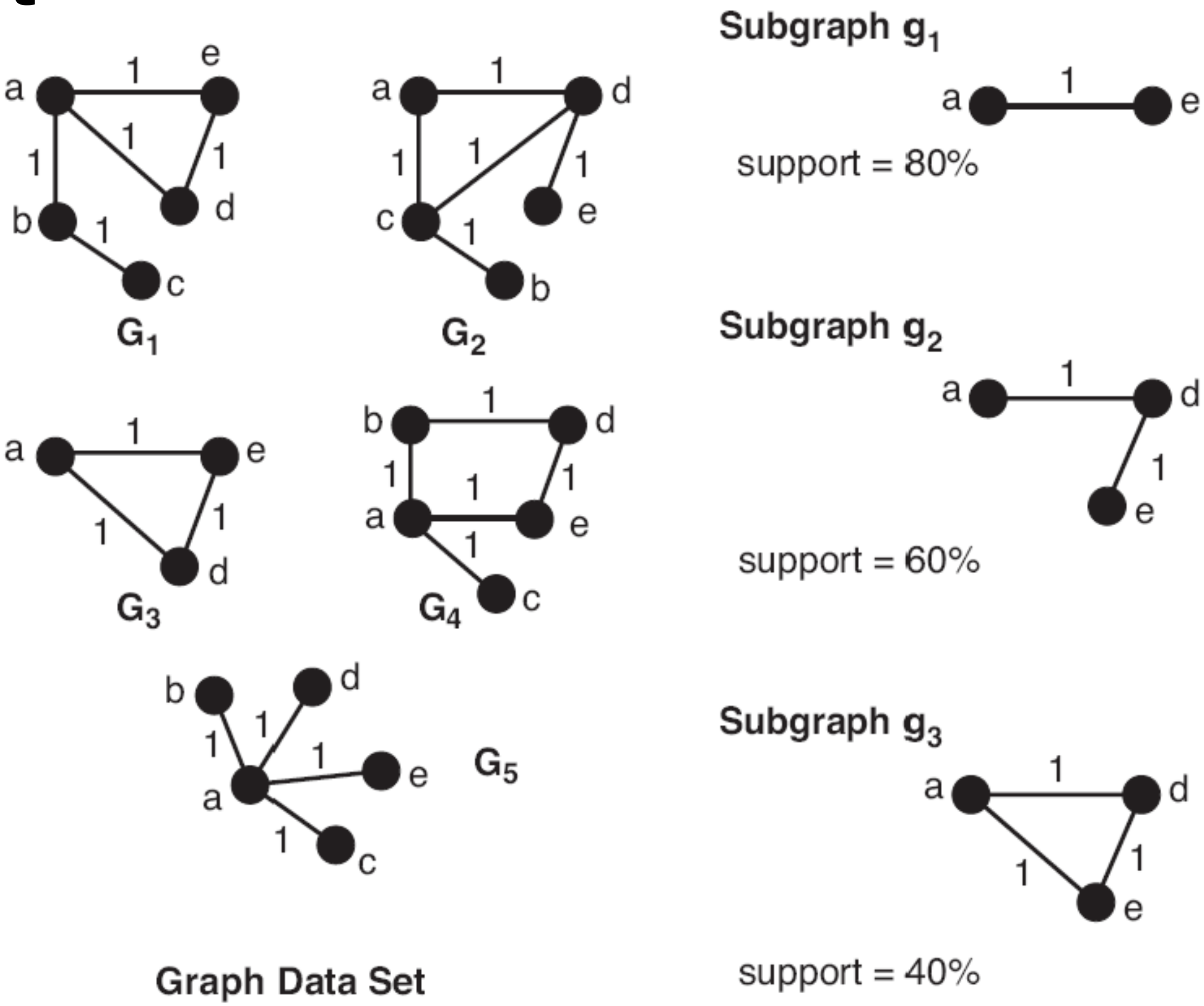
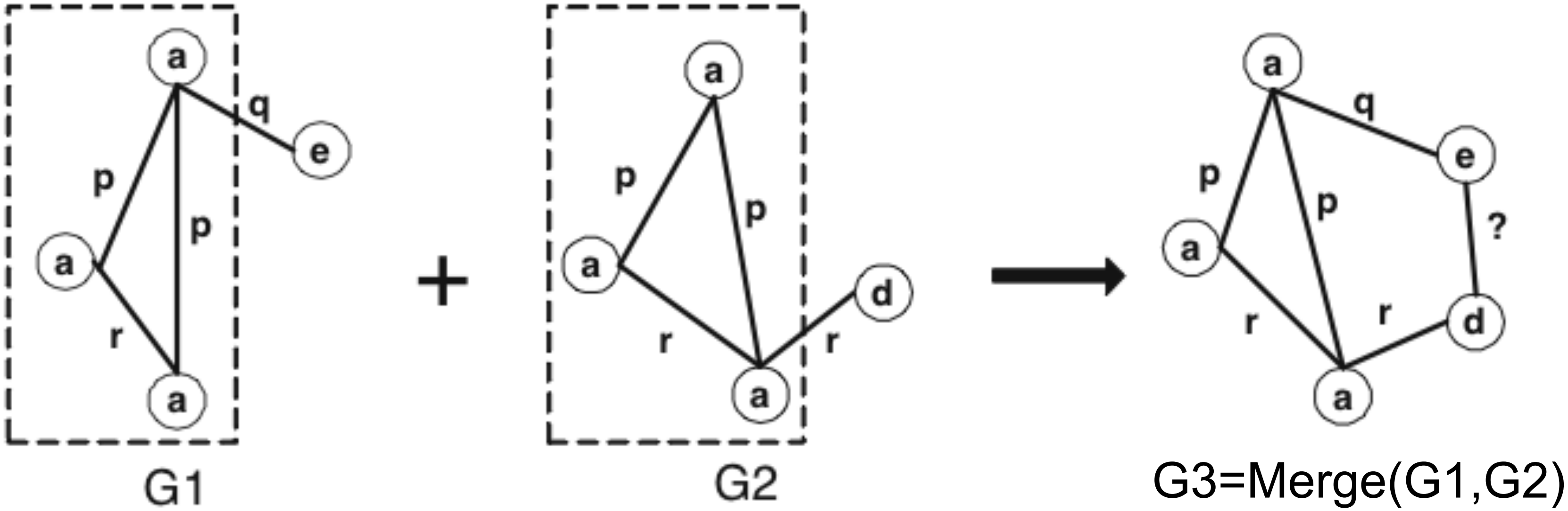


Figure 7.10. Computing the support of a subgraph from a set of graphs.

Challenges…

- Support:
  - number of graphs in a given graph DB that contain a particular subgraph
- Apriori principle still holds
- Level-wise (Apriori-like) approaches:
  - Vertex growing:
    - k is the number of vertices
  - Edge growing:
    - k is the number of edges

Vertex Growing

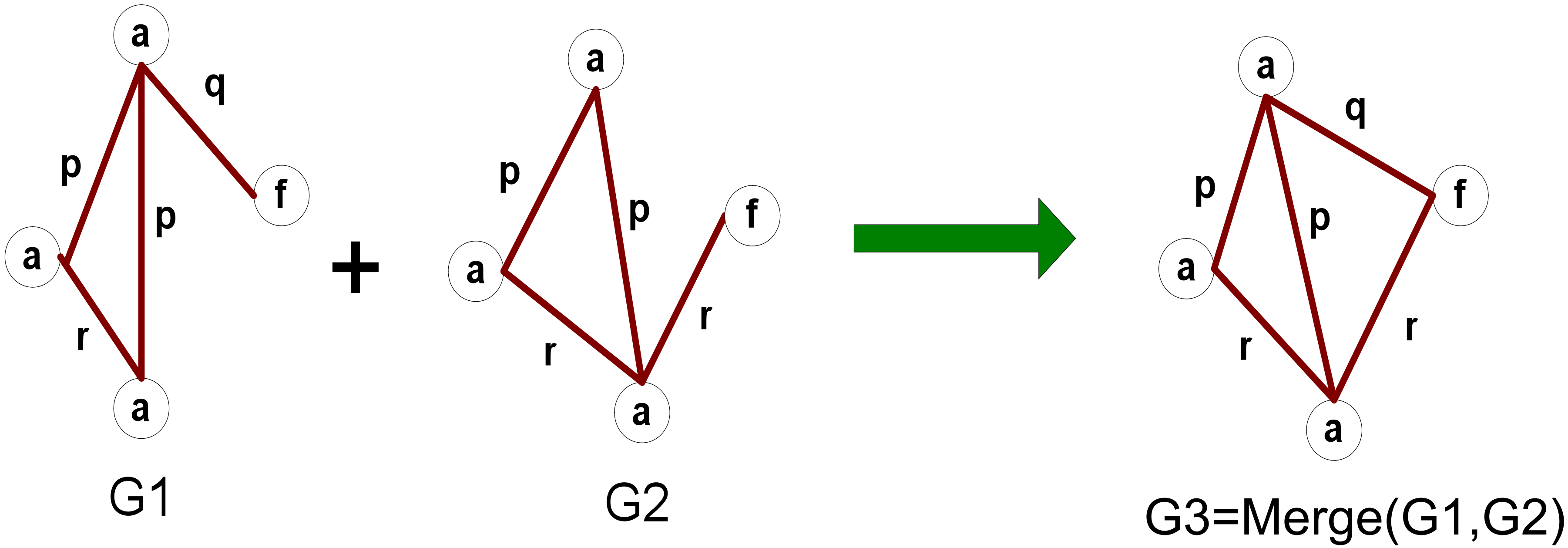


$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix} \quad M_{G2} = \begin{pmatrix} 0 & p & p & 0 \\ p & 0 & r & 0 \\ p & r & 0 & r \\ 0 & 0 & r & 0 \end{pmatrix} \quad M_{G3} = \begin{pmatrix} 0 & p & p & q & 0 \\ p & 0 & r & 0 & 0 \\ p & r & 0 &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

Figure 7.13 Vertex-growing strategy.

?: try no edge or arbitrary edge label

Edge Growing

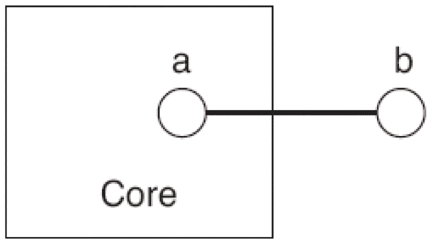


But this is not so simple !!!

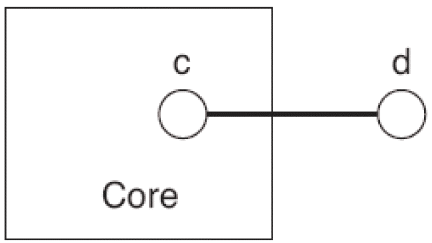
Exkurs:

Edge Growing: General rules

G1



G2



The cores (subgraphs after removing an edge) must be topologically equivalent.

The merging depends on whether a/c *are topologically equivalent(“*a=c*“) and* b/d *have identical labels (*b=d*)*

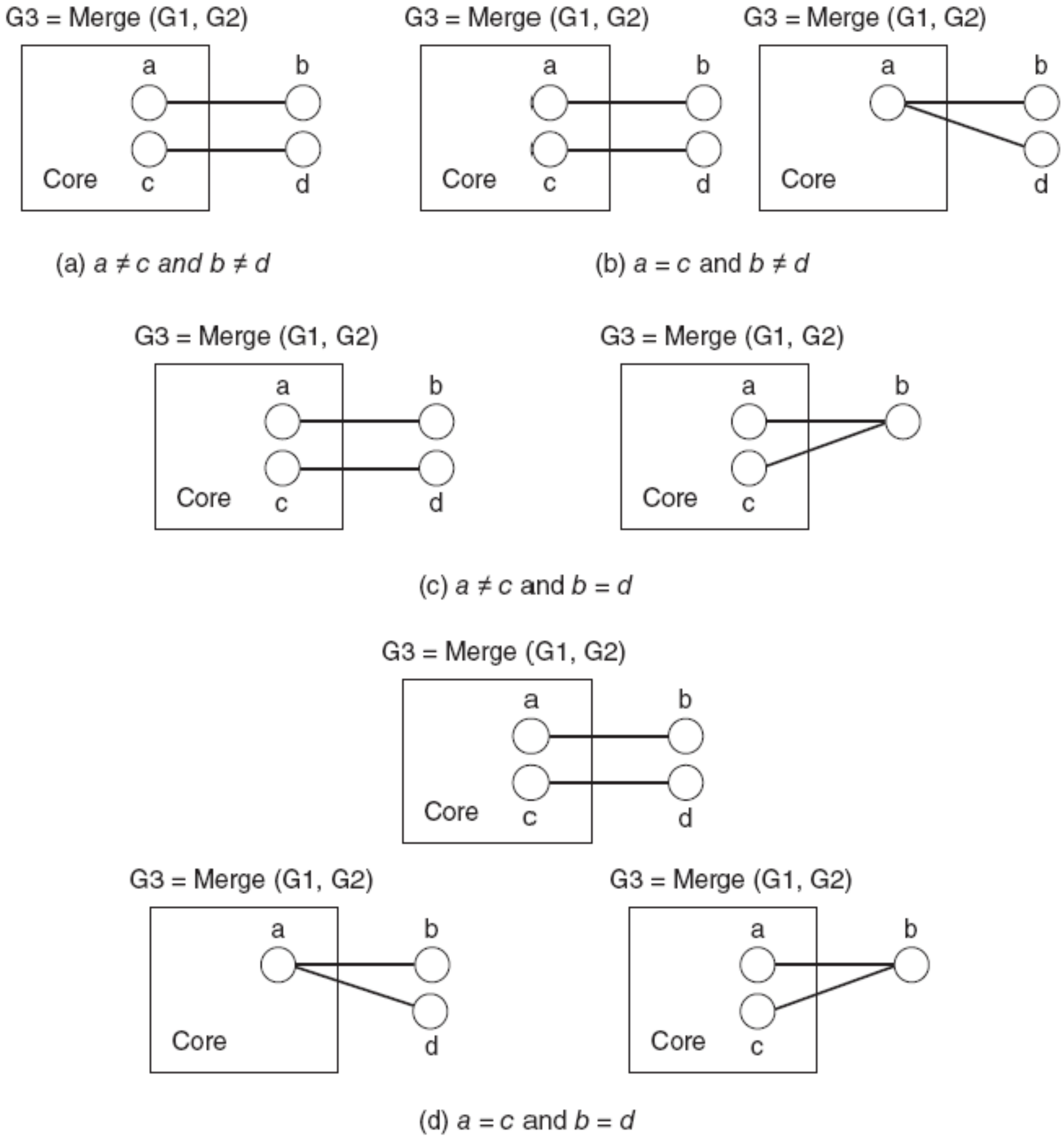


Figure 7.17. Candidate subgraphs generated via edge growing.

Apriori-like Algorithm

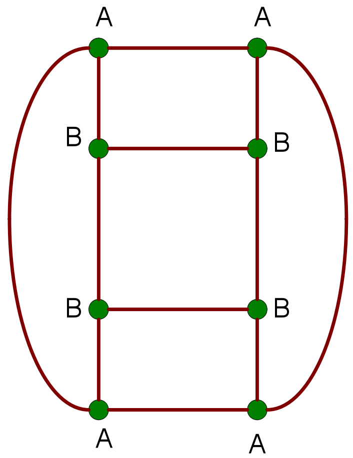
- Find frequent 1-subgraphs
- Repeat
  - Candidate generation
    - Use frequent (*k-1*)-subgraphs to generate candidate *k*-subgraph
  - Candidate pruning
    - Prune candidate subgraphs that contain infrequent (*k-1*)-subgraphs
  - Support counting
    - Count the support of each remaining candidate
  - Eliminate candidate *k*-subgraphs that are infrequent

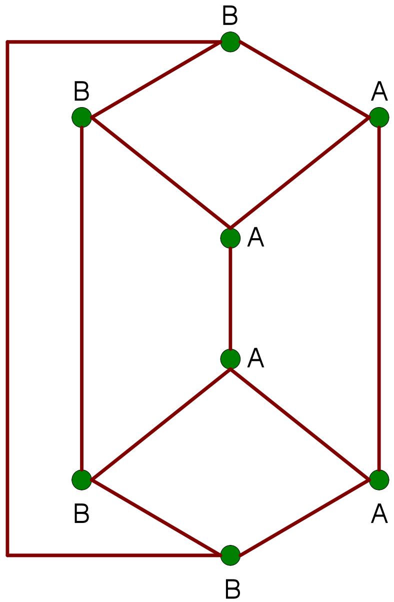
➢ *But there are many complications, e.g.:*

- ➢ Merging two frequent *k*-subgraphs may produce ***multiple*** candidate (*k+1*)-subgraphs
- ➢ How to check for graph identity/containment …

Graph Isomorphism

- Two graphs are isomorphic if they are topologically equivalent.

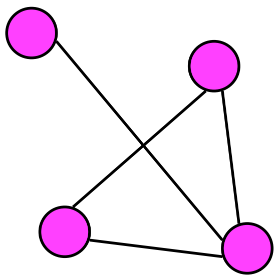




- Tests for (sub)graph isomorphism are needed:
  - During candidate generation step, to determine whether a new candidate has been generated
  - During candidate pruning step, to check whether its (*k-1*)-subgraphs are among the frequent (*k-1*)-graphs
  - During candidate counting, to check whether a candidate is contained within another graph

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
  - Example:
    - Lexicographically largest adjacency matrix (as a string)



$$\longrightarrow \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

String: 0010001111010110 **Canonical: 0111101011001000**