Data Mining
Mining Complex Data

Lecture Notes for Chapter 6

Data Mining
by
Zhaonian Zou

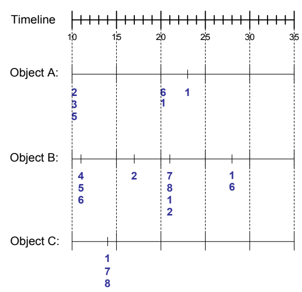
6.1 Mining Sequence Data

6.1.1 Sequence Data

Sequence Data

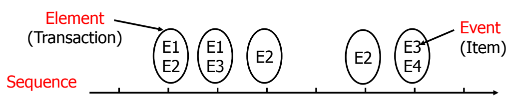
Sequence Database:

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 set of events (items)

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

  – Each element is attributed to a specific time or location
- 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

- Web sequence:

  < {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} >

- Sequence of initiating events causing the nuclear accident at 3-mile Island:
  (http://stellar-one.com/nuclear/staff_reports/summary_SOE_the_initiating_event.htm)

  < {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 at a library:

  <{Fellowship of the Ring} {The Two Towers} {Return of the King}>

6.1 Mining Sequence Data

6.1.2 Sequential Pattern Mining

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 $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i1}$, …, $a_n \subseteq b_{in}$

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

- 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} {g h i}>
  - Examples of subsequences:
    <{a} {c d} {f} {g} >, < {c d e} >, < {b} {g} >, etc.
- How many k-subsequences can be extracted from a given n-sequence?

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

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

<{a} {d e} {i}>

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

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

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%

Extracting Sequential Patterns

- Given n events: $i_1, i_2, i_3, \ldots, i_n$
- Candidate 1-subsequences:
  $<\{i_1\}>, <\{i_2\}>, <\{i_3\}>, \ldots, <\{i_n\}>$
- Candidate 2-subsequences:
  $<\{i_1, i_2\}>, <\{i_1, i_3\}>, \ldots, <\{i_1\}\ \{i_1\}>, <\{i_1\}\ \{i_2\}>, \ldots, <\{i_{n-1}\}\ \{i_n\}>$
- Candidate 3-subsequences:
  $<\{i_1, i_2, i_3\}>, <\{i_1, i_2, i_4\}>, \ldots, <\{i_1, i_2\}\ \{i_1\}>, <\{i_1, i_2\}\ \{i_2\}>, \ldots,$
  $<\{i_1\}\ \{i_1, i_2\}>, <\{i_1\}\ \{i_1, i_3\}>, \ldots, <\{i_1\}\ \{i_1\}\ \{i_1\}>, <\{i_1\}\ \{i_1\}\ \{i_2\}>, \ldots$

6.1 Mining Sequence Data

6.1.3 GSP Algorithm

Generalized Sequential Pattern (GSP)

- **Step 1**:
  - Make the first pass over the sequence database D to yield all the 1-item frequent 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 $<\{i_1\}>$ and $<\{i_2\}>$ will produce three candidate 2-sequences: $<\{i_1\}\ \{i_2\}>$, $<\{i_2\}\ \{i_1\}>$ and $<\{i_1\ i_2\}>$
- 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 in w_1
      - Otherwise, the last event in w_2 becomes a separate element appended to the end of w_1

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
- 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 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} >

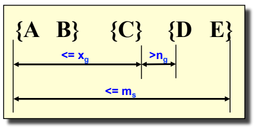
Candidate Pruning

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

6.1 Mining Sequence Data

6.1.4 Sequential Pattern Mining under Timing Constraints

Timing Constraints (I)



x_g: max-gap

n_g: min-gap

m_s: maximum span

$x_g = 2$, $n_g = 0$, $m_s = 4$

Data sequence	Subsequence	Contain?
< {2,4} {3,5,6} {4,7} {4,5} {8} >	< {6} {5} >	Yes
< {1} {2} {3} {4} {5}>	< {1} {4} >	No
< {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

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
  - Question:
    - 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

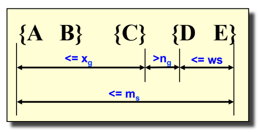
Contiguous Subsequences

- s is a contiguous subsequence of
  $w = e_1, e_2, \ldots, e_k$
  if any of the following conditions hold:
  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 more than 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 < {2} {1} {3} {2}>

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:
  - A candidate *k*-sequence is pruned if at least one of its **contiguous** (*k-1*)-subsequences is infrequent

Timing Constraints (II)



x_g: max-gap

n_g: min-gap

ws: window size

m_s: maximum span

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

Data sequence	Subsequence	Contain?
< {2,4} {3,5,6} {4,7} {4,6} {8} >	< {3} {5} >	No
< {1} {2} {3} {4} {5}>	< {1,2} {3} >	Yes
< {1,2} {2,3} {3,4} {4,5}>	< {1,2} {3,4} >	Yes

Modified Support Counting Step

- Given a candidate pattern: <{a, c}>
  - Any data sequences that contain

    <… {a c} … >,
    <… {a} … {c}…>   (where time({c}) – time({a}) ≤ ws)
    <…{c} … {a} …>   (where time({a}) – time({c}) ≤ ws)

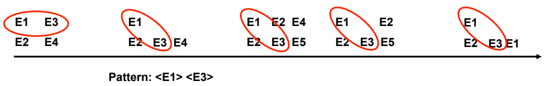
    will contribute to the support count of candidate pattern

6.1 Mining Sequence Data

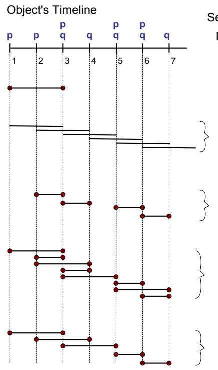
6.1.5 Variants of Sequential Pattern Mining

Other Formulation

- In some domains, we may have only one very long time series
  - Example:
    - monitoring network traffic events for attacks
    - monitoring telecommunication alarm signals
- Goal is to find frequent sequences of events in the time series
  - This problem is also known as frequent episode mining



General Support Counting Schemes



Sequence: (p) (q)

Method	Support Count
COBJ	1
CWIN	6
CMINWIN	4
CDIST_O	8
CDIST	5

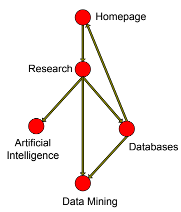
Assume:
x_g = 2 (max-gap)
n_g = 0 (min-gap)
ws = 0 (window size)
m_s = 2 (maximum span)

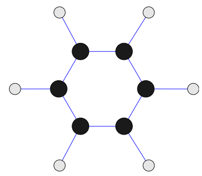
6.2 Mining Graph Data

6.2.1 Graph Data

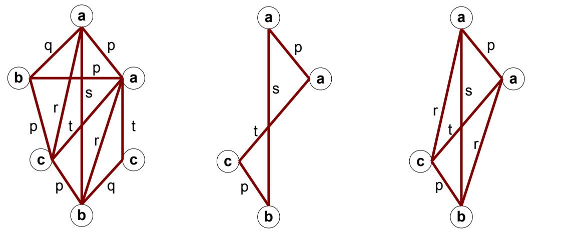
Frequent Subgraph Mining

- Extend association rule mining to finding frequent subgraphs
- Useful for Web Mining, computational chemistry, bioinformatics, spatial data sets, etc





Graph Definitions

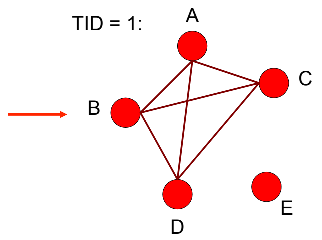


(a) Labeled Graph (b) Subgraph (c) Induced Subgraph

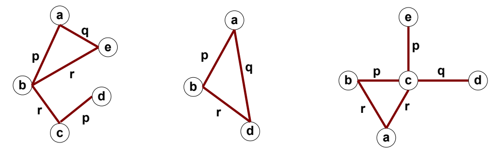
Representing Transactions as Graphs

- Each transaction is a clique of items

Transaction Id	Items
1	{A,B,C,D}
2	{A,B,E}
3	{B,C}
4	{A,B,D,E}
5	{B,C,D}



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	…	…	…	…	…	…	…	…

6.2 Mining Graph Data

6.2.2 Subgraph Pattern Mining

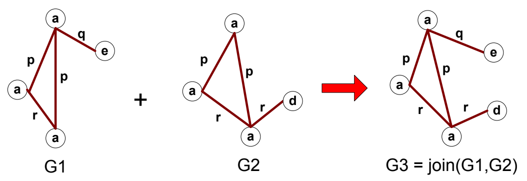
Challenges

- Node 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?

Challenges…

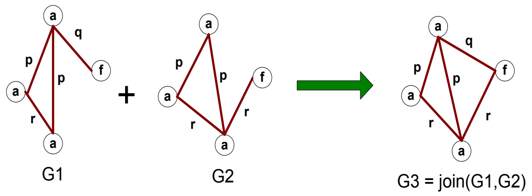
- Support:
  - number of graphs that contain a particular subgraph
- Apriori principle still holds
- Level-wise (Apriori-like) approach:
  - Vertex growing:
    - k is the number of vertices
  - Edge growing:
    - k is the number of edges

Vertex Growing



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

Edge Growing

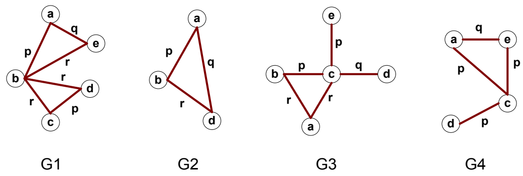


Apriori-like Algorithm

- 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

In practice, it is not as easy. There are many other issues

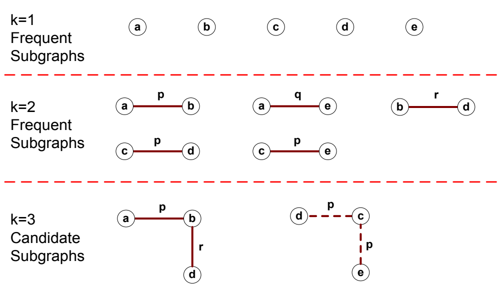
Example: Dataset



	(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

Example

Minimum support count = 2

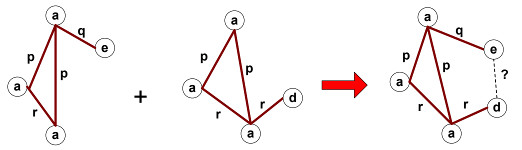


(Pruned candidate)

Candidate Generation

- In Apriori:
  - Merging two frequent *k*-itemsets will produce a candidate (*k+1*)-itemset
- In frequent subgraph mining (vertex/edge growing)
  - Merging two frequent *k*-subgraphs may produce more than one candidate (*k+1*)-subgraph

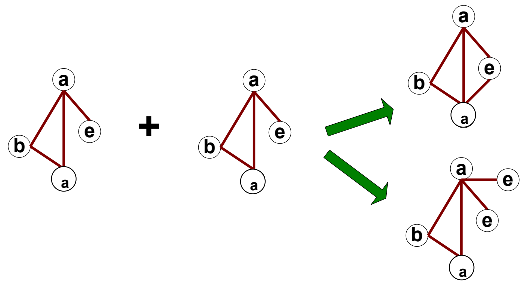
Multiplicity of Candidates (Vertex Growing)



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

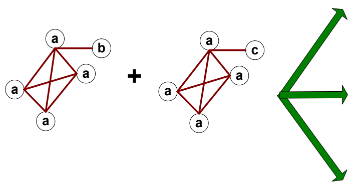
Multiplicity of Candidates (Edge growing)

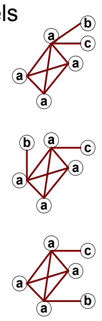
- Case 1: identical vertex labels



Multiplicity of Candidates (Edge growing)

- Case 2: Core contains identical labels

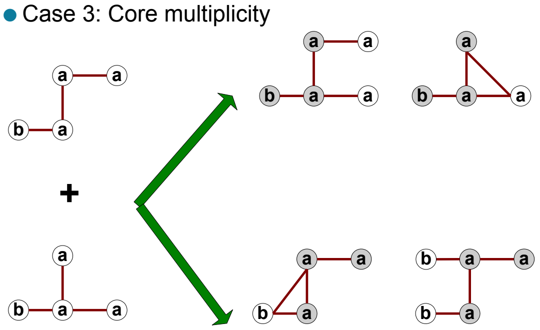




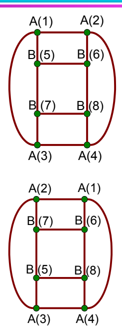
Core: The (k-1) subgraph that is common between the joint graphs

Multiplicity of Candidates (Edge growing)

- Case 3: Core multiplicity



Adjacency Matrix Representation



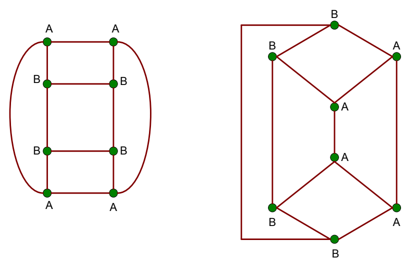
	A(1)	A(2)	A(3)	A(4)	B(5)	B(6)	B(7)	B(8)
A(1)	1	1	1	0	1	0	0	0
A(2)	1	1	0	1	0	1	0	0
A(3)	1	0	1	1	0	0	1	0
A(4)	0	1	1	1	0	0	0	1
B(5)	1	0	0	0	1	1	1	0
B(6)	0	1	0	0	1	1	0	1
B(7)	0	0	1	0	1	0	1	1
B(8)	0	0	0	1	0	1	1	1

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

- **The same graph can be represented in many ways**

Graph Isomorphism

- A graph is isomorphic if it is topologically equivalent to another graph

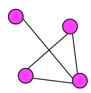


Graph Isomorphism

- Test for graph isomorphism is needed:
  - During candidate generation step, to determine whether a candidate has been generated
  - During candidate pruning step, to check whether its (*k-1*)-subgraphs are frequent
  - 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



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

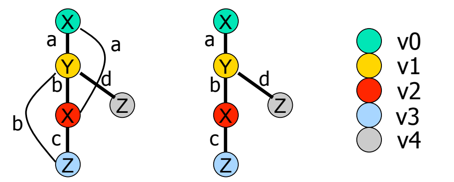
String: 0010001111010110 **Canonical: 0111101011001000**

6.2 Mining Graph Data

6.2.3 gSpan Algorithm

DFS Tree

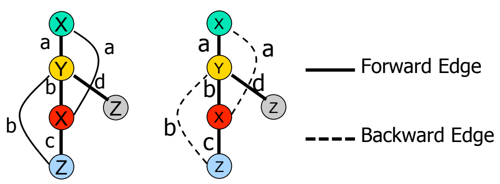
- Perform a DFS on a graph and obtain a DFS tree



Graph DFS Tree DFS Order

Forward and Backward Edges

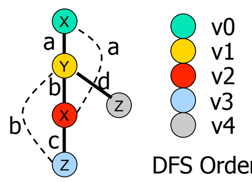
- Any edge in the DFS tree is a forward edge
- Any edge not in the DFS tree is a backward edge



Graph

DFS Code

- An edge (v_i,v_j) is represented by a 5-tuple $(v_i, v_j, l(v_i), l(v_j), l(v_i,v_j))$ to represent an edge
  - v_i and v_j are the ID of v_i and v_j
  - $l(v_i)$ and $l(v_j)$ are the labels of v_i and v_j
  - $l(v_i,v_j)$ is the label of the edge (v_i,v_j)

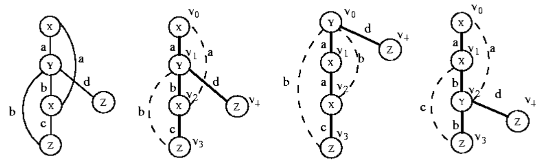


DFS Order

e0: (0,1,x,y,a)
e1: (1,2,y,x,b)
e2: (2,0,x,x,a)
e3: (2,3,x,z,c)
e4: (3,1,x,y,b)
e5: (1,4,x,z,d)

Minimum DFS Code

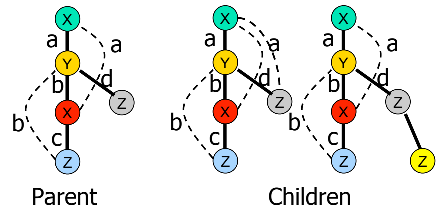
- A graph may have multiple DFS codes



- The minimum DFS code of a graph is the smallest one in lexicographic order.

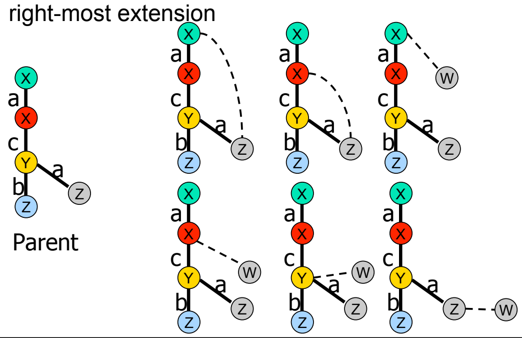
Parent and its Children

- A DFS code $(e_0,e_1,\ldots,e_n)$ is the parent of another DFS code $(e'_0,e'_1,\ldots,e'_m)$ if
  - $m = n + 1$
  - $e_i = e'_i$ for $i = 0, 1, \ldots, n$



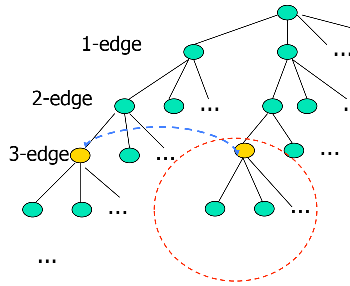
Right-Most Extension

- All children of a DFS code can be obtained by right-most extension



DFS Code Enumeration Tree

- All DFS codes can be organized as a DFS tree with respect to the parent-child relationship



If a DFS code is not minimum, then the subtree rooted at the DFS code can be pruned!

gSpan Algorithm

- Scan the database to find all frequent edges
- For each frequent edge e do
  - Perform a DFS on the subtree rooted at e
  - If a DFS code C visited in the DFS is minimum, then
    - Count the support of C
    - If sup(C) >= minsup then
      - output C
      - Get the children of C by right-most extension

Thank You!