CS522 Advanced Database Systems
Mining Frequent Patterns

Chengyu Sun
California State University, Los Angeles

Sales Transactions

TID	Transactions
1	Beef, Chicken, Milk
2	Beef, Cheese
3	Cheese, Boots
4	Beef, Chicken, Cheese
5	Beef, Chicken, Clothes, Cheese, Milk
6	Chicken, Clothes, Milk
7	Chicken, Clothes, Milk
8	Beef, Milk

Support Count

- The support count, or frequency, of a itemset is the number of the transactions that contain the itemset
  - Item, Itemset, and Transaction
- Examples:
  - `support_count({beef})=5`
  - `support_count({beef,chicken,milk})=??`

Frequent Itemset

- An itemset is frequent if its support count is greater than or equals to a minimum support count threshold
  - `support_count(X)≥min_sup`

The Need for Closed Frequent Itemsets

- Two transactions
  - $<a_1, a_2, \ldots, a_{100}>$ and $<a_1, a_2, \ldots, a_{50}>$
- `min_sup=1`
- \# of frequent itemsets??

Closed Frequent Itemset

- An itemset **X** is closed if there exists no *proper superset* of **X** that has the same support count
- A closed frequent itemset is an itemset that is both *closed* and *frequent*

Closed Frequent Itemset Example

- Two transactions
  - $\langle a_1, a_2, \ldots, a_{100} \rangle$ and $\langle a_1, a_2, \ldots, a_{50} \rangle$
- `min_sup=1`
- Closed frequent itemset(s)??

Maximal Frequent Itemset

- An itemset **X** is a maximal frequent itemset if **X** is frequent and there exists no *proper superset* of **X** that is also frequent
- Example: if `{a,b,c}` is a maximal frequent itemset, which one of these *cannot* be a MFI
  - `{a,b,c,d}, {a,c}, {b,d}`

Maximal Frequent Itemset Example

- Two transactions
  - $\langle a_1, a_2, \ldots, a_{100} \rangle$ and $\langle a_1, a_2, \ldots, a_{50} \rangle$
- `min_sup=1`
- Maximal frequent itemset(s)??
- Maximal frequent itemset vs. closed frequent itemset??

From Frequent Itemsets to Association Rules

- `{chicken,cheese}` is a frequent set
- `{chicken}`⇒`{cheese}`??
- Or is it `{cheese}`⇒`{chicken}`??

Association Rules

- **A**⇒**B**
  - **A** and **B** are itemsets
  - **A**∩**B**=∅

Support

- The support of **A**⇒**B** is the percentage of the transactions that contain **A**∪**B**

$$\text{support}(A \Rightarrow B) = P(A \cup B) = \frac{\text{support\_count}(A \cup B)}{|D|}$$

$P(A \cup B)$ is the probability that a transaction contains $A \cup B$
D is the set of the transactions

Confidence

- The confidence of **A**$\Rightarrow$**B** is the percentage of the transactions containing **A** that also contains **B**

$$\text{confidence}(A \Rightarrow B) = P(B \mid A) = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)}$$

Support and Confidence Example

- `{chicken}`$\Rightarrow$`{cheese}`??
- `{cheese}`$\Rightarrow$`{chicken}`??

Strong Association Rule

- An association rule is strong if it satisfies both a minimum support threshold (`min_sup`) and a minimum confidence threshold (`min_conf`)
- Why do we need both *support* and *confidence*??

Association Rule Mining

- Find strong association rules
  - Find all frequent itemsets
  - Generate strong association rules from the frequent itemsets

The Apriori Property

- All nonempty subsets of a frequent itemset must also be frequent
- Or, if an itemset is not frequent, its supersets cannot be frequent either

Finding Frequent Itemsets – The Apriori Algorithm

- Given `min_sup`
- Find the frequent 1-itemsets L_1
- Find the the frequent k-itemsets L_k by joining the itemsets in L_{k-1}
- Stop when L_k is empty

Apriori Algorithm Example

beef	1
chicken	2
milk	3
cheese	4
boots	5
clothes	6

TID	Transactions
1	1, 2, 3
2	1, 4
3	4, 5
4	1, 2, 4
5	1, 2, 6, 4, 3
6	2, 6, 3
7	2, 6, 3
8	1, 3

- Support 25%

L_1

- Scan the data once to get the count of each item
- Remove the items that do not meet `min_sup`

C_1	support_count	L_1
{1}	5	{1}
{2}	5	{2}
{3}	5	{3}
{4}	4	{4}
{5}	1	
{6}	3	{6}

L_2

- $C_2=L_1 \times L_1$
- Scan the dataset again for the support_count of C_2, then remove non-frequent itemsets from C_2, i.e. $C_2 \rightarrow L_2$

C_2	support_count	L_2
{1,2}	3	{1,2}
{1,3}	3	{1,3}
{1,4}	3	{1,4}
{1,6}	1	
{2,3}	4	{2,3}
{2,4}	2	{2,4}
{2,6}	3	{2,6}
{3,4}	1	
{3,6}	3	{3,6}
{4,6}	1	

L_3

- ??

From L_{k-1} to C_k

- Let l_i be an itemset in L_{k-1}, and $l_i[j]$ be the jth item in l_i
- Items in an itemset are sorted, i.e. $l_i[1]<l_i[2]<\ldots<l_i[k-1]$
- l_1 and l_2 are joinable if
  - Their first k-2 items are the same, and
  - $l_1[k-1]<l_2[k-1]$

From C_k to L_k

- Reduce the size of C_k using the Apriori property
  - any (k-1)-subset of an candidate must be frequent, i.e. in L_{k-1}
- Scan the dataset to get the support counts

Generate Association Rules from Frequent Itemsets

- For each frequent itemset `l`, generate all nonempty subset of `l`
- For every nonempty subset of `s` of `l`, output rule `s⇒(l-s)` if `conf(s⇒(l-s))≥min_conf`

Confidence-based Pruning …

- `conf({a,b}⇒{c,d})<min_conf`
  - `conf({a}⇒{c,d})??`
  - `conf({a,b,e}⇒{c,d})??`
  - `conf({a}⇒{b,c,d})??`

… Confidence-based Pruning

- If `conf(s⇒(l-s))<min_conf`, then `conf(s'⇒(l-s'))<min_conf` where `s'⊆s`.
- Example: `conf({a,b}⇒{c,d})<min_conf`
  - ??

Limitations of the Apriori Algorithm

- Multiple scans of the datasets
  - How many??
- Need to generate a large number of candidate sets

FP-Growth Algorithm

- Frequent-pattern Growth
- Mine frequent itemsets *without candidate generation*

FP-Growth Example

TID	Transactions
1	I1, I2, I5
2	I2, I4
3	I2, I3, I6
4	I1, I2, I4
5	I1, I3
6	I2, I3
7	I1, I3
8	I1, I2, I3, I5
9	I1, I2, I3

min_sup=2

L

- Scan the dataset and find the frequent 1-itemsets
- Sort the 1-itemsets by support count in descending order

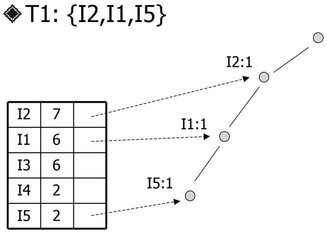
L
I2: 7
I1: 6
I3: 6
I4: 2
I5: 2

FP-tree

- Each transaction is processed in `L` order (why??) and becomes a branch in the FP tree
- Each node is linked from `L`

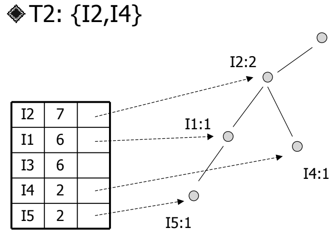
FP-tree Construction …

- T1: {I2,I1,I5}



… FP-tree Construction …

- T2: {I2,I4}



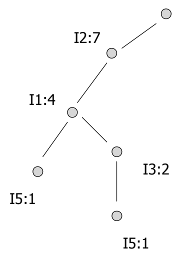
… FP-tree Construction

- ??

Mining the FP-tree

- For each item `i` in `L` (in ascending order), find the branch(s) in the FP tree that ends in `i` – Prefix Paths
  - If there is only one path, generate all the frequent patterns ended in `i`
  - Else create the Conditional FP-tree for `i` and recursively run the mining algorithm on the conditional FP-tree

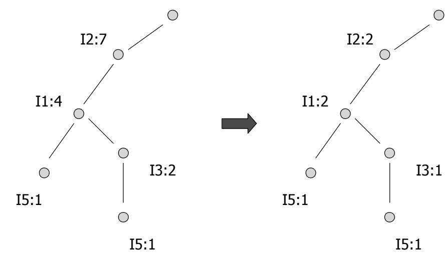
Prefix Paths of I5



I5 is frequent ➔ {I5:2}

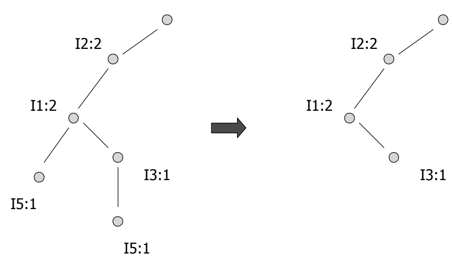
From Prefix Paths to Conditional FP-tree

◈Adjust the support counts



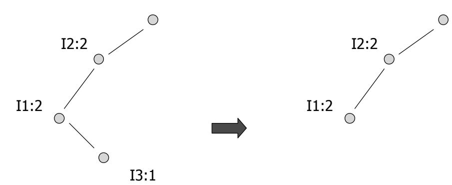
From Prefix Paths to Conditional FP-tree

◈Remove the suffix

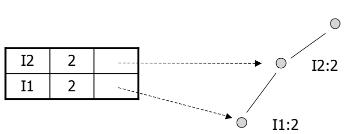


From Prefix Paths to Conditional FP-tree

◈Remove the infrequent items

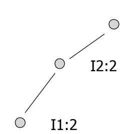


Conditional FP-tree for I5



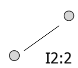
◈A FP-tree with *suffix pattern* {I5}

Prefix Paths of {I1,I5}



{I1,I5} is frequent and there is a single path
➔ {I1,I5:2}, {I2,I1,I5:2}

Prefix Paths of {I2,I5}

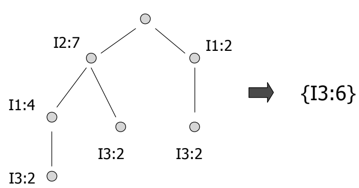


{I2,I5} is frequent and there is a single path
➔ {I2,I5:2}

All Frequent Itemsets with Suffix I5

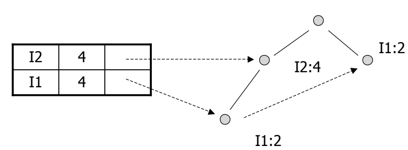
- {I5:2}
- {I1,I5:2}, {I2,I1,I5:2}
- {I2,I5:2}

Mining The FP-tree – I3 …



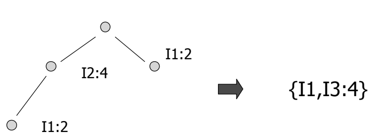
Prefix Paths of I3

… Mining The FP-tree – I3 …



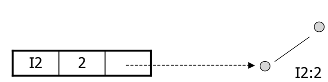
Conditional FP-tree with suffix pattern {I3}

… Mining The FP-tree – I3 …



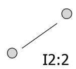
Prefix Paths of {I1,I3}

… Mining The FP-tree – I3 …



Conditional FP-tree for {I2,I1,I3}

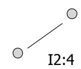
… Mining The FP-tree – I3 …



➡ {I2,I1,I3:2}

Prefix Paths of {I2,I1,I3}

… Mining The FP-tree – I3



➡ {I2,I3:4}

Prefix Paths of {I2,I3}

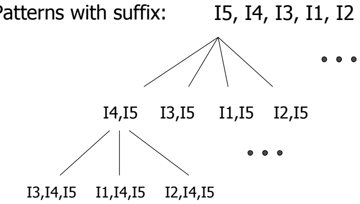
All Frequent Itemsets with Suffix I3

- {I3:6}
- {I1,I3:4}
- {I2,I1,I3:2}
- {I2,I3:4}

About FP-tree Mining

- A divide-and-conquer approach

Patterns with suffix: I5, I4, I3, I1, I2



Data Partitioning

- Divide dataset into `n` non-overlapping partitions such that *each partition fits into main memory*
- Find local frequent itemsets in each partition (1 scan)
  - Local `min_sup`??
- All local frequent itemsets form a candidate set
  - Will it include all the global frequent itemsets??
- Find global frequent itemsets from candidates (1 scan)

Vertical Data Format

Item	TID_set
I1	T1,T4,T5,T7,T8,T9
I2	T1,T2,T3,T4,T6,T8,T9
I3	T3,T5,T6,T7,T8,T9
I4	T2,T4
I5	T1,T8

- And how does it help??

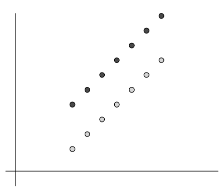
Strong Association Rules Could Be Misleading …

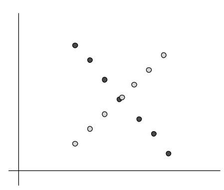
- Example:
  - 10,000 transactions
  - 6,000 transactions included games
  - 7,500 transactions included videos
  - 4,000 transactions included both
- {game} $\Rightarrow$ {video}
  - Support?? Confidence??

… Strong Association Rules Could Be Misleading

- Does buying game really imply buying video as well??

Correlation





Multiplication Rule

- If two events A and B are independent of each other

$$P(AB) = P(A)P(B)$$

From Multiplication Rule to Lift

$$lift(A,B) = \frac{P(AB)}{P(A)P(B)}$$

- `lift({game},{video})=??`

Problem of Lift

datasets	mc	m'c	mc'	m'c'	total	lift
A_1	100	100	100	100	400	??
A_2	100	100	100	1,000	1,300	??
A_3	100	100	100	10,000	10,300	??
A_4	100	100	100	100,000	100,300	??

mc: # of transactions that contain both milk and coffee
mc': # of transactions that contain milk but not coffee
m'c: # of transactions that contain coffee but not milk
m'c': # of transactions that contain neither milk nor coffee

Null-invariant

- A *null-transaction* is a transaction that does not contain any of the itemsets being examined
- A correlation measure is null-invariant if its value is not affected by the number of null-transactions.

Some Null-invariant Measures

Cosine Measure $$\frac{P(AB)}{\sqrt{P(A)\times P(B)}}$$

All_confidence $$\min\{P(A \mid B), P(B \mid A)\}$$

Max_confidence $$\max\{P(A \mid B), P(B \mid A)\}$$

Kulczynski Measure $$\frac{1}{2}(P(A \mid B)+P(B \mid A))$$

Cosine vs. Lift

$$lift(A,B)=\frac{P(AB)}{P(A)P(B)}=\frac{\frac{\sup(A\cup B)}{N}}{\frac{\sup(A)}{N}\frac{\sup(B)}{N}}=\frac{N\sup(A\cup B)}{\sup(A)\sup(B)}$$

$$\text{cosine}(A,B)=\frac{P(AB)}{\sqrt{P(A)\times P(B)}}=\frac{\frac{\sup(A\cup B)}{N}}{\sqrt{\frac{\sup(A)\sup(B)}{N^2}}}=\frac{\sup(A\cup B)}{\sqrt{\sup(A)\sup(B)}}$$

cosine({milk}, {coffee}) = ??

Ranges of Correlation Measures

	Perfectly positively correlated	Perfectly negatively correlated
Cosine		
All_conf		
Max_conf		
Kulc		

Choosing Correlation Measures …

datasets	mc	m′c	mc′	m′c′	cosine	all conf	max conf	Kulc
D_1	10,000	1000	1000	100,000	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100,000	0.09	0.09	0.09	0.91
D_4	1,000	1000	1000	100,000	0.5	0.5	0.5	0.5
D_5	1,000	100	10,000	100,000	0.29	0.09	0.91	0.5
D_6	1,000	10	100,000	100,000	0.10	0.01	0.99	0.5

… Choosing Correlation Measures

- Recommended:
  - Kulczynski + Imbalance Ratio (IR)

$$IR(A,B)=\frac{|\sup(A)-\sup(B)|}{\sup(A)+\sup(B)-\sup(A\cup B)}$$

Mining Sequential Patterns

- <{computer},{printer},{printer cartridge}>
- <{bread,milk},{bread,milk},{bread,milk }…>
- <{home.jsp},{search.jsp},{product.jsp} ,{product.jsp},{search.jsp}…>

Terminology and Notations

- Item, itemset
- Event = itemset
- A sequence is an ordered list of events
  - $<e_1e_2e_3\ldots e_l>$
  - E.g. <(a)(abc)(bc)(d)(ac)(f)>
- The length of a sequence is the number of items in the sequence, i.e. *not the number of events*

Sequences vs. Itemsets

- {a,b,c}
  - # of 3-itemset(s)??
  - # of 3-sequence(s)??

Subsequence

- $A=<a_1a_2a_3\ldots a_n>$
- $B=<b_1b_2b_3\ldots b_m>$
- A is a *subsequence* of B if there exists $1\leq j_1<j_2<\ldots<j_n \leq m$ such that $a_1\subseteq b_{j1},a_2 \subseteq b_{j2},\ldots,a_n \subseteq b_{jn}$

Subsequence Example

- `s=<(abc)(de)(f)>`
- Which of these are subsequences of `s`??
  - `s1=<(ab)(d)>`
  - `s2=<(ab)(f)>`
  - `s3=<(ac)(f)>`
  - `s4=<(abcde)>`
  - `s5=<(a)(de)>`
  - `s6=<(de)(a)(f)>`

Sequential Pattern

- If A is a subsequence of B, we say B *contains* A
- The support count of A is the number of sequences that contain A
- A is *frequent* if `support_count(A)`≥`min_sup`
- A frequent sequence is called a sequential pattern

Apriori Property Again

- Every nonempty subsequence of a frequent sequence is frequent

GSP Algorithm

- *Generalized Sequential Patterns*
- An extension of the Apriori algorithm for mining sequential patterns

GSP Example

SID	Sequence
1	<(a)(ab)(a)>
2	<(a)(c)(bc)>
3	<(ab)(c)(b)>
4	<(a)(c)(c)>

min_sup=2

L_1

C_1	support_count	L_1
<(a)>	4	<(a)>
<(b)>	3	<(b)>
<(c)>	3	<(c)>

L_2

C_2	support_count	L_2
<(a)(a)>	1	
<(a)(b)>	3	<(a)(b)>
<(a)(c)>	3	<(a)(c)>
<(b)(a)>	1	
<(b)(b)>	1	
<(b)(c)>	1	
<(c)(a)>	0	
<(c)(b)>	2	<(c)(b)>
<(c)(c)>	2	<(c)(c)>
<(ab)>	2	<(ab)>
<(ac)>	0	
<(bc)>	1	

From L_{k-1} to C_k

- Two sequences s_1 and s_2 are joinable if the subsequence obtained by dropping the first item in s_1 is the same as the subsequence obtained by dropping the last item in s_2
- The joined sequence is s_1 concatenated with the last item i of s_2
  - If the last two items in s_2 are in the same event, i is merged into the last event of s_1;
  - Otherwise i becomes a separate event

L3

C_3	support_count	L_3
<(a)(c)(b)>	2	<(a)(c)(b)>
<(a)(c)(c)>	2	<(a)(c)(c)>
<(c)(c)(b)>	0	

Candidate Pruning

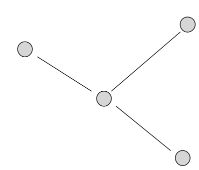
- A k-sequence can be pruned if one of its (k-1)-subsequence is not frequent

L_3	Candidate generation → C_4	Pruning → C_4
<(1)(2)(3)>	<(1)(2)(3)(4)>	<(1)(2 5)(3)>
<(1)(2 5)>	<(1)(2 5)(3)>	
<(1)(5)(3)>	<(1)(5)(3 4)>	
<(2)(3)(4)>	<(2)(3)(4)(5)>	
<(2 5)(3)>	<(2 5)(3 4)>	
<(3)(4)(5)>		
<(5)(3 4)>		

Subgraph Patterns

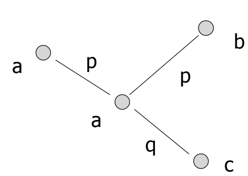
- Applications in web mining, computational chemistry, bioinformatics, network computing …

Vertices and Edges

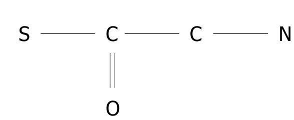


Labels

- Vertex labels: {a,b,c}
- Edge labels: {p,q}



Why Labels?

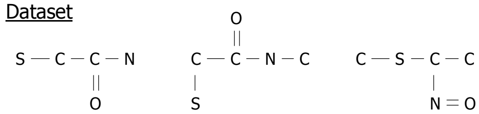


Subgraph

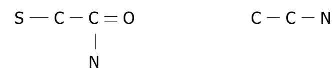
- A graph `G'=(V',E')` is a subgraph of another graph `G=(V,E)` if its vertex set `V'` is a subset of `V` and its edge set `E'` is a subset of `E`

Support

Dataset



Subgraph Patterns



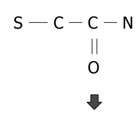
support count = ?? support count = ??

Frequent Subgraph Mining

- Given a set of graphs and `min_sup`, find all subgraphs `g` with `support(g)` ≥ `min_sup`
  - Typically we only consider graphs that *undirected* and *connected*

Transform Graph to Itemset

- Each combination of an edge label with its corresponding vertex labels is mapped to an "item"



Problem??

{ (1,S,C), (1,C,C), (1,C,N), (2,C,O)}

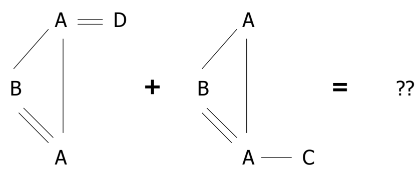
Apriori-based Approach

- Candidate generation
- Candidate pruning
- Support counting
- Candidate elimination

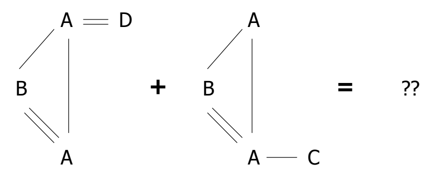
Candidate Generation

- Merge two frequent (k-1)-subgraphs to form a candidate `k`-subgraph
  - What is `k`??
- The two (k-1)-subgraphs must share a common (k-2)-subgraph, referred to as their *core*

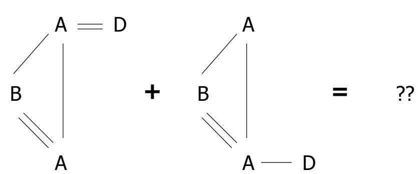
Vertex Growing



Edge Growing …



… Edge Growing

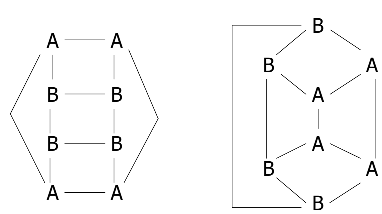


Candidate Pruning

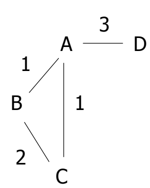
- Remove an edge from a candidate k-subgraph and check if the resulting (k-1)-subgraph is connected and frequent

Graph Isomorphism Problem

- Determine whether two graphs are topologically equivalent, i.e. *isomorphic*

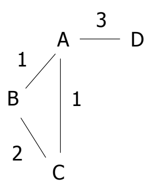


Adjacency Matrix …



$$\Rightarrow \begin{bmatrix} 0 & 1 & 1 & 3 \\ 1 & 0 & 2 & 0 \\ 1 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

… Adjacency Matrix



$$\begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 1 & 1 & 3 & 0 \end{bmatrix}$$

How many adjacency matrices can a graph with k vertices have??

String Representation of an Adjacency Matrix

$$\begin{bmatrix} 0 & 1 & 1 & 3 \\ 1 & 0 & 2 & 0 \\ 1 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 1 & 1 & 3 & 0 \end{bmatrix}$$

112300   200113

Graph Code

- A.K.A. Canonical label
- The string representation of the adjacency matrix that has the lowest (or highest) lexicographic value

Support Counting

- Isomorphism test a candidate k-subgraph against the k-subgraphs of each graph

Summary

- Frequent itemsets, association rules, sequential patterns, subgraph patterns
  - Measures: support, confidence, correlation
  - Algorithms: Apriori, FP-Growth, association rule generation, GPS
  - Optimizations: partitioning, vertical data format, various pruning techniques

Readings

- Textbook Chapter 6