大数据计算及应用(二)

Association Rules and Frequent Pattern Mining

Slides adapted from http://www.mmds.org

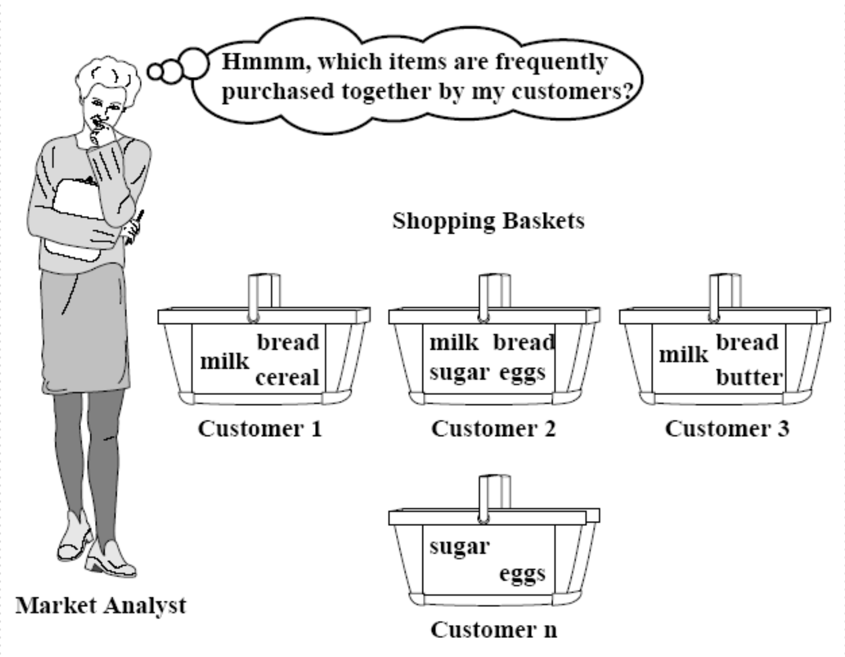
Agenda

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Association Rule

- Items frequently purchased together:

beer → diaper



- Uses:
  - Placement
  - Advertising
  - Sales
  - Coupons
- Objective: increase sales and reduce costs

The Market-Basket Model

- A large set of *items*, e.g., things sold in a supermarket
- A large set of *baskets*, each of which is a small set of the items, e.g., the things one customer buys on one day

TID	*Items*
1	**Beer, Diaper, Milk**
2	**Coke, Diaper, Eggs**
3	**Beer, Coke, Diaper, Eggs**
4	**Coke, Eggs**

The Market-Basket Model

- A general many-many mapping (association) between two kinds of things
  - But we ask about connections among “items” not “baskets”
- The technology focuses on common events, not rare events (“long tail”)

Applications – (1)

- Items = products; baskets = sets of products someone bought in one trip to the store
- Example application: given that many people buy beer and diapers together
  - Run a sale on diapers; raise price of beer
- Only useful if many buy diapers & beer

Applications – (2)

- Items = words; Baskets = Web pages;
- Unusual words appearing together in a large number of documents, e.g., "Brad" and "Angelina" may indicate an interesting relationship

Applications – (3)

- Items = sentences; baskets = documents containing those sentences
- Items that appear together too often could represent plagiarism

Association Rule Mining Applications

- Basket Data Analysis
- Genomic Data
- Telecommunication
- Credit Cards/ Banking Services
- Medical Treatments
- Web Personalization
- etc.

Scale of the Problem

- WalMart sells 100,000 items and can store billions of baskets
- The Web has billions of words and many billions of pages

Some Definition - Support

An itemset is supported by a basket (transaction) if it is included in the basket

Market-Basket transactions

TID	*Items*
1	**Beer, Diaper, Milk**
2	**Coke, Diaper, Eggs**
3	**Beer, Coke, Diaper, Eggs**
4	**Coke, Eggs**

<Beer, Diaper> is supported by basket 1, and 3, and its support is 2/4=50%.

Some Definition – Frequent Itemset

If the support of an itemset exceeds user specified *min_support* (threshold), this itemset is called a frequent itemset (pattern).

Market-Basket transactions

TID	*Items*
1	Beer, Diaper, Milk
2	Coke, Diaper, Eggs
3	Beer, Coke, Diaper, Eggs
4	Coke, Eggs

min_support=50%
<Beer, Diaper> is a frequent itemset
<Beer, Milk> is not a frequent itemset

Association Rules

- **Association Rules:**
  If-then rules about the contents of baskets
- $\{i_1, i_2,\ldots,i_k\} \rightarrow j$ means: "if a basket contains all of $i_1,\ldots,i_k$ then it is ***likely*** to contain j"
- **In practice there are many rules, want to find significant/interesting ones!**
- ***Confidence*** of this association rule is the probability of j given $I = \{i_1,\ldots,i_k\}$

$$\mathrm{conf}(I \rightarrow j) = \frac{\mathrm{support}(I \cup j)}{\mathrm{support}(I)}$$

Example: Confidence

$T_1 = \{m, c, b\}$ $T_2 = \{m, p, j\}$

$T_3 = \{m, b\}$ $T_4 = \{c, j\}$

$T_5 = \{m, p, b\}$ $T_6 = \{m, c, b, j\}$

$T_7 = \{c, b, j\}$ $T_8 = \{b, c\}$

- Association rule: $\{m, b\} \rightarrow c$
  - Support(m,b)=4/8, Support(m,b,c)=2/8
  - Confidence = 2/4 = 0.5

Association Rules Mining

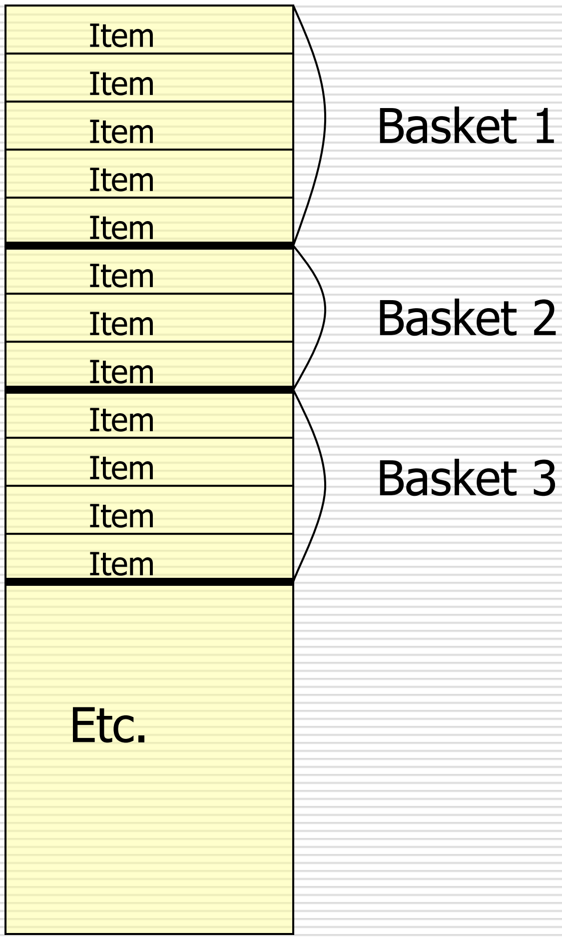
- Question: "find all association rules with support $\geq s$ and confidence $\geq c$"

- Hard part: finding the frequent itemsets

Computation Model

- Typically, data is kept in flat files rather than in a database system
  - Stored on disk
  - Stored basket-by-basket
  - Expand baskets into pairs, triples, etc. as you read baskets

File Organization



Example: items are positive integers, and boundaries between baskets are –1

Computation Model – (2)

- The true cost of mining disk-resident data is usually the number of disk I/O's
- In practice, association-rule algorithms read the data in *passes* – all baskets read in turn
- Thus, we measure the cost by the number of passes an algorithm takes

Main-Memory Bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource
  - As we read baskets, we need to count something, e.g., occurrences of pairs
  - The number of different things we can count is limited by main memory
  - Swapping counts in/out is a disaster

Finding Frequent Pairs

- The hardest problem often turns out to be finding the frequent pairs
  - Why? Often frequent pairs are common, frequent triples are rare
    - Why? Probability of being frequent drops exponentially with size; number of sets grows more slowly with size
- We'll concentrate on pairs, then extend to larger sets

Naïve Algorithm

- Read file once, counting in main memory the occurrences of each pair
  - From each basket of n items, generate its $n(n-1)/2$ pairs by two nested loops
- Fails if $(\#\text{items})^2$ exceeds main memory
  - Remember: #items can be 100K (Wal-Mart) or 10B (Web pages)

Example: Counting Pairs

- Suppose 10^5 items
- Suppose counts are 4-byte integers
- Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$ (approximately)
- Therefore, $2*10^{10}$ (20 gigabytes) of main memory needed

Details of Main-Memory Counting

- Two approaches:

  (1) Count all pairs, using a triangular matrix

  - requires only 4 bytes/pair

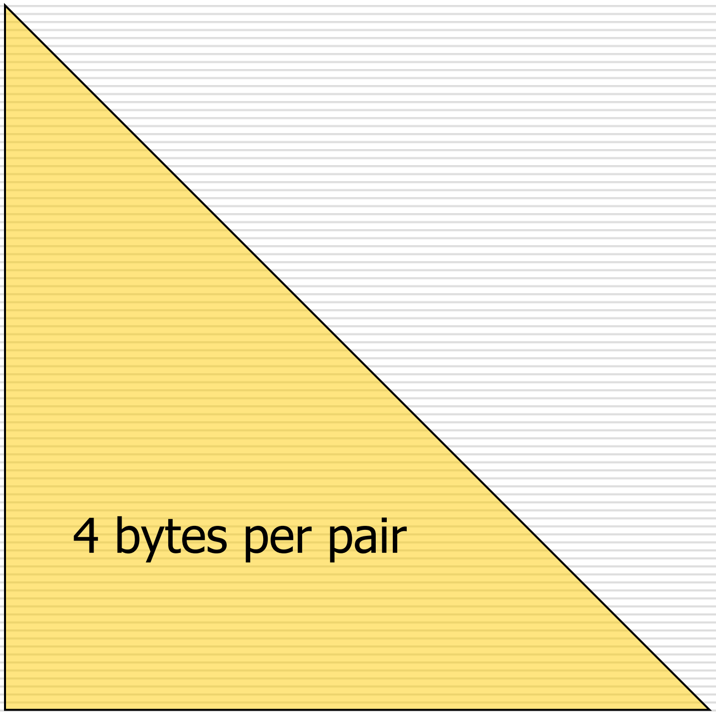
    always assume integers are 4 bytes

  (2) Keep a table of triples [*i*, *j*, *c*] = "the count of the pair of items {*i*, *j* } is *c*"

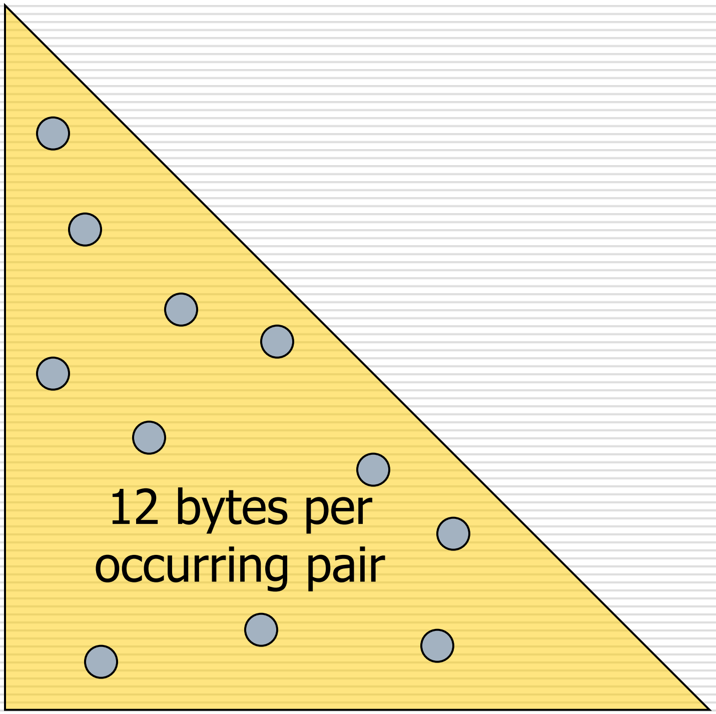
  - requires 12 bytes

    but only for those pairs with count > 0

Details of Main-Memory Counting



Method (1)



Method (2)

Comparing the Two Approaches

- **Approach 1: Triangular Matrix**
  - **n** = total number of items
  - Count pair of items $\{i, j\}$ only if $i<j$
  - Keep pair counts in lexicographic order:
    - {1,2}, {1,3},…, {1,n}, {2,3}, {2,4},…,{2,n}, {3,4},…{n-1,n}
  - Pair $\{i, j\}$ is at position $(i-1)(n-i/2)+j-i$
  - Total number of pairs $\boldsymbol{n(n-1)/2}$; total bytes= $\boldsymbol{2n^2}$
  - **Triangular Matrix** requires 4 bytes per pair
- **Approach 2** uses **12 *bytes*** per occurring pair *(but only for pairs with count > 0)*
  - Beats Approach 1 if less than **1/3** of possible pairs actually occur

Comparing the Two Approaches

- **Approach 1: Triangular Matrix**
  - **n** = total number items
  - Count pair of items {*i*, *j*} only if *i*<*j*
  - …
  - …s= $\boldsymbol{2n^2}$
  - …
- **A**… pair
  (b…
  - … possible pairs actually occur

Problem is if we have too many items so the pairs do not fit into memory.

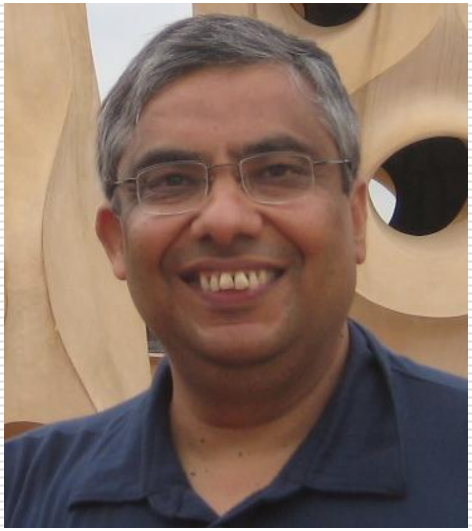
Can we do better?

Outline

- Association Rules
- Frequent Itemset Mining Algorithms
  - Apriori
  - FP-growth
- Sequential Pattern Mining Algorithms

Apriori Algorithm

- Proposed by Rakesh Agrawal [*VLDB'94*]
- Key idea:
  - Candidate generation-and-test
  - Anti-monotone property



http://www.vldb.org › conf PDF

Fast Algorithms for Mining Association Rules - VLDB ...

by R Agrawal · Cited by 28692 — We consider the problem of discovering **association rules** between items in a large database of sales transactions. We present two new **algorithms** for…

13 pages

Apriori Algorithm – (1)

- A two-pass approach called *Apriori* limits the need for main memory
- *Monotonicity* : if a set of items appears at least *s* times, so does every subset
  - Contrapositive for pairs: if item *i* does not appear in *s* baskets, then no pair including *i* can appear in *s* baskets

Apriori Algorithm – (2)

- Pass 1: Read baskets and count in main memory the occurrences of each item
  - Requires only memory proportional to #items
- Items that appear at least *s* times are the *frequent items*

Apriori Algorithm – (3)

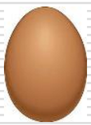
- Pass 2: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent
  - Requires memory proportional to square of *frequent* items only (for counts), plus a list of the frequent items (so you know what must be counted)

Apriori Algorithm

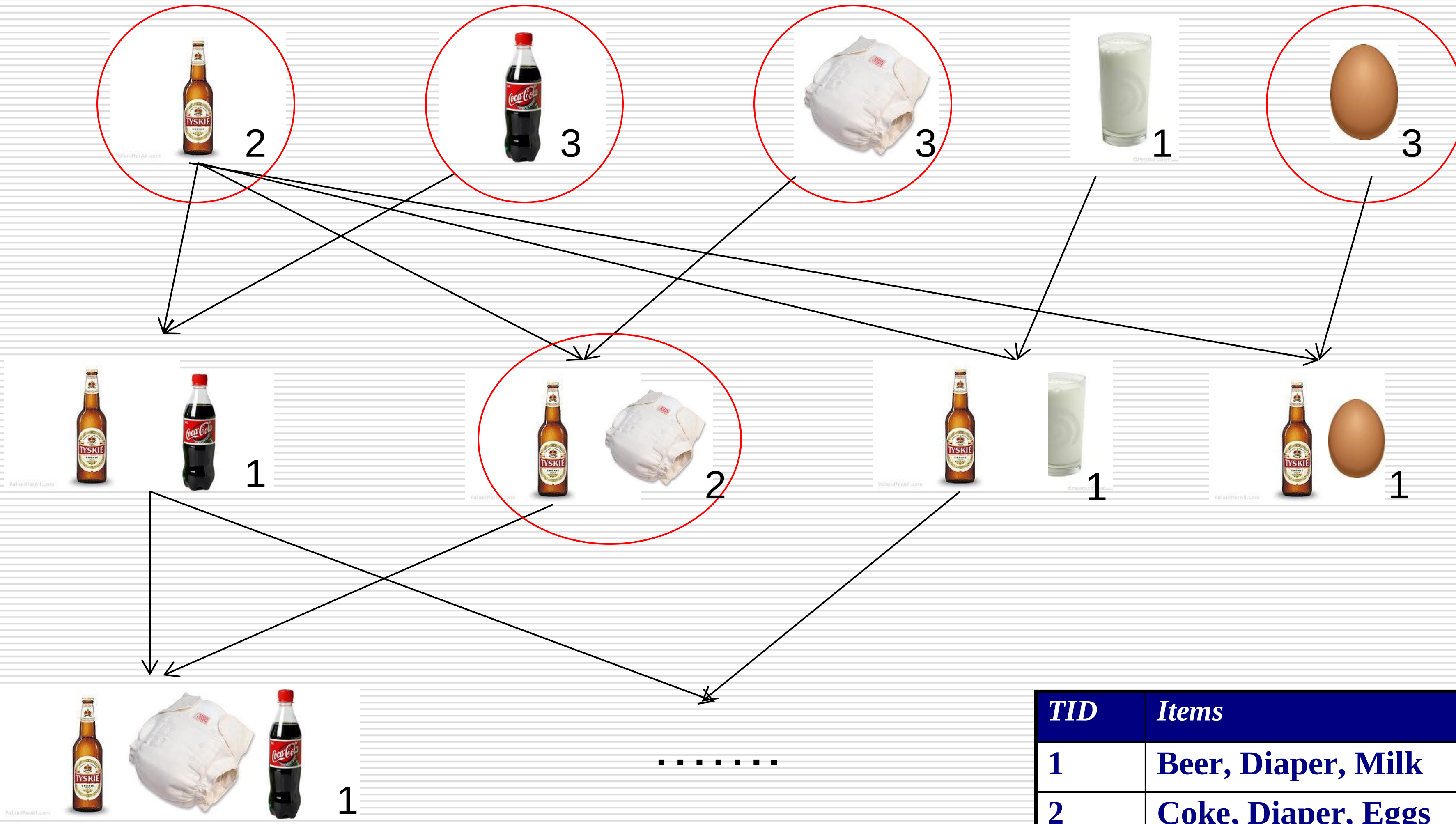
Market-Basket transactions

TID	*Items*
1	**Beer, Diaper, Milk**
2	**Coke, Diaper, Eggs**
3	**Beer, Coke, Diaper, Eggs**
4	**Coke, Eggs**





Naive Algorithm

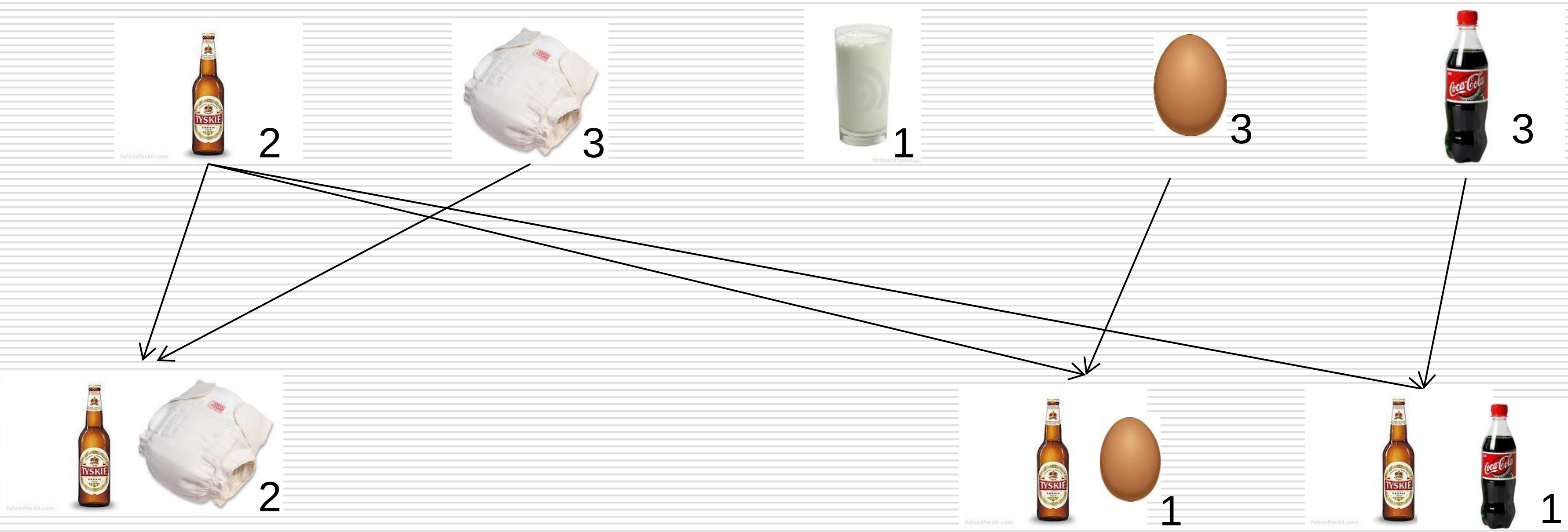


TID	*Items*
1	**Beer, Diaper, Milk**
2	**Coke, Diaper, Eggs**
3	**Beer, Coke, Diaper, Eggs**
4	**Coke, Eggs**

$Sup_{min}=2$

Apriori Algorithm

- Anti-monotone property: If an itemset is not frequent, then any of its superset is not frequent



$Sup_{min}=2$

TID	*Items*
1	**Beer, Diaper, Milk**
2	**Coke, Diaper, Eggs**
3	**Beer, Coke, Diaper, Eggs**
4	**Coke, Eggs**

Apriori Algorithm

$Sup_{min} = 2$

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan →

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

→ L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

← 2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

← L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

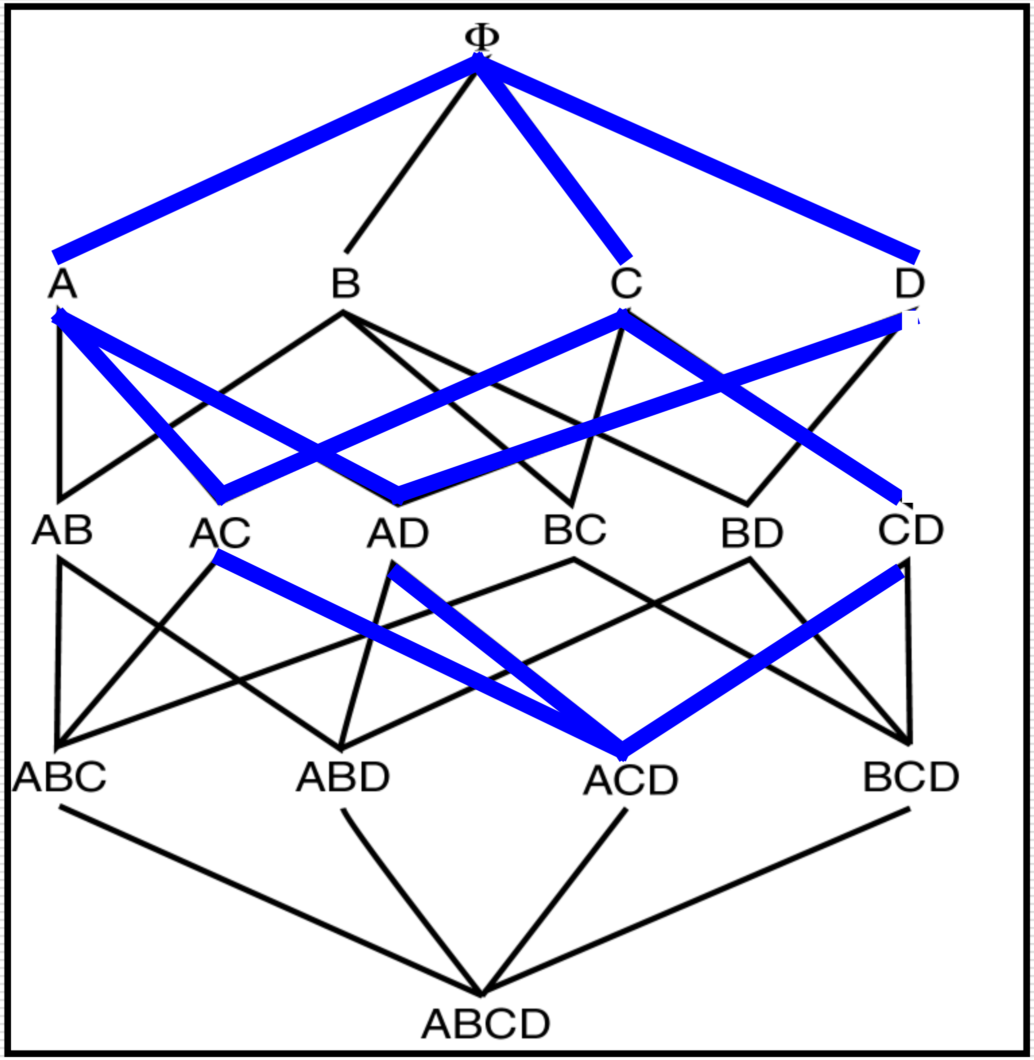
3rd scan → L_3

Itemset	sup
{B, C, E}	2

Apriori Algorithm

1. C_1 = Itemsets of size one in I;
2. Determine all large itemsets of size 1, $L_{1;}$
3. i = 1;
4. Repeat
5. i = i + 1;
6. C_i = Apriori-Gen(L_{i-1});
7. Count C_i to determine $L_{i;}$
8. until no more large itemsets found;

Frequent Itemset Property



Drawbacks of Apriori

- Multiple scans of transaction database
  - Multiple database scans are costly
- Huge number of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $2^{100}-1 = 1.27*10^{30}$

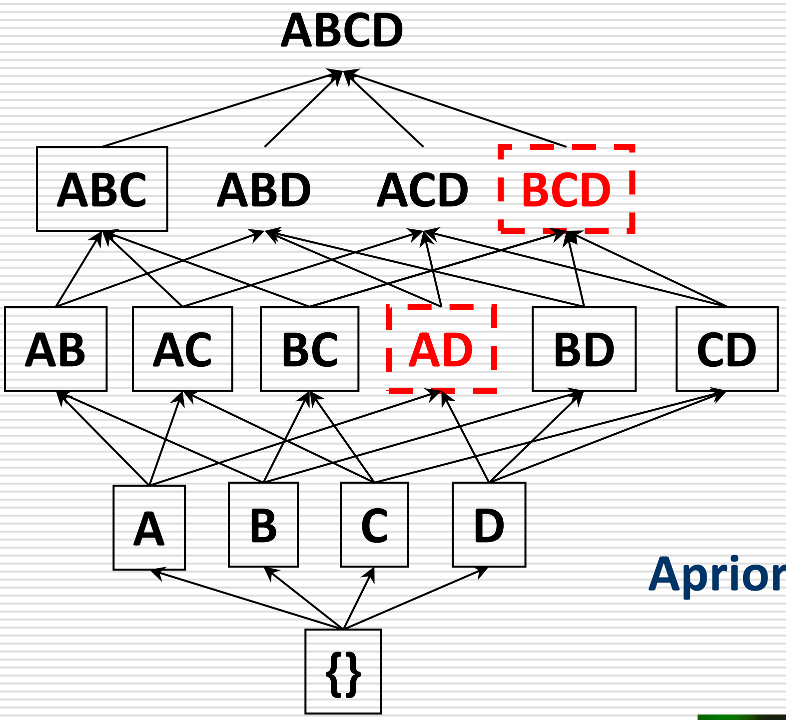
Improving Apriori: General Ideas

- Reduce passes of transaction database scans
- Shrink number of candidates
- Facilitate support counting of candidates

Improving Apriori's Efficiency

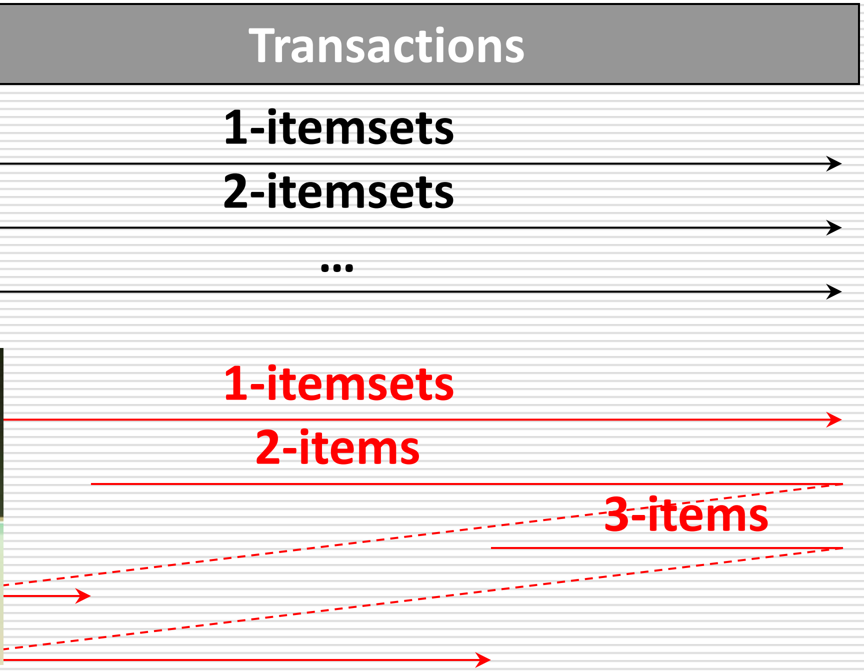
- **Hash-based itemset counting**: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
- **Transaction reduction**: A transaction that does not contain any frequent k-itemset is useless in subsequent scans
- **Partitioning:** Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
- **Sampling**: mining on a subset of given data, need a lower support threshold + a method to determine the completeness
- **Dynamic itemset counting**: add new candidate itemsets immediately (unlike Apriori) when all of their subsets are estimated to be frequent

DIC: Reduce Number of Scans



Itemset lattice

- Once both A and D are determined frequent, the counting of AD begins
- Once all length-2 subsets of BCD are determined frequent, the counting of BCD begins



S. Brin , R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*

FP-growth Algorithm

FP-Growth

- Proposed by Jiawei Han [*SIGMOD'00*]
- Uses the Apriori pruning principle
- Scan DB only twice
  - Once to find frequent 1-itemset (single item pattern)
  - Once to construct FP-tree (prefix tree, Trie), the data structure of FP-growth

Mining Frequent Patterns without Candidate Generation

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - highly condensed, but complete for frequent pattern mining
  - avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: sub-database test only!

FP-Growth

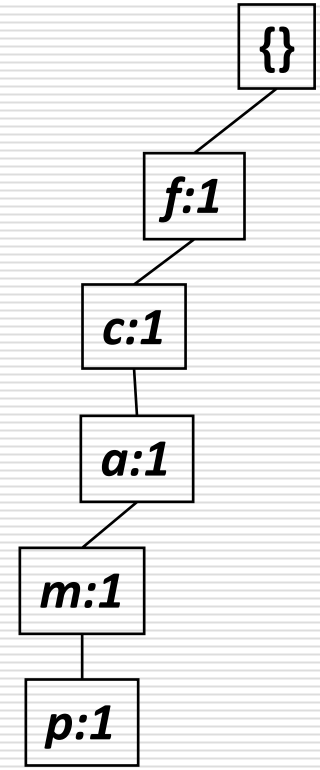
$Sup_{min} = 2$

TID	*Items bought*
10	*{f, a, c, d, g, i, m, p}*
20	*{a, b, c, f, l, m, o}*
30	*{b, f, h, j, o, w}*
40	*{b, c, k, s, p}*
50	*{a, f, c, e, l, p, m, n}*

Header Table

Item	*frequency*
f	*4*
c	*4*
a	*3*
b	*3*
m	*3*
p	*3*

TID	*(ordered) frequent items*
10	*{f, c, a, m, p}*
20	*{f, c, a, b, m}*
30	*{f, b}*
40	*{c, b, p}*
50	*{f, c, a, m, p}*



FP-Growth

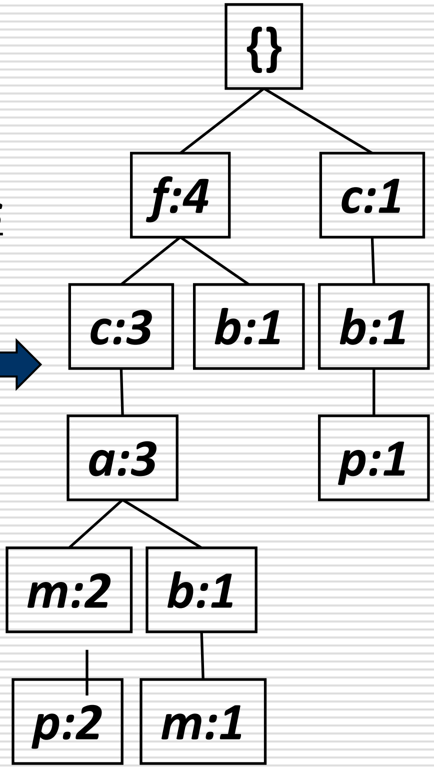
$Sup_{min} = 2$

TID	*Items bought*
10	*{f, a, c, d, g, i, m, p}*
20	*{a, b, c, f, l, m, o}*
30	*{b, f, h, j, o, w}*
40	*{b, c, k, s, p}*
50	*{a, f, c, e, l, p, m, n}*

Header Table

Item	*frequency*
f	*4*
c	*4*
a	*3*
b	*3*
m	*3*
p	*3*

TID	*(ordered) frequent items*
10	*{f, c, a, m, p}*
20	*{f, c, a, b, m}*
30	*{f, b}*
40	*{c, b, p}*
50	*{f, c, a, m, p}*



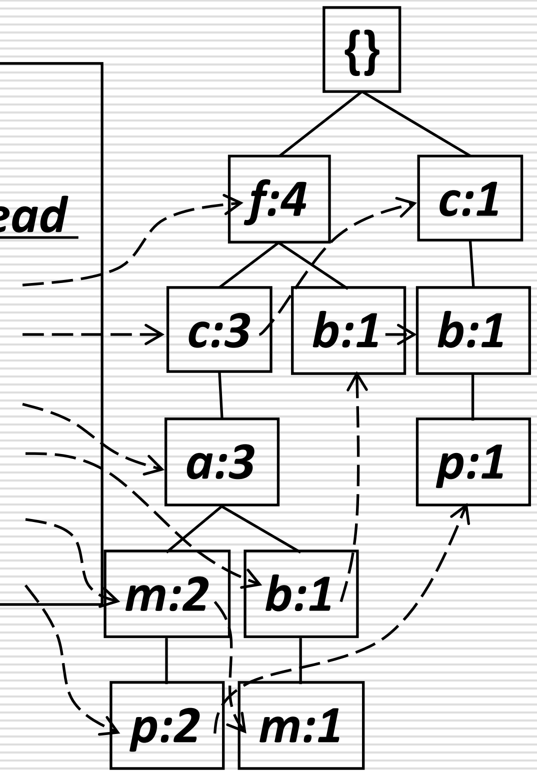
FP-Growth

$Sup_{min} = 2$

TID	(ordered) frequent items
10	{f, c, a, m, p}
20	{f, c, a, b, m}
30	{f, b}
40	{c, b, p}
50	{f, c, a, m, p}

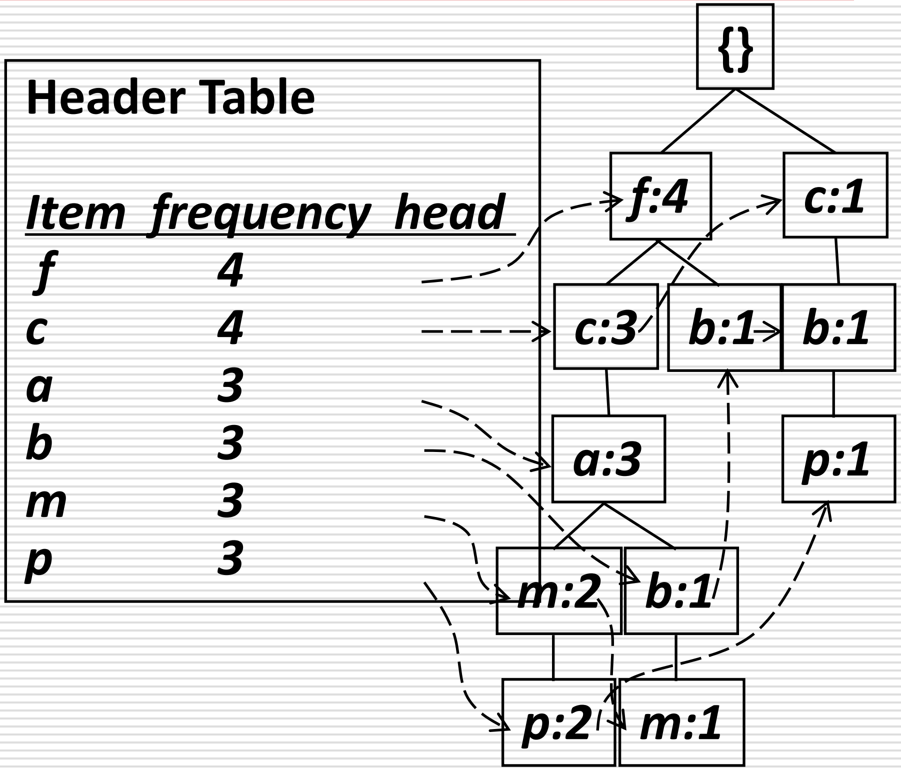
Header Table

Item	frequency	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



FP-Growth

$Sup_{min} = 2$



Conditional **pattern bases**

Item	***cond. pattern base***	***freq. itemset***
p	***fcam:2, cb:1***	***fp, cp, ap, mp, fcp, fap, fmp, cap, cmp, amp, camp, facp, fcmp, famp, fcamp***
m	***fca:2, fcab:1***	***fm, cm, am, fcm, fam, cam, fcam***
b	***fca:1, f:1, c:1***	***…***
a	***fc:3***	***…***
c	***f:3***	***…***

Why Is Frequent Pattern Growth Fast?

- The performance study shows
  - FP-growth is faster than Apriori (in most cases), and is also faster than tree-projection (an order of magnitude on some datasets)
- Reasoning
  - No candidate generation (claimed by the authors)
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

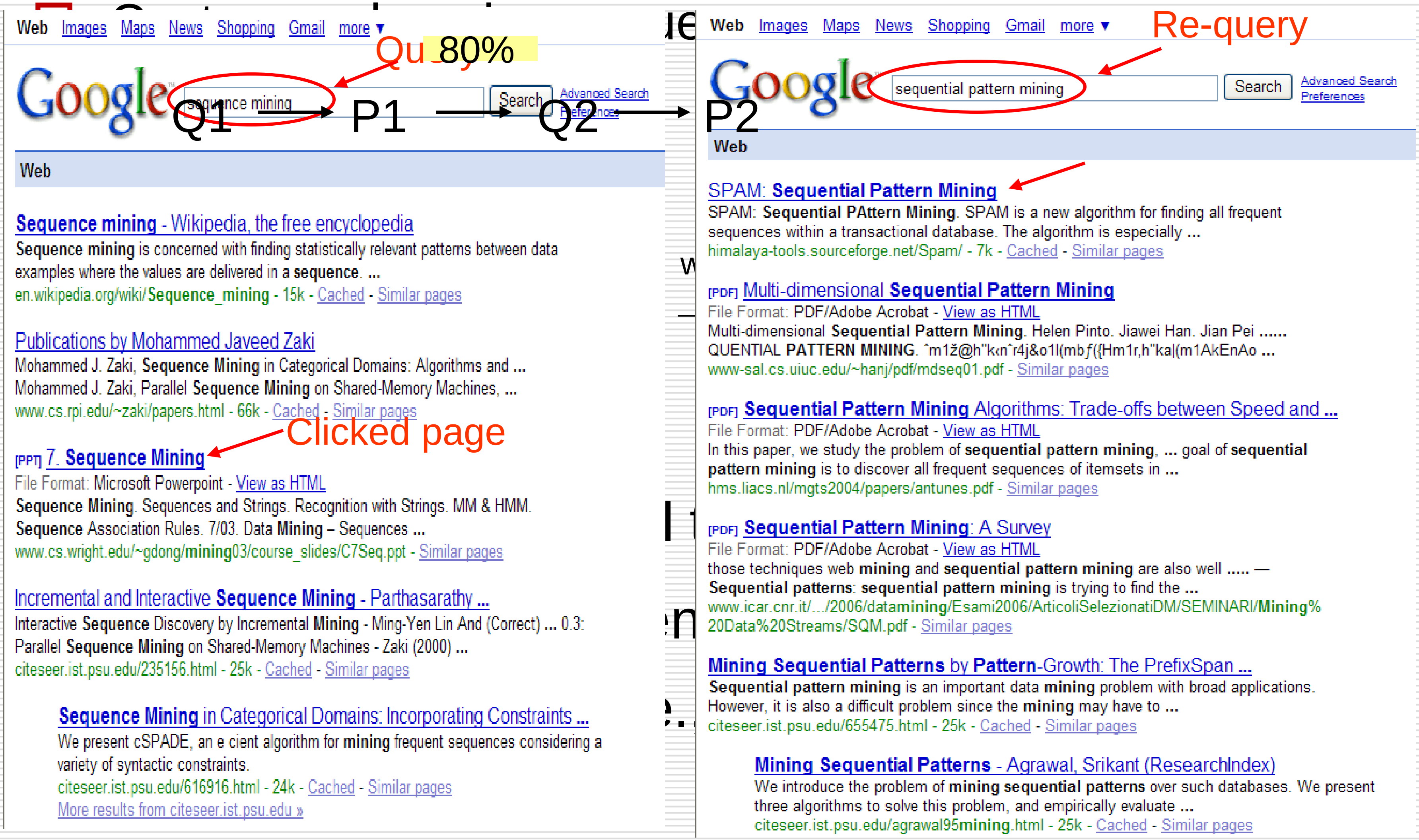
Extension of Association Rule Mining

- Association rule mining has been extensively studied in the data mining community.
- There are many efficient algorithms and model variations.
- Other related work includes
  - Multi-level or generalized rule mining
  - Sequential pattern mining
  - Constrained rule mining
  - Incremental rule mining
  - Maximal and closed frequent itemset mining
  - Numeric association rule mining
  - Rule interestingness and visualization
  - Parallel algorithms
  - …

Extension of Association Rule Mining

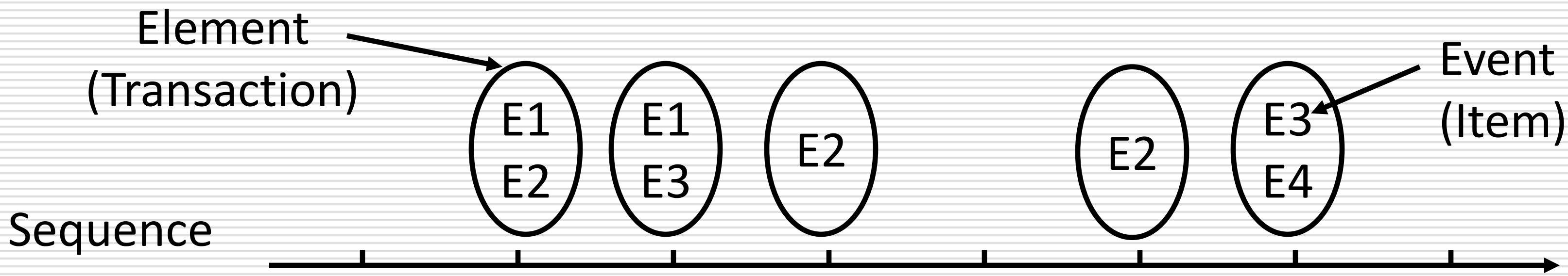
- Association rule mining has been extensively studied in the data mining community.
- There are many efficient algorithms and model variations.
- Other related work includes
  - Multi-level or generalized rule mining
  - Sequential pattern mining
  - Constrained rule mining
  - Incremental rule mining
  - Maximal and closed frequent itemset mining
  - Numeric association rule mining
  - Rule interestingness and visualization
  - Parallel algorithms
  - …

Applications



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 items

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

  - Each element is attributed to a specific time

- A k-sequence is a sequence that contains k items

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

- 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:

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

Outline

- Association Rules
- Frequent Itemset Mining Algorithms
- Sequential Pattern Mining Algorithms
  - GSP
  - SPADE
  - SPAM

GSP (Generalized Sequential Pattern Mining)

- Proposed by Srikant and Agrawal [*EDBT'96*]
- Uses the Apriori pruning principle

Finding Length-1 Sequential Patterns

- Initial candidates:
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once, count support for candidates

min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Cand	Sup
<a>	**3**
<b>	**5**
<c>	**4**
<d>	**3**
<e>	**3**
<f>	**2**
~~<g>~~	1
~~<h>~~	1

Generating Length-2 Candidates

51 length-2 Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Without Apriori property, 8*8+8*7/2=92 candidates

Apriori prunes 44.57% candidates

GSP Mining Process

5th scan: 1 cand. 1 length-5 seq. pat. **<(bd)cba>**

Cand. cannot pass sup. threshold

4th scan: 8 cand. 6 length-4 seq. pat. **<abba> <(bd)bc> …**

Cand. not in DB at all

3rd scan: 46 cand. 19 length-3 seq. pat. 20 cand. not in DB at all **<abb> <aab> <aba> <baa> <bab> …**

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all **<aa> <ab> … <af> <ba> <bb> … <ff> <(ab)> … <(ef)>**

1st scan: 8 cand. 6 length-1 seq. pat. **<a> <b> <c> <d> <e> <f> <g> <h>**

***min_sup* =2**

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

GSP Algorithm

- Take sequences in form of <x> as length-1 candidates
- Scan database once, find F_1, the set of length-1 sequential patterns
- Let k=1; while F_k is not empty do
  - Form C_{k+1}, the set of length-(k+1) candidates from F_k;
  - If C_{k+1} is not empty, scan database once, find F_{k+1}, the set of length-(k+1) sequential patterns
  - Let k=k+1;

GSP Algorithm

- Benefits from the Apriori pruning
  - Reduces search space
- Bottlenecks
  - Scans the database multiple times
  - Generates a huge set of candidate sequences

SPADE Algorithm

SPADE Algorithm

- Proposed by Zaki *et al.* [*MLJ'01*]
- Candidate generation-and-test
- Vertical ID-list data representation based on Lattice-theory
- Counting support through temporal joins
- Reduced I/O costs (three DB scans)

SPADE Algorithm

ID	Data Sequence
1	< a c (b c) d (a b c) a d >
2	< b (c d) a c (b d) >
3	< d (b c) (a c) (c d) >

ID-List DB

a		b		c		d	
SID	Pos	SID	Pos	SID	Pos	SID	Pos
1	1	1	3	1	2	1	4
1	5	1	5	1	3	1	7
1	6	2	1	1	5	2	2
2	3	2	5	2	2	2	5
3	3	3	2	2	4	3	1
				3	2	3	4
				3	3		
				3	4		

Temporal Joins

{a}	
SID	Pos
1	1
1	5
1	6
2	3
3	3

\+

{b}	
SID	Pos
1	3
1	5
2	1
2	5
3	2

{a, b}	
SID	Pos
1	3
1	5
2	5

Supp{ab}=2

{b, a}	
SID	Pos
1	5
1	6
3	3

Supp{ba}=2

{(ab)}	
SID	Pos
1	5

Supp{(ab)}=1

min_support=2

SPAM Algorithm

SPAM Algorithm

- Proposed by Ayres *et al.* [*KDD'02*]
- Key idea based on SPADE
- Using bitmap data representation
- Faster than SPADE yet space consuming

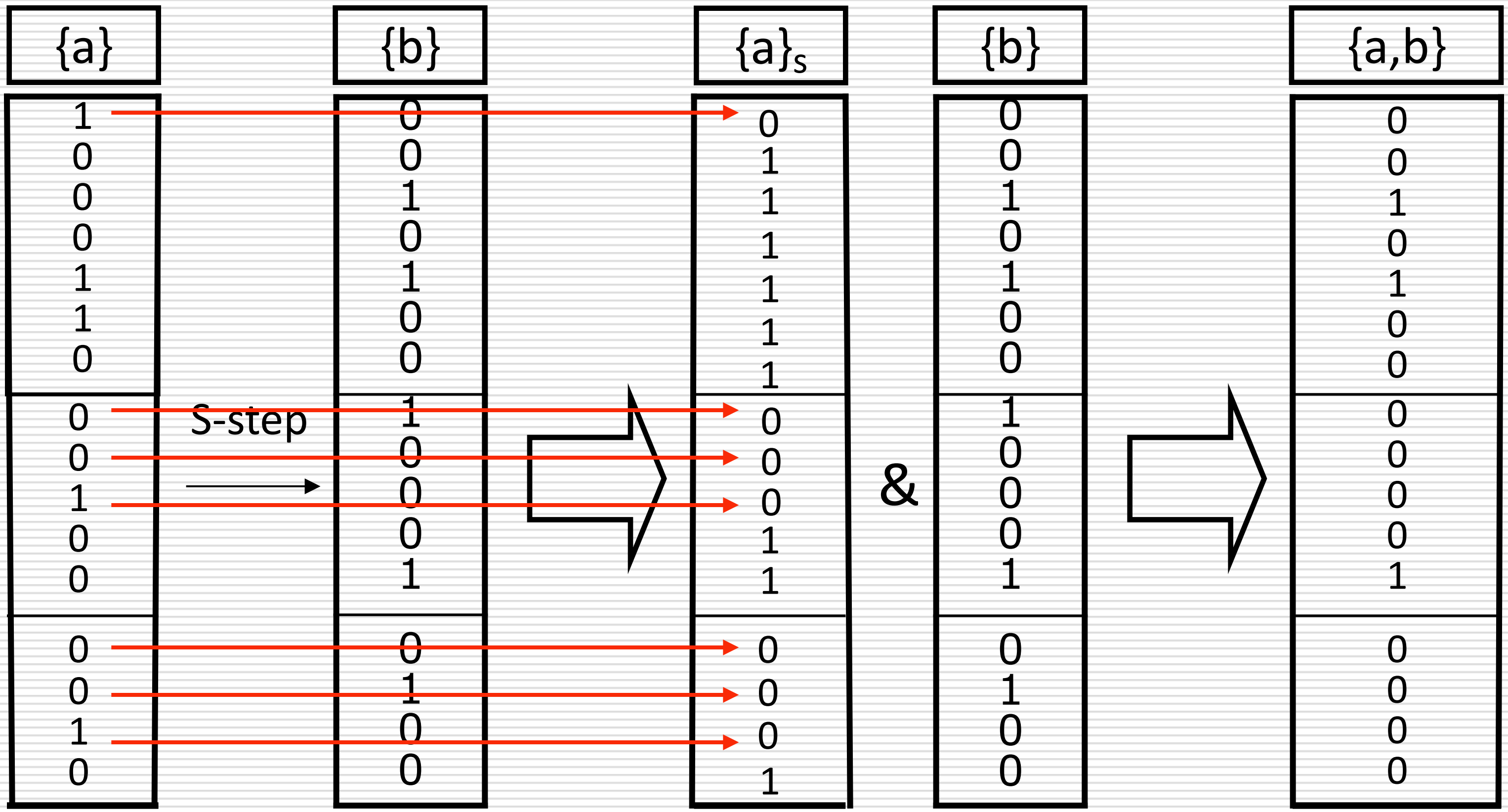
SPAM Algorithm

ID	Data Sequence
10	< a c (b c) d (a b c) a d >
20	< b (c d) a c (b d) >
30	< d (b c) (a c) (c d) >

SID	EID	{a}	{b}	{c}	{d}
10	1	1	0	0	0
10	2	0	0	1	0
10	3	0	1	1	0
10	4	0	0	0	1
10	5	1	1	1	0
10	6	1	0	0	0
10	7	0	0	0	1
20	1	0	1	0	0
20	2	0	0	1	1
20	3	1	0	0	0
20	4	0	0	1	0
20	5	0	1	0	1
30	1	0	0	0	1
30	2	0	1	1	0
30	3	1	0	1	0
30	4	0	0	1	1

SPAM Temporal Joins

Sequence extended step:



Sup(ab)=2

min_support=2

Problem of SPAM

- Bitmap representation is space consuming i.e., data is commonly very sparse

Acknowledgement

- Slides are adapted from:
  - Prof. Jeffrey D. Ullman
  - Dr. Jure Leskovec
  - Dr. Wujun Li