Data mining - Association rules

(Ref:Data Mining Concepts by Arun K Pujari)

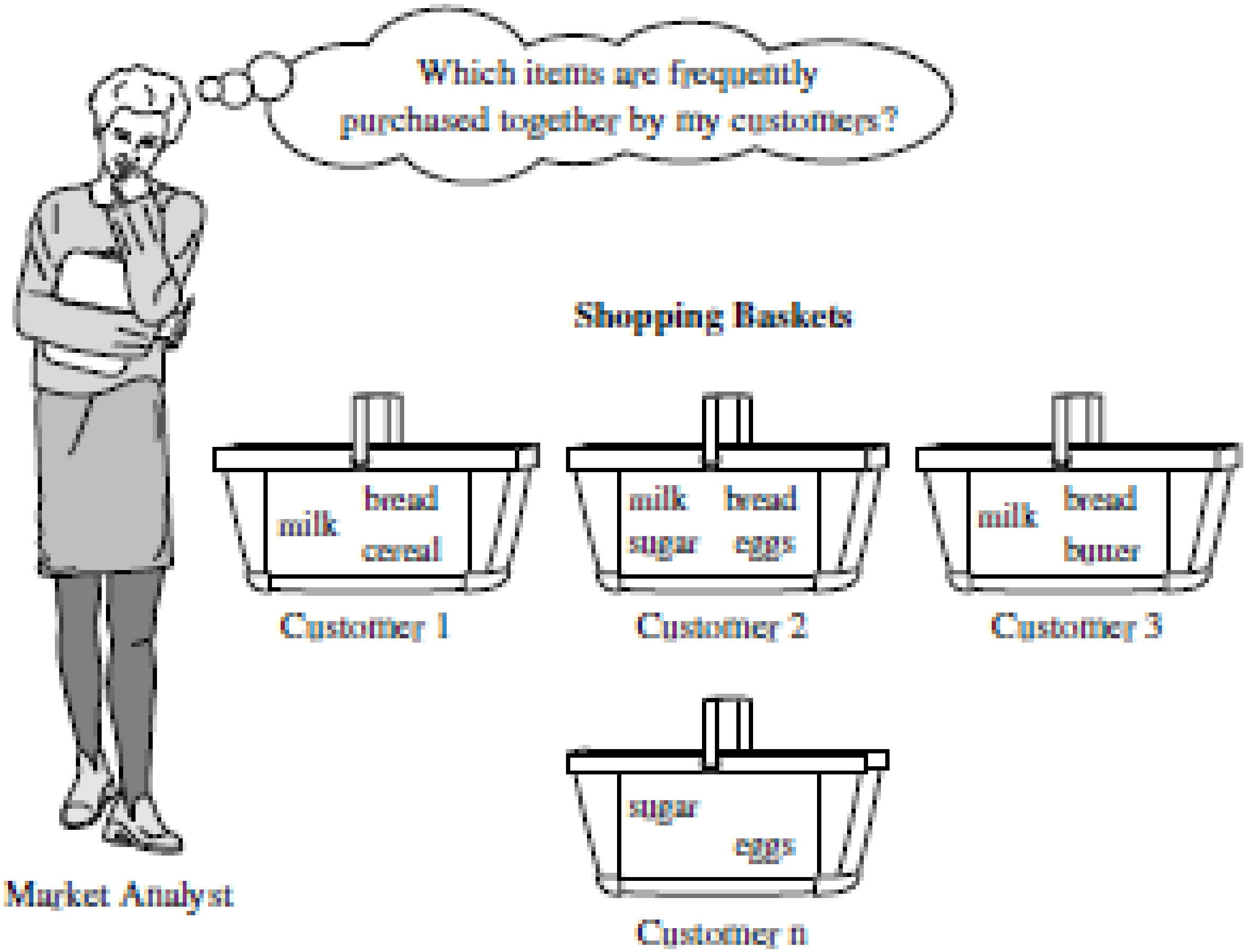
DATA MINING

- Data mining is the non trivial extraction of implicit, previously unknown and potentially useful information from the data

What Is Frequent Pattern Analysis?

- **Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set**
- **First proposed by Agrawal, Imielinski, and Swami in the context of frequent itemsets and association rule mining**
- **Motivation: Finding inherent regularities in data**
  - **What products were often purchased together?**
  - **What are the subsequent purchases after buying a PC?**
  - **Can we automatically classify web documents?**
- **Applications**
  - **Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.**


Market basket



Develop marketing strategies, inventory management, sale promotion strategies etc.

Association Rules: Basic Concepts

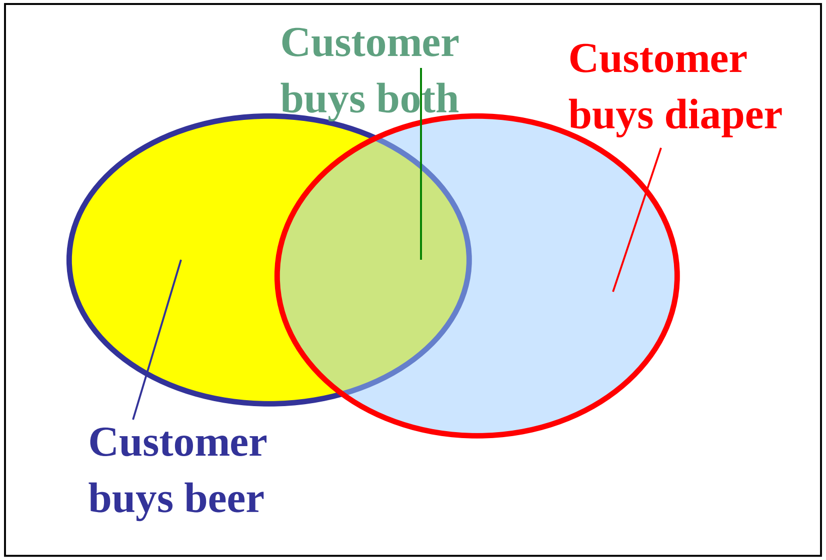
- Given: (1) database of transactions, (2) each transaction is a list of items (purchased by a customer in a visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items

Problem Statement

- $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$: a set of literals, called items
- Transaction $\mathcal{T}$: a set of items s.t. $\mathcal{T} \subseteq \mathcal{I}$
- Database $\mathcal{D}$: a set of transactions
- A transaction contains X, a set of items in $\mathcal{I}$, if $X \subseteq \mathcal{T}$
- An association rule is an implication of the form $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$
- The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with confidence c if c% of transactions in $\mathcal{D}$ that contain X also contain Y
- The rule $X \Rightarrow Y$ has support s in the transaction set $\mathcal{D}$ if s% of transactions in $\mathcal{D}$ contain $X \cup Y$
- Find all rules that have support and confidence greater than user-specified min support and min confidence

Example : Frequent Patterns and Association Rules

Transaction-id	Items bought
10	A, B, D
20	A, C, D
30	A, D, E
40	B, E, F
50	B, C, D, E, F



- Itemset X = $\{x_1, \ldots, x_k\}$
- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - support, s, probability that a transaction contains $X \cup Y$
  - confidence, c, conditional probability that a transaction having X also contains Y

Let $sup_{min} = 50\%$, $conf_{min} = 50\%$

Freq. Pat.: {A:3, B:3, D:4, E:3, AD:3}

Association rules:

$A \rightarrow D$ (60%, 100%)

$D \rightarrow A$ (60%, 75%)

Example: Database with transactions (customer_# : item_a1, item_a2, …)

1:   1, 3, 5.

2:   1, 8, 14, 17, 12.

3:   4, 6, 8, 12, 9, 104.

4:   2, 1, 8.

support {8,12} = 2 (,or 50% ~ 2 of 4 customers)

support {1, 5} = 1 (,or 25% ~ 1 of 4 customers)

support {1} = 3 (,or 75% ~ 3 of 4 customers)

Example: Database with transactions (customer_# : item_a1, item_a2, …)

1: 3, 5, 8.
2: 2, 6, 8.
3: 1, 4, 7, 10.
4: 3, 8, 10.
5: 2, 5, 8.
6: 1, 5, 6.
7: 4, 5, 6, 8.
8: 2, 3, 4.
9: 1, 5, 7, 8.
10: 3, 8, 9, 10.

Conf ({5} => {8}) ?

supp({5}) = 5 , supp({8}) = 7 , supp({5,8}) = 4,

then **conf({5} => {8}) = 4/5 = 0.8 or 80%**

Example: Database with transactions (customer_# : item_a1, item_a2, …)

1: 3, 5, 8.
2: 2, 6, 8.
3: 1, 4, 7, 10.
4: 3, 8, 10.
5: 2, 5, 8.
6: 1, 5, 6.
7: 4, 5, 6, 8.
8: 2, 3, 4.
9: 1, 5, 7, 8.
10: 3, 8, 9, 10.

Conf ({5} => {8}) ? 80% Done. Conf ({8} => {5}) ?

supp({5}) = 5 , supp({8}) = 7 , supp({5,8}) = 4,

then **conf({8} => {5}) = 4/7 = 0.57 or 57%**

Conf ({5} => {8}) ? 80% Done.

Conf ({8} => {5}) ? 57% Done.

Rule ({5} => {8}) more meaningful than Rule ({8} => {5})

Example: Database with transactions (customer_# : item_a1, item_a2, …)

1: 3, 5, 8.
2: 2, 6, 8.
3: 1, 4, 7, 10.
4: 3, 8, 10.
5: 2, 5, 8.
6: 1, 5, 6.
7: 4, 5, 6, 8.
8: 2, 3, 4.
9: 1, 5, 7, 8.
10: 3, 8, 9, 10.

Conf ({9} => {3}) ?

supp({9}) = 1 , supp({3}) = 4 , supp({3,9}) = 1,

then **conf({9} => {3}) = 1/1 = 1.0 or 100%. OK?**

Conf({9} => {3}) = 100%. Done.

Notice: High Confidence, Low Support.

-> Rule ({9} => {3}) not meaningful

Frequent set

- An item set $X \subseteq I$ is said to be a frequent item set in T, if
- $S(X)_T \geq$ user specified minimum support

The downward closure property of frequent sets

Any subset of a frequent itemset must be frequent

If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**

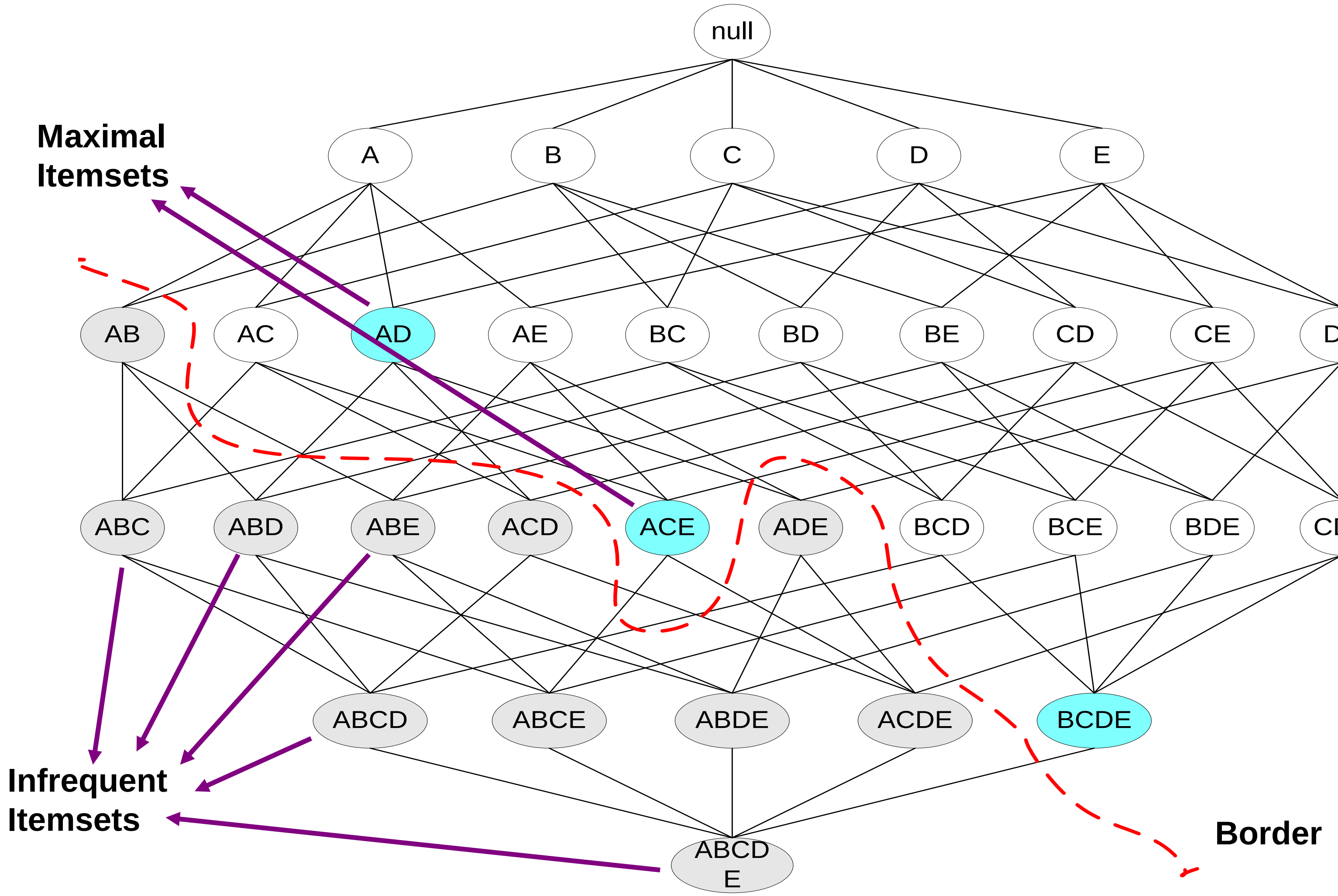
i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}

The upward closure property of frequent sets

Any superset of an infrequent set is an infrequent set

- Maximal frequent set: a frequent set is a maximal frequent set if it is a frequent set and no superset of this is a frequent set
- Border set: an itemset is a border set if it is not a frequent set, but all its proper subsets are frequent sets
- Note: set of all maximal frequent sets can act as a compact representation of the set of all frequent sets

Lattice of subsets



ID	apples,	beer,	cheese,	dates,	eggs,	fish,	glue,	honey,	ice-cream
1	1	1	0	1	0	0	1	1	0
2	0	0	1	1	1	0	0	0	0
3	0	1	1	0	0	1	0	0	0
4	0	1	0	0	0	1	0	0	1
5	0	0	0	0	1	0	1	0	0
6	0	0	0	0	0	1	0	0	1
7	1	0	0	1	0	0	0	1	0
8	0	0	0	0	0	1	0	0	1
9	0	0	1	0	1	0	0	0	0
10	0	1	0	0	0	0	1	0	0
11	0	0	0	0	1	0	1	0	0
12	1	0	0	0	0	0	0	0	0
13	0	0	1	0	0	1	0	0	0
14	0	0	1	0	0	1	0	0	0
15	0	0	0	0	0	0	0	1	1
16	0	0	0	1	0	0	0	0	0
17	1	0	0	0	0	1	0	0	0
18	1	1	1	1	0	0	0	1	0
19	1	1	0	1	0	0	1	1	0
20	0	0	0	0	1	0	0	0	0

Numbers

Our example transaction DB has 20 records of supermarket transactions, from a supermarket that only sells 9 things

One month in a large supermarket with five stores spread around a reasonably sized city might easily yield a DB of 20,000,000 baskets, each containing a set of products from a pool of around 1,000

1GB database, 125,000 block reads for a single pass (blocksize=8KB)

if algorithm requires 10 passes, 1,250,000 block read

assume avg. read time=12ms per page/block

time spent in I/O=1,250,000 x 12 ms= 4 hrs.

Apriori Algorithm

- the **Apriori algorithm** was proposed by Agarwal and Srikant in 1994.
- Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as *candidate generation)*, and groups of candidates are tested against the data.
- Also called level-wise algorithm
- The algorithm terminates when no further successful extensions are found.
- Apriori uses breadth-first search to count candidate item sets efficiently.
- Since, algorithm uses prior knowledge of frequent item set properties, it is called as apriori.

Apriori Algorithm

Uses a ***Level-wise search***, where ***k*-itemsets** (An itemset that contains *k* items is a ***k-itemset***) are used to explore (*k*+1)-itemsets, to mine frequent itemsets from transactional database for Boolean association rules.

First, the set of frequent 1-itemsets is found. This set is denoted L1. L1 is used to find L2, the set of frequent 2-itemsets, which is used to fine L3, and so on, until no more frequent *k*-itemsets can be found.

Apriori candidate generation

- The candidate-gen function takes L_{k-1} and returns a superset (called the candidates) of the set of all frequent k-itemsets. It has two steps
  - *join* step: Generate all possible candidate itemsets C_k of length k
  - *prune* step: Remove those candidates in C_k that cannot be frequent.

***gen_candidate_itemsets** with the given* L_{k-1} as follows:

$C_k = \varnothing$
for all itemsets $l_1 \in L_{k-1}$ ***do***
for all itemsets $l_2 \in L_{k-1}$ ***do***
    if $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$
    then $c = l_1[1], l_1[2] \ldots l_1[k-1], l_2[k-1]$
    $C_k = C_k \cup \{c\}$

prune(C_k)

for all $c \in C_k$
for all $(k-1)$-subsets d of c ***do***
    if $d \notin L_{k-1}$
    then $C_k = C_k \setminus \{c\}$

Apriori Algorithm

Initialize: $k := 1$, C_1 = all the 1-itemsets;
read the database to count the support of C_1 to determine L_1.
$L_1 :=$ {frequent 1-itemsets};
$k := 2$; *// k represents the pass number//*
while $(L_{k-1} \neq \emptyset)$ ***do***
begin
  $C_k :=$ ***gen_candidate_itemsets*** *with the given* L_{k-1}
  prune(C_k)
  for all transactions $t \in T$ ***do***
  increment the count of all candidates in C_k that are contained in t;
  $L_k :=$ All candidates in C_k with minimum support ;
  $k := k + 1$;
end
Answer $:= \cup_k L_k$;

Min. sup. = 3

T1	**{Mango, Onion, Nestle Bar, Key-chain, Eggs, Yogurt}**
T2	{Doll, Onion, Nestle Bar, Key-chain, Eggs, Yogurt}
T3	{Mango, Apple, Key-chain, Eggs}
T4	{Mango, Umbrella, Corn, Key-chain, Yogurt}
T5	{Corn, Onion, Onion, Key-chain, Ice-cream, Eggs}

Transaction ID	Items Bought	Items Bought (order)
T1	{M, O, N, K, E, Y }	{E, K, M, N, O, Y}
T2	{D, O, N, K, E, Y }	{D, E, K, N, O, Y}
T3	{M, A, K, E}	{A, E, K, M}
T4	{M, U, C, K, Y }	{C, K, M, U , Y}
T5	{C, O, O, K, I, E}	{C, E, I, K, O}

TDB

Tid	Items
T1	{E, K, M, N, O, Y }
T2	{D, E, K, N, O,Y }
T3	{ A, E, K, M}
T4	{C, K, M, U, Y }
T5	{C, E, I, K, O}

C_1

1st scan

Itemset	sup
{A}	1
{C}	2
{D}	1
{E}	4
{I}	1
{K}	5
{M}	3
{N}	2
{O}	3
{U}	1
{Y}	3

L_1

Item set
{E}
{K}
{M}
{O}
{Y}

C_2

Items et
{E,K}
{E, M}
{E,O}
{E, Y}
{K,M}
{K, O}
{K, Y}
{M, O}
{M, Y}
{O, Y}

C_2

2nd scan

Items et	SUP
{E,K}	4
{E, M}	2
{E,O}	3
{E, Y}	2
{K,M}	3
{K, O}	3
{K, Y}	3
{M, O}	1
{M, Y}	2
{O, Y}	2

L_2

Itemset
{E, K}
{E, O}
{K, M}
{K, O}
{K, Y}

C_3

Itemset
{E, K, O}
{K, M, O}
{K, M, Y}
{K, O, Y}

3rd scan

C_3

Item set	SUP
{E, K, O}	3

L_3

Itemset
{E, K, O}

Freaquent items (L)=L_1 U L_2 U L_3

Freaquent items = {{E}, {K}, {M},{O},{Y},{E,K},{E,O},{K,M},{K,O},{K,Y},{E,K,O}}

Database TDB

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

$Sup_{min} = 2$

1[st] 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}

2[nd] 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}

3[rd] scan

L_3

Itemset	sup
{B, C, E}	2

Generating association rules

- For each frequent itemset *l*, generate all nonempty subsets of *l*.
- For every nonempty subset *s* of *l*, output the rule "*s* =>*l-s*" if (*support count*(*l*) / *support count*(*s*)) >= *min conf*, where *min conf* is the minimum confidence threshold.
- Because the rules are generated from frequent itemsets, each one automatically satisfies minimum support.
- Let *l* ={B,C,E} , given, min conf threshold=60%
- Subsets of *l* ={{B,C},{B,E},{C,E},{B},{C},{E}}
- s = {B,C}, {B,C}=> {E} , conf({B,C}=>{E})=2/2=100%
- s = {B,E}, {B,E}=> {C} , conf({B,E}=>{C})=2/3=66%
- s = {C,E}, {C,E}=> {B} , conf({C,E}=>{B})=2/2=100%
- s = {B}, {B}=> {C,E} , conf({B}=>{C,E})=2/3=66%
- s = {C}, {C}=> {B,E} , conf({C}=>{B,E})=2/3=66%
- s = {E}, {E}=> {B,C} , conf({E}=>{B,C})=2/3=66%

Apriori Advantages/Disadvantages

- Advantages
  - Uses large itemset property
  - Easily parallelized
  - Easy to implement
- Disadvantages
  - Assumes transaction database is memory resident.
  - Requires many database scans.

Database 2

1:bread, milk

2:bread, meat, orange juice, eggs

3:milk, meat, orange juice, cola

4:bread, milk, meat, orange juice

5:bread, milk, meat, cola

Database 1

Tid	Items
10	1,2,5
20	2,4
30	2,3
40	1,2,4
50	1,3
60	2,3
70	1,3
80	1,2,3,5
90	1,2,3

Itemsets

{1,2,3,4}

{1,2,4}

{1,2}

{2,3,4}

{2,3}

{3,4}

{2,4} min support count>=3

Partition Algorithm

- Frequent sets are very few compared to the set of all itemsets
- Partition set of transaction to smaller segments so that, each segment can be accommodated in main memory
- Compute set of frequent sets for each partition

Partition algorithm

- 2 scan of database
  - Generate set of all potentially frequent itemsets
  - Measure actual support
- 2 phases
  - Logically divide database into number of non overlapping partitions
    - If ‘n’ partitions – ‘n’ iterations
    - Merge frequent item sets
    - Local frequent item sets of same length from all partitions are combined to generate global candidate item sets

Partition algorithm

- Identify frequent item sets by generating actual support
- NOTE
  - A partition P refers to any subset of transaction in DB
  - Local support – fraction of transactions containing given item set
  - Local frequent item set – local support in a partition is at least minimum support
  - If an item set is not frequent in any partition, it is not frequent in whole DB.

Apriori Limitations

- Apriori algorithm can be very slow and the bottleneck is candidate generation.
- For example, if the transaction DB has 10^4 frequent 1-itemsets, they will generate 10^7 candidate 2-itemsets even after employing the downward closure.
- To compute those with support more than minimum support, the database need to be scanned at every level. It needs ($n + 1$) scans, where n is the length of the longest pattern.

```
P=partition_database(T); n=number of partitions;
//Phase I
for i = 1 to n do begin
  read_in_partition(T_i in P)
  L^i=generate all frequent itemsets of T_i using apriori method in main memory
end
//Merge Phase
for (k=1; L_k^i ≠ Φ, i=1,2,…,n; k++) do begin
  C_k^G = ∪_{i=1}^{n} L_k^i
End
//Phase II
for i = 1 to n do begin
  read_in_partition(T_i in P)
  for all candidates c ∈ C^G compute s(c)_Ti
End
L^G= {c ∈ C^G | s(c)_Ti ≥ σ }
Answer = L^G
```

Tid	Items
10	1,2,5
20	2,4
30	2,3
40	1,2,4
50	1,3
60	2,3
70	1,3
80	1,2,3,5
90	1,2,3

- Min. sup.count>=3
- Local sup. Count >=1

- L^1
  - $L_1=\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
  - $L_2=\{\{1,2\},\{1,5\},\{2,3\},\{2,4\},\{2,5\}\}$
  - $L_3=\{\{1,2,5\}\}$
- L^2
  - $L_1=\{\{1\},\{2\},\{3\},\{4\}\}$
    $L_2=\{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\}\}$
  - $L_3=\{\{1,2,4\}\}$
- L^3
  - $L_1=\{\{1\},\{2\},\{3\},\{5\})$
  - $L_2=\{\{1,2\},\{1,3\},\{1,5\},\{2,3\},\{2,5\},\{3,5\}\}$
  - $L_3=\{\{1,2,3\},\{1,2,5\},\{1,3,5\},\{2,3,5\}\}$
  - $L_4=\{\{1,2,3,5\}\}$

Tid	Items
10	1,2,5
20	2,4
30	2,3
40	1,2,4
50	1,3
60	2,3
70	1,3
80	1,2,3,5
90	1,2,3

- C^G_1={{1},{2},{3},{4},{5}}
- C^G_2={{1,2},{1,3},{1,4},{1,5},{2,3},{2,4},{2,5},{3,5}}
- C^G_3={{1,2,3},{1,2,4},{1,2,5},{1,3,5},{2,3,5}}
- C^G_4={{1,2,3,5}}

C^G={C^G_1, C^G_2, C^G_3, C^G_4}

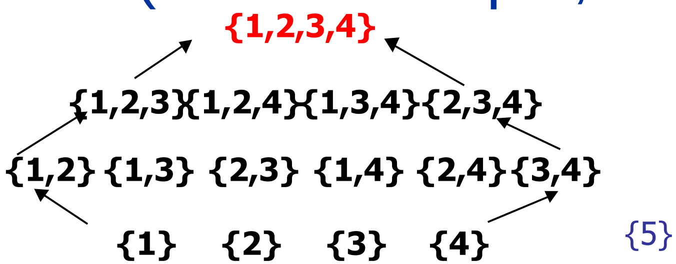
L^G={{1},{2},{3},{1,2},{1,3},{2,3}}

Pincer Search Method

- Apriori
  - Bottom-up, breadth-first search
  - # DB passes = largest size of frequent item set
  - Performance – decreases
- Solution – pincer search method
  - Bi-directional search
  - Top-down and bottom-up search
  - Find frequent item sets by bottom-up and maintain list of maximal frequent item sets
  - Count support for both

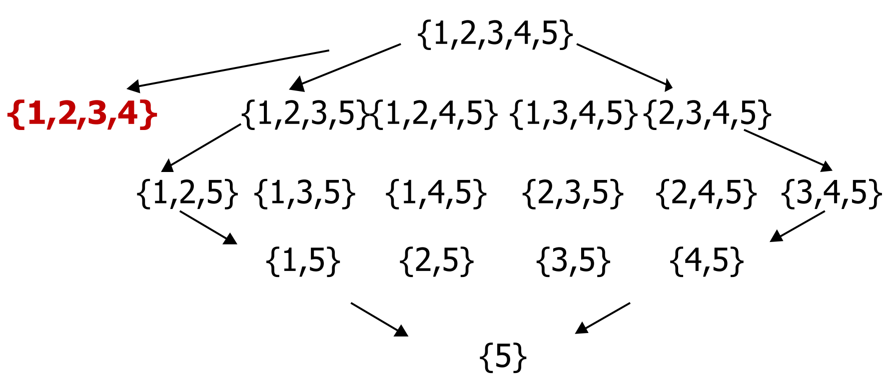
Complexity of One-Way Searches

For bottom-up search, every frequent itemset is explicitly examined (in the example, until {1,2,3,4} is examined)



Black: frequent itemsets
Red: maximal frequent itemsets
Blue: infrequent itemsets

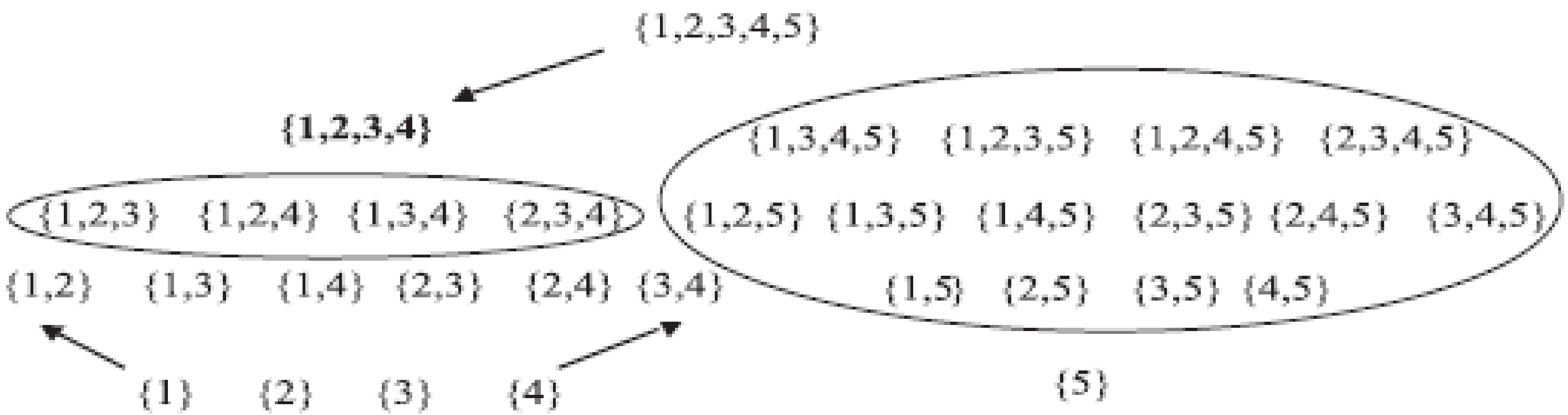
- For top-down search, every infrequent itemset is explicitly examined (in the example until {5} is examined)



Red: maximal frequent itemsets
Black: infrequent itemsets

Use Property 1 to eliminate candidates in the top-down search

Use Property 2 to eliminate candidates in the bottom-up search



Algorithm: The Pincer-Search algorithm

L_0 := Ø; k := 1; C_1 := {{i} | $i \in I$ }; S_0 := Ø

MFCS := {{1, 2, . . . , n}}; MFS := Ø

while $C_k \neq$ Ø

read database and count supports for C_k and MFCS

remove frequent itemsets from MFCS and add them to MFS

L_k:={frequent k-itemset}

S_k := {infrequent itemsets in C_k}

call the *MFCS-gen* algorithm if $S_k \neq$ Ø // MFS=MFS U {frequent itemsets in MFCS }

call MFS-pruning procedure

generate C_{k+1} from L_k (apriori join)

if any frequent itemset in L_k is removed in MFS-pruning procedure

call the *recovery* procedure to recover candidates to C_{k+1}

call MFCS *prune* procedure to prune candidates in C_{k+1}

k := k + 1

end-while

Answer = U_k L_k U MFS

MFCS-Gen Algorithm

for all itemset s in S_k

  for all itemsets m in MFCS

    if s is a subset of m

      MFCS := MFCS $\setminus$ $\{ m \}$

      for all items e in itemset s

        if $m \setminus \{ e \}$ is not a subset of any itemset in the MFCS

          MFCS := MFCS $\cup$ $\{ m \setminus \{ e \} \}$

return MFCS

Recovery

for all items l in L_k

for all items m in MFS

if the first k-1 items in l are also in m

for i from j+1 to |m|

/*suppose $m.item_j = l.item_{k-1}$ */

$C_{k+1}=C_{k+1} \cup \{l.item_1, l.item_2, \ldots, l.ltem_k, m.item_i\}$

MFS-Prune

for all items l in L_k
if l is a subset of any itemset in the current MFS
delete l from L_k

MFCS-Prune

for all items c in C_{k+1}
if c is not a subset of any itemset in the current MFCS
delete c from C_{k+1}

Tid	Items
10	1,2,5
20	2,4
30	2,3
40	1,2,4
50	1,3
60	2,3
70	1,3
80	1,2,3,5
90	1,2,3

Support Count >= 2

L_0={ } k=1 C_1={{1},{2},{3},{4},{5}} S_0={}
MFCS={ {1,2,3,4,5} }, MFS={ }

C_1={{1}-6,{2}-7,{3}-6,{4}-2,{5}-2}
MFCS={ {1,2,3,4,5}-0 }, MFS={ }

L_1={{1},{2},{3},{4},{5}}
S_1={ }

call the *MFCS-gen* algorithm if $S_1 \neq \emptyset$
MFS=MFS U {frequent itemsets in MFCS }
call MFS-pruning procedure
generate C_2 from L_1 (apriori)

C_2 ={ {1,2}, {1,3},{1,4},{1,5},{2,3},{2,4},{2,5},{3,4},{3,5},{4,5}}

MFS Prune, MFCS Prune, k=2

C_2 ={ {1,2}-4, {1,3}-4,{1,4}-1,{1,5}-2,{2,3}-4,{2,4}-2,{2,5}-2,{3,4}-0,{3,5}-1,{4,5}-0}
MFCS = { {1,2,3,4,5}-0} MFS={}

L_2 ={ {1,2}, {1,3},{1,5},{2,3},{2,4},{2,5}}

S_2 ={{1,4},{3,4},{3,5},{4,5}}

MFCS Gen
s={1,4}, m={1,2,3,4,5}, s is subset of m, MFCS=MFCS/m={ }
e=1, m/e={1,2,3,4,5}/{1}={2,3,4,5} not subset of MFCS, MFCS={2,3,4,5}
e=4, m/e= {1,2,3,4,5}/{4}={1,2,3,5} not subset of MFCS,
MFCS={{1,2,3,5},{2,3,4,5}}

s={3,4}, m={1,2,3,5}, s is not a subset of m
s={3,4}, m={2,3,4,5}, s is a subset of m, MFCS=MFCS/m={{1,2,3,5}}
e=3, m/e={2,3,4,5}/{3}={2,4,5} not subset of MFCS, MFCS={{2,4,5},{1,2,3,5}}
e=4, m/e= {2,3,4,5}/{4}={2,3,5} is subset of MFCS, MFCS={{2,4,5},{1,2,3,5}}

MFCS={{1,2,3,5},{2,4,5}}

s={3,5}, m={1,2,3,5}, s is a subset of m, MFCS={{2,4,5}}
e=3, m/e={1,2,3,5}/{3}={1,2,5} not subset of MFCS, MFCS={{1,2,5},{2,4,5}}
e=5, m/e= {1,2,3,5}/{5}={1,2,3} not subset of MFCS,
MFCS={{1,2,3},{1,2,5},{2,4,5}}

s={4,5}, m={1,2,3}, s is not a subset of m,
s={4,5}, m={1,2,5}, s is not a subset of m,
s ={4,5}, m={2,4,5}, s is a subset of m, MFCS=MFCS/m={{1,2,3},{1,2,5}}
e=4, m/e={2,4,5}/{4}={2,5} subset of MFCS, MFCS={{1,2,3},{1,2,5}}
e=5, m/e= {2,4,5}/{5}={2,4} not a subset of MFCS, MFCS={{2,4},{1,2,3},{1,2,5}}

MFS={{2,4}}, MFCS={{1,2,3},{1,2,5}}

MFS Prune

L_2 ={ {1,2}, {1,3},{1,5},{2,3},{2,4},{2,5}}   // before MFS prune

L_2 ={ {1,2}, {1,3},{1,5},{2,3},{2,5}}   // after MFS prune

MFS={{2,4}},   MFCS={{1,2,3},{1,2,5}}

Generate C_3

C_3 = {{1,2,3},{1,2,5},{1,3,5},{2,3,5}}

Recovery

MFS={2,4}, L_2={2,3} => {2,3,4}

MFS={2,4}, L_2={2,5} => {2,4,5}

C_3 = {{1,2,3},{1,2,5},{1,3,5},{2,3,5},{2,3,4},{2,4,5}} //after recovery

MFCS Prune

MFCS={{1,2,3},{1,2,5}}
C_3 *= {{1,2,3},{1,2,5},{1,3,5},{2,3,5},{2,3,4},{2,4,5}} //before prune*

C_3 *= {{1,2,3},{1,2,5}} //after prune*

k=3

MFCS={{1,2,3}-2,{1,2,5}-2}
C_3 *= {{1,2,3}-2,{1,2,5}-2}*

MFS={{2,4},{1,2,3},{1,2,5}}, MFCS={ }

L_3 *= {{1,2,3},{1,2,5}},* S_3 *={ }, MFCS Gen – not called*

MFS Prune
L_3 *= { },* C_4 *= { } , MFCS Prune*
k=4

Frequent item sets= L_1 U L_2 U L_3 U MFS

Database1

1: 1,5,6,8
2: 2,4,8
3: 2,3,7,8
4: 2,3,4
5: 2,6,7,9
6: 2,3,6,7,9
7: 2,4,6,7,9
8: 1,3,5,7
9: 2,3,7
10: 2,3,8,9

Database2

1: *a b c d e f*
2: *a b c g*
3: *a b d h*
4: *b c d e k*
5: *a b c*

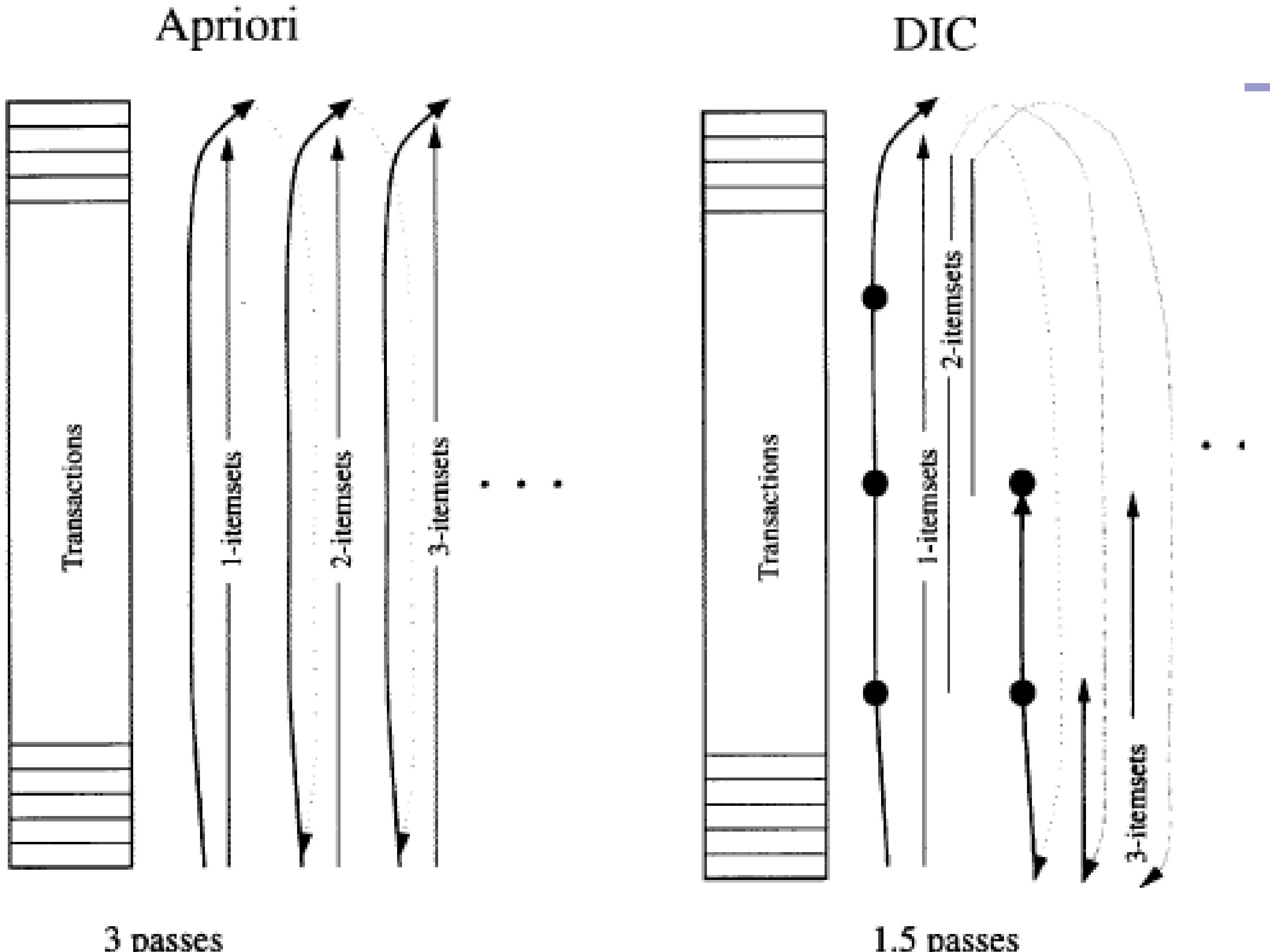
Min supp=2

Dynamic Itemset Counting (DIC) Algorithm

- Bin et al. -1997
- Basics: works like a train running over the data
- Stops at intervals M between transactions
- Passengers – itemsets
- Requirement – get on and off at the same stop
- Assumption – records are read sequentially

Apriori vs. DIC

- Apriori
  - level-wise
  - many passes
- DIC
  - reduce the number of passes
  - fewer candidate itemsets than sampling
- example : 40,000 transaction, M = 10,000


Apriori
DIC
Transactions
1-itemsets
2-itemsets
3-itemsets
Transactions
1-itemsets
2-itemsets
3-itemsets
3 passes
1.5 passes

Counting large itemsets

- Itemsets : a large lattice
- count just the minimal small itemsets
  - the itemsets that do not include any other small itemsets
- mark itemset
  - Solid box - confirmed frequent itemset
  - Solid circle - confirmed infrequent itemset
  - Dashed box – item with support > min. support
  - Dashed circle – fresh items sets

DIC Algorithm

Initially,

solid-box – empty itemset;

solid-circle – empty;

dashed-box – empty;

dashed-circle – all 1-itemset with stop-number as 0;

current-stop-number = 0;

do until the dashed-circle or dashed-box is empty

read database till next stop and increase the counters for itemsets in the dashed-box & dashed-circle to reach the next stop;

increase current-stop-number by 1;

for each itemset in the dashed-circle

{ if count of itemset is greater than σ

move the itemset to dashed-box;

generate new itemsets if possible from dashed-box and solid-box, put into dashed-circle with counter value = 0 and stop-number = current-stop-number;

}

```
        for each itemset in the dashed-circle
            if  stop-number = current-stop-number and itemset
             is counted through all transactions then
                move this itemset to solid-circle;

        for each itemset in the dashed-box
            if  stop-number = current-stop-number
                move this itemset to solid-box;
end // do
return items in solid-box
```