Mining Frequent patterns, Associations and Correlations: Basic Concepts and Methods

What Is Pattern Discovery?

- Patterns represent intrinsic and important properties of datasets
- Frequent Patterns: A set of items, subsequences, or substructures that occur frequently together (or strongly correlated) in a data set
- Eg: Set of items milk and bread appear frequently together in a transaction
- A subsequence such as buying pc, then digital camera and then memory card
- Motivation examples:

– What products were often purchased together?

– What are the subsequent purchases after buying an iPad?

– What kinds of DNA are sensitive to this new drug?

– What word sequences likely form phrases in this corpus?

Pattern Discovery: Why Is It Important?

- Finding inherent regularities in a data set
- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Mining sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: Discriminative pattern-based analysis
  - Cluster analysis: Pattern-based subspace clustering
- Broad applications: Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log analysis, biological sequence analysis

Market Basket Analysis

- Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets.
- Industries are interested in such pattern of data.
- Helps in many business-decision making processes such as
    - To develop marketing strategies
    - catalog design
    - cross-marketing
    - customer shopping behavior analysis.
- **Market basket analysis: Process analyses customer buying habits by finding association between the different items that customer place in their “shopping baskets”**

From Frequent Itemsets to Association Rules

- The patterns can be represented as association rules.
- Support and confidence are two measures of rule interestingness.
- Association rules are considered interesting if satisfy minimum support and confidence.
- Computer => software[support 2%,confidence= 60%]
- Association rules: X => Y
- Support, *s*: The probability that a transaction contains $X \cup Y$
  - Support $(X \Rightarrow Y) = P(X \cup Y)$
- Confidence, c: The conditional probability that a transaction containing X also contains Y
  - $c(X \Rightarrow Y) = P(Y/X) = sup(X \cup Y) / sup(X)$
  - c = support_count(x $\cup$ *Y)/support_count(X)*
- Association rules are considered if they satisfy minimum support and confidence threshold

From Frequent Itemsets to Association Rules

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- **Association rule mining**: Find all of the rules, *X* → *Y*, with minimum support and confidence
- Frequent itemsets: Let *minsup = 50%*
  - Freq. 1-itemsets: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3
  - Freq. 2-itemsets: {Beer, Diaper}: 3
- Association rules: Let *minconf = 50%*
  - *Beer* → *Diaper* (60%, 100%)
  - *Diaper* → *Beer* (60%, 75%)

{Beer}   {Diaper} = {Beer, Diaper}

Basic Concepts: Frequent Itemsets (Patterns)

- Itemset: A set of one or more items
- k-itemset: X = {x_1, …, x_k}
- *(absolute) support (count)* of X:

  Occurrence frequency of an itemset is the number of transactions that contain itemset.
- *(relative) support*, *s*: The probability that a transaction contains P(A U B)
- An itemset X is *frequent* if the support of X is no less than a *minsup* threshold (denoted as σ)

Let minsup = 50%

- Freq. 1-itemsets:
- Beer: 3 (60%); Nuts: 3 (60%)
- Diaper: 4 (80%); Eggs: 3 (60%)
- Freq. 2-itemsets:
- {Beer, Diaper}: 3 (60%)

Association Rule Mining

- **Association rule mining can be viewed as two step process**
  - Find all frequent item sets
  - Generate strong association rules from frequent item sets.
- **Finding all frequent item sets:**
  - Each of these item sets will occur at least as frequently as a predetermined minimum support count, min_sup
- **Generate strong association rules from the frequent item sets:**
  - These rules should satisfy minimum support and confidence

Challenge: There Are Too Many Frequent Patterns!

- A long pattern contains a combinatorial number of shorter frequent item sets.
- How many frequent itemsets does the following TDB_1 contain?
  - $TDB_{1:}$ T_1: $\{a_1, a_2 \ldots, a_{100}\}$ Assuming (absolute) minsup = 1

    1-itemsets: $\{a_1\}, \{a_2\}\ldots.\{a_{100}\}$ contains $\binom{100}{1}$

    2-itemsets: $\{a_{1,} a_2\},\{a_{1,}a_3\}$ … contains $\binom{100}{2}$

    99-itemsets: $\{a_1, a_2, \ldots, a_{99}\}$: 1, …, $\{a_2, a_3, \ldots, a_{100}\}$: 1

    100-itemset: $\{a_1, a_2, \ldots, a_{100}\}$: 1 contains $\binom{100}{100}$
  - In total: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100}$
  - $= 2^{100} - 1$ sub-patterns! How to handle such a challenge?

A too huge set for any computer to compute or store!

Closed and Maximal

Solution 1: **Closed patterns**: A pattern (itemset) X is closed in a dataset D if X is *frequent*, and there exists *no proper super-itemset Y such that Y has the same support* as X in D.

- Solution 2: **Max-patterns:** A pattern X is a max-pattern if X is frequent and there exists no (immediate) super-itemset Y such that XCY and Y is frequent

Expressing Patterns in Compressed Form: Closed Patterns and Maxim

- *My dataset*: 1:*A,B,C,E* 2:*A,C,D,E,* 3:B,*C,E* 4:*A,C,D,E*
  5: *C,D,E* 6: *A,D,E*
- {A} = 4 ; {B} = 2 ; {C} = 5 ; {D} = 4 ; {E} = 6
- {A,B} = 1; {A,C} = 3; {A,D} = 3; {A,E} = 4; {B,C} = 2; {B,D} = 0; {B,E} = 2; {C,D} = 3; {C,E} = 5; {D,E} = 3
- {A,B,C} = 1; {A,B,D} = 0; {A,B,E} = 1; {A,C,D} = 2; {A,C,E} = 3; {A,D,E} = 3; {B,C,D} = 0; {B,C,E} = 2; {C,D,E} = 3
- {A,B,C,D} = 0; {A,B,C,E} = 1; {B,C,D,E} = 0
- Min_sup=0.5

Closed and Maximal-Example

- {A} = 4 ; not closed due to {A,E}
- {B} = 2 ; not frequent => ignore
- {C} = 5 ; not closed due to {C,E}
- {D} = 4 ; closed, but not maximal due to e.g. {A,D}
- {E} = 6 ; closed, but not maximal due to e.g. {D,E}
- {A,C,E} = 3; maximal frequent
- {A,D,E} = 3; maximal frequent
- {C,D,E} = 3; maximal frequent

Frequent Itemset Mining Methods

- **The Downward Closure Property of Frequent Patterns**
- **The Apriori Algorithm**
- **Extensions or Improvements of Apriori**
- **Mining Frequent Patterns by Exploring Vertical Data Format**
- **FPGrowth: A Frequent Pattern-Growth Approach**
- **Mining Closed Patterns**

Apriori Pruning and Scalable Mining Methods

- Scalable mining Methods: Three major approaches
  - Level-wise, join-based approach: Apriori
  - Vertical data format approach
  - Frequent pattern projection and growth

Apriori Pruning and Scalable Mining Methods

- Apriori is a seminal algorithm, uses priori knowledge of frequent itemset properties.
- Apriori employs level wise search where k-itemsets are used to explore (k+1) item sets
- The set of frequent 1-itemsets is found by following
  - Scan the database to accumulate the count for each item
  - Accumulate the items that satisfy minimum support .
  - The resulting set is denoted as L_1.
  - L_1 is used to find L_2 the set of frequent-2 item sets which used to find L_3 until no frequent item sets can be found
- To improve the efficiency of level-wise generation Apriori property is used to reduce search space.

Apriori Property

- **All nonempty subsets of a frequent itemsets must also frequent. (or) If there is any itemset which is infrequent, its superset should not be generated/tested!**
  - If an item does not support min_sup (P(I)<min_sup)
  - If A is added to I then (I U A) cannot occur more frequent than I i.e, P (IUA)<min_sup
- This property is called **antimonotonicity:** if a set cannot pass a test all its superset will fail for the same test
- Algorithm make use of Apriori property follows two-step process consisting of **join and prune actions**

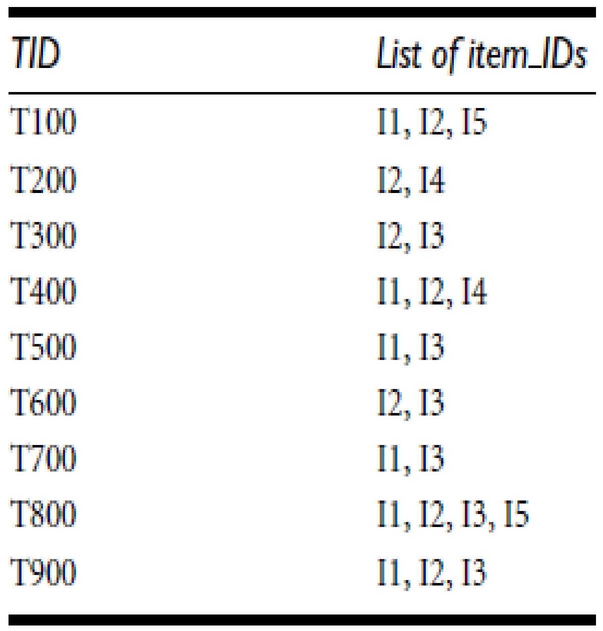
Join step for k>=2

- To find L_k : Generate a set of candidate k-itemsets by joining L_{k-1} by itself
  - Set of candidates is denoted by $C_{k.}$
  - Let I1 and I2 be itemsets in L_{k-1}
  - Apriori assumes itemset are sorted in lexicographic order
  - The join can be performed when $L_{k-1} \bowtie L_{k-1}$ if their first (k-2) items are in common
  - Members are joined if (I1[1]=I2[1] ^ I1[2]=I2[2] ^.....^(I1[k-2]=I2[k-2]) ^ (I1[k-1]<I2[k-1])
  - The resulting data set formed by joining I1 and I2 is {I1[1],I1[2], ……., I1[k-2],I1[k-1],I2[k-1]

Prune Step

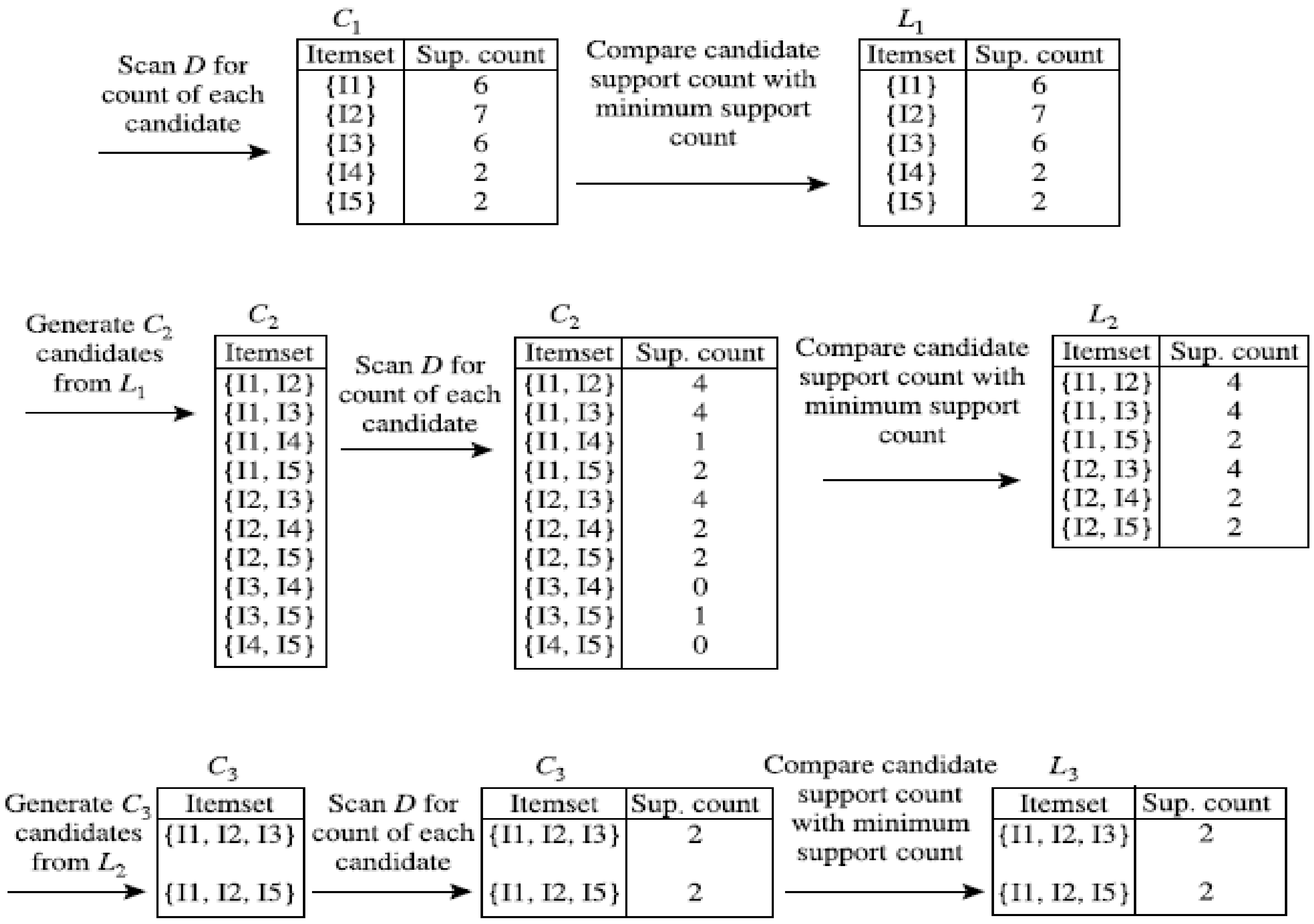
- Initially, scan DB once to get frequent 1-itemset
- Let C_k is a superset of L_k , members of C_k may or may not be frequent but all frequent k-itemsets are included in C_k .
- Data base scan done to determine the count of candidates in Ck.
- Apriori property is used any (k-1) itemset that is not frequent cannot be subset of frequent k-itemset and so removed from C_k
- Subtest testing can be done using Hash tree.

Frequent itemsets-generation



TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Frequent itemsets-generation



Scan D for count of each candidate →

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Compare candidate support count with minimum support count →

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Generate C_2 candidates from L_1 →

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

Scan D for count of each candidate →

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

Compare candidate support count with minimum support count →

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

Generate C_3 candidates from L_2 →

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

Scan D for count of each candidate →

C_3

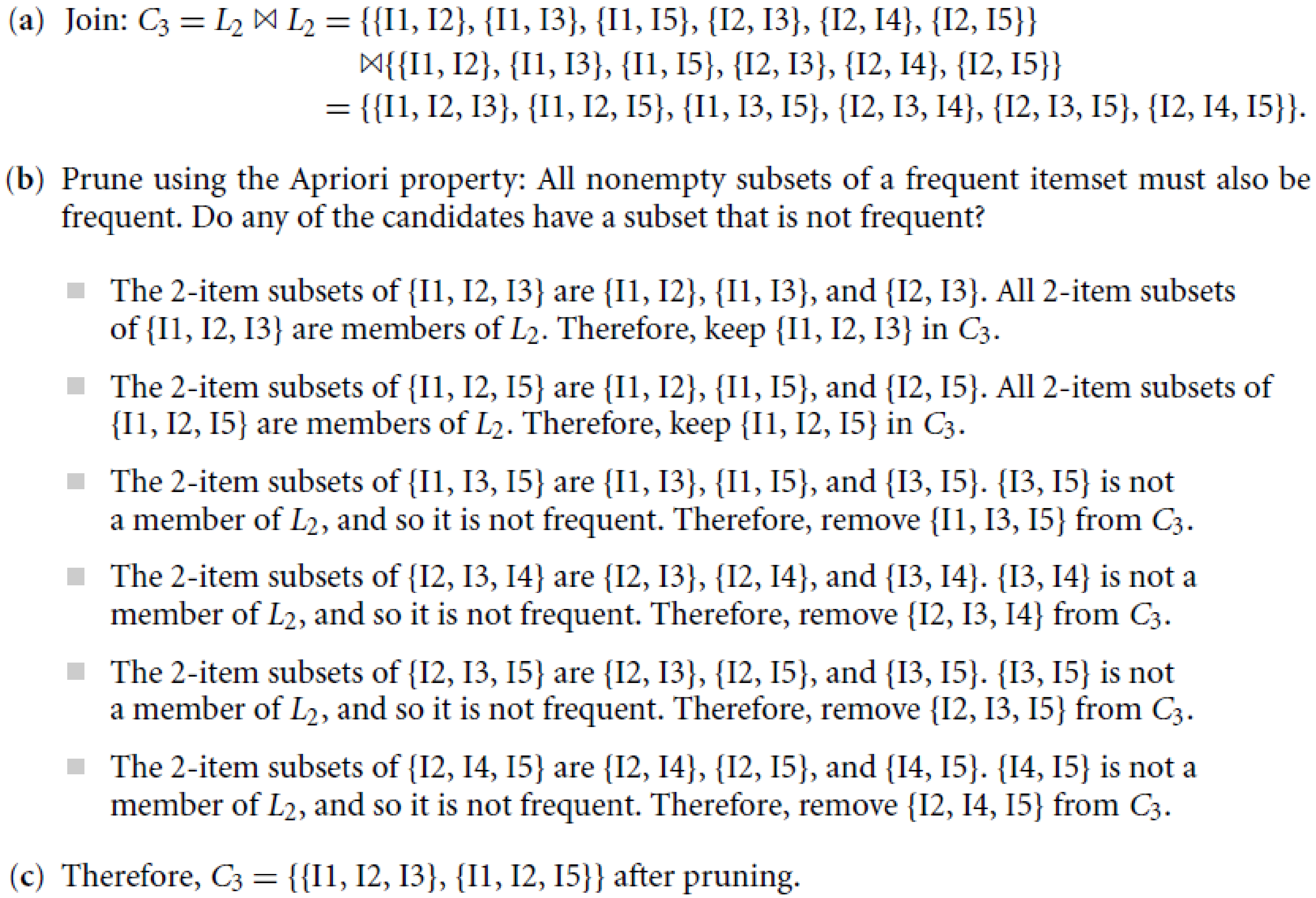
Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Compare candidate support count with minimum support count →

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Join & Prune STEPS



(a) Join: $C_3 = L_2 \bowtie L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$\bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$.

(b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of L_2. Therefore, keep {I1, I2, I3} in C_3.
- The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of L_2. Therefore, keep {I1, I2, I5} in C_3.
- The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I1, I3, I5} from C_3.
- The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I4} from C_3.
- The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I5} from C_3.
- The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I4, I5} from C_3.

(c) Therefore, $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

Generation of frequent Itemset

- In 1st iteration each item is a member of the set of candidate 1-itemsets C_1.
- Set of frequent item sets L_1 is determined by considering C_1 satisfying min_sup.
- To obtain L_K, the algorithm
  - Generate C_K by joining (L_{K-1} **join** L_{K-1} **)**
  - Prune: Prune the item sets based on the Apriori property all subsets of frequent itemset must also frequent.
  - Accumulate support of each item set in C_k and determine L_k with min_sup

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

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

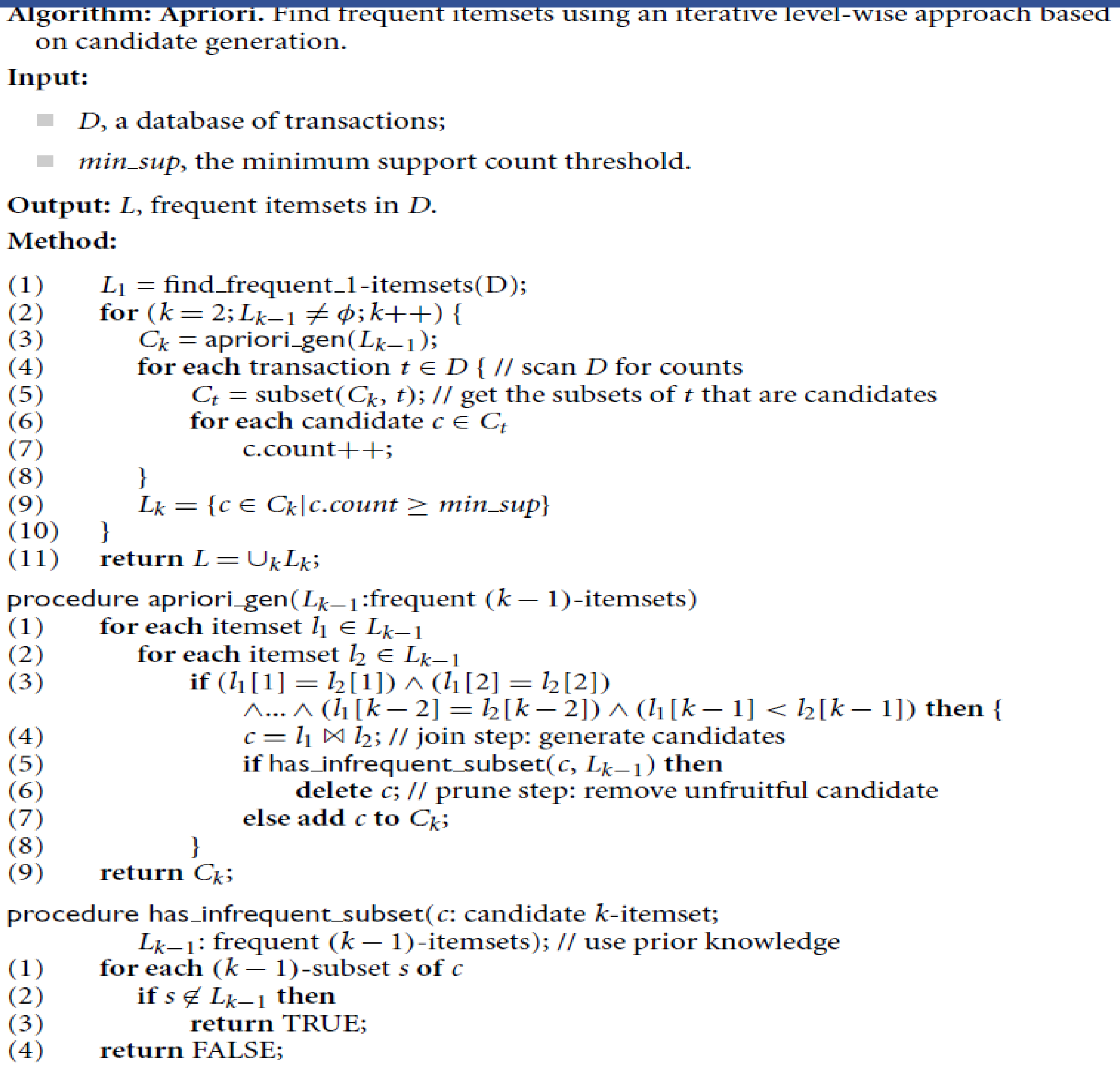
Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Apriori Algorithm



Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)      $L_1$ = find_frequent_1-itemsets(D);
(2)      for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
(3)          $C_k$ = apriori_gen($L_{k-1}$);
(4)          for each transaction $t \in D$ { // scan $D$ for counts
(5)              $C_t$ = subset($C_k$, $t$); // get the subsets of $t$ that are candidates
(6)              for each candidate $c \in C_t$
(7)                  c.count++;
(8)          }
(9)          $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)     }
(11)     return $L = \cup_k L_k$;

procedure apriori_gen($L_{k-1}$:frequent ($k-1$)-itemsets)
(1)      for each itemset $l_1 \in L_{k-1}$
(2)          for each itemset $l_2 \in L_{k-1}$
(3)              if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2])$
                     $\wedge ... \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
(4)                  $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)                  if has_infrequent_subset($c$, $L_{k-1}$) then
(6)                      delete $c$; // prune step: remove unfruitful candidate
(7)                  else add $c$ to $C_k$;
(8)              }
(9)      return $C_k$;

procedure has_infrequent_subset($c$: candidate $k$-itemset;
             $L_{k-1}$: frequent ($k-1$)-itemsets); // use prior knowledge
(1)      for each ($k-1$)-subset $s$ of $c$
(2)          if $s \notin L_{k-1}$ then
(3)              return TRUE;
(4)      return FALSE;
```

Generation of Association rules from Frequent Itemsets

- Strong association rules can be derived from frequent itemsets.
- Strong association rules always satisfy minimum support and confidence

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

- Association rules can be generated:
  - For each frequent itemset I generate all non-empty subsets of I
  - For every non-empty subset s of I output the rule "s=>(I-s)" if

  Support_count(Ius)/support_count(s)>=min_conf

Association Rules- Example

- X={I1,I2,I5} what are association rules generated from X
- The non-empty subsets are:
  - {I1,I2},{I1,I5},{I2,I5},{I1},{I2},{I5}.
  - Association rules are

$$\begin{aligned}
&\{I1,I2\} \Rightarrow I5, && \mathit{confidence} = 2/4 = 50\% \\
&\{I1,I5\} \Rightarrow I2, && \mathit{confidence} = 2/2 = 100\% \\
&\{I2,I5\} \Rightarrow I1, && \mathit{confidence} = 2/2 = 100\% \\
&I1 \Rightarrow \{I2,I5\}, && \mathit{confidence} = 2/6 = 33\% \\
&I2 \Rightarrow \{I1,I5\}, && \mathit{confidence} = 2/7 = 29\% \\
&I5 \Rightarrow \{I1,I2\}, && \mathit{confidence} = 2/2 = 100\%
\end{aligned}$$

  - If minimum support count is 70% then second,third and last rules are output

Improving the Efficiency of Apriori

Improving the Efficiency of Apriori

- Several variation of Apriori algorithm have been proposed on improving the efficiency of Apriori original Algorithm
  - Hash-based Technique
  - Transaction reduction
  - Partitioning
  - Sampling
  - Dynamic itemset counting

Hash based Technique

- Idea: Reduces the size of candidate k-itemsets, C_k
- **To generate the frequent-1 itemsets (L1)**
  - Generate 2-itemsets for each transaction
  - Map them into different buckets of hash-table structure
  - Increase the corresponding bucket counts
  - Remove the non-frequent candidate set
  - Substantially reduces the number of candidate k-itemsets

Hash-Based Technique (Example)

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$ $\longrightarrow$

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Transaction Reduction

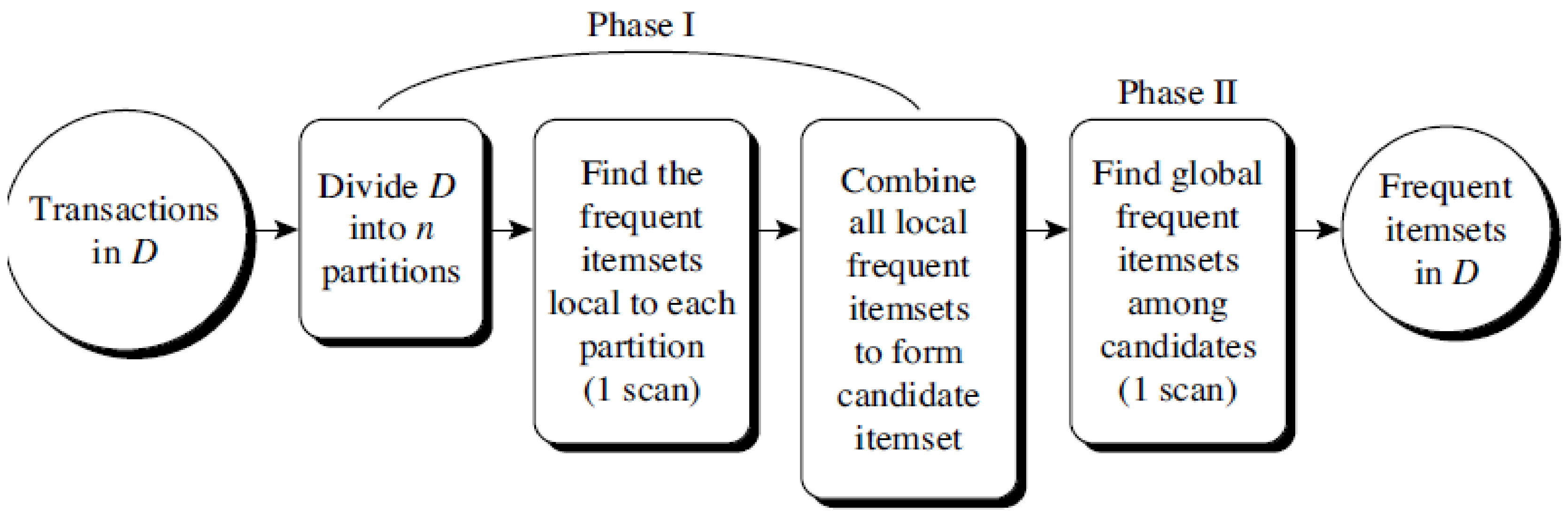
- Idea: Reduces the number of transactions scanned in future iterations
- A transaction that does not contain any frequent k-itemsets cannot contain (k+1) itemsets
- Such a transaction can be marked or removed from further

Partitioning

- Idea: Partitioning the data to find candidate itemsets
- Algorithm consists of 2 phases
- Phase-1
  - Algorithm divides the transactions D into n overlapping partitions
  - Each partition obtain local frequent itemsets
  - Collection of all frequent itemsets from all partitions forms the global candidate itemsets with respect to D

Partitioning

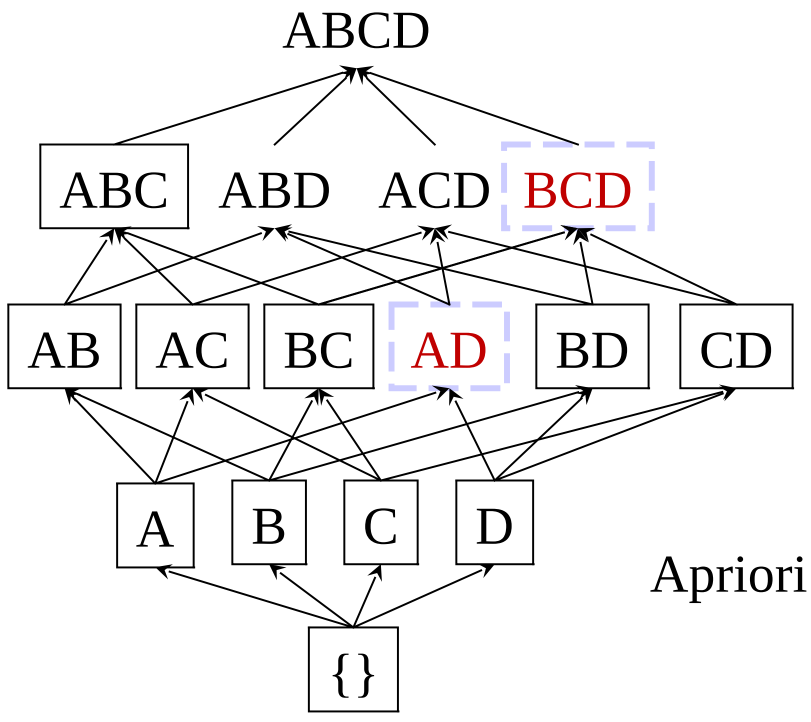
- Phase –II
  - Assess the support count of candidate itemsets
  - Determine the global frequent itemsets


Phase I
Phase II
Transactions in D
Divide D into n partitions
Find the frequent itemsets local to each partition (1 scan)
Combine all local frequent itemsets to form candidate itemset
Find global frequent itemsets among candidates (1 scan)
Frequent itemsets in D

Sampling

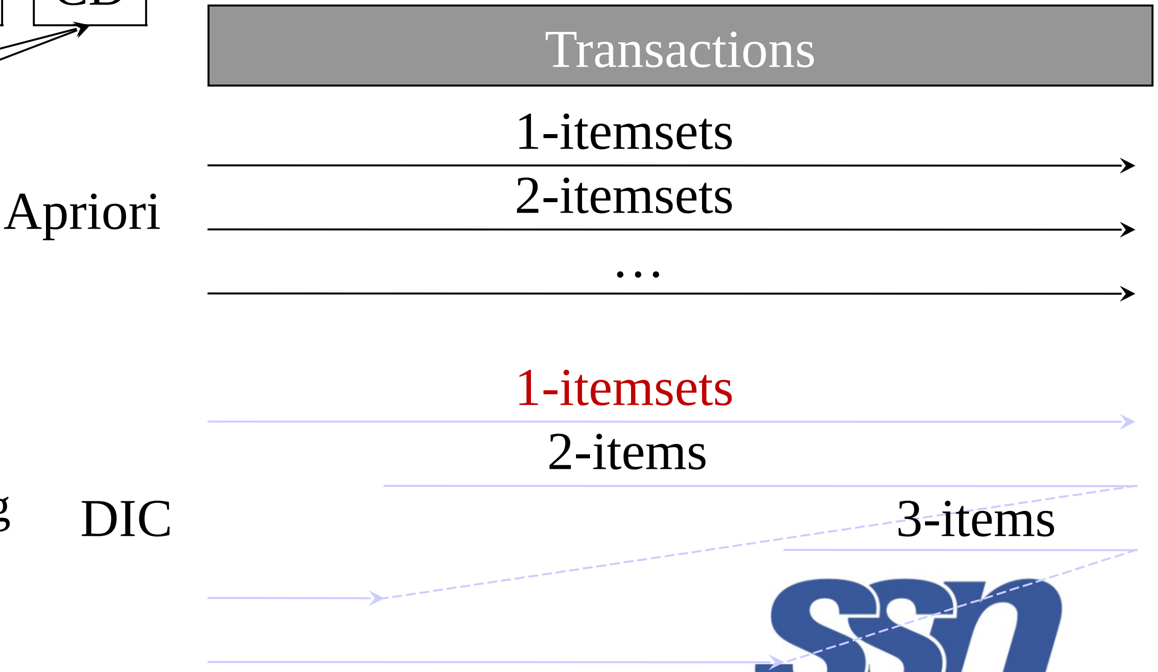
- Basic idea: To pick random sample S of D, search for frequent itemsets in S instead of D
- Attains some degree of accuracy against efficiency
- Searching in S there is possibility of missing global frequent itemsets.
- Use lower support threshold than minimum support to find frequent itemsets local to S (L^S)
- A mechanisms is used to determine whether all global frequent itemsets are included in L^S
- Approach is best for computationally intensive applications that must run frequently

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



Dynamic itemset counting and implication rules for market basket data *SIGMOD'97*