Pattern Mining: Basic Concepts and Methods

- Basic Concepts
- Frequent Itemset Mining Methods – Apriori Algorithm
- Which Patterns Are Interesting? - Pattern Evaluation Methods

The Downward Closure Property of Frequent Patterns

- Observation: From TDB$_{1:}$ T$_1$: {a$_1$, …, a$_{50}$}; T$_2$: {a$_1$, …, a$_{100}$}
  - We get a frequent itemset: {a$_1$, …, a$_{50}$}
  - Also, its subsets are all frequent: {a$_1$}, {a$_2$}, …, {a$_{50}$}, {a$_1$, a$_2$}, …, {a$_1$, …, a$_{49}$}, …
  - There are some hidden relationships among frequent patterns!
- The downward closure (also called "Apriori") property of frequent patterns
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - Every transaction containing {beer, diaper, nuts} also contains {beer, diaper}
  - Apriori: Any subset of a frequent itemset must be frequent
- Efficient mining methodology
  - If any subset of an itemset S is infrequent, then there is no chance for S to be frequent—why do we even have to consider S!?

A sharp knife for pruning!

Apriori Pruning and Scalable Mining Methods

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not even be generated! (Agrawal & Srikant @VLDB’94, Mannila, et al. @ KDD’ 94)
- Scalable mining Methods: Three major approaches
  - Level-wise, join-based approach: Apriori (Agrawal & Srikant@VLDB’94)
  - Vertical data format approach: Eclat (Zaki, Parthasarathy, Ogihara, Li @KDD’97)
  - Frequent pattern projection and growth: FPgrowth (Han, Pei, Yin @SIGMOD’00)

Apriori: A Candidate Generation & Test Approach

- Outline of Apriori (level-wise, candidate generation and test)
    - Scan DB once to get frequent 1-itemset
    - Repeat
        - Generate length-(k+1) candidate itemsets from length-k frequent itemsets
        - Test the candidates against DB to find frequent (k+1)-itemsets
        - Set k := k +1
    - Until no frequent or candidate set can be generated
    - Return all the frequent itemsets derived

The Apriori Algorithm (Pseudo-Code)

C_k: Candidate itemset of size k

F_k : Frequent itemset of size k

```
K := 1;
$F_k$ := {frequent items};   // frequent 1-itemset
**While** ($F_k$ != $\emptyset$) **do {**    // when $F_k$ is non-empty
   $C_{k+1}$ := candidates generated from $F_k$;  // candidate generation
   Derive $F_{k+1}$ by counting candidates in $C_{k+1}$ with respect to *TDB* at minsup;
   k := k + 1
   **}**
**return** $\cup_k F_k$            // return $F_k$ generated at each level
```

The Apriori Algorithm—An Example

minsup = 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

F_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

F_2

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

C_3

Itemset
{B, C, E}

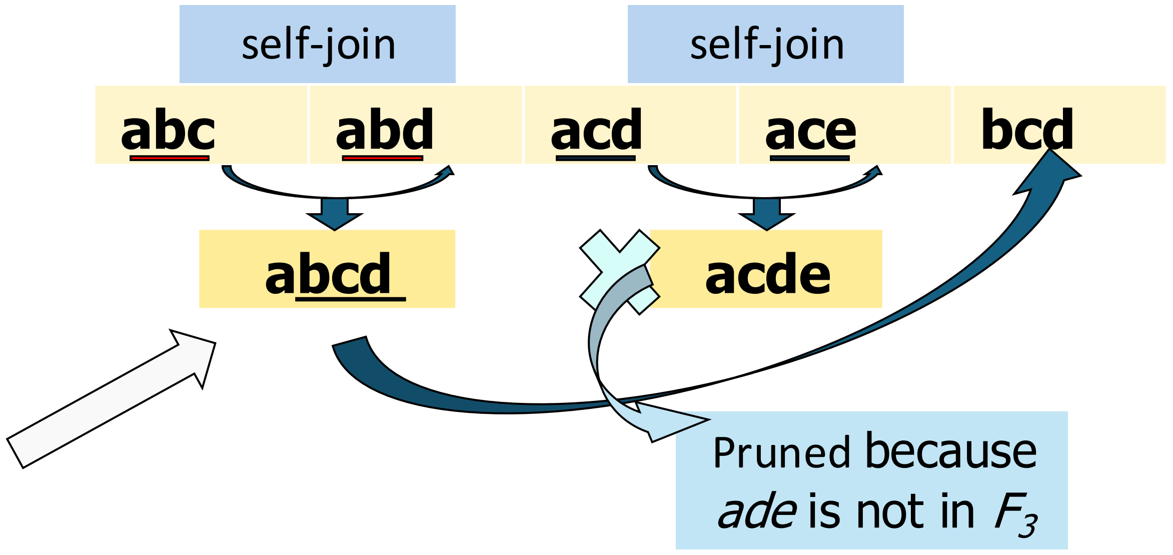
3rd scan

F_3

Itemset	sup
{B, C, E}	2

Apriori: Implementation Tricks

- How to generate candidates?
  - Step 1: self-joining F_k
  - Step 2: pruning
- Example of candidate-generation
  - F_3 = {*abc, abd, acd, ace, bcd*}
  - Self-joining: $F_3 * F_3$
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in F_3
  - C_4 = {*abcd*}



Apriori Advantages/Disadvantages

- Advantages:
  - Uses large itemset property (same as Apriori Property)
  - Easily parallelized
  - Easy to implement

- Disadvantages:
  - Assumes transaction database is memory resident
  - Requires up to m database scans

Apriori (Another Example Page 151)

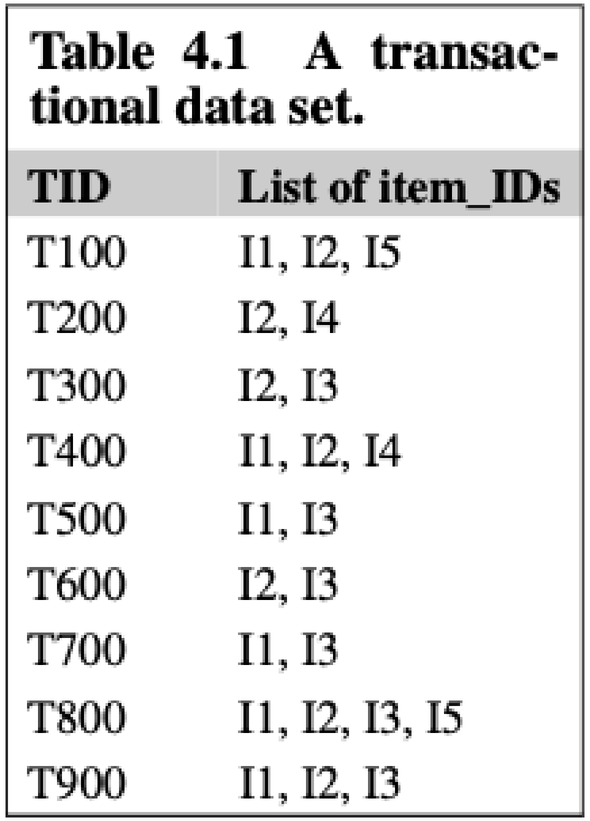
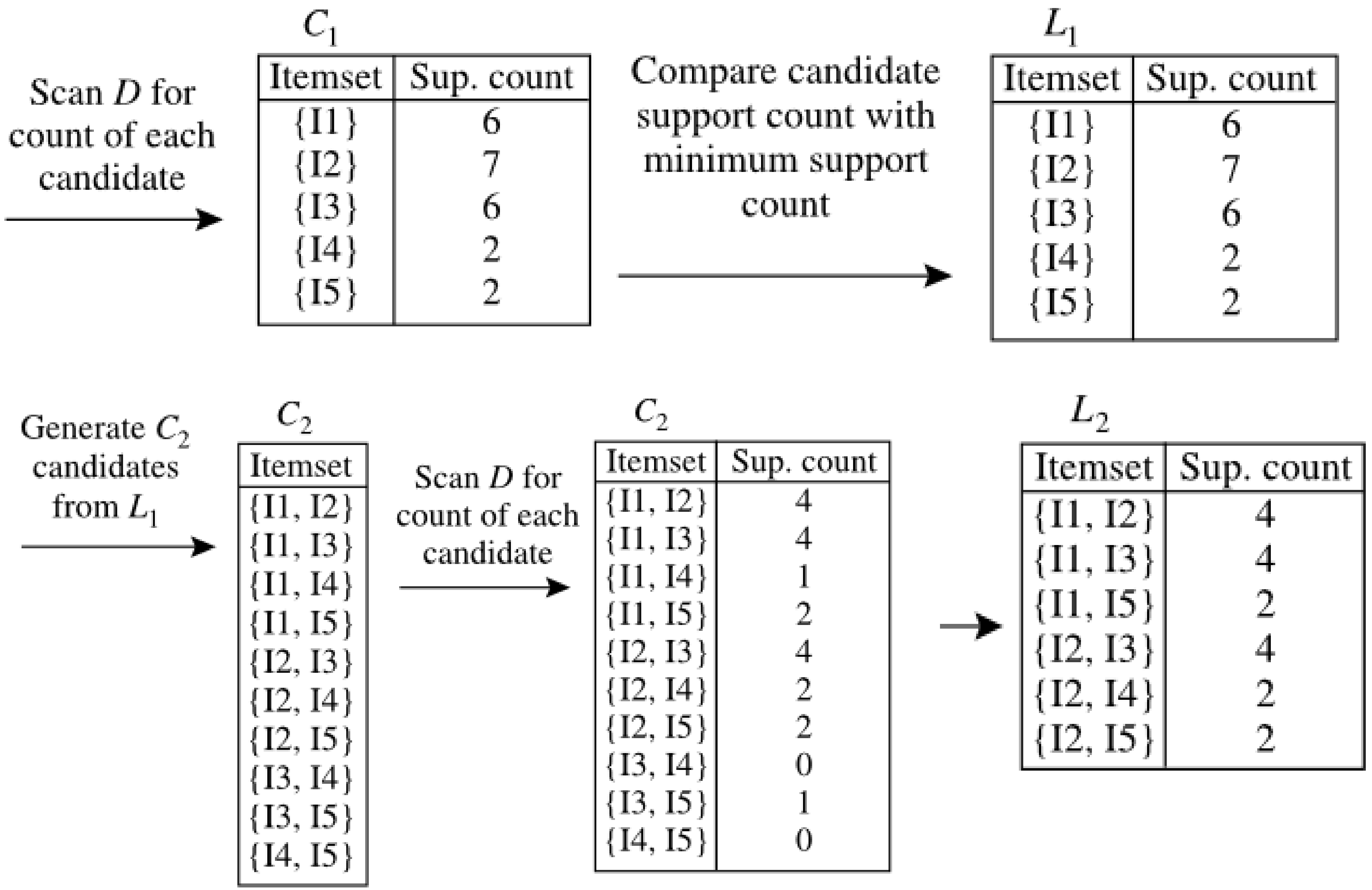


Table 4.1 A transactional data set.

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

$min\_sup = 2$



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

→

L_2

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

$min\_sup = 2$

Apriori Example Continued

Table 4.1 A transactional data set.

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

When we go from L2 to C3, we generate all the possible combinations from the Itemset but then as per the Apriori property, we reject the itemsets in C3 which do have a subset such that the subset is not part of L2.

For Example:

{I1, I3} U {I1, I5} => {I1, I3, I5}

But the subset {I3, I5} is not in L2

Therefore, we reject {I1, I3, I5} and do not include in C3. **This is called pruning**.

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 $\longrightarrow$

C_3

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

Scan D for count of each candidate $\longrightarrow$

C_3

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

Compare candidate support count with minimum support count $\longrightarrow$

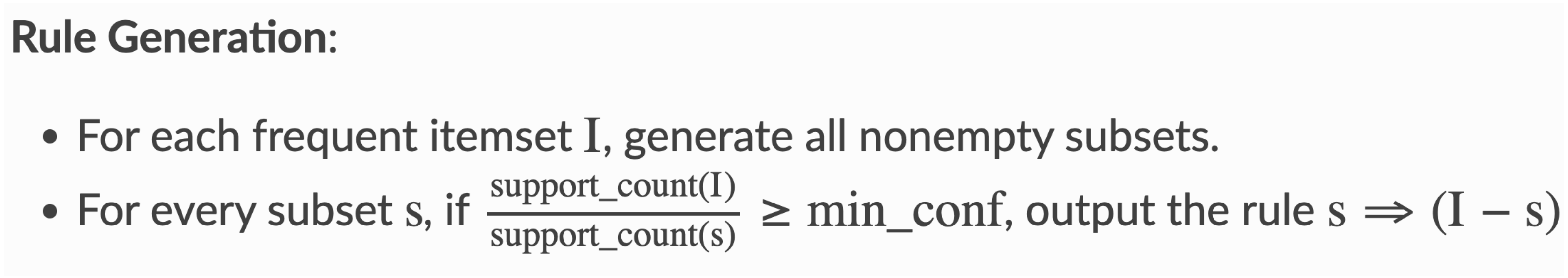
L_3

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

Generating association rules from frequent itemsets

- Strong association rules are derived from frequent itemsets in transaction databases.
- They meet specific minimum support (min_sup) and confidence (min_conf) thresholds.

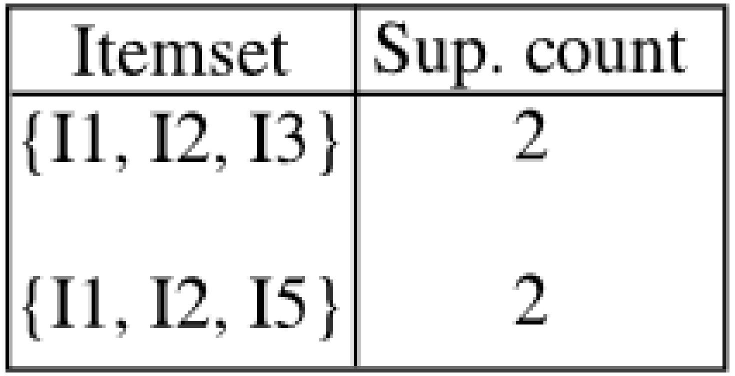
$$confidence\,(A \Rightarrow B) = P(B|A) = \frac{support\_count\,(A \cup B)}{support\_count\,(A)}.$$



Rule Generation:

- For each frequent itemset I, generate all nonempty subsets.
- For every subset s, if $\frac{\mathrm{support\_count(I)}}{\mathrm{support\_count(s)}} \geq \mathrm{min\_conf}$, output the rule $\mathrm{s} \Rightarrow (\mathrm{I} - \mathrm{s})$

Apriori Example Continued



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

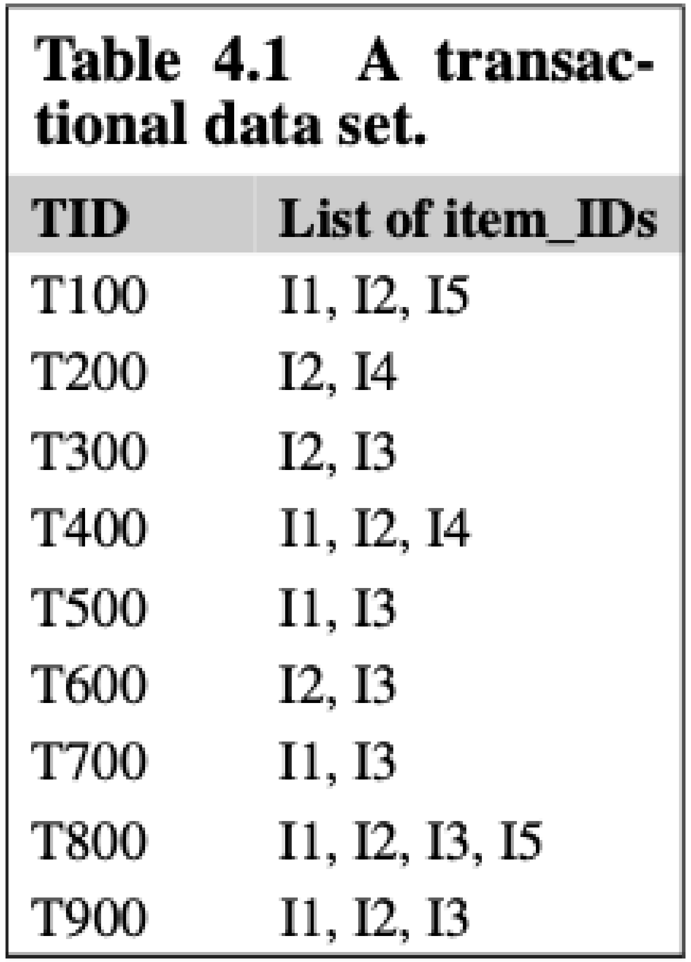


Table 4.1 A transactional data set.

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

- Consider X = {I1, I2, I5}
- What are the association rules that can be generated from X?
- The nonempty subsets of X are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}

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

If minimum confidence = 70%, then only 3 rules make the cut

Approaches to make Apriori faster

- Hash Based Technique
- Transaction Reduction
- Partitioning
- Sampling
- And more…

Hash-based technique

- **Purpose**: To reduce the number of candidate itemsets (like pairs, triples of items) that need to be checked for frequency.
- **Hashing**: This process involves converting itemsets into hash values that point to specific "buckets".
- **Advantages**:
    - It quickly filters out many itemsets that can't be frequent, so you don't waste time checking them.
    - This is particularly helpful when you're looking for 2-itemsets (pairs).

Hash-based technique (continued)

- **Steps**:
  - Start by scanning the database to find frequent single items (1-itemsets).
  - Combine these to form larger itemsets (like 2-itemsets).
  - Apply a hash function to each larger itemset. This function calculates a number based on the items.
  - The resulting number tells you which bucket to put the itemset in a hash table.
- **Bucket Counts**:
  - Each bucket has a count of how many itemsets ended up there.
  - If the bucket count is less than the minimum support count (which is the threshold for an itemset to be considered frequent), all itemsets in that bucket are ignored in the next round.

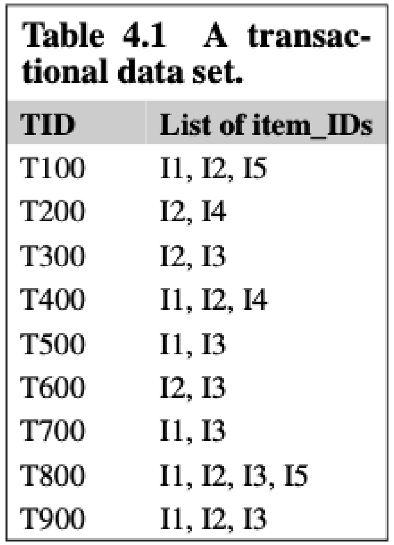
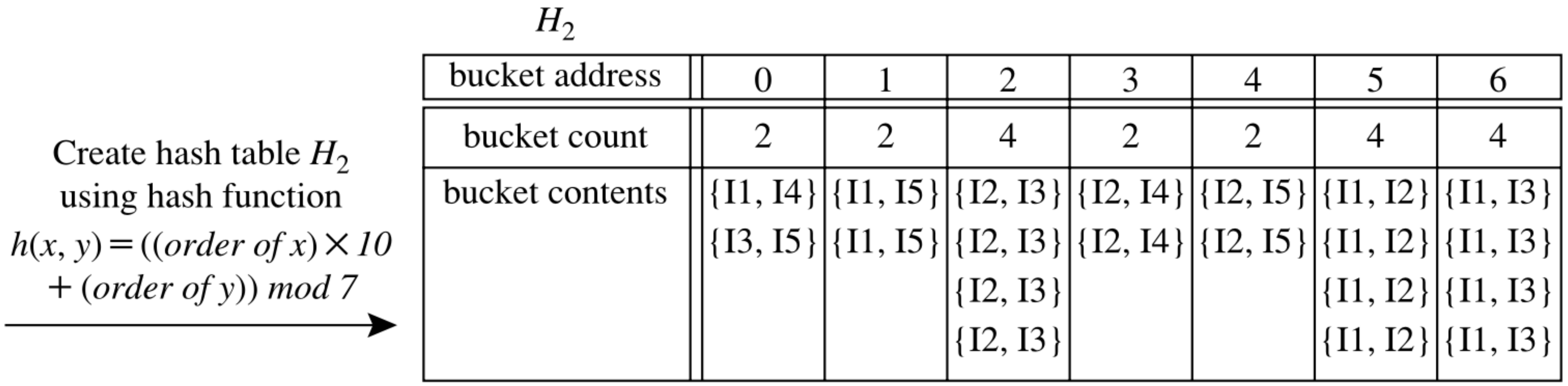


Table 4.1 A transactional data set.

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 $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$

H_2

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

2 Simple but effective methods

Transaction Reduction

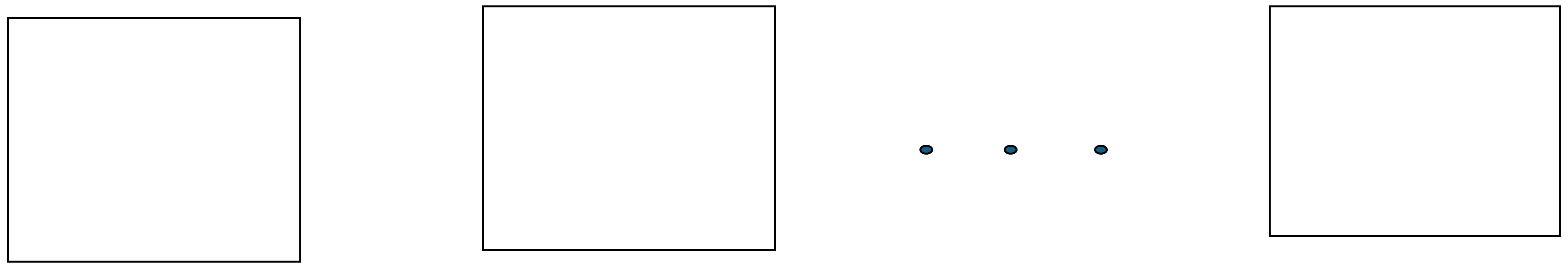
- Ignore transactions without frequent items for future scans.
  - A transaction that does not contain any frequent k-itemsets cannot contain any frequent (k+1)-itemsets.
- This speeds up the search for larger itemsets.
- It cuts down the amount of data to process.

Sampling

- Taking a smaller, random part (sample) of the data to find frequent itemsets quickly.
- This smaller size lets you work faster but might miss some frequent itemsets.
- To minimize misses, a lower support threshold is used on the sample.
- This method is good for quick, repeated analysis when full accuracy isn't critical

Partitioning based approach

- Theorem: *Any itemset that is potentially frequent in TDB must be frequent in at least one of the partitions of TDB*

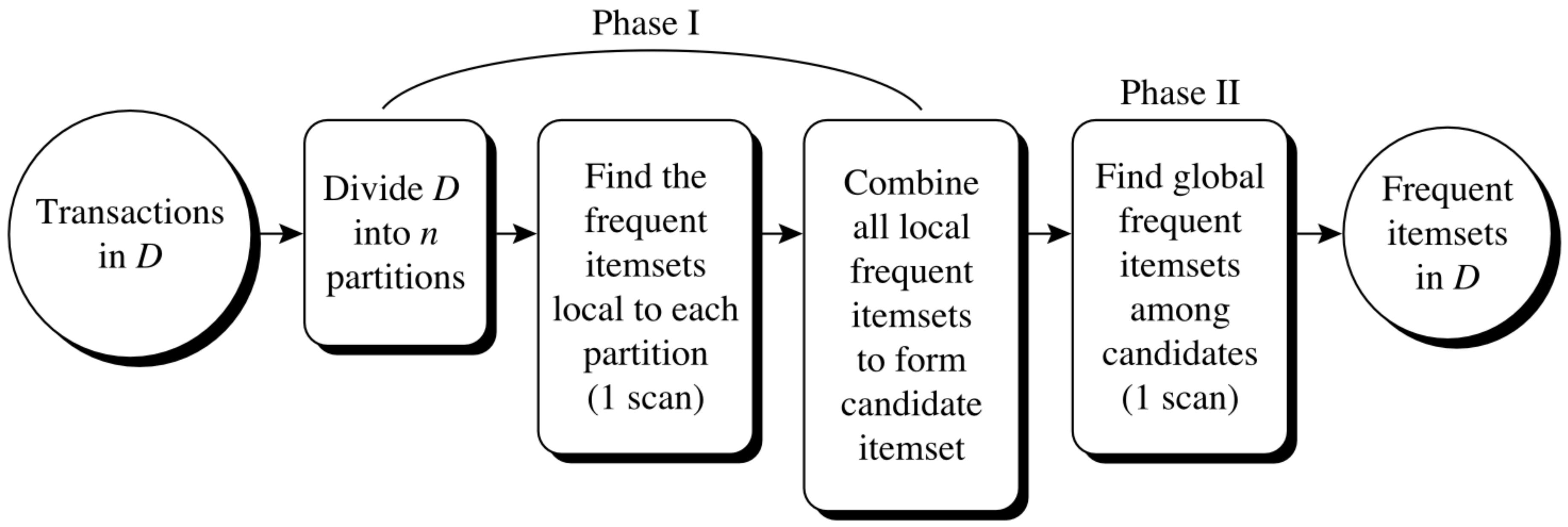


$$\begin{array}{ccccccccc} TDB_1 & + & TDB_2 & + & \cdots & + & TDB_k & = & TDB \\ sup_1(X) < \sigma|TDB_1| & & sup_2(X) < \sigma|TDB_2| & & \cdots & & sup_k(X) < \sigma|TDB_k| & & sup(X) < \sigma|TDB| \end{array}$$

- Method: Scan DB twice (A. Savasere, E. Omiecinski and S. Navathe, *VLDB'95*)
  - Scan 1: Partition database so that each partition can fit in main memory (why?)
    - Mine local frequent patterns in this partition
  - Scan 2: Consolidate global frequent patterns
    - Find global frequent itemset candidates (those frequent in at least one partition)
    - Find the true frequency of those candidates, by scanning TDB_i one more time

Partitioning based approach

- Partitioning is a two-phase technique for mining frequent itemsets:
- **Phase I**:
    - Split the dataset *D* into *n* parts.
    - Find frequent itemsets in each part (only needs one scan per part).
    - Combine these to form a global candidate itemset list.
- **Phase II**:
    - Do one more scan of *D* to identify truly frequent itemsets from the global candidates numbers.


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