

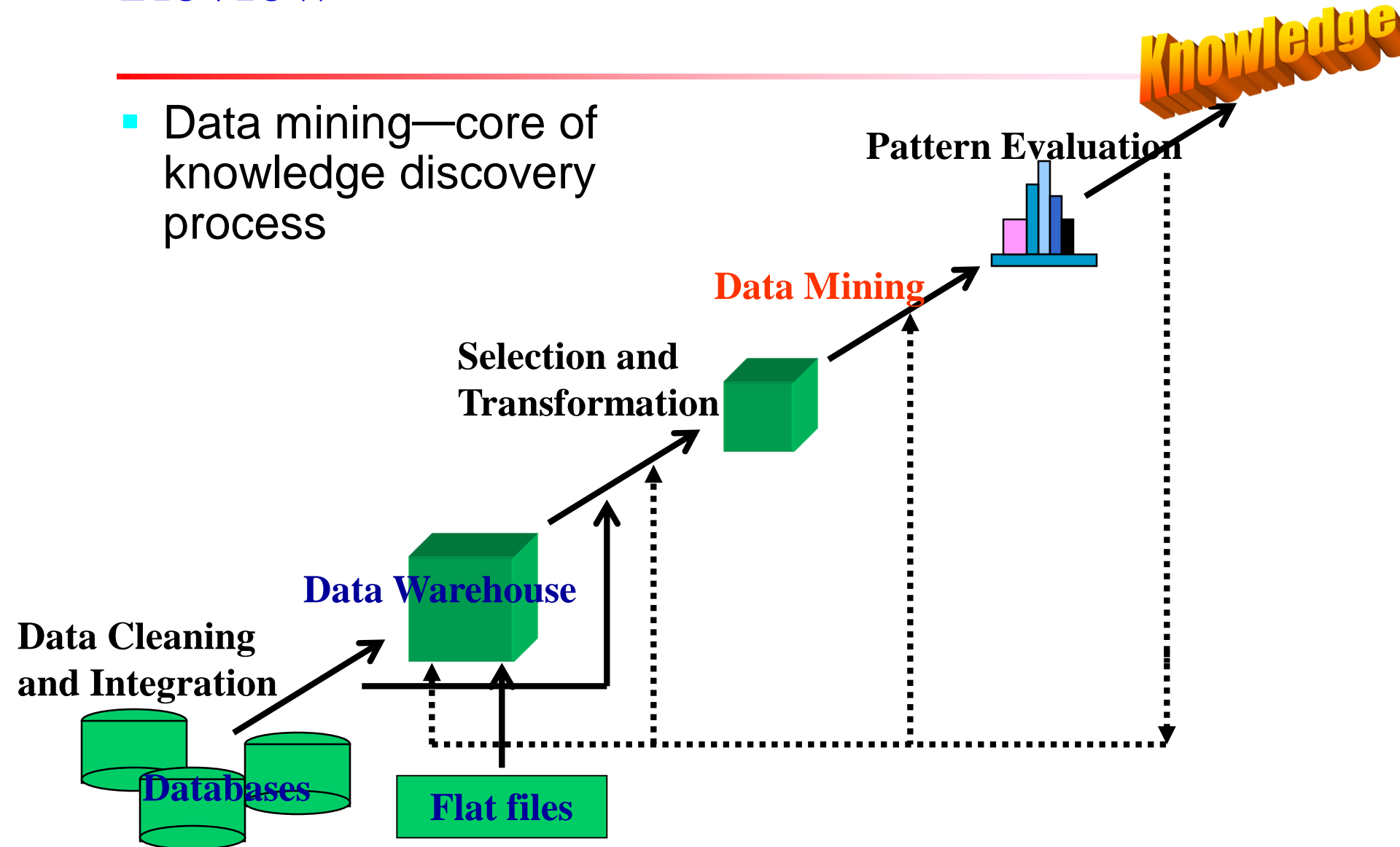
Data Mining

Ying Liu, Prof., Ph.D

University of Chinese Academy of Sciences

Review

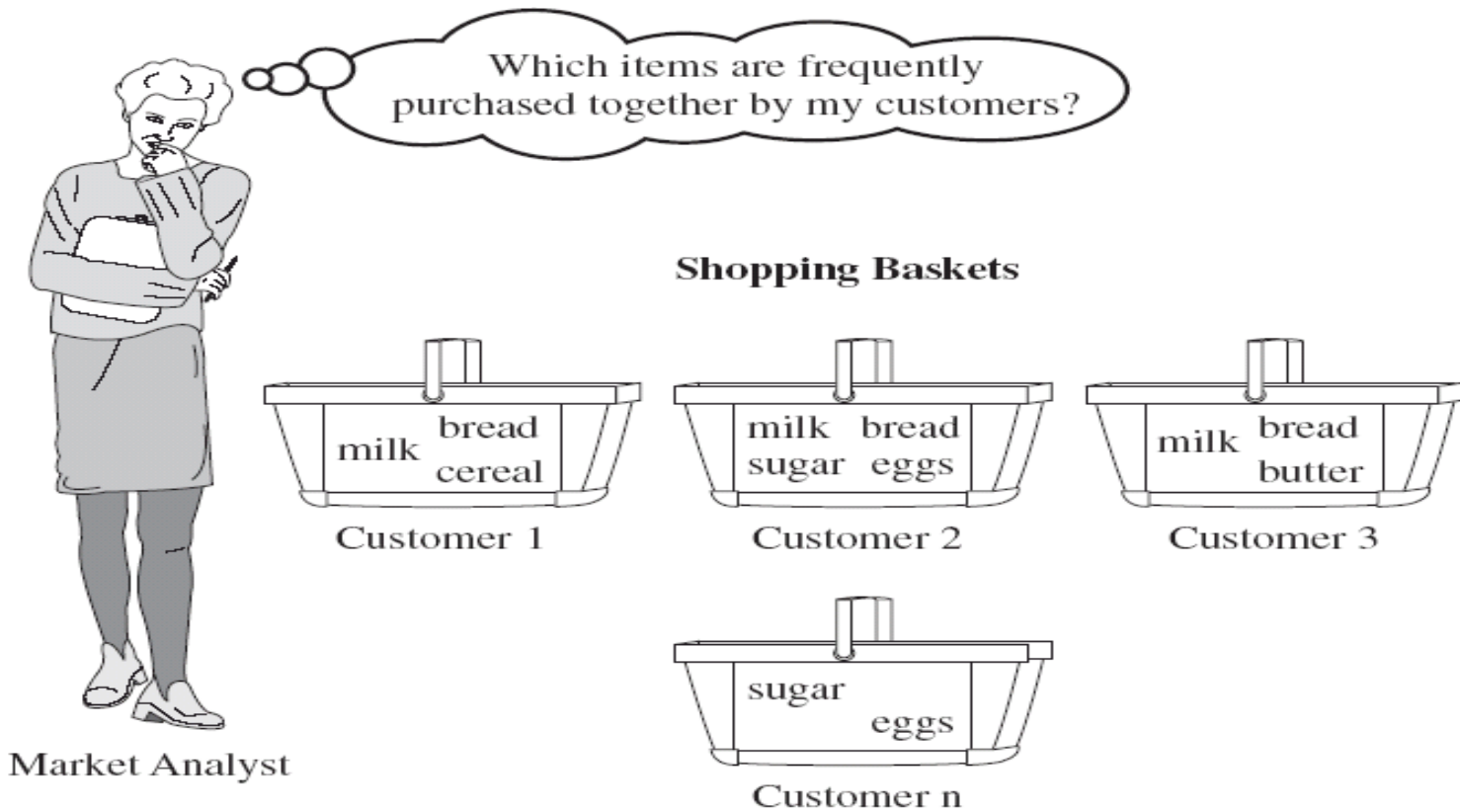
- Data mining—core of knowledge discovery process



Mining Association Rules in Large Databases

- Basic concepts and a road map
- Mining single-dimensional Boolean association rules
- Mining multilevel association rules
- Mining multidimensional association rules
- Summary

Market Basket Analysis



What Is Association Rules Mining?

■ Association rules mining

- Finding frequent patterns, associations among sets of items or objects in transaction databases, relational databases, and other information repositories.

■ Examples

- What products were often purchased together? — Beer and diapers?!
- What DNA segments often occur together in DNA sequences?

What Is Association Rules Mining?

- Where does the data come from?
 - supermarket transactions, membership cards, discount coupons, customer complaint calls
- Applications
 - Basket data analysis
 - Cross-marketing
 - Catalog design
 - Sale campaign analysis
 - Web log (click stream) analysis
 - DNA sequence analysis

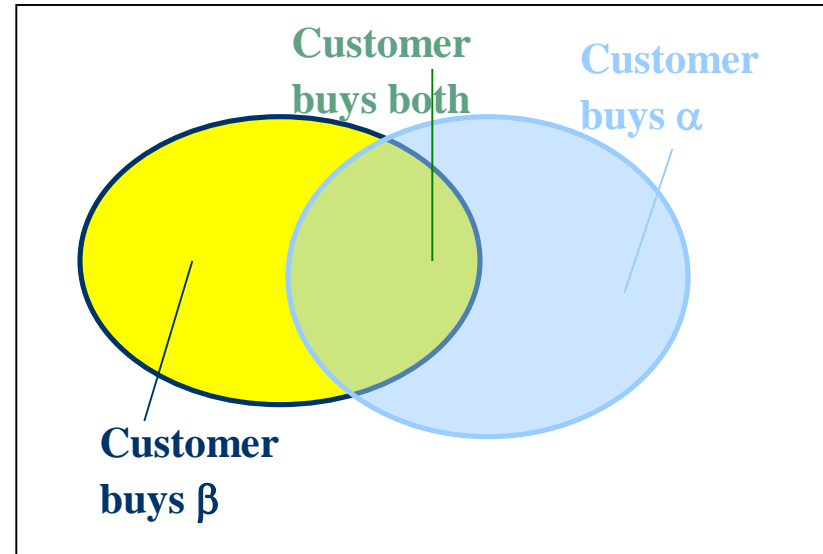
Basic Concepts

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

- Item collection $X = \{x_1, \dots, x_m\}$
- Itemset: a set of items, k -itemset
- Transaction $T \subseteq X$, each T associates a unique Tid and items bought by a customer
- Rule form $\alpha \Rightarrow \beta$, $\alpha \subset X$, $\beta \subset X$, $\alpha \cap \beta = \emptyset$

Basic Concepts

- support, s , probability that a transaction contains α and β
 - support $(\alpha \Rightarrow \beta) = P(\alpha \cap \beta)$
- Frequent itemset, occurrence greater than a min_support
- Frequent itemset mining, find all the rules $\alpha \Rightarrow \beta$ satisfying min_support
- Let $\text{sup}_{\min} = 50\%$,
frequent Itemsets $\{A:3, B:3, D:4, E:3, AD:3\}$
support $(A) = 3/5 = 60\%$, support $(AD) = 3/5 = 60\%$



Basic Concepts

- confidence, c , conditional probability that a transaction having α also contains β

$$\text{Confidence}(\alpha \Rightarrow \beta) = P(\beta \mid \alpha) = \frac{P(\alpha \cap \beta)}{P(\alpha)} = \frac{\text{count}(\alpha \cap \beta)}{\text{count}(\alpha)}$$

- Measure of rule interestingness
- Rules satisfy min_support and min_confidence are strong
- Let $\text{sup}_{\min} = 50\%$, $\text{conf}_{\min} = 50\%$,
frequent itemsets $\{A:3, B:3, D:4, E:3, AD:3\}$

Association rules:

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

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

Interestingness Measure: Correlations (Lift)

- *play basketball* \Rightarrow *eat cereal* [40%, 66.7%] is misleading
 - The overall % of students eating cereal is 75% > 66.7%.
- Measure of dependent/correlated events: **lift**

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

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(B, C) = \frac{2000 / 5000}{3000 / 5000 * 3750 / 5000} = 0.89 \quad lift(B, \neg C) = \frac{1000 / 5000}{3000 / 5000 * 1250 / 5000} = 1.33$$

Association Rule Mining: A Road Map

- **Boolean vs. quantitative** associations (based on the types of values handled)
 - Boolean association rules, only concern presence or absence of items, $\text{buys}(x, \text{"SQLServer"}) \wedge \text{buys}(x, \text{"DMBook"}) \Rightarrow \text{buys}(x, \text{"DBMiner"})$ [0.2%, 60%]
 - Quantitative association rules, concern quantitative attributes, $\text{age}(x, \text{"30...39"}) \wedge \text{income}(x, \text{"42...48K"}) \Rightarrow \text{buys}(x, \text{"high resolution TV"})$ [1%, 75%]
- **Single level vs. multiple-level** analysis (based on the levels of abstraction involved)
 - $\text{age}(x, \text{"30...39"}) \Rightarrow \text{buys}(x, \text{"laptop computer"})$
 - $\text{age}(x, \text{"30...39"}) \Rightarrow \text{buys}(x, \text{"computer"})$
- **Single dimension vs. multiple dimensional** associations (based on dimensions involved)

Mining Association Rules in Large Databases

- Basic concepts and a road map
- Mining single-dimensional Boolean association rules
- Mining multilevel association rules
- Mining multidimensional association rules
- Summary

Handling Exponential Complexity

- Given n transactions and m different items:
 - Number of possible association rules: $O(2^m)$
 - Computation complexity: $O(nm2^m)$
- **Apriori Principle**
 - Collect single item counts, find large items
 - Find candidate pairs, count them => large pairs of items
 - Find candidate triplets, count them => large triplets of items, And so on...
 - **Guiding Principle: Every subset of a frequent itemset has to be frequent**
 - Used for pruning many candidates

Apriori: A Candidate Generation-and-Test Approach

- **Apriori** uses prior knowledge of frequent itemsets
- Iterative approach, level-wise search
- The Apriori property (**downward closure property**, **anti-monotone**) of frequent patterns
 - Any subset of a frequent itemset must be frequent
 - If any itemset is infrequent, its superset should not be generated/tested
 - If {beer, diaper, nuts} is frequent, so is {beer, diaper}, every transaction having {beer, diaper, nuts} also contains {beer, diaper}
 - If {beer, diaper} is infrequent, {beer, diaper, nut} cannot be frequent at all

Apriori: A Candidate Generation-and-Test Approach

■ Method:

- Initially, scan DB once to get frequent 1-itemset
- **Generate** length $(k+1)$ **candidate** itemsets from length k **frequent** itemsets
- **Test** the candidates against DB
- Terminate when no frequent or candidate set can be generated

Apriori Algorithm — An Example

$\text{Sup}_{\min} = 2$

Database D

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

L_2

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

C_2

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

2nd scan

C_2

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

C_3

Itemset	sup
{B, C, E}	2

3rd scan

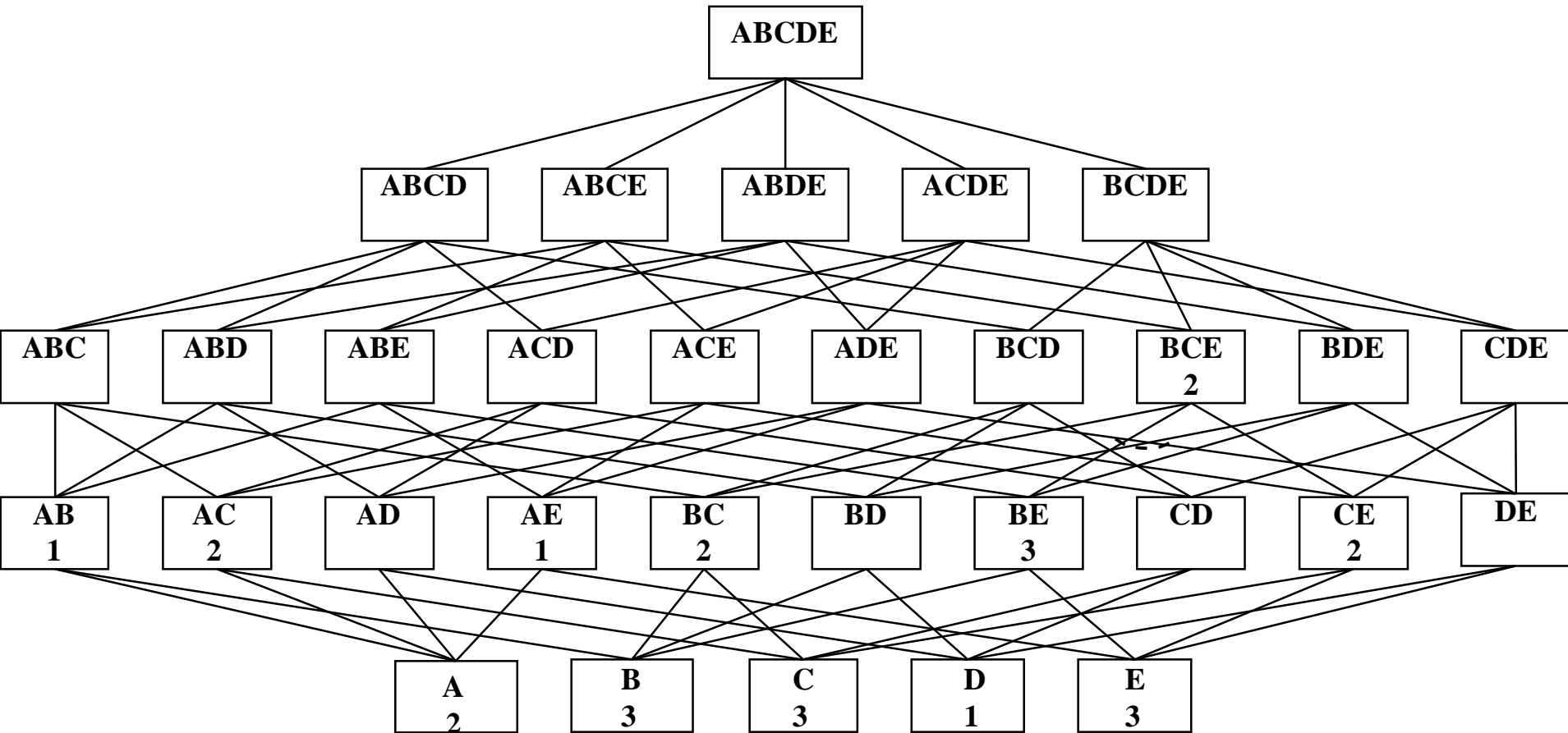
C_3

Itemset	sup
{B, C, E}	2

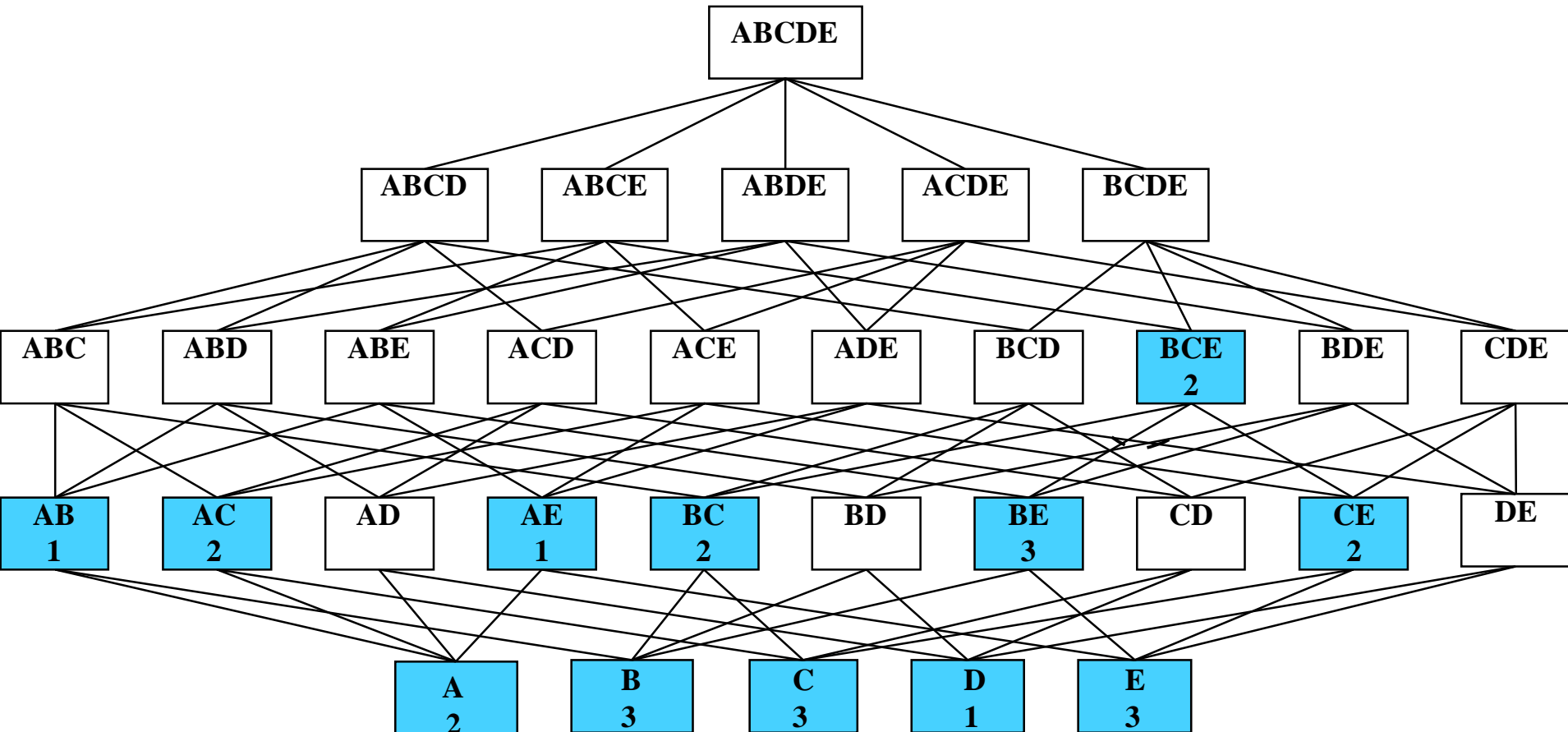
L_3

Itemset	sup
{B, C, E}	2

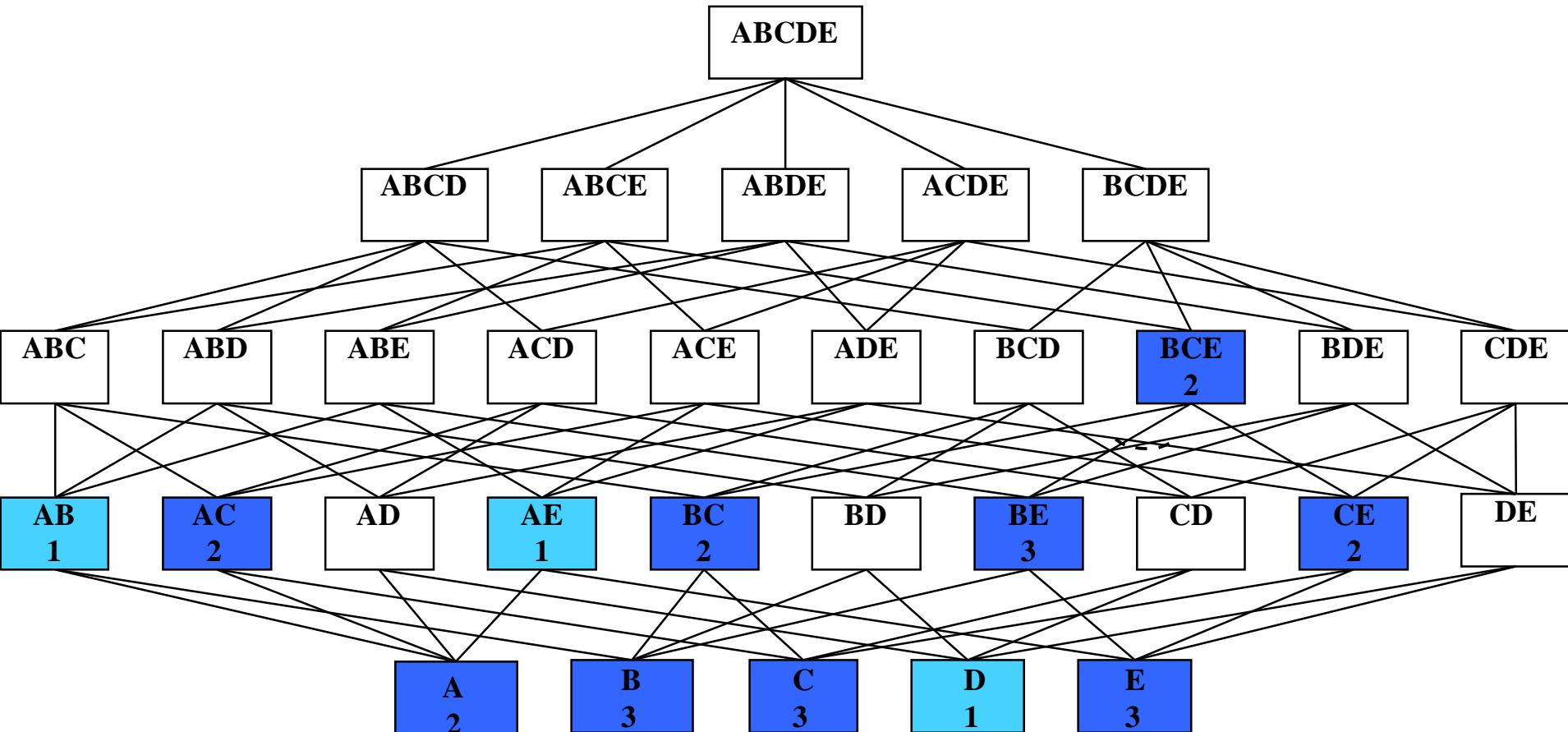
Apriori Algorithm — An Example



Apriori Algorithm — An Example



Apriori Algorithm — An Example



Apriori Algorithm

■ Pseudo-code

C_k : Candidate itemset of size k

L_k : frequent itemset of size k

Input: Database D , min_sup

Output: frequent itemsets L

$L_1 = \{\text{frequent single items from } D\};$

for ($k = 2$; $L_{k-1} \neq \emptyset$; $k++$) **do begin**

$C_k =$ candidates generated from L_{k-1} ;

for each transaction $t \in D$ **do**

increment the count of all candidates in C_k which are
contained in t

end

$L_k =$ candidates in C_k with $min_support$

end

return $L = \cup_k L_k$;

How to Generate Candidates?

■ How to generate candidates?

- Step 1: self-joining L_k
- Step 2: pruning

■ Example

- $L_3 = \{abc, abd, acd, ace, bcd\}$
- Self-joining: $L_3 * L_3$
 - abc and $abd \rightarrow abcd$, acd and $ace \rightarrow acde$
- Pruning:
 - $acde$ is pruned because ade is not in L_3
- $C_4 = \{abcd\}$

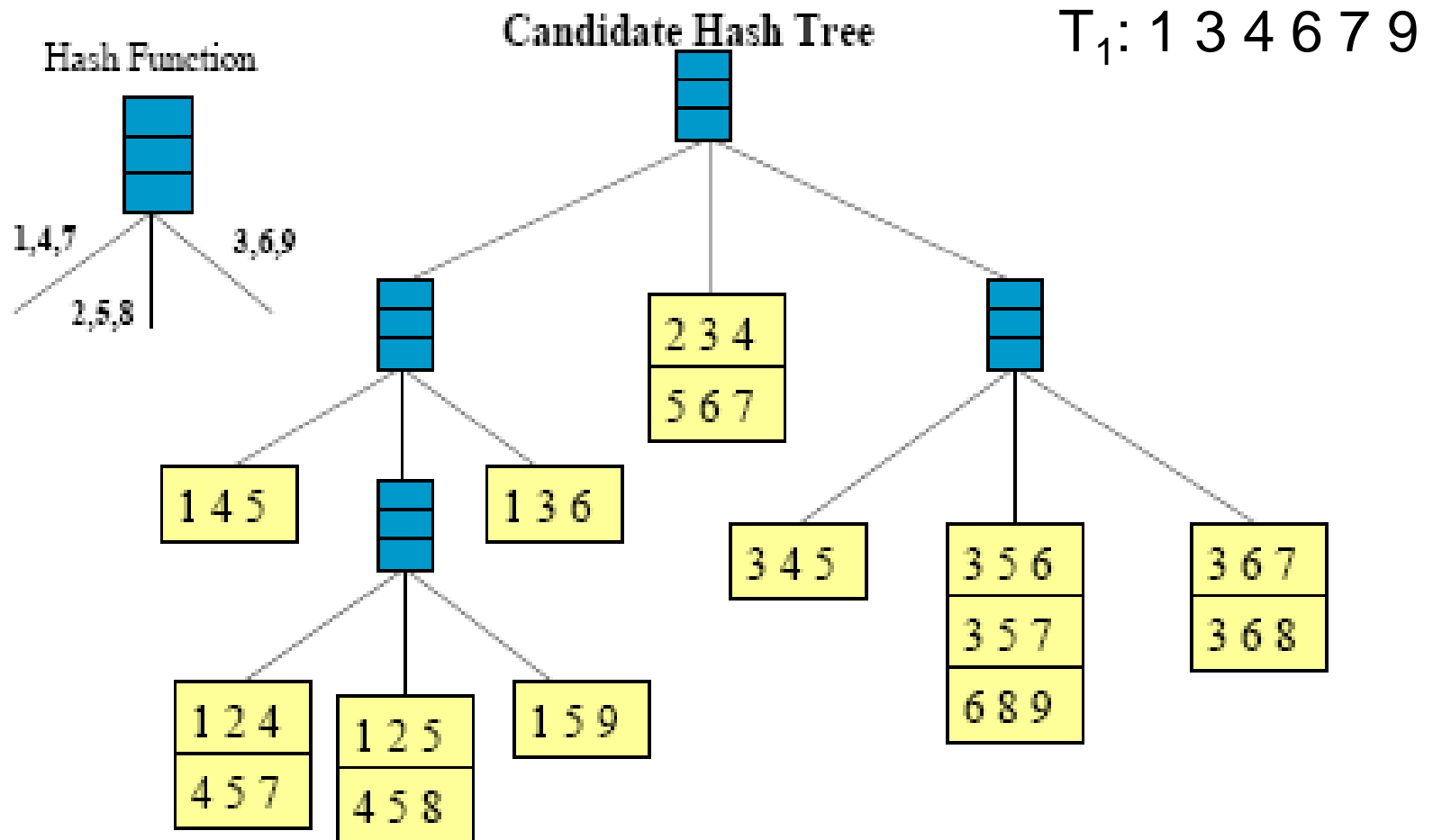
How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in order
- Step 1: self-joining L_{k-1}
 - for each itemset $l_1 \in L_{k-1}$
 - for each itemset $l_2 \in L_{k-1}$
 - if $(l_1[1]=l_2[1]) \wedge (l_1[2]=l_2[2]) \wedge \dots \wedge (l_1[k-2]=l_2[k-2])$ then
 - $c = l_1 \text{ join } l_2$
 - pruning (c)
 - end
- end
- Step 2: pruning
 - forall $(k-1)$ -subsets s of c do
 - if (s is not in L_{k-1}) then delete c

How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
 - The total number of candidates can be very huge
 - One transaction may contain many candidates
- Method:
 - Candidate itemsets are stored in a *hash-tree*
 - *Leaf node* of hash-tree contains a list of itemsets and counts
 - *Interior node* contains a hash table

Example: Counting Supports of Candidates



Exercise

1. A database has 9 transactions. Let $min_sup = 20\%$. Please present all the candidates and frequent itemsets at each iteration.

TID	List of items_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

Challenges of Frequent Pattern Mining

■ Challenges

- Multiple scans of transaction database
- Huge number of candidates
- Tedious workload of support counting for candidates

■ Improving Apriori

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

Partition: Scan Database Only Twice

- A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In *VLDB'95*.
- Partitioning technique
 - Partition the data into N small partitions
 - **Phase 1**: find local frequent itemsets on each data partition. Record all local frequent itemsets.
 - **Phase 2**: Integrate all local frequent itemsets, scan database, find global frequent itemsets.
- **Correctness**: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions

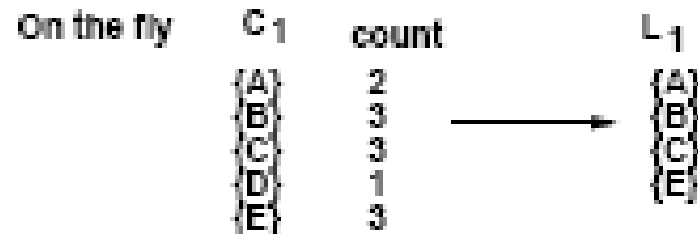
Partition: Scan Database Only Twice

- Each partition can be fit into memory
- Scan database only twice! Reduce I/O cost!
- Execution time scales linearly
- Good for very large-scale database
- Applicable to parallel/distributed computing systems
 - Each processor performs FIM on its local data
 - Central server aggregates local frequent itemsets, broadcast potential global itemsets
 - Each processor scans local data to count the frequency
 - Central server aggregates the counts, find the global itemsets

DHP: Reduce the Number of Candidates

- J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*
- Hash-based technique
 - When scanning transactions to generate frequent k -itemsets, L_k , generate all $(k+1)$ -itemsets for each transaction
 - Hash all $(k+1)$ -itemsets into buckets, increase bucket count
 - If a $(k+1)$ -itemset bucket count is below min_sup , it must be removed from $(k+1)$ candidate itemsets, C_{k+1}
- **Correctness:** A k -itemset whose corresponding hashing bucket count is below the threshold cannot be frequent

DHP: Reduce the Number of Candidates



minimum support, $s = 2$

Making a hash table

100 $\{A\ C\}, \{A\ D\}, \{C\ D\}$
 200 $\{B\ C\}, \{B\ E\}, \{C\ E\}$
 300 $\{A\ B\}, \{A\ C\}, \{A\ E\}, \{B\ C\}, \{B\ E\}, \{C\ E\}$
 400 $\{B\ E\}$

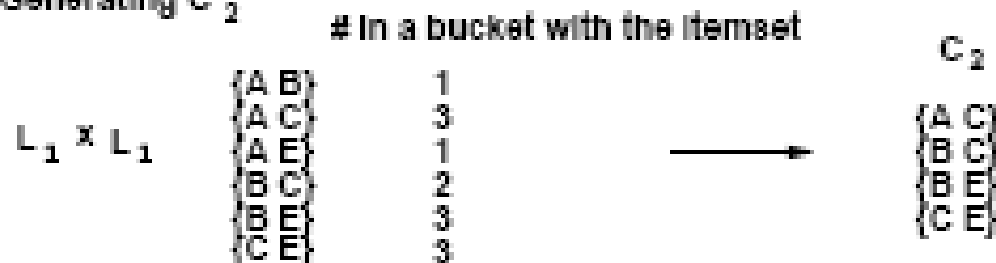
$$h(\{x\ y\}) = ((\text{order of } x) * 10 + (\text{order of } y)) \bmod 7;$$

$\begin{Bmatrix} \{C\ E\} \\ \{C\ E\} \\ \{A\ D\} \end{Bmatrix}$	$\{A\ E\}$	$\begin{Bmatrix} \{B\ C\} \\ \{B\ C\} \end{Bmatrix}$		$\begin{Bmatrix} \{B\ E\} \\ \{B\ E\} \\ \{B\ E\} \end{Bmatrix}$	$\{A\ B\}$	$\begin{Bmatrix} \{A\ C\} \\ \{C\ D\} \\ \{A\ C\} \end{Bmatrix}$
3	1	2	0	3	1	3
0	1	2	3	4	5	6

Hash table H_2
Hash address

The number of items hashed to bucket 2

Generating C_2



DHP: Reduce the Number of Candidates

■ Pros

- Reduce the number of candidates, C_k , especially for C_2 . Size of C_2 is usually huge, reduce C_2 is crucial
- Execution time scales linearly when varying the size of data

Comparison of time (T15.I4.D100)

	Apriori number	DHP number
L_1	820	820
C_2	335,790	338
L_2	207	207
C_3	618	618
L_3	201	201
C_4	184	184
L_4	98	98
C_5	30	30
L_5	23	23
C_6	1	1
L_6	1	1
total time	39.39	13.91

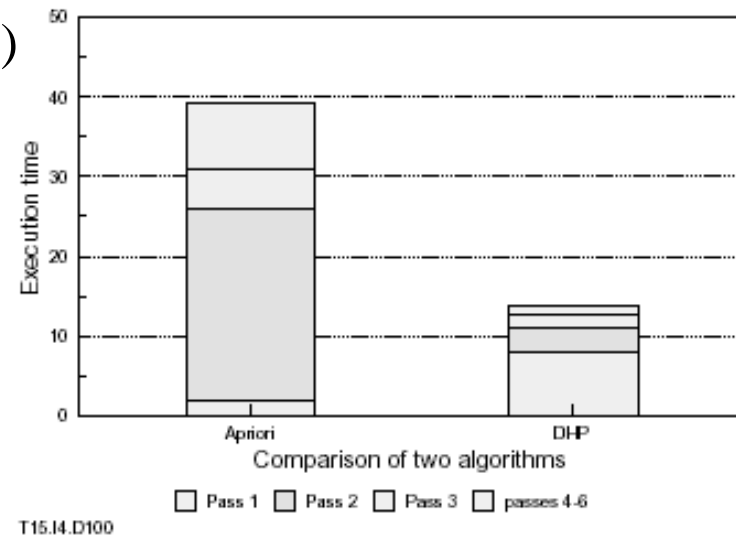
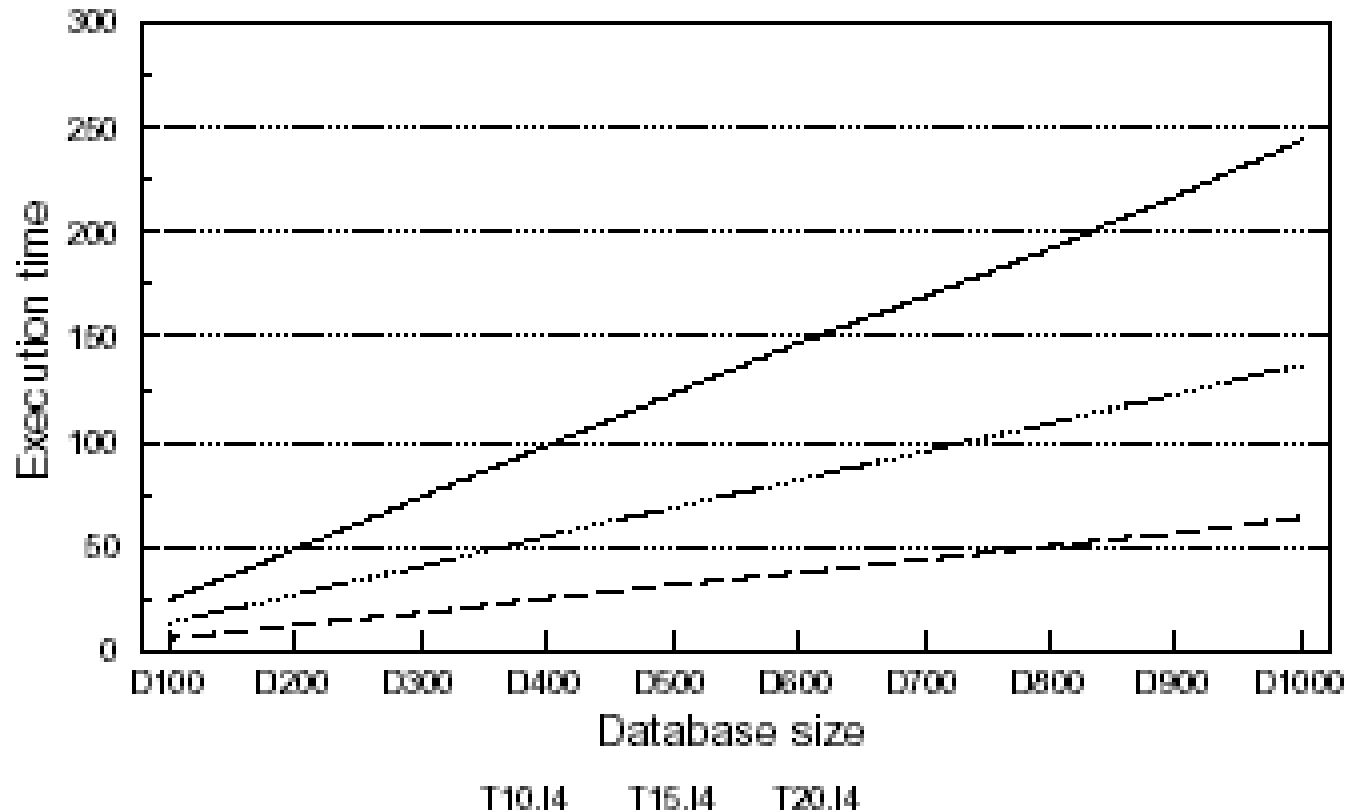


Figure 8: Execution time of Apriori and DHP

Comparison of time (T15.I4.D100)

DHP: Reduce the Number of Candidates



Performance of DHP when increasing the size of database

DHP: Reduce the Number of Candidates

■ Cons

- Consume more memory, for hash table
- The larger the hash table, the smaller C_k and L_k

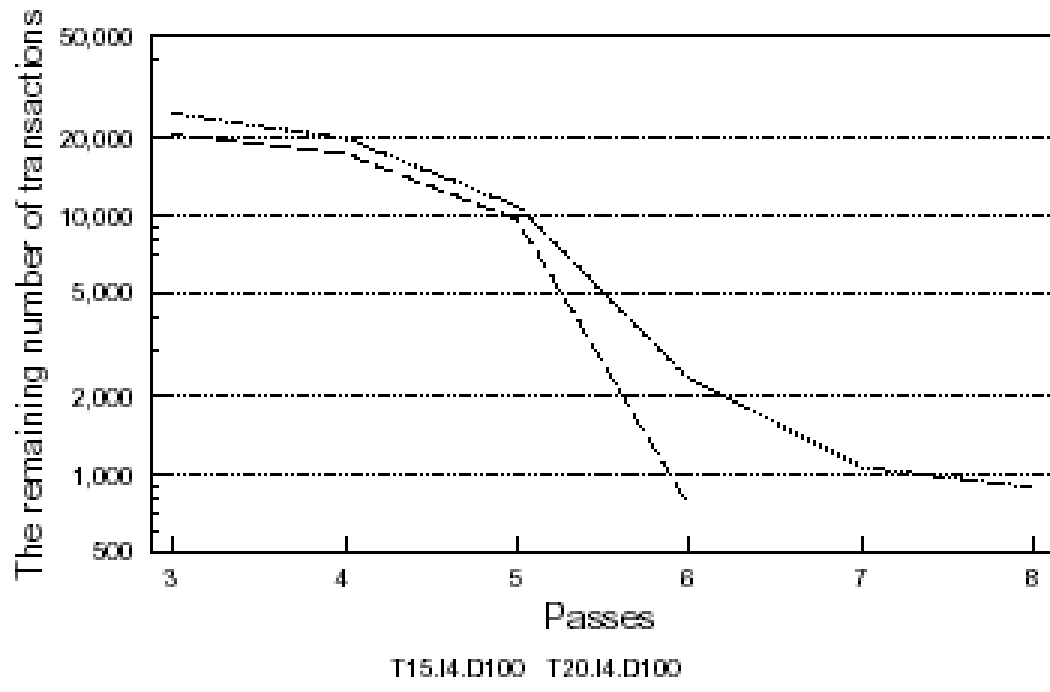
Results from varying hash table sizes
(T10.I4.D100)

$ H_2 $	524,288	262,144	131,072	95,536	32,768
L_1	559	559	559	559	559
$ \{H_2 \geq s\} $	58	61	75	96	182
C_2	81	120	199	394	1355
L_2	45	45	45	45	45
α	0.0314	0.0320	0.0345	0.0386	0.0545
size of D_3	498KB	500KB	507KB	539KB	603KB
$ D_3 $	19,732	19,741	19,755	20,501	21,607
total time	6.44	6.43	6.24	6.77	7.23

Transaction Reduction

- J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*
- Transaction reduction
 - When scanning transactions to generate frequent k -itemsets, L_k , mark the transaction that contains no k -candidate
 - Remove all the marked transaction
 - The number of transactions drops dramatically

Transaction Reduction



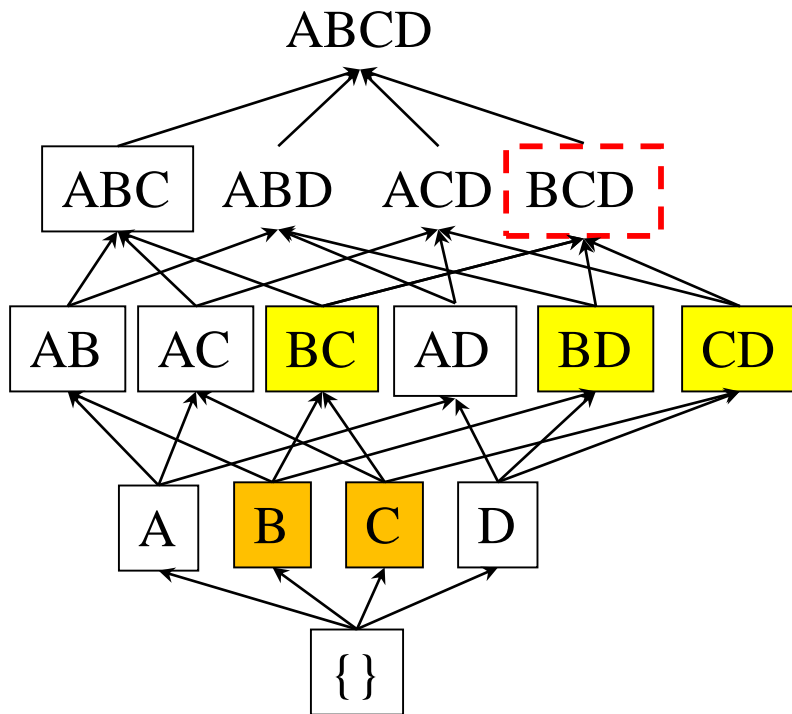
The number of original tx's: 100,000
 $s=0.75\%$

The remaining number of transaction in each pass

DIC: Reduce Number of Scans

- S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*
- *Sergey Brin, founder of Google!*
- Partition database into blocks marked by starting points
- New candidate can be added at any starting point once all its subsets are determined frequent
- Reduce the number of database scans

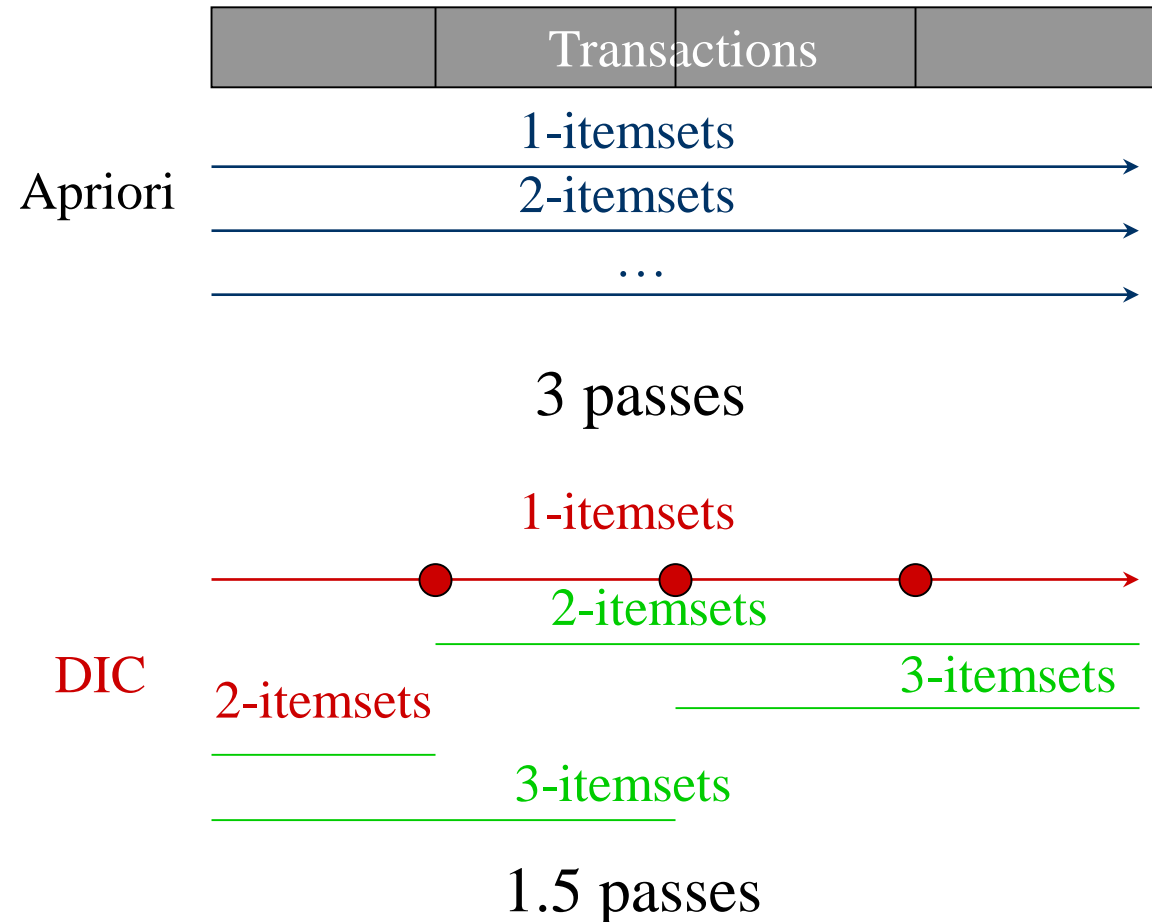
DIC: Reduce Number of Scans



Itemset lattice

- Once both B and C are determined frequent, new candidate BC is added, the counting of BC begins at the next starting point
- Once all length-2 subsets of BCD are determined frequent, new candidate BCD is added, the counting of BCD begins at the next starting point

DIC: Reduce Number of Scans



- Assume 40000 transactions, 4 partitions
- Begin counting 2-itemsets after the first 10000 have been read
- Begin counting 3-itemsets after the first 20000 have been read
- Scan database again, count 2 and 3-itemsets
- After 10000 transactions, finish counting 2-itemsets
- After 20000 transactions, finish counting 3-itemsets

Exercise

2. A database has 9 transactions. Let $min_sup = 20\%$. Please present all the frequent itemsets generated by DIC in the first iteration. (Note: partition the data into 3 blocks)

TID	List of items_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

Bottleneck of Frequent-pattern Mining

- Multiple database scans are **costly**
- Mining long patterns needs many passes of scanning and generates lots of candidates
 - To find frequent itemset $i_1 i_2 \dots i_{100}$
 - # of scans: **100**
 - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \dots + \binom{100}{100} = 2^{100} - 1 =$
 1.27×10^{30} !
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

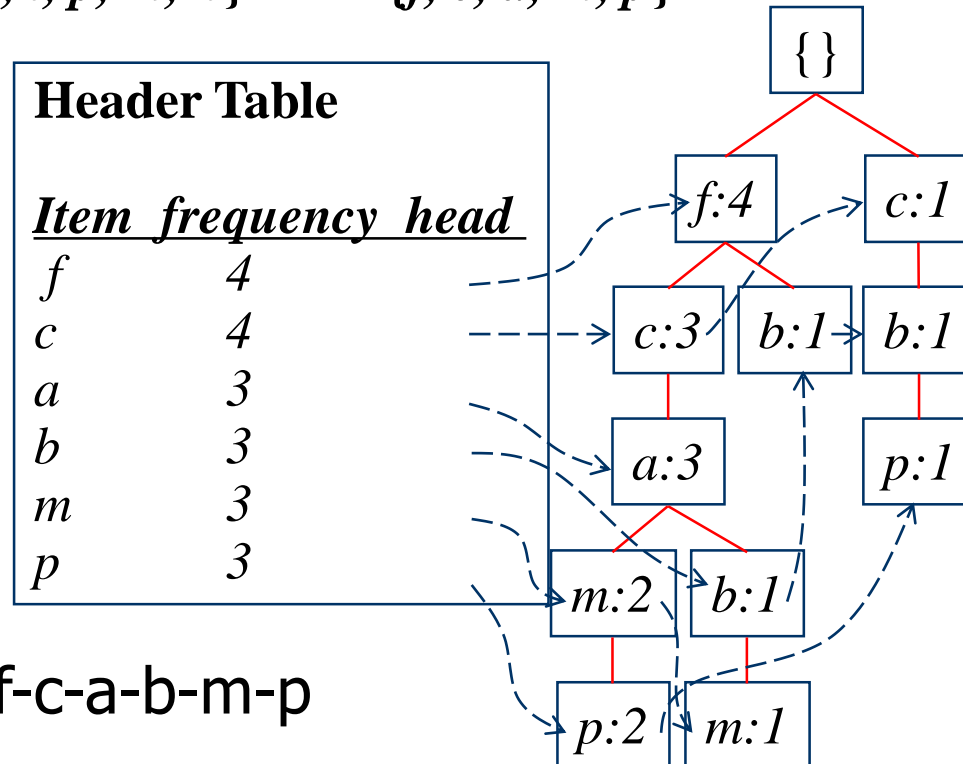
Construct FP-tree from a Transaction Database

- Scan DB once, find frequent 1-itemset (single item pattern)
- Sort frequent items in frequency descending order L
- Create the root of the tree, labeled with “null”
- Scan DB again, sort each transaction in L order, a branch is created for each transaction
 - Increment the count of each node along a common prefix by 1
 - Create nodes for the items following the prefix
- Build a header table, connect each item point in the tree

Construct FP-tree from a Transaction Database

<i>TID</i>	<i>Items bought</i>	<i>(ordered) frequent items</i>
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3



F-list = f-c-a-b-m-p

Construct FP-tree from a Transaction Database

- 1. Scan the transaction database D once. Collect F , the set of frequent items, and their support counts. Sort F in support count descending order as L , the *list* of frequent items.
- 2. Create the root of an FP-tree, and label it as “null.” For each transaction $Trans$ in D do the following:
 - Select and sort the frequent items in $Trans$ according to the order of L . Let the sorted frequent item list in the $Trans$ be $[p|P]$, where p is the first element and P is the remaining list.
 - Call `insert_tree` ($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item-name = p.item-name$, then increment N 's count by 1; else create a new node N , and let its count be 1, its parent link be linked to T , and its node-link to the nodes with the same *item-name* via the node-link structure.
 - If P is nonempty, call `insert_tree`(P , N) recursively.

Benefits of the FP-tree Structure

■ Completeness

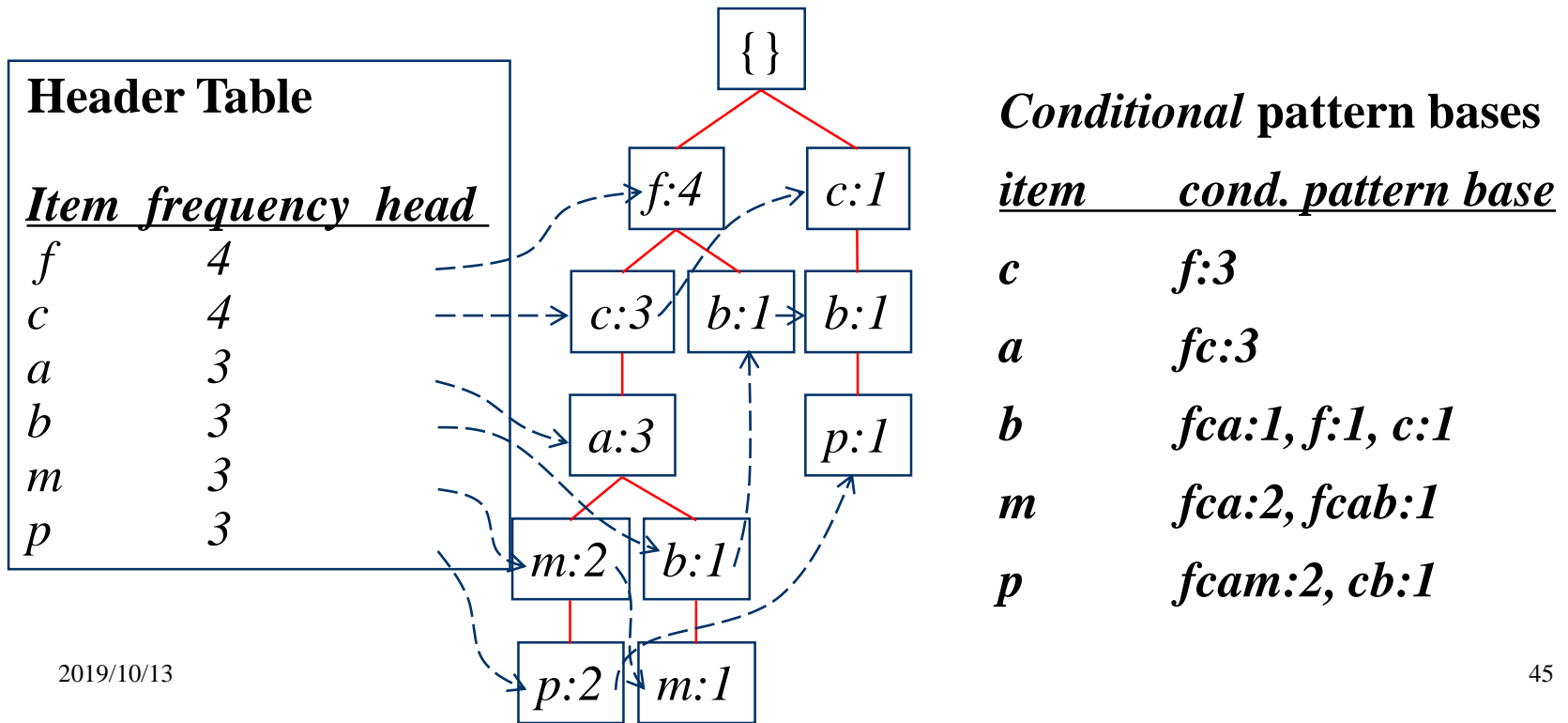
- Preserve complete information for frequent pattern mining
- Never break a long pattern of any transaction

■ Compactness

- Reduce irrelevant info—infrequent items are gone
- Items in frequency descending order: the more frequently occurring, the more likely to be shared
- Never be larger than the original database

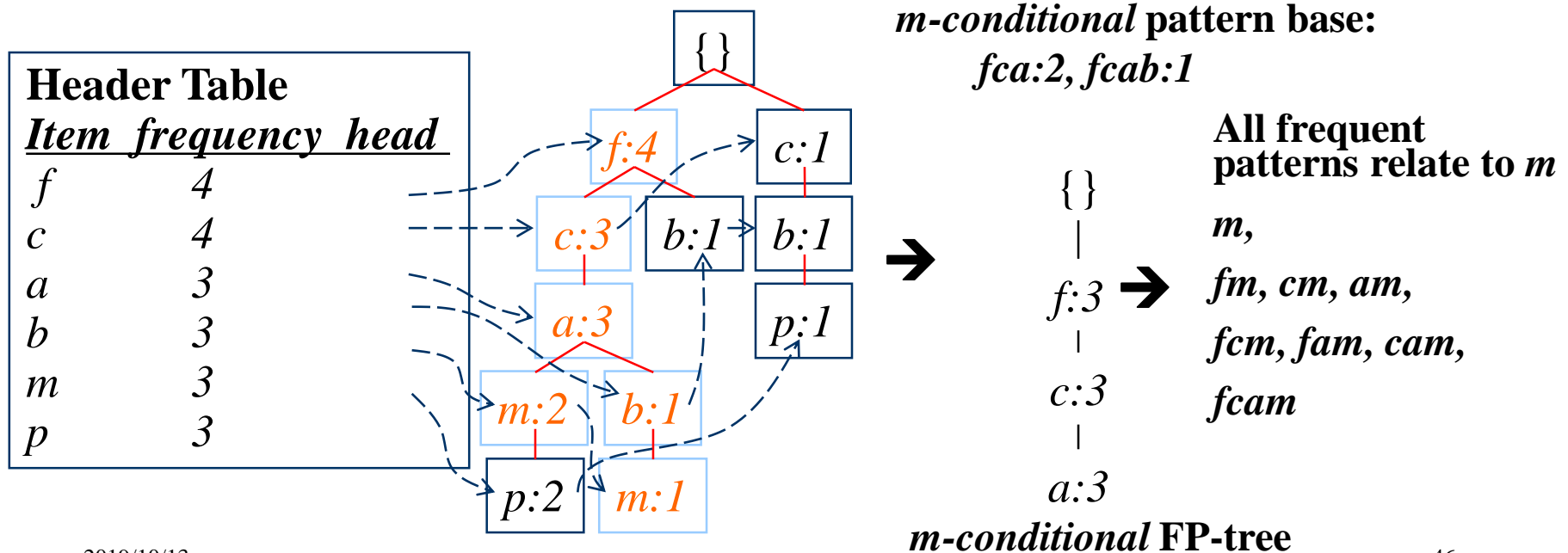
Construct Conditional Pattern Base

- Start at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item x
- Accumulate all of *transformed prefix paths* of item x into form x 's conditional pattern base

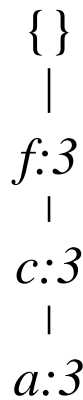


From Conditional Pattern-bases to Conditional FP-trees

- For each pattern-base
 - Accumulate the count for each item in the base
 - Construct the FP-tree for the frequent items of the pattern base

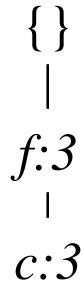


Recursion: Conditional FP-tree



m-conditional FP-tree

Cond. pattern base of “am”: (fc:3)



am-conditional FP-tree

Cond. pattern base of “cm”: (f:3)



cm-conditional FP-tree

Cond. pattern base of “cam”: (f:3)



cam-conditional FP-tree

Mining Frequent Patterns With FP-trees

procedure **FP_growth**($Tree, \alpha$)

- (1) **if** $Tree$ contains a single path P then
- (2) **for each** combination (denoted as β) of the nodes in the path P
- (3) generate pattern $\beta \cup \alpha$ with *support_count* = *minimum support count of nodes in β* ;
- (4) **else for each** a_i in the header of $Tree$ {
- (5) generate pattern $\beta = a_i \cup \alpha$ with *support_count* = $a_i.support_count$;
- (6) construct β 's conditional pattern base and then β 's conditional FP_tree $Tree_\beta$;
- (7) **if** $Tree_\beta$ then
- (8) call **FP_growth**($Tree_\beta, \beta$); }

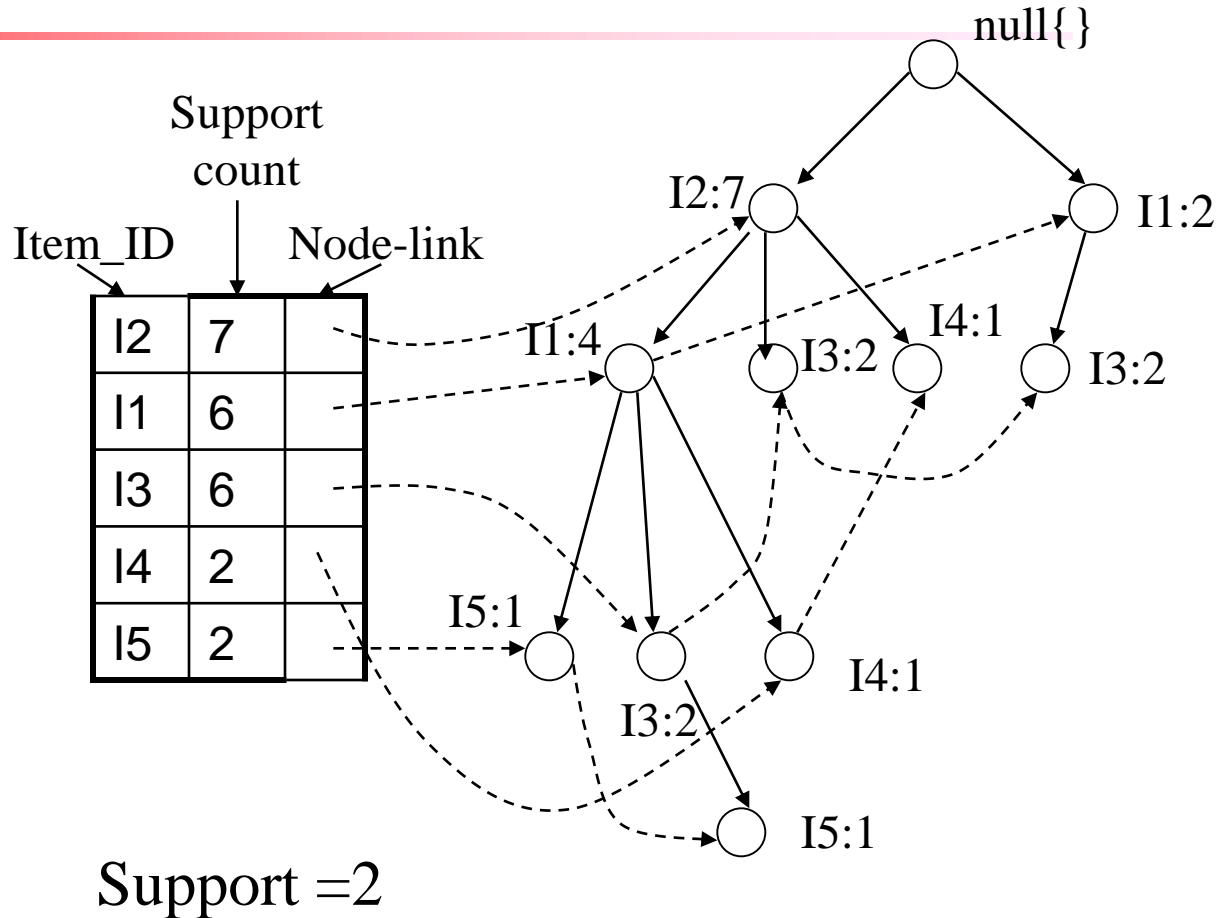
Exercise

3. A database has 9 transactions. Let $min_sup = 20\%$. Please construct the FP-tree for the database, the conditional FP-trees, and all the frequent itemsets.

TID	List of items_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

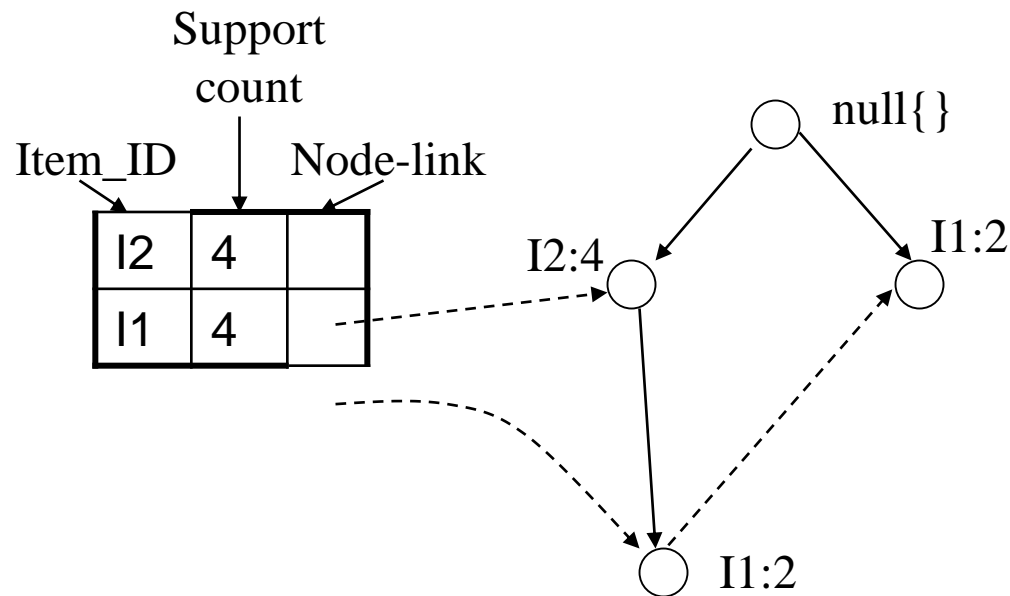
Solution

TID	List of items_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

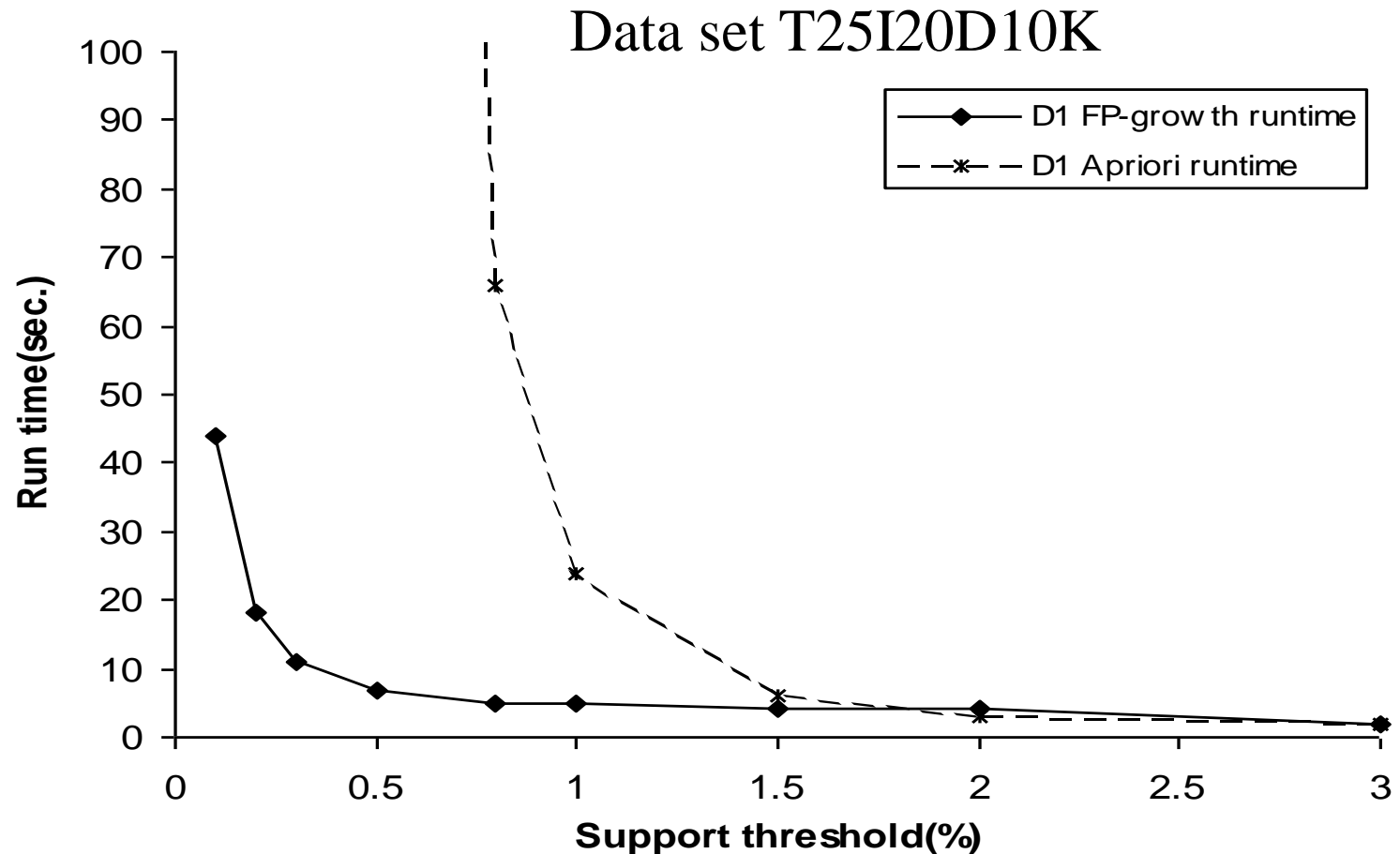


Solution

<i>item</i>	<i>conditional pattern base</i>	<i>conditional FP-tree</i>	<i>frequent patterns generated</i>
I5	$\{\{I2, I1: 1\}, \{I2, I1, I3: 1\}\}$	$\langle I2: 2, I1: 2 \rangle$	$\{I2, I5: 2\}, \{I1, I5: 2\}, \{I2, I1, I5: 2\}$
I4	$\{\{I2, I1: 1\}, \{I2: 1\}\}$	$\langle I2: 2 \rangle$	$\{I2, I4: 2\}$
I3	$\{\{I2, I1: 2\}, \{I2: 2\}, \{I1: 2\}\}$	$\langle I2: 4, I1: 2 \rangle, \langle I1: 2 \rangle$	$\{I2, I3: 4\}, \{I1, I3: 4\}, \{I2, I1, I3: 2\}$
I1	$\{\{I2: 4\}\}$	$\langle I2: 4 \rangle$	$\{I2, I1: 4\}$



FP-Growth vs. Apriori: Scalability With the Support Threshold



Why Is FP-Growth the Winner?

■ Divide-and-conquer:

- Decompose both the mining task and DB according to the frequent patterns obtained so far
- Focus searching on smaller databases

■ Other factors

- No candidate generation, no candidate test
- Compressed database: FP-tree structure
- Two scans of entire database
- Basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

Cons

- Building FP-trees
 - A stack of FP-trees
- Redundant information
 - Transaction abcd appears in a-, ab-, abc-, ac-, c-FP-trees

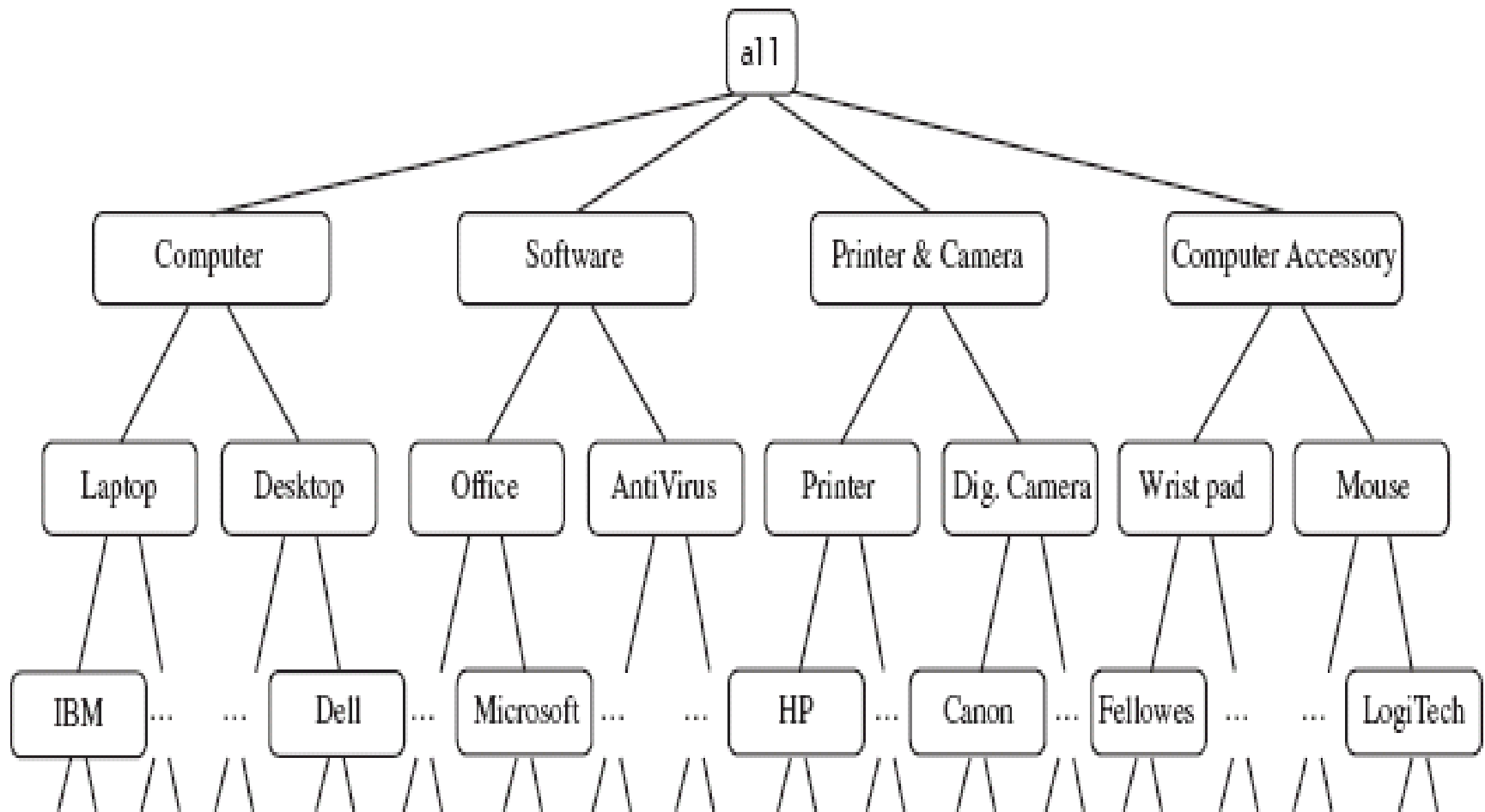
Mining Association Rules in Large Databases

- Basic concepts and a road map
- Mining single-dimensional Boolean association rules
- Mining multilevel association rules
- Mining multidimensional association rules
- Summary

Mining Multiple-Level Association Rules

- Association rules at high concept levels may represent common sense knowledge
- Hard to find association rules at low concept level
 - Items at the lower level usually have lower support, less than min_support threshold
- Mining association rules at multiple levels of abstraction
- Example: sales in AllElectronics store computer sector

Example



Mining Multiple-Level Association Rules

■ Uniform support

- Top-down, level-wise
- Use uniform minimum support for each level
- Perform Apriori at each level
- Optimization: if an ancestor is infrequent, the search on the descendants can be avoided

uniform support

Level 1
min_sup = 5%

Milk
[support = 10%]

Level 2
min_sup = 5%

2% Milk
[support = 6%]

Skim Milk
[support = 4%]

Mining Multiple-Level Association Rules

uniform support

Level 1
min_sup = 5%

Milk
[support = 10%]

Level 2
min_sup = 5%

2% Milk
[support = 6%]

Skim Milk
[support = 4%]

■ Drawbacks

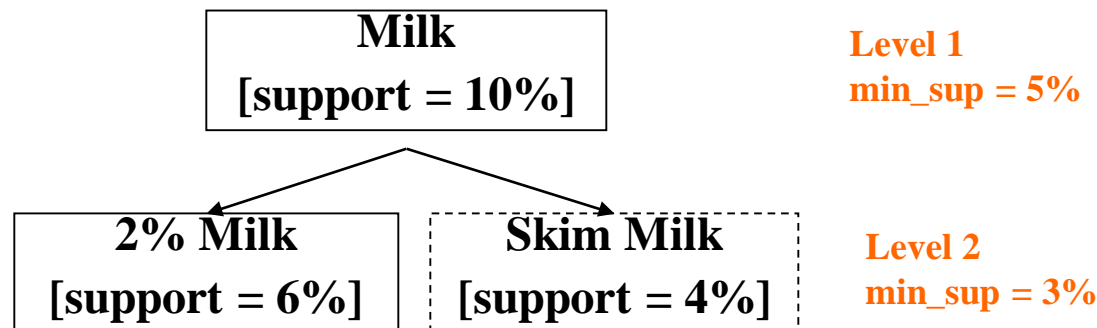
- Miss interesting associations with too high threshold
- Generate too many uninteresting rules with too low threshold

Mining Multiple-Level Association Rules

■ Reduced support

- Top-down, level-wise
- Each concept level has its own minimum support threshold
- The lower level, the smaller threshold
- Perform Apriori at each level

reduced support



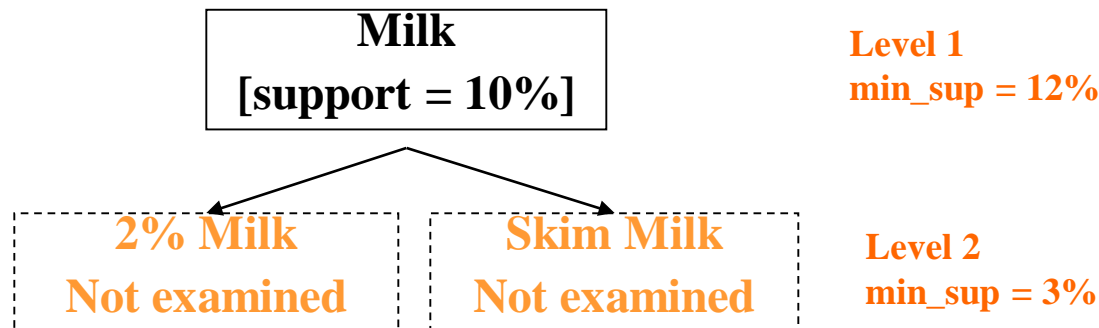
Mining Multiple-Level Association Rules

■ Reduced support

■ Optimization -- level-cross filtering by single item

- An item at the i th concept level is examined *iff* its parent concept at the $(i-1)$ th level is frequent
- If a concept is infrequent, its descendents are pruned from the database
- Drawbacks
 - Miss associations at low level items which are frequent based on a reduced min_support, but whose ancestors do not satisfy min_support

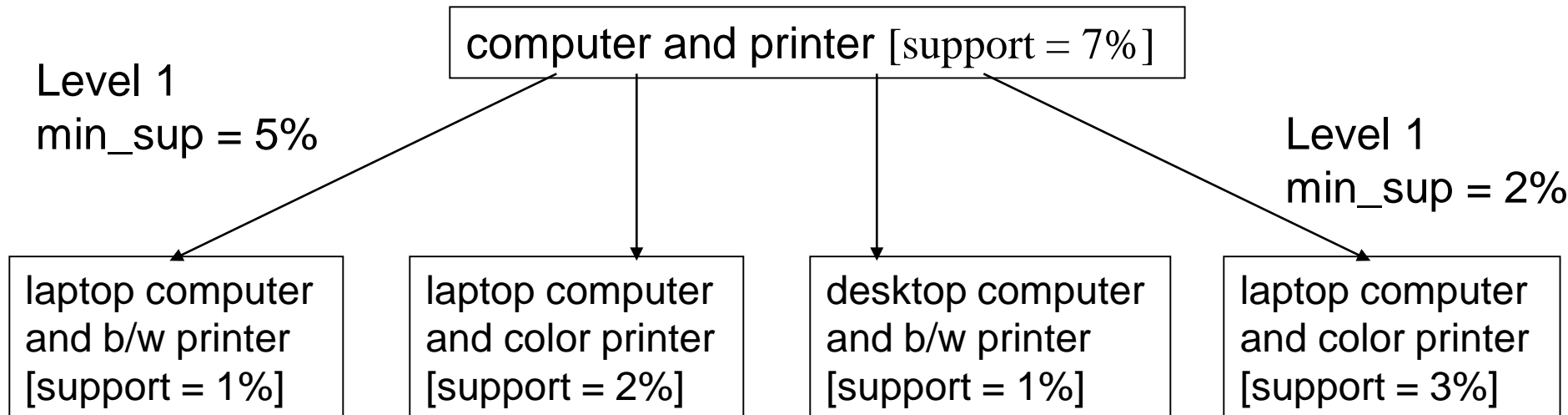
reduced support



Mining Multiple-Level Association Rules

■ Reduced support

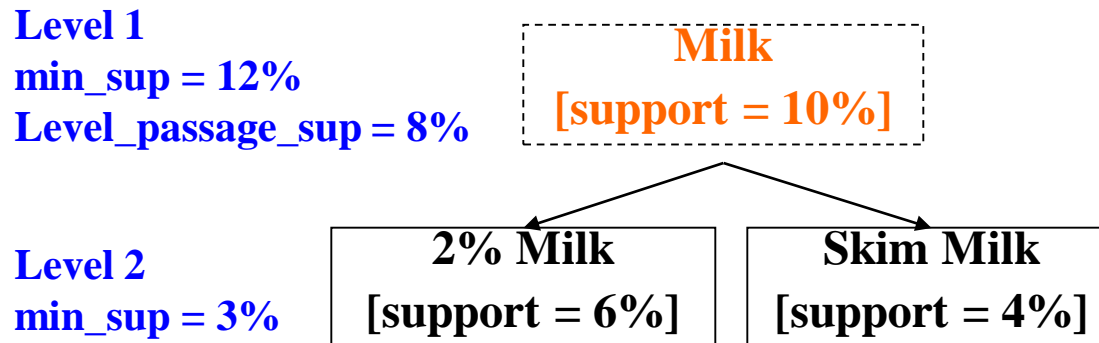
- Optimization -- level-cross filtering by k -itemset
 - Only the children of frequent k -itemsets are examined
 - Drawback: many valuable patterns may be filtered out



Mining Multiple-Level Association Rules

■ Reduced support

- Optimization -- Controlled level-cross filtering by single item
 - next level min sup < level passage threshold < min sup
 - Allow the children of items that do not satisfy the min_sup to be examined if they satisfy the level passage threshold



Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items
- Example
 - milk \Rightarrow wheat bread [support = 8%, confidence = 70%]
 - 2% milk \Rightarrow wheat bread [support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor

Mining Association Rules in Large Databases

- Basic concepts and a road map
- Mining single-dimensional Boolean association rules
- Mining multilevel association rules
- Mining multidimensional association rules
- Summary

Mining Multi-Dimensional Association

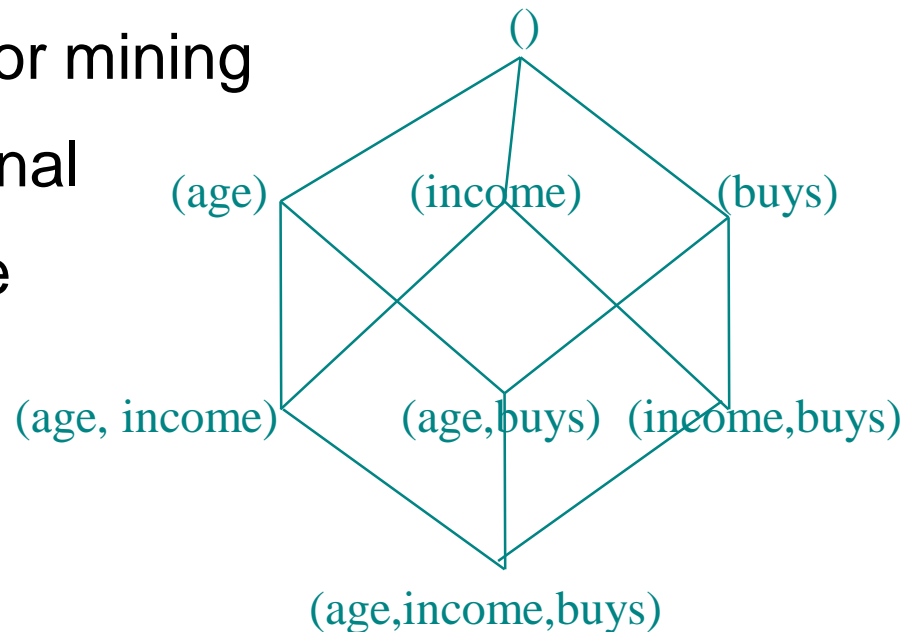
- Single-dimensional rules:
 $\text{buys}(X, \text{"milk"}) \Rightarrow \text{buys}(X, \text{"bread"})$
- Multi-dimensional rules: ≥ 2 dimensions or predicates
 - Inter-dimension assoc. rules (*no repeated predicates*)
 $\text{age}(X, \text{"19-25"}) \wedge \text{occupation}(X, \text{"student"}) \Rightarrow \text{buys}(X, \text{"coke"})$
 - hybrid-dimension assoc. rules (*repeated predicates*)
 $\text{age}(X, \text{"19-25"}) \wedge \text{buys}(X, \text{"popcorn"}) \Rightarrow \text{buys}(X, \text{"coke"})$
- Categorical Attributes: finite number of possible values, no ordering among values
- Quantitative Attributes: numeric, implicit ordering among values — discretization, clustering approaches

Mining Quantitative Associations

- Techniques can be used to categorize numerical attributes
 - Static discretization based on predefined concept hierarchies
 - Dynamic discretization based on data distribution
 - Clustering: Distance-based association
 - one dimensional clustering then association

Static Discretization of Quantitative Attributes

- Discretized prior to mining using concept hierarchy
- Numeric values are replaced by ranges
- In relational database, finding all frequent k -predicate sets will require k or $k+1$ table scans
- Data cube is well suited for mining
- The cells of a n -dimensional cuboid correspond to the dimensions
- Mining from data cubes can be much faster



Quantitative Association Rules

- Numeric attributes are *dynamically* discretized
 - Such that the confidence or compactness of the rules mined is maximized
- 2-D quantitative association rules: $A_{\text{quan1}} \wedge A_{\text{quan2}} \Rightarrow A_{\text{cat}}$
- Association rule clustering system (ARCS)
 - Binning: 2-D grid, manageable size
 - Finding frequent predicate sets: scan the database, count the support for each grid cell
 - Clustering the rules: cluster adjacent cells to form a rule

Quantitative Association Rules

■ Example

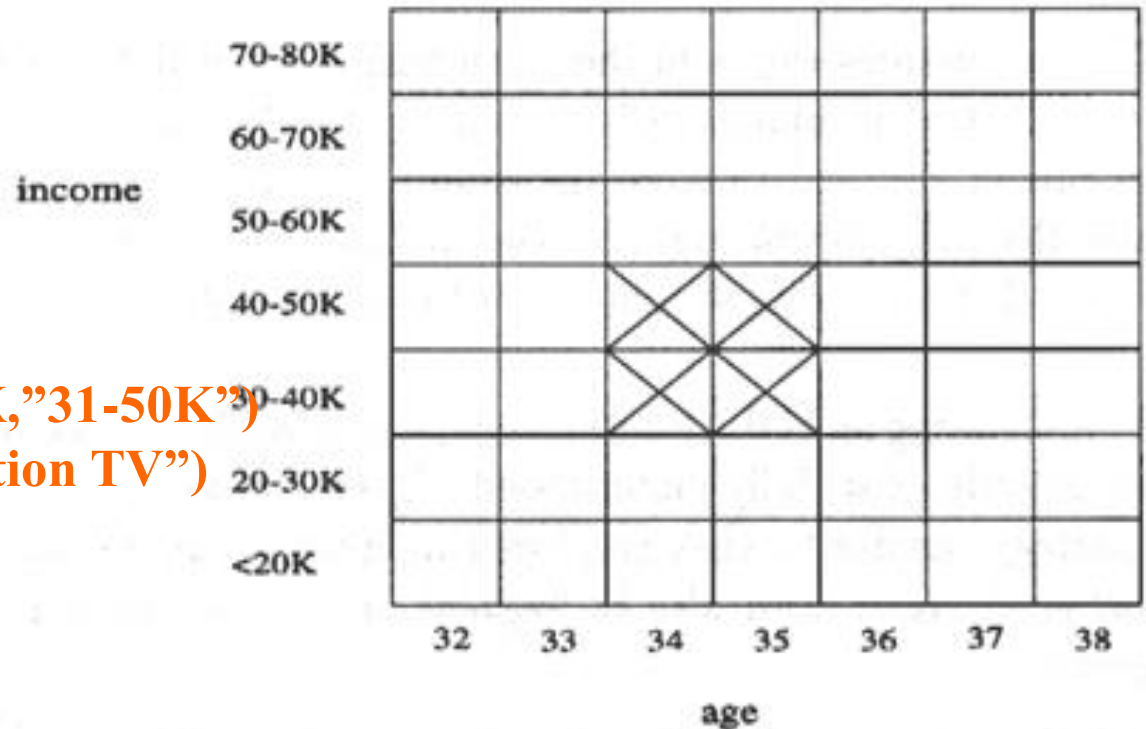
$\text{age}(X, "34") \wedge \text{income}(X, "31-40K") \Rightarrow \text{buys}(X, "high\ resolution\ TV")$

$\text{age}(X, "35") \wedge \text{income}(X, "31-40K") \Rightarrow \text{buys}(X, "high\ resolution\ TV")$

$\text{age}(X, "34") \wedge \text{income}(X, "41-50K") \Rightarrow \text{buys}(X, "high\ resolution\ TV")$

$\text{age}(X, "35") \wedge \text{income}(X, "41-50K") \Rightarrow \text{buys}(X, "high\ resolution\ TV")$

$\text{age}(X, "34-35") \wedge \text{income}(X, "31-50K")$
 $\Rightarrow \text{buys}(X, "high\ resolution\ TV")$



Mining Association Rules in Large Databases

- Basic concepts and a road map
- Mining single-dimensional Boolean association rules
- Mining multilevel association rules
- Mining multidimensional association rules
- Summary

Summary

- Frequent pattern mining—an important task in data mining
- Scalable frequent pattern mining methods
 - Apriori (Candidate generation & test)
 - Partition, DIC, DHP, etc.
 - Projection-based (FP-growth)
- Mining a variety of rules and interesting patterns