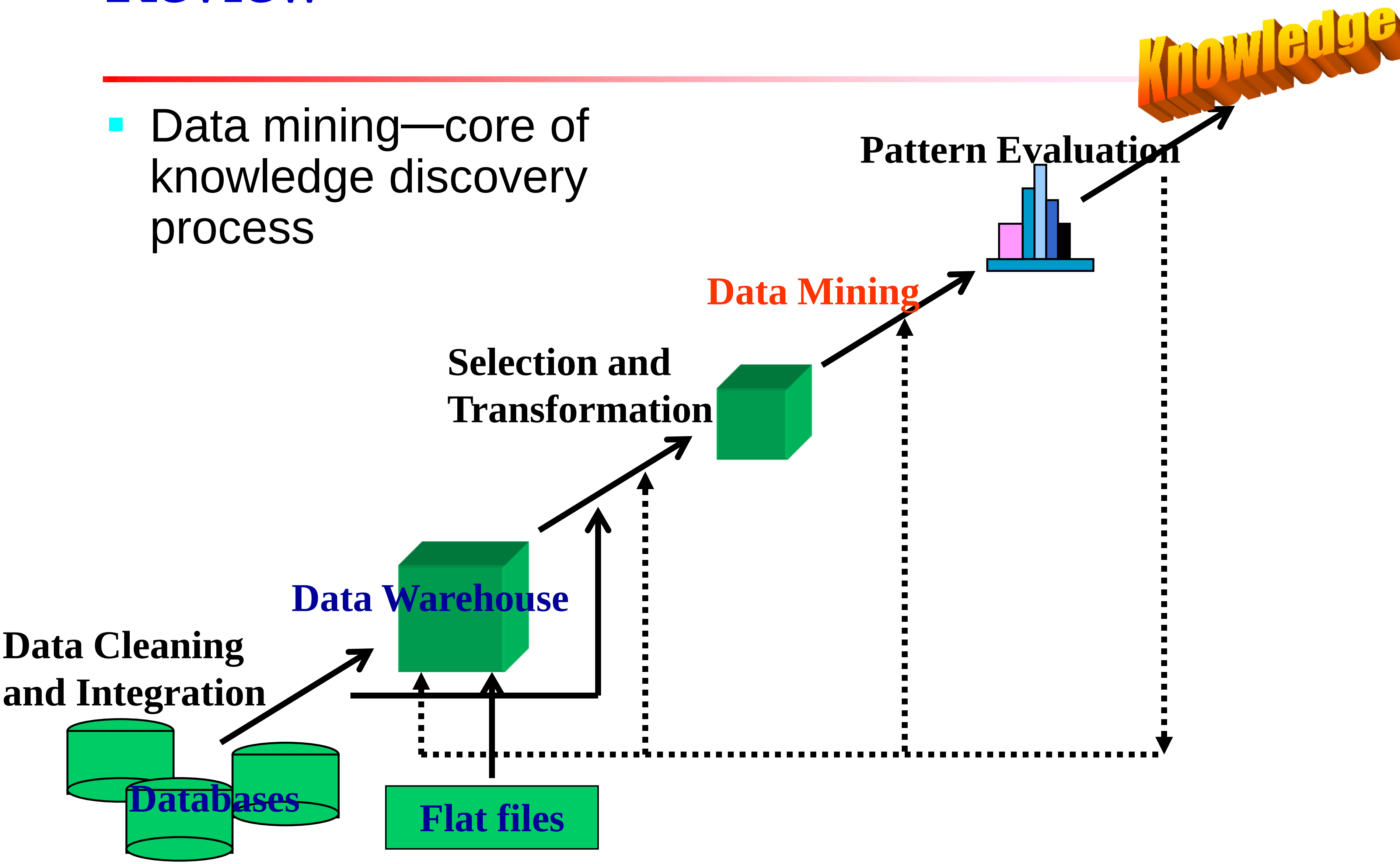
Data Mining

Instructor: Associate Prof. Ying Liu

University of Chinese Academy of Sciences

Review

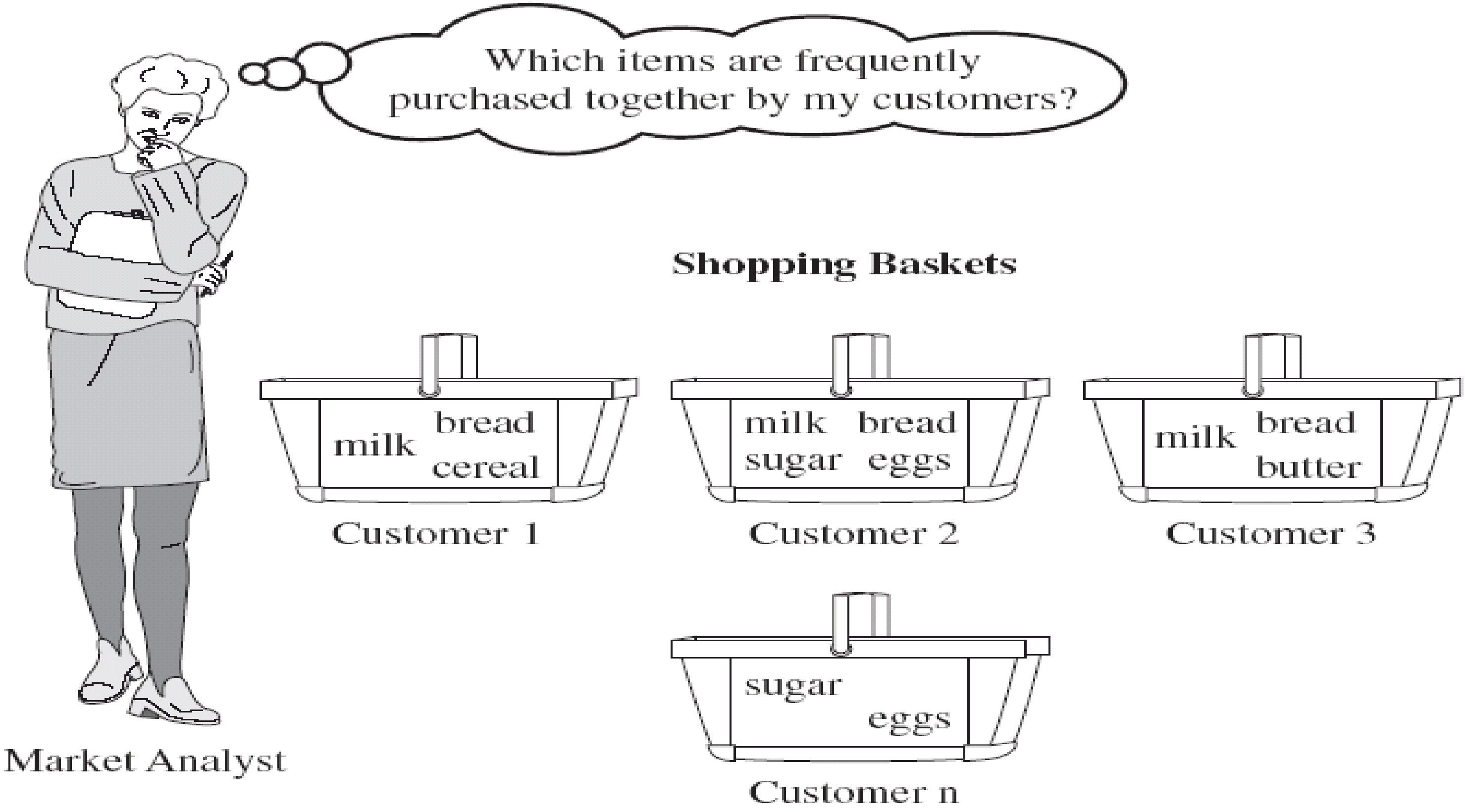
- Data mining—core of knowledge discovery process



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 are the subsequent purchases after buying a PC?

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

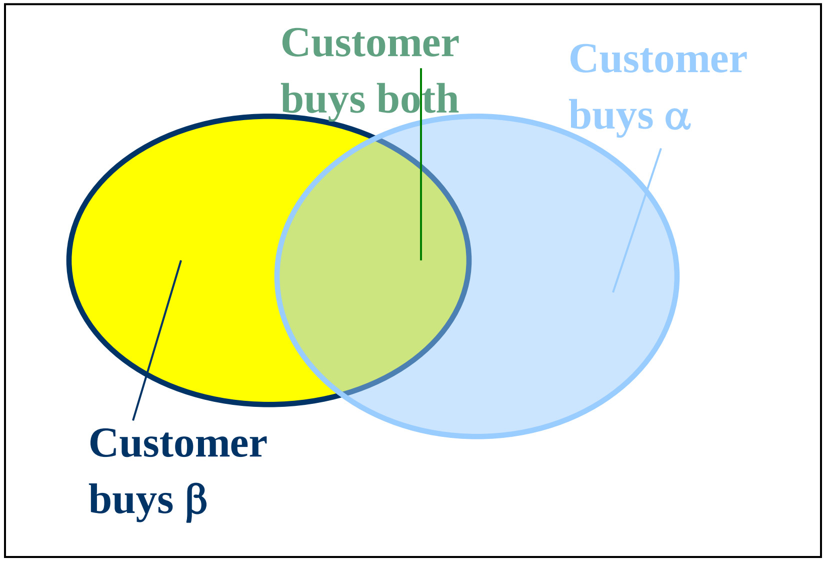
- Item collection X = $\{x_1, \ldots, x_m\}$
- Itemset: a set of items, *k*-itemset
- Transaction T $\subseteq$ X, each T associates a unique Tid and items bought by a customer
- Rule form $\alpha => \beta$, $\alpha \subset X$, $\beta \subset X$, $\alpha \cap \beta = \varnothing$

Basic Concepts

- support, *s*, probability that a transaction contains α and β
  - support (α => β) = P(α∩β)
- Frequent itemset, occurrence greater than a min_support
- Frequent itemset mining, find all the rules *α* => β satisfying min_support
- Let 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

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

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

  Association rules:

  A => D (60%, 100%)

  D => 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

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

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, buys(x, “SQLServer”) ^ buys(x, “DMBook”) => buys(x, “DBMiner”) [0.2%, 60%]
  - Quantitative association rules, concern quantitative attributes, age(x, “30…39”) ^ income(x, “42…48K”) => buys(x, “high resolution TV”) [1%, 75%]
- Single level vs. multiple-level analysis (based on the levels of abstraction involved)
  - age(x, “30…39”) => buys(x, “laptop computer”)
  - age(x, “30…39”) => buys(x, “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

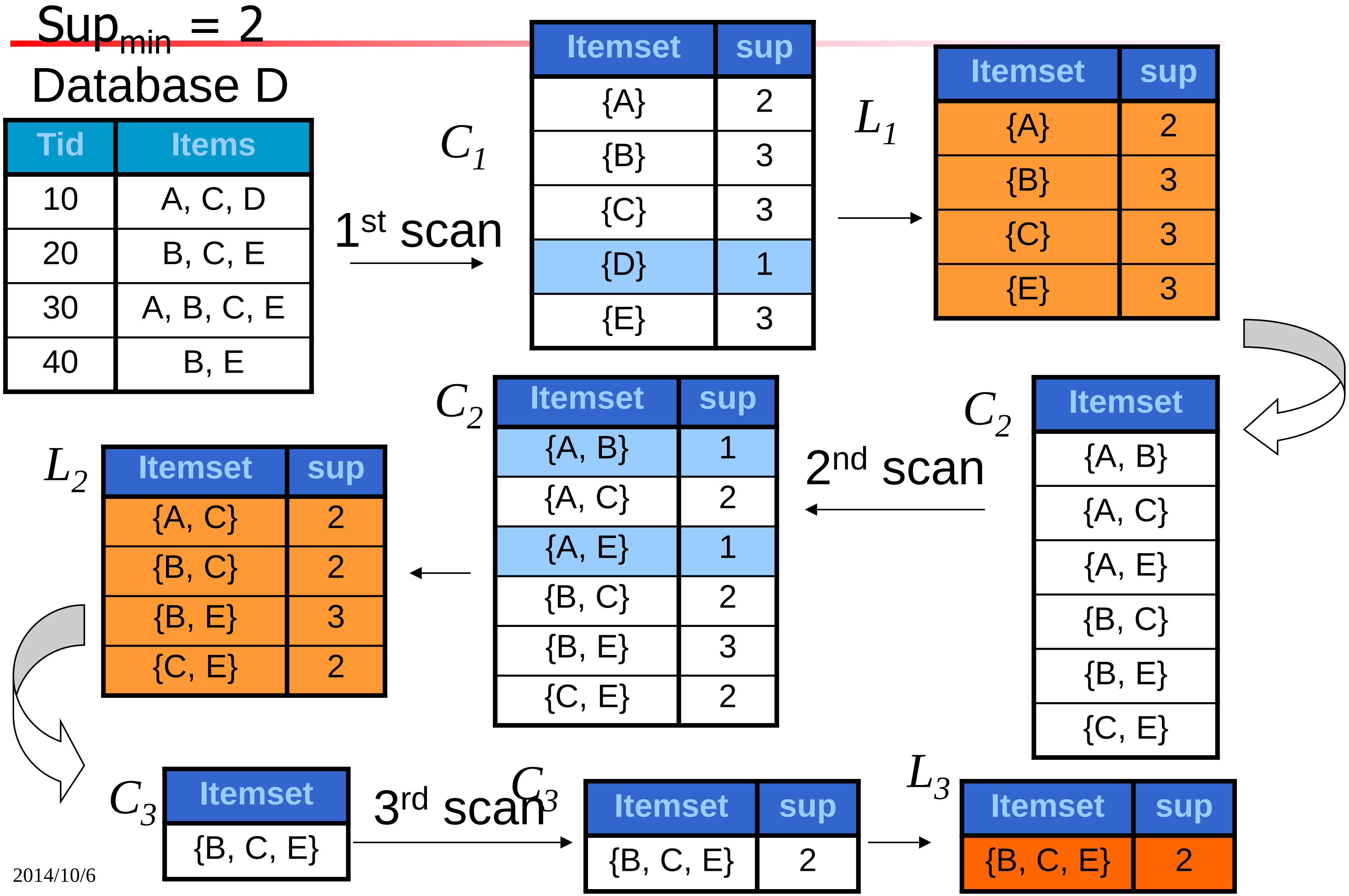
Apriori: A Candidate Generation-and-Test Approach

- 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



$Sup_{min} = 2$

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

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

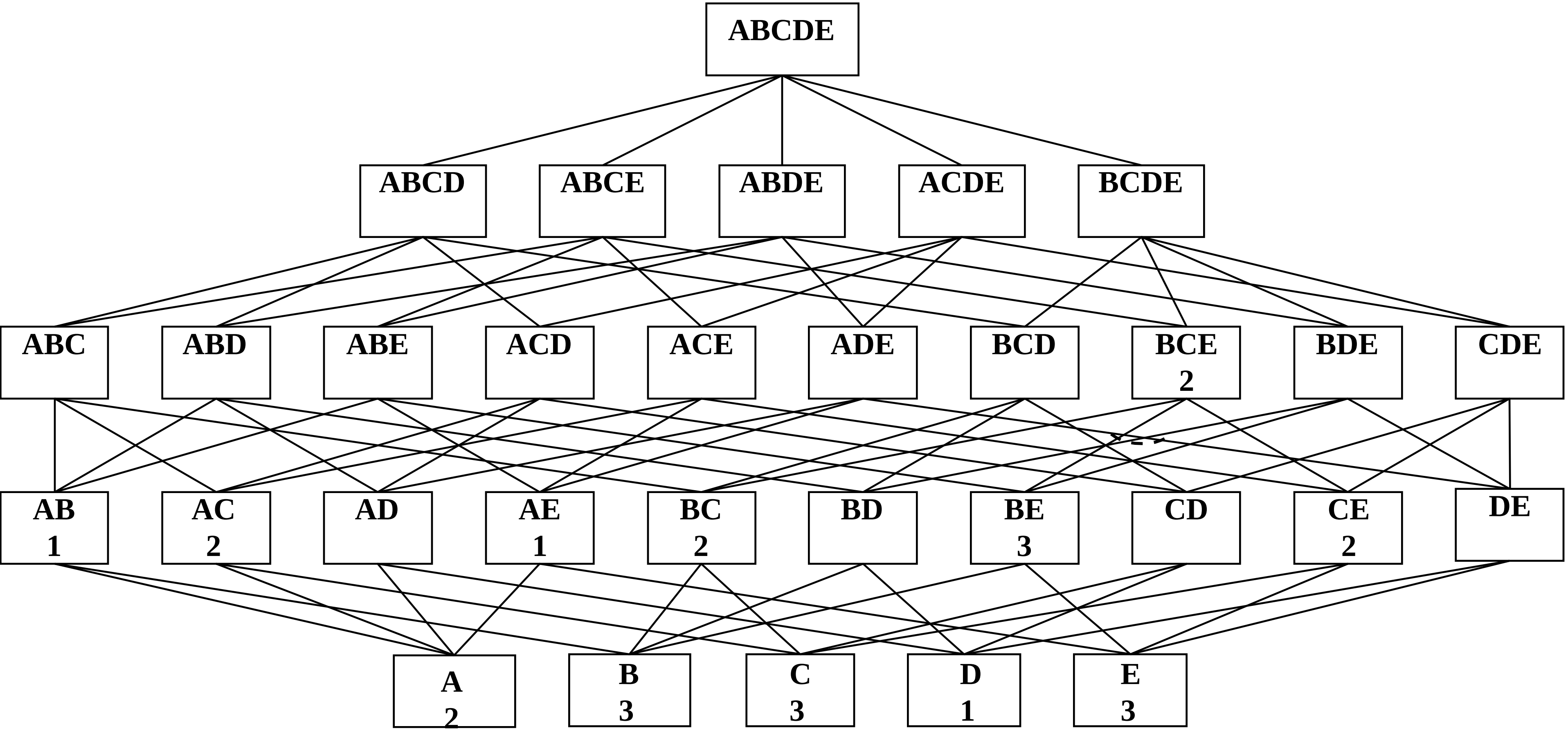
C_3

Itemset	sup
{B, C, E}	2

L_3

Itemset	sup
{B, C, E}	2

Apriori Algorithm — An Example



ABCDE

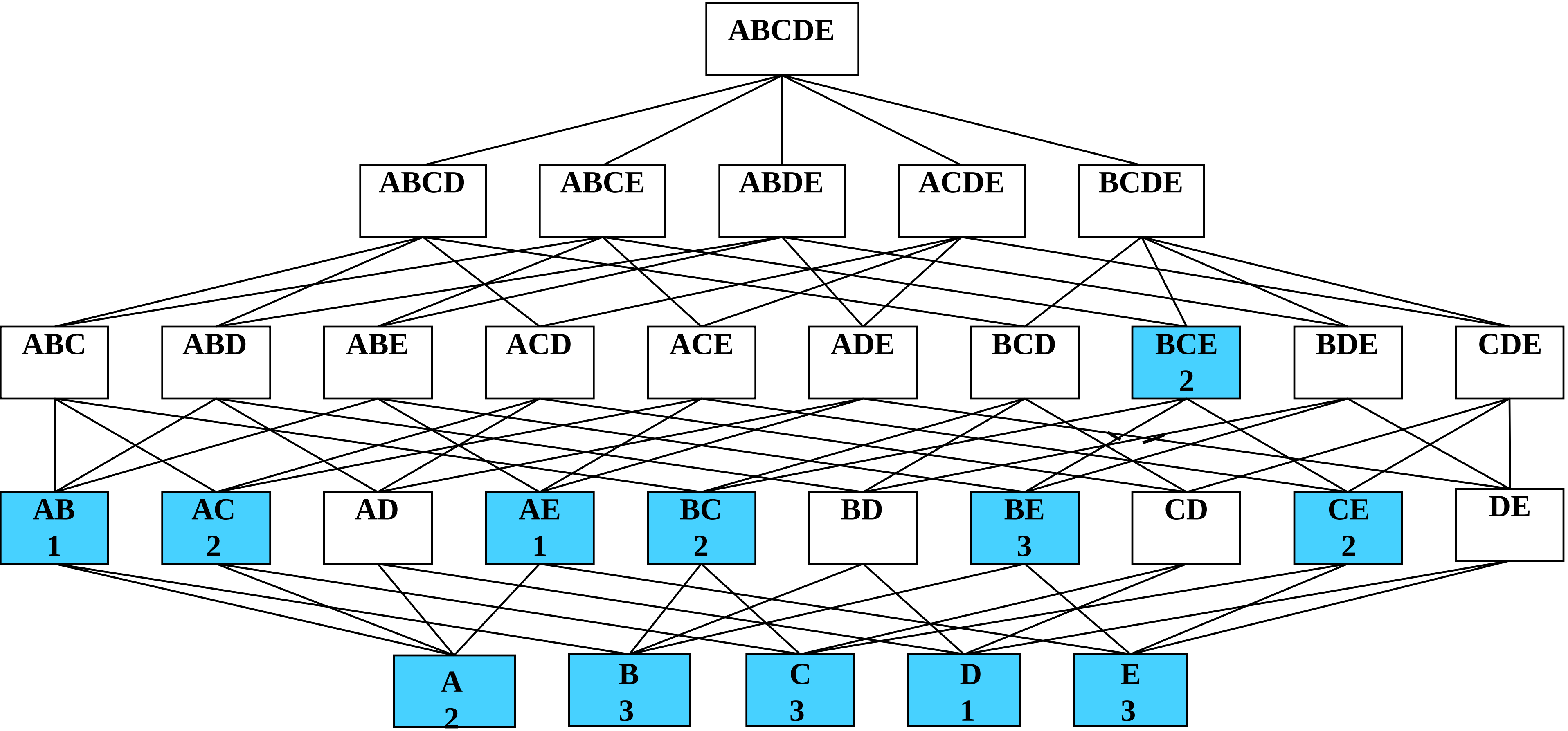
ABCD ABCE ABDE ACDE BCDE

ABC ABD ABE ACD ACE ADE BCD BCE 2 BDE CDE

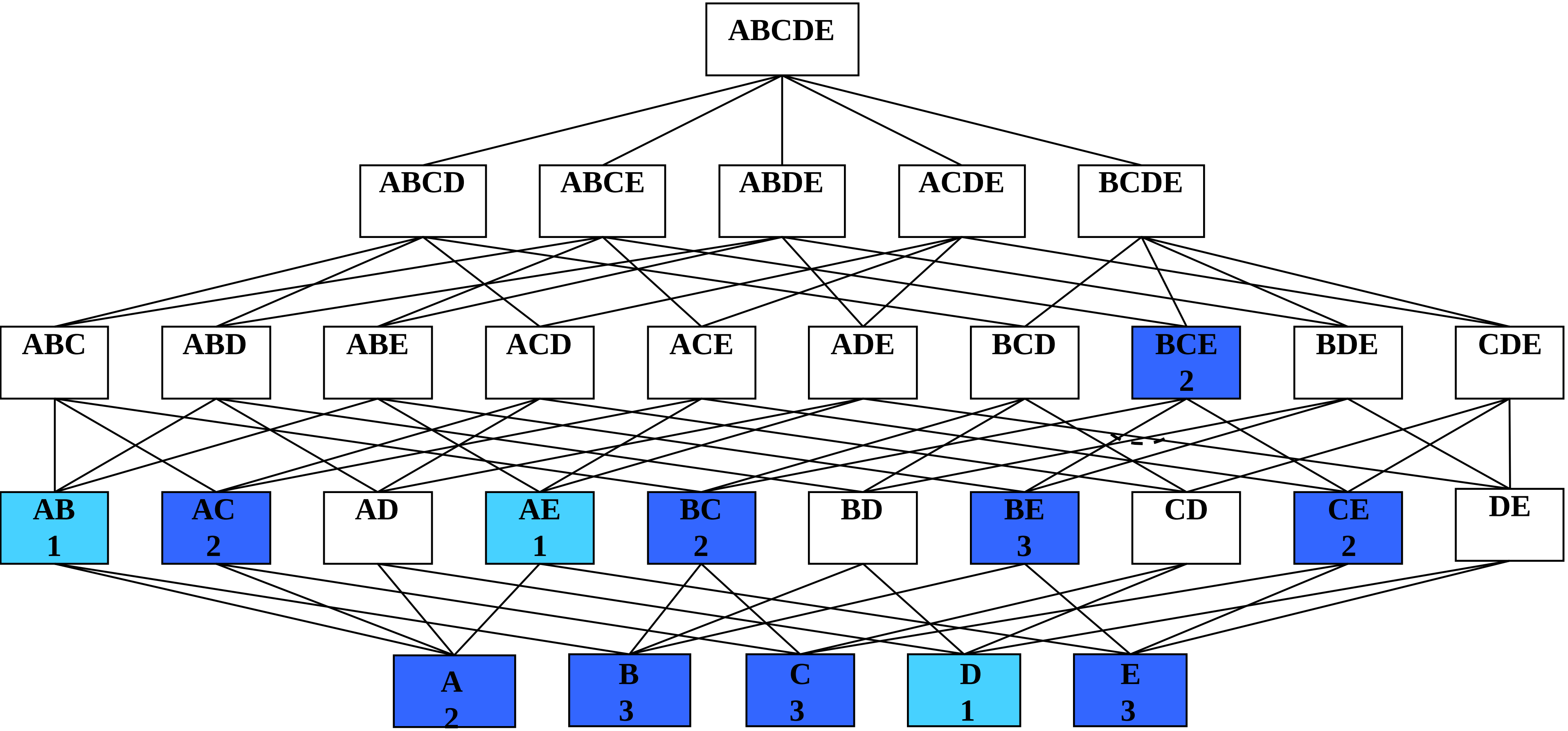
AB 1 AC 2 AD AE 1 BC 2 BD BE 3 CD CE 2 DE

A 2 B 3 C 3 D 1 E 3

Apriori Algorithm — An Example



Apriori Algorithm — An Example



ABCDE

ABCD ABCE ABDE ACDE BCDE

ABC ABD ABE ACD ACE ADE BCD BCE 2 BDE CDE

AB 1 AC 2 AD AE 1 BC 2 BD BE 3 CD CE 2 DE

A 2 B 3 C 3 D 1 E 3

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 $L$

        $L_1$ = {frequent single items from $D$};
        for ($k$ = 2; $L_{k-1}$ !=$\emptyset$; $k$++) 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* -> *abcd, acd* and *ace* -> *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 $(l_1[1]=l_2[1])$^$(l_1[2]=l_2[2])$^…^$(l_1[k-2]=l_2[k-2])$ then
            $c = l_1$ join $l_2$
            pruning ($c$)
    end
end
```

- Step 2: pruning

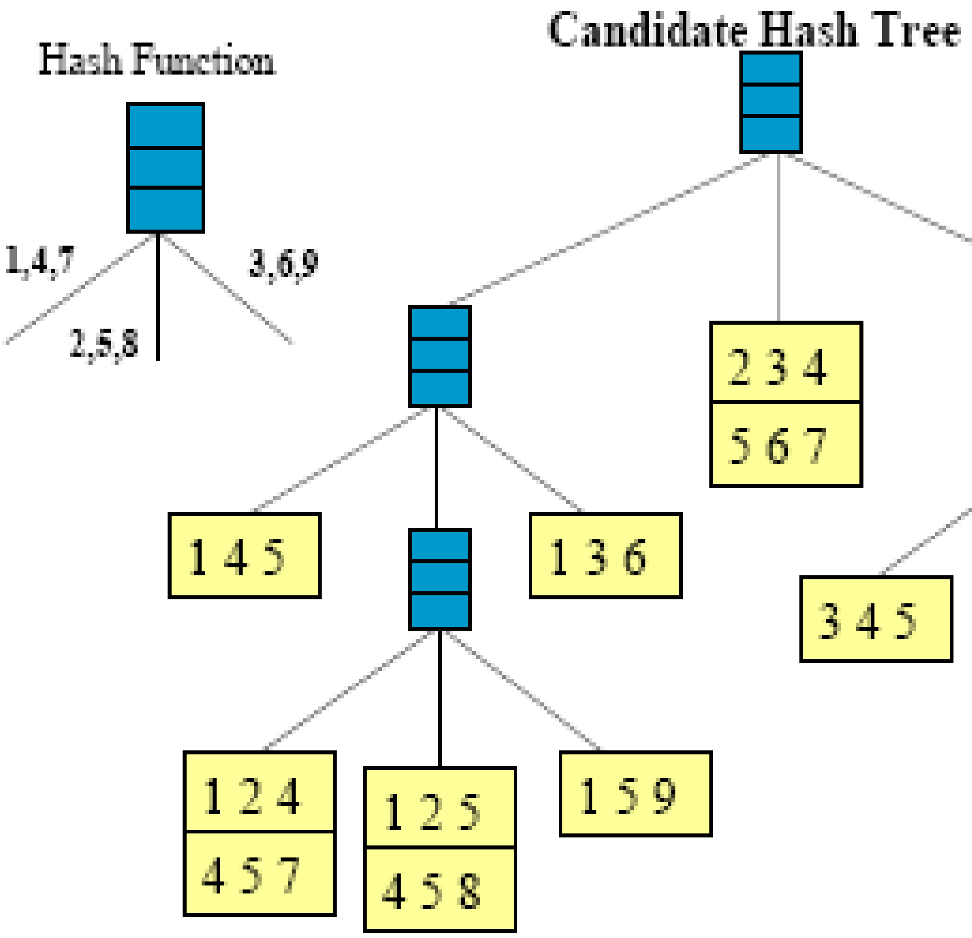
```
forall $(k-1)$-subsets $s$ of $c$ do
    if ($s$ is not in $L_{k-1}$) then delete $c$
```

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

T_1: 1 3 4 6 7 9



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

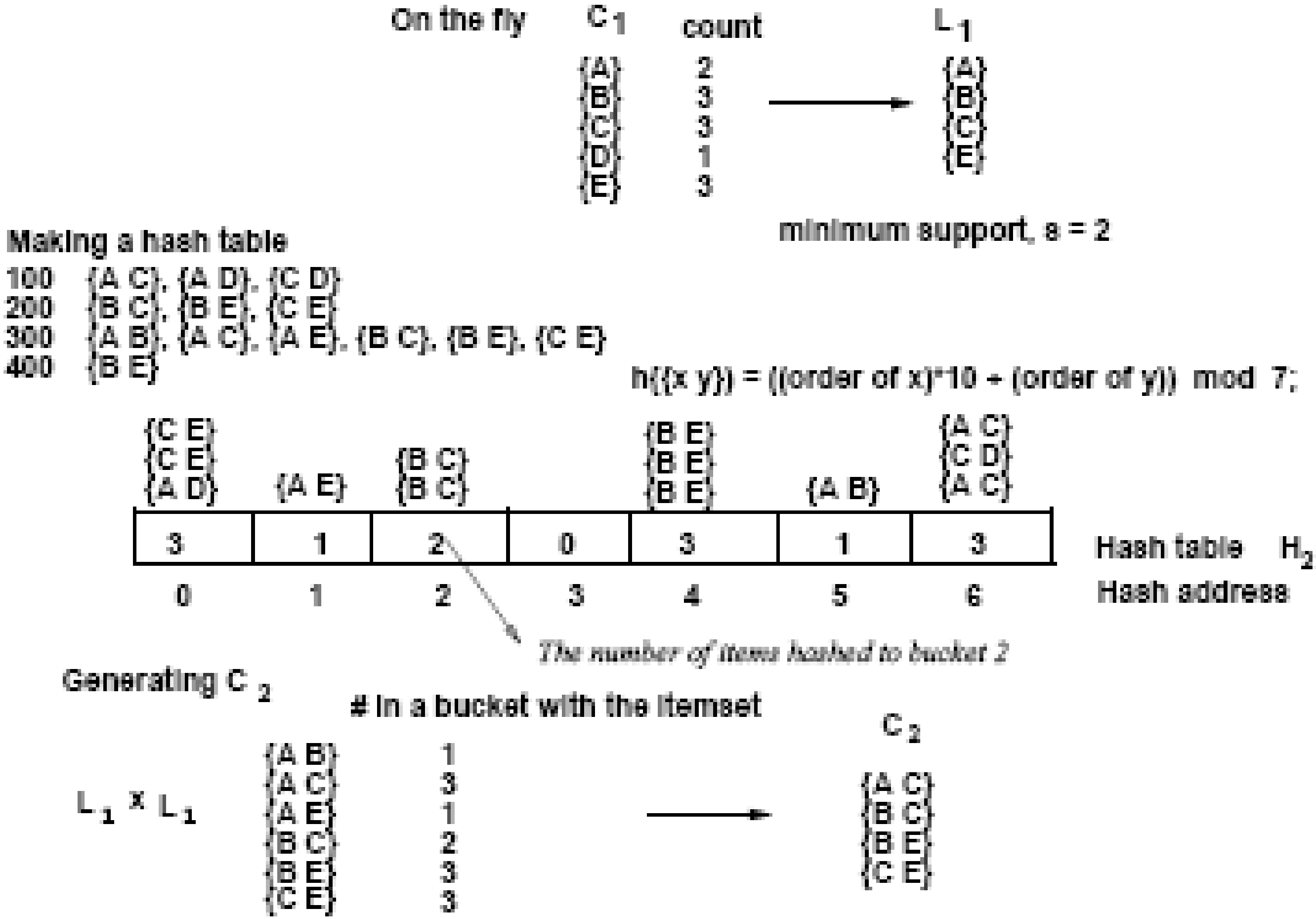
Partition: Scan Database Only Twice

- 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



On the fly

C_1	count
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

→ L_1: {A}, {B}, {C}, {E}

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}) = ((order of x)*10 + (order of y)) mod 7;

Hash address	0	1	2	3	4	5	6
Itemsets	{C E} {C E} {A D}	{A E}	{B C} {B C}		{B E} {B E} {B E}	{A B}	{A C} {C D} {A C}
Hash table H_2	3	1	2	0	3	1	3

The number of items hashed to bucket 2

Generating C_2

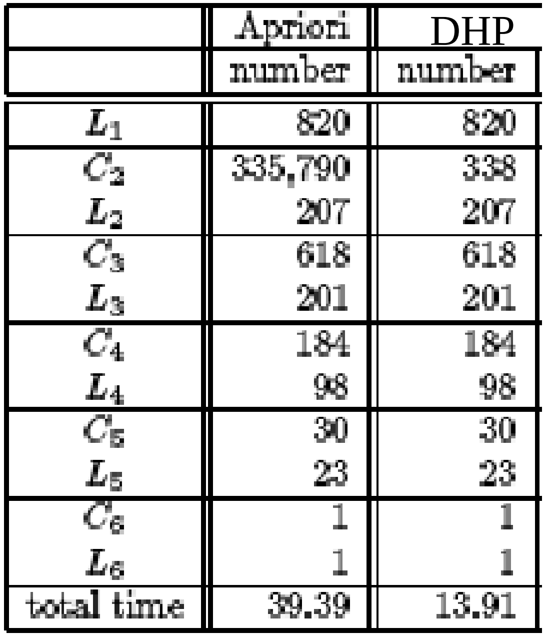
$L_1 \times L_1$	# in a bucket with the itemset
{A B}	1
{A C}	3
{A E}	1
{B C}	2
{B E}	3
{C E}	3

→ C_2: {A C}, {B C}, {B E}, {C E}

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	DHP
	number	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

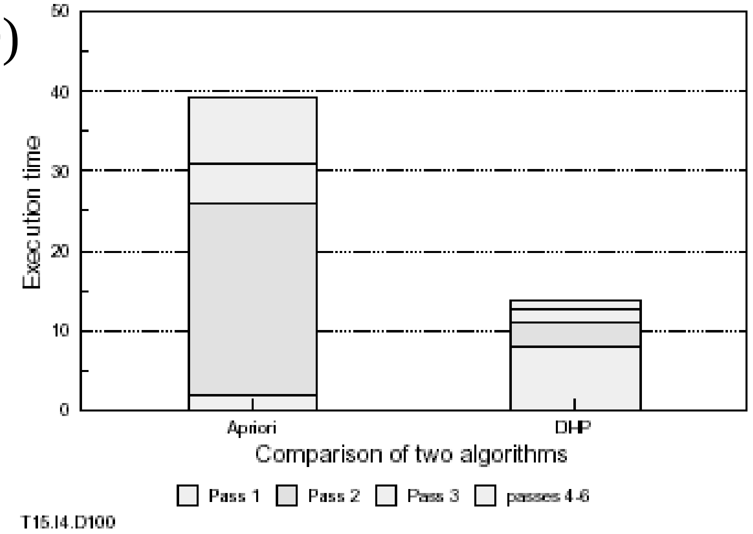
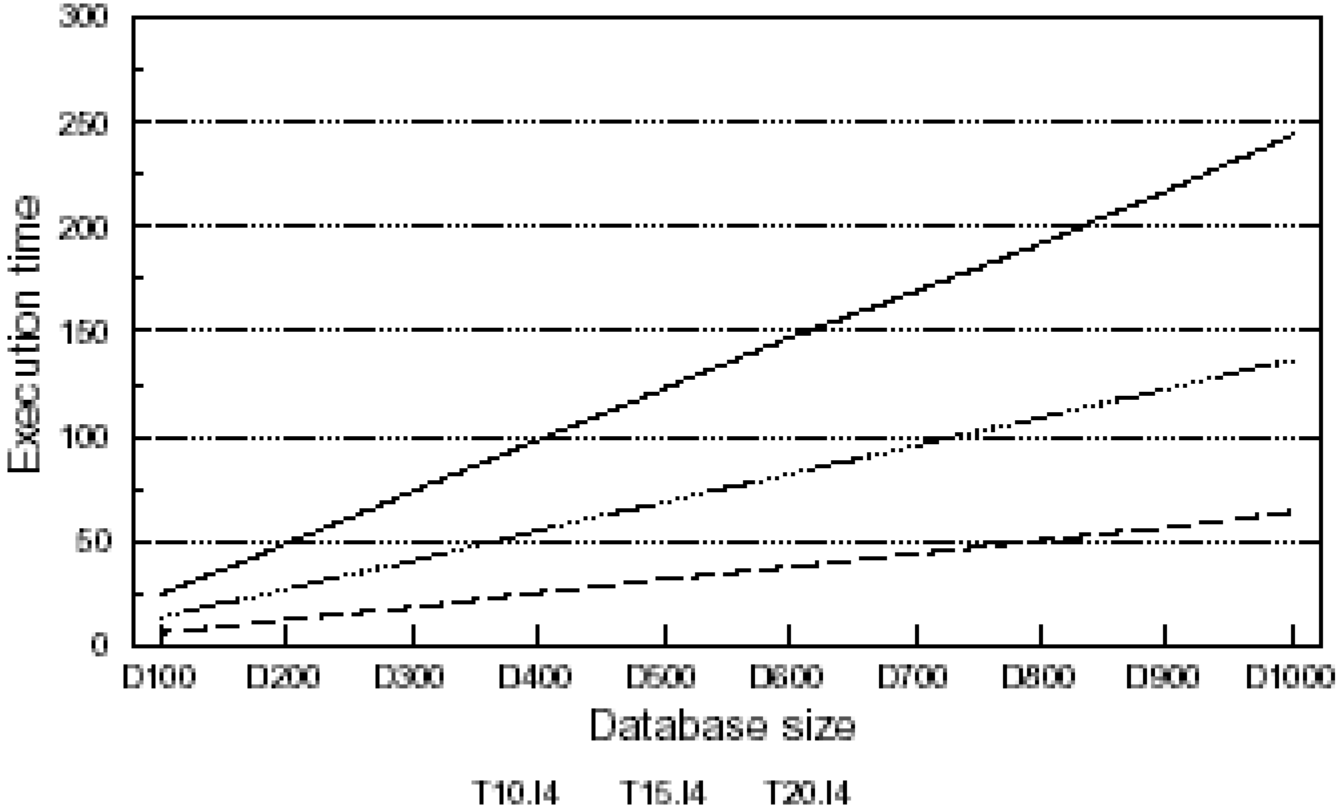


Figure 8: Execution time of Apriori and DHP

Comparison of time (T15.I4.D100)

DHP: Reduce the Number of Candidates

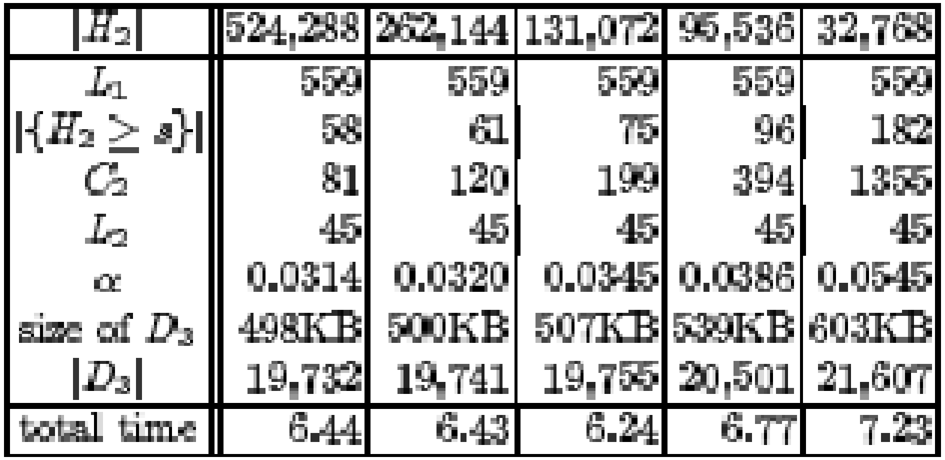


Performance of DHP when increasing the size of database

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)

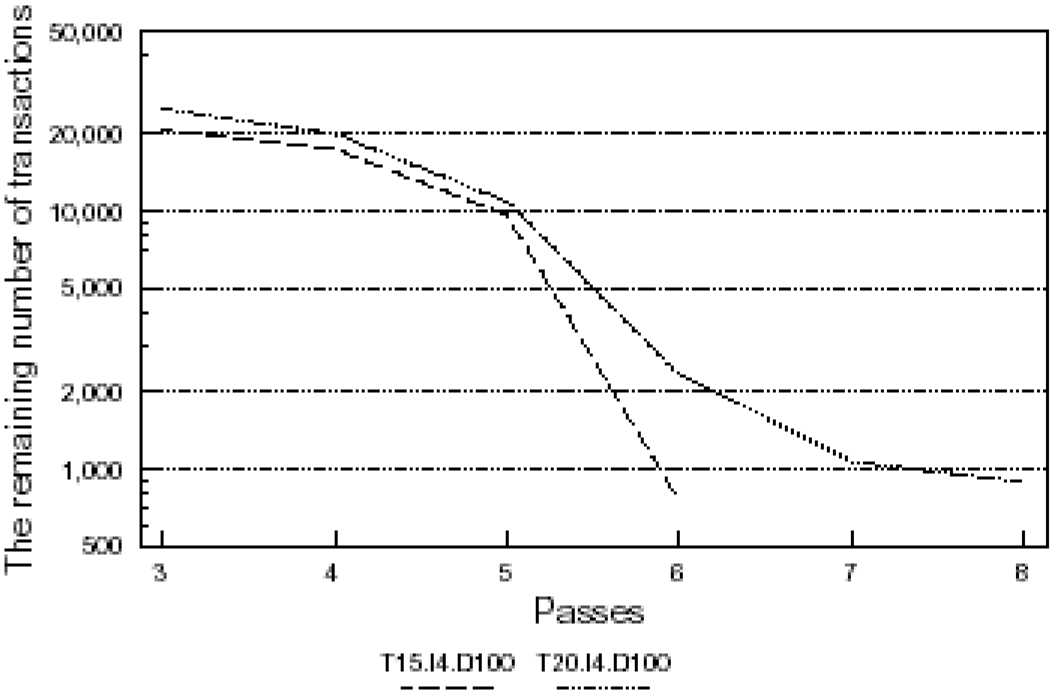


$\vert H_2 \vert$	524,288	262,144	131,072	95,536	32,768
L_1	559	559	559	559	559
$\vert\{H_2 \geq s\}\vert$	58	61	75	96	182
C_2	81	120	199	394	1355
L_2	45	45	45	45	45
α	0.0314	0.0320	0.0345	0.0386	0.0545
size of D_3	498KB	500KB	507KB	539KB	603KB
$\vert D_3 \vert$	19,732	19,741	19,755	20,501	21,607
total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

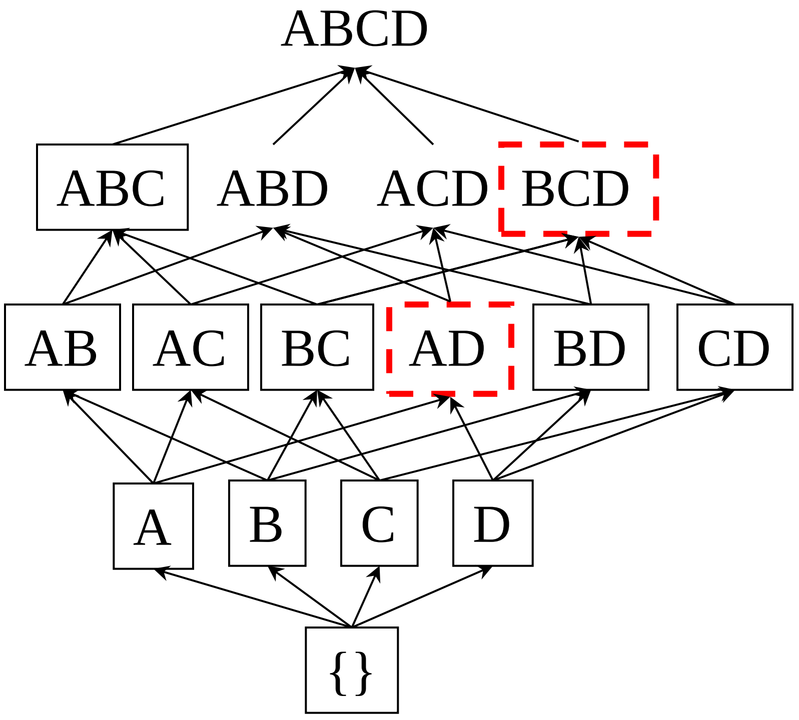
s=0.75%

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

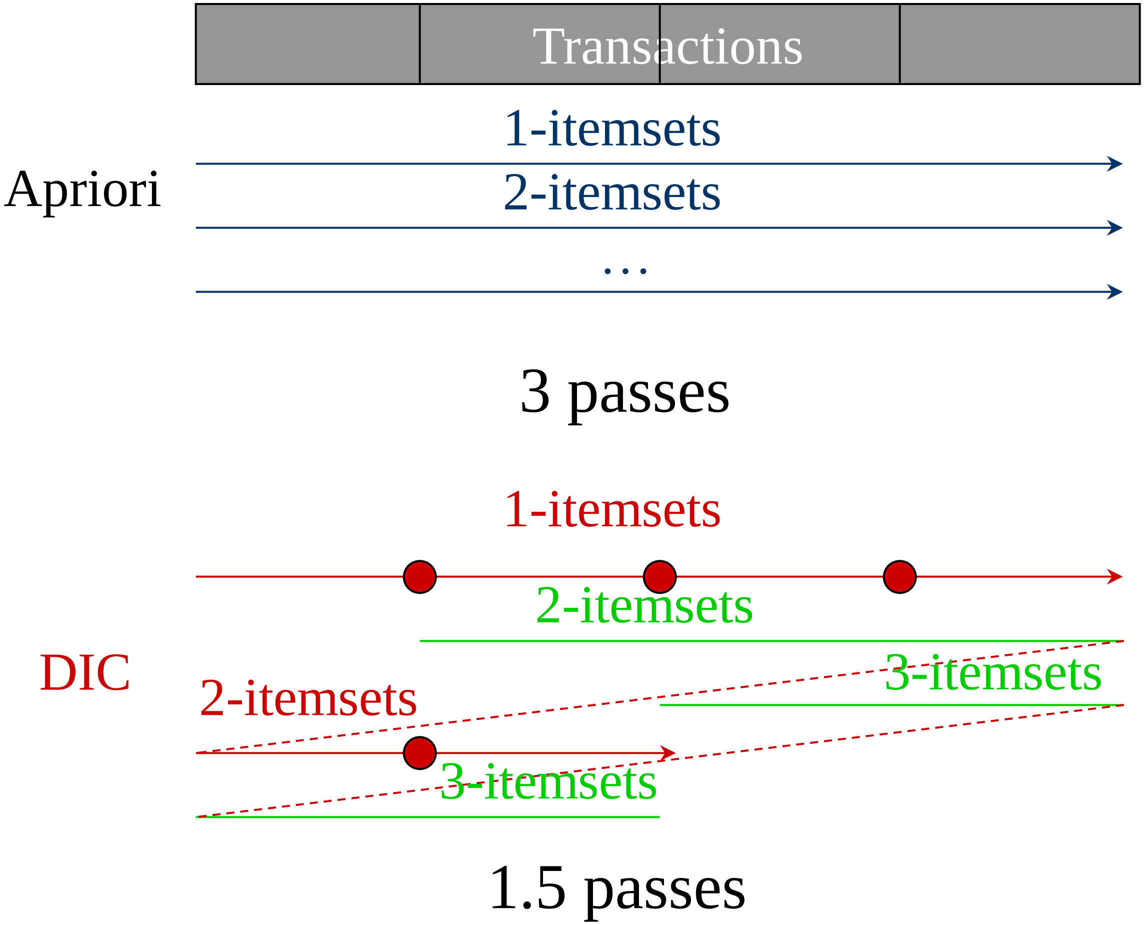
DIC: Reduce Number of Scans



Itemset lattice

- Once both A and D are determined frequent, new candidate AD is added, the counting of AD 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.

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 $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $({}_{100}{}^{1}) + ({}_{100}{}^{2}) + \ldots + ({}_{1}{}^{1}{}_{0}{}^{0}{}_{0}{}^{0}) = 2^{100}-1 = 1.27*10^{30}$!
- 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

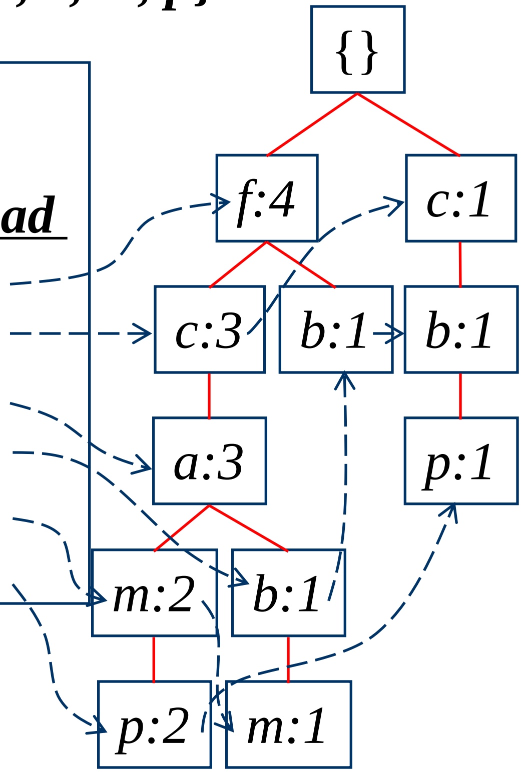
Construct FP-tree from a Transaction Database

TID	*Items bought*	*(ordered) frequent items*
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**

Header Table

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



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

Construct FP-tree from a Transaction Database

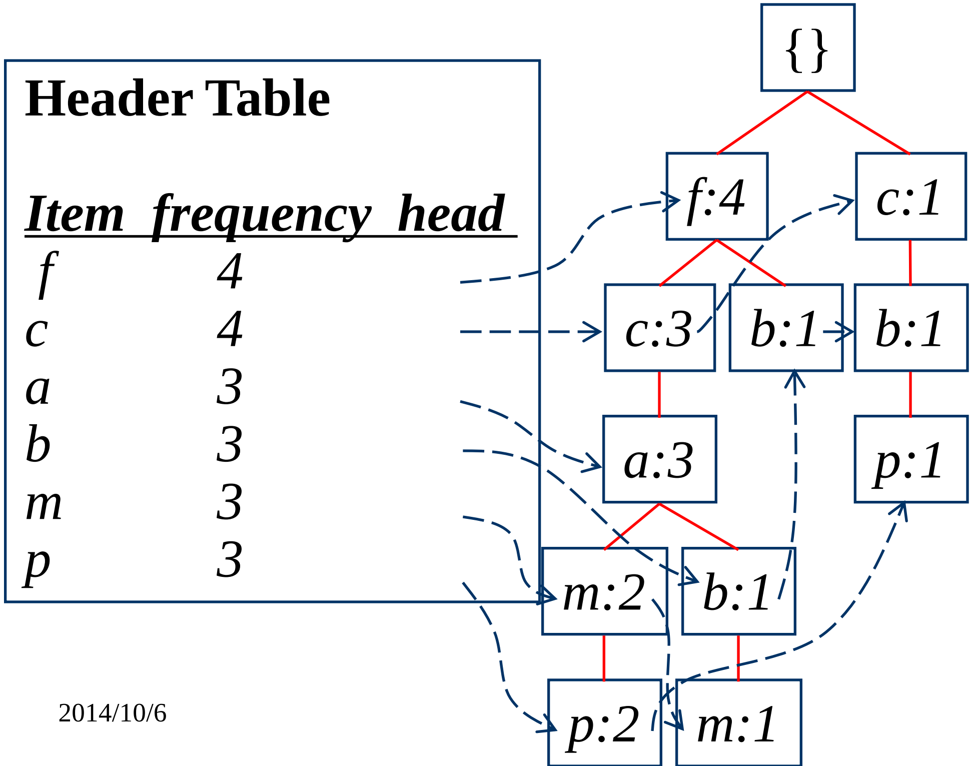
- (a) Scan the transaction database *D* once. Collect *F*, the set of frequent items, and their support counts. Sort *F* in support count descending order as *L*, the *list* of frequent items.
- (b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in *D* do the following:
  - 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



Header Table

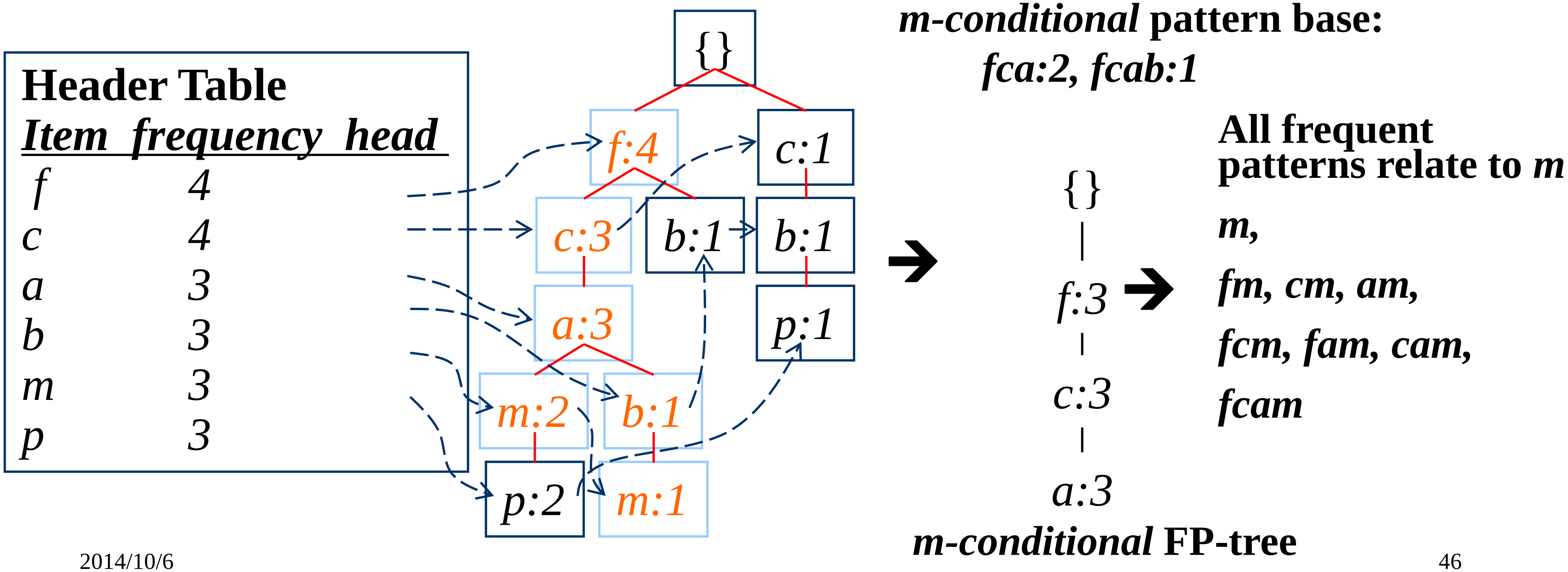
Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

Conditional **pattern bases**

item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

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



Header Table

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

m-conditional **pattern base:**

fca:2, fcab:1

All frequent patterns relate to *m*

m,

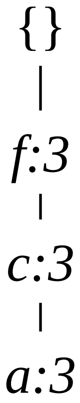
fm, cm, am,

fcm, fam, cam,

fcam

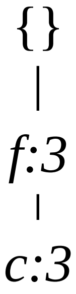
m-conditional **FP-tree**

Recursion: Conditional FP-tree



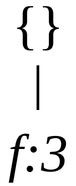
m-conditional **FP-tree**

Cond. pattern base of “am”: (fc:3)



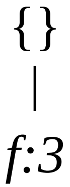
am-conditional **FP-tree**

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

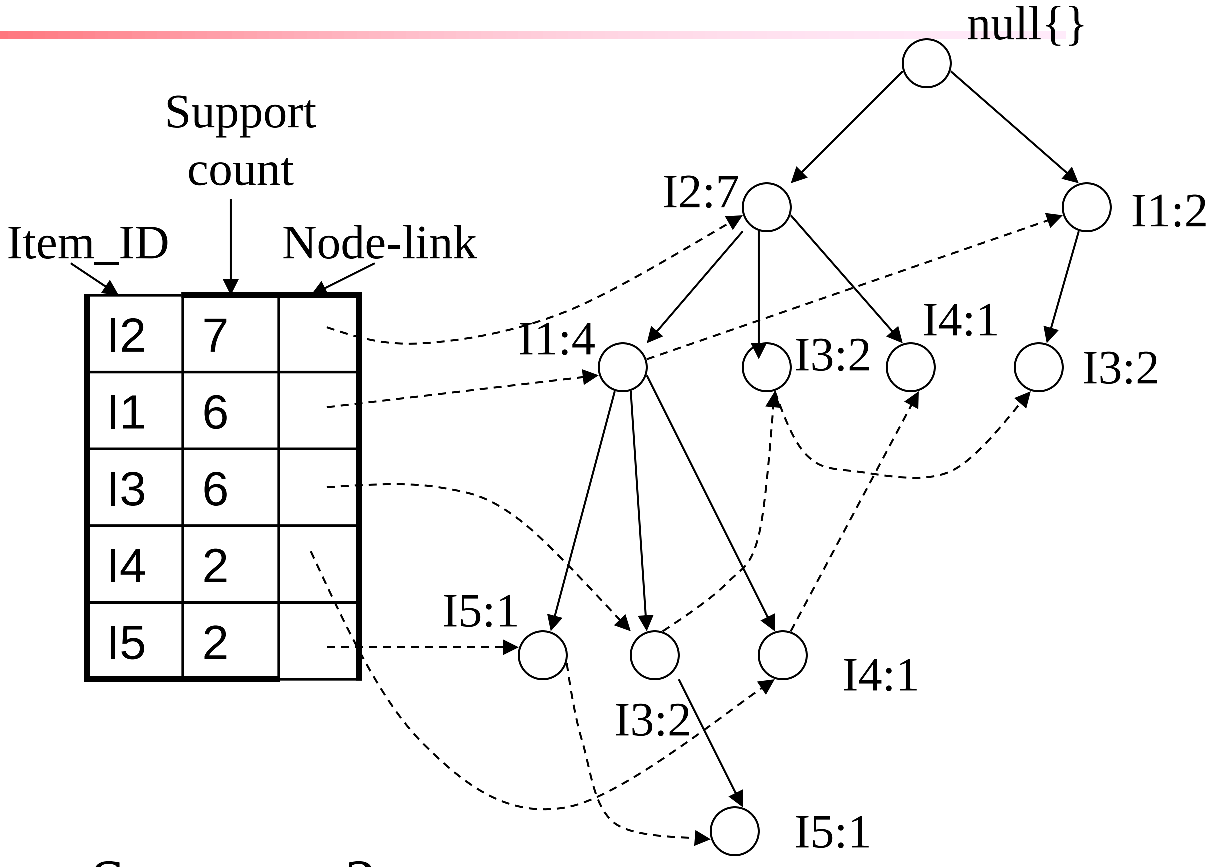
Exercise

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

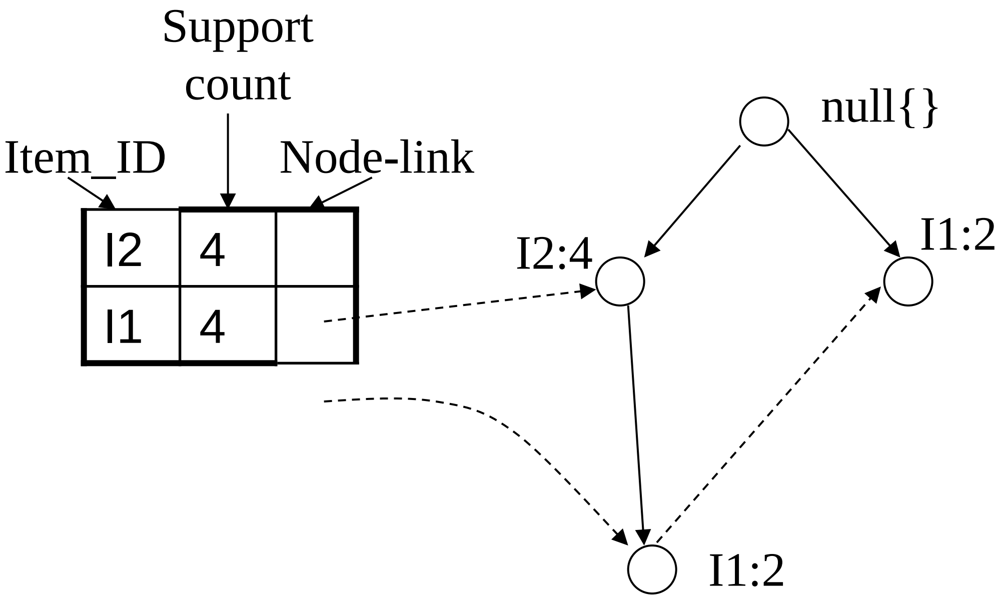


Item_ID	Support count	Node-link
I2	7	
I1	6	
I3	6	
I4	2	
I5	2	

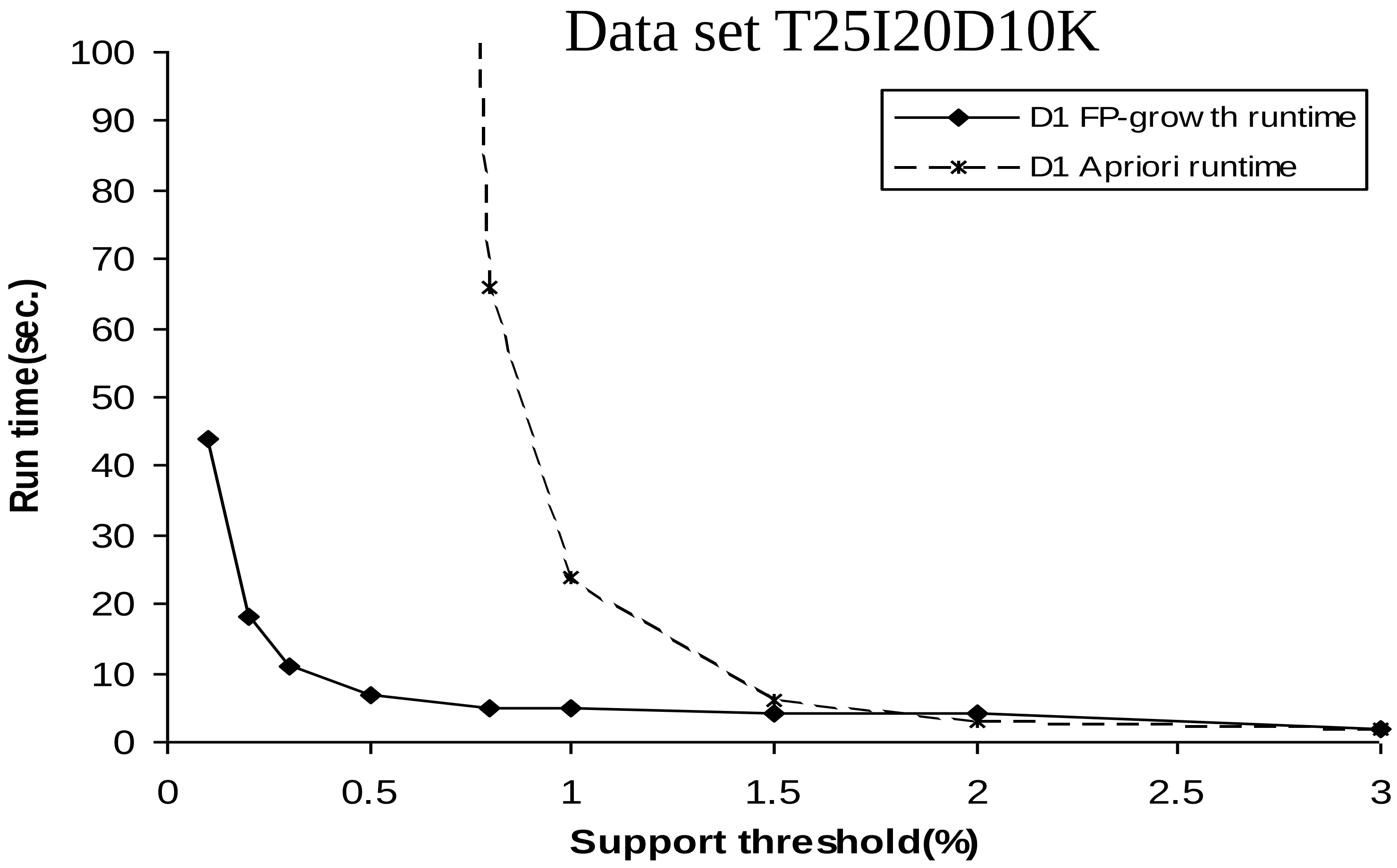
Support =2

Solution

item	*conditional pattern base*	*conditional FP-tree*	*frequent patterns generated*
I5	{{I2,I1: 1}, {I2,I1,I3: 1}}	⟨I2: 2, I1: 2⟩	{I2,I5: 2}, {I1,I5: 2}, {I2,I1,I5: 2}
I4	{{I2,I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2,I4: 2}
I3	{{I2,I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2,I3: 4}, {I1,I3: 4}, {I2,I1,I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{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

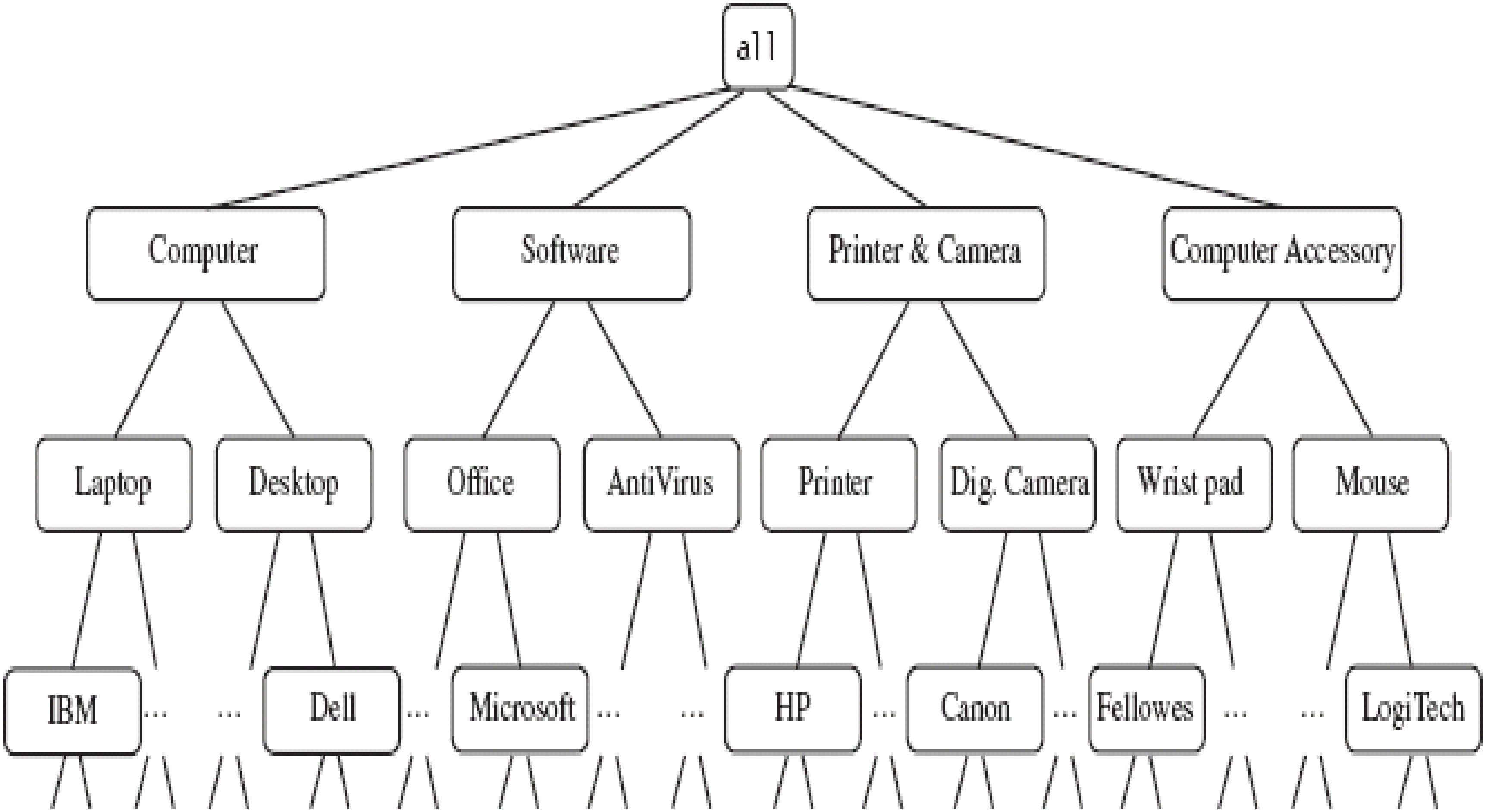
Mining Association Rules in Large Databases

- 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

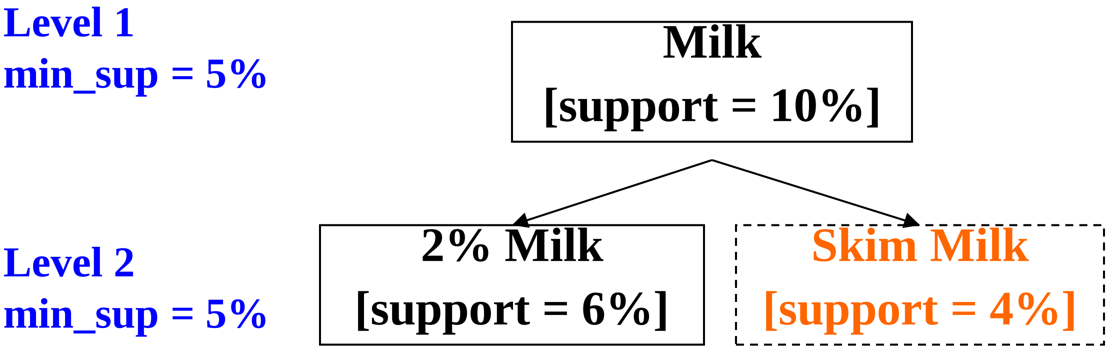
Example


all
Computer
Software
Printer & Camera
Computer Accessory
Laptop
Desktop
Office
AntiVirus
Printer
Dig. Camera
Wrist pad
Mouse
IBM
…
…
Dell
…
Microsoft
…
…
HP
…
Canon
…
Fellowes
…
…
LogiTech

Mining Multiple-Level Association Rules

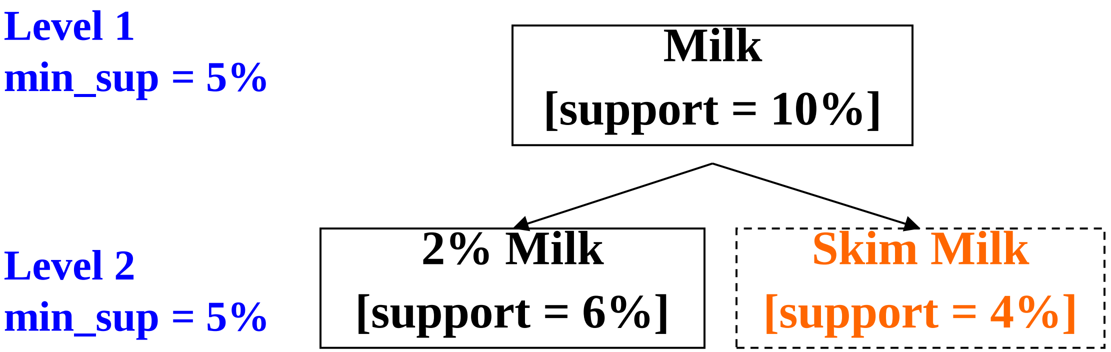
- Uniform support
  - Top-down, level-wise
  - Use uniform minimum support for each level
  - Perform Apriori at each level
  - Optimization: if an ancestor is infrequent, the search on the descendants can be avoided

uniform support



Mining Multiple-Level Association Rules

uniform support

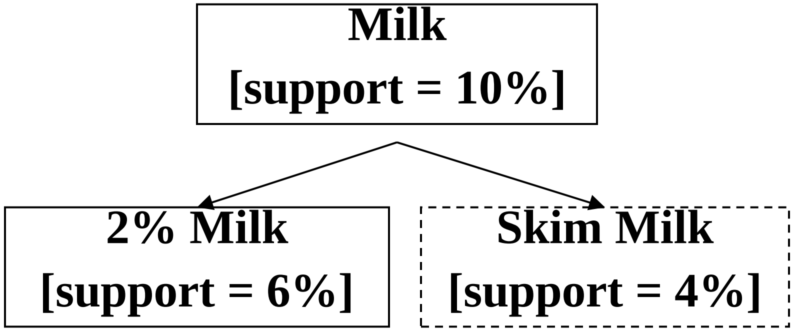


- Drawbacks
  - Miss interesting associations with too high threshold
  - Generate too many uninteresting rules with too low threshold

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



Level 1
min_sup = 5%

Level 2
min_sup = 3%

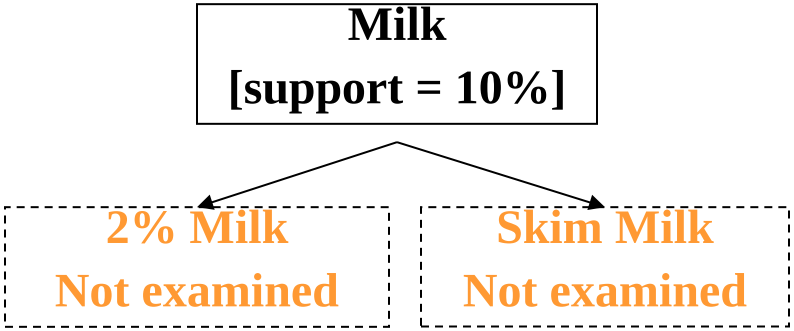
Mining Multiple-Level Association Rules

- Reduced support
  - Optimization -- level-by-level independent search
    - Search items at each level, regardless of whether or not its parent concept is frequent or not
    - Generate numerous associations of little significance at low level

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

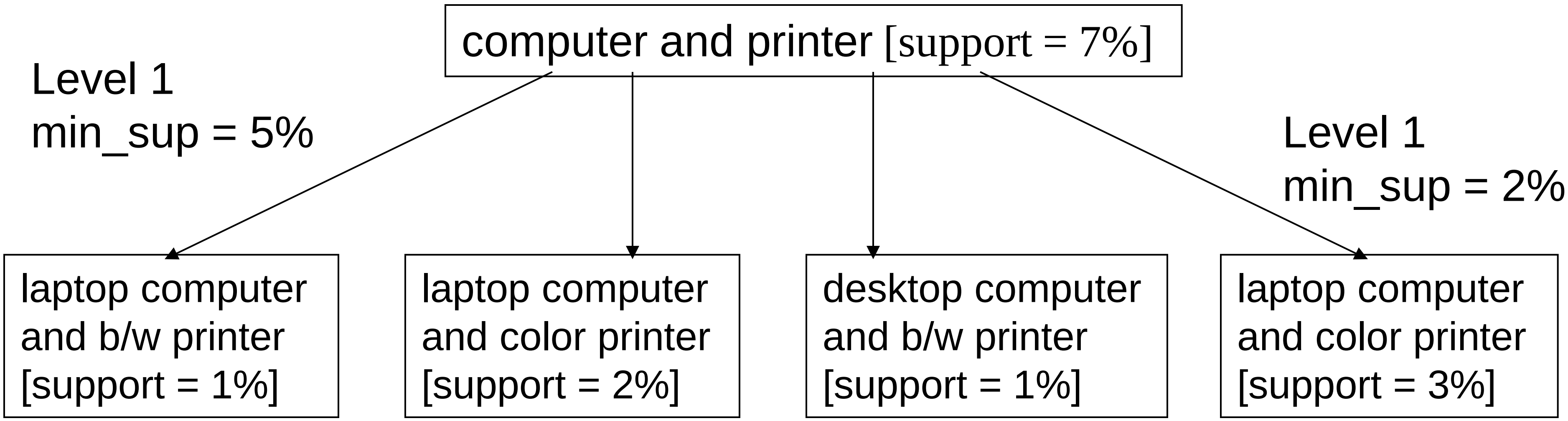


Level 1
min_sup = 12%

Level 2
min_sup = 3%

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



computer and printer [support = 7%]

Level 1
min_sup = 5%

Level 1
min_sup = 2%

laptop computer and b/w printer [support = 1%]

laptop computer and color printer [support = 2%]

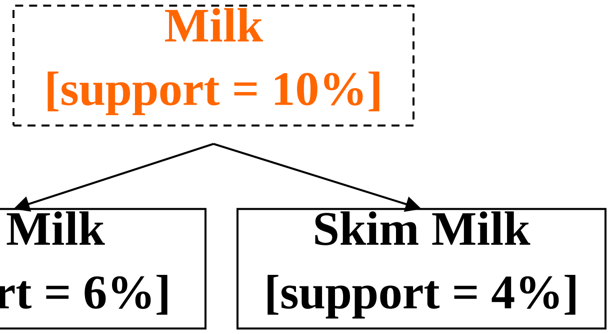
desktop computer and b/w printer [support = 1%]

laptop computer and color printer [support = 3%]

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

Level 1
min_sup = 12%
Level_passage_sup = 8%


Milk
[support = 10%]

2% Milk
[support = 6%]

Skim Milk
[support = 4%]

Level 2
min_sup = 3%

Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items
- Example
  - milk ⇒ wheat bread [support = 8%, confidence = 70%]
  - 2% milk ⇒ 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:

  buys(X, “milk”) ⇒ buys(X, “bread”)

- Multi-dimensional rules: ≥ 2 dimensions or predicates
  - Inter-dimension assoc. rules (*no repeated predicates*)

    age(X,”19-25”) ∧ occupation(X,“student”) ⇒ buys(X, “coke”)
  - hybrid-dimension assoc. rules (*repeated predicates*)

    age(X,”19-25”) ∧ buys(X, “popcorn”) ⇒ buys(X, “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

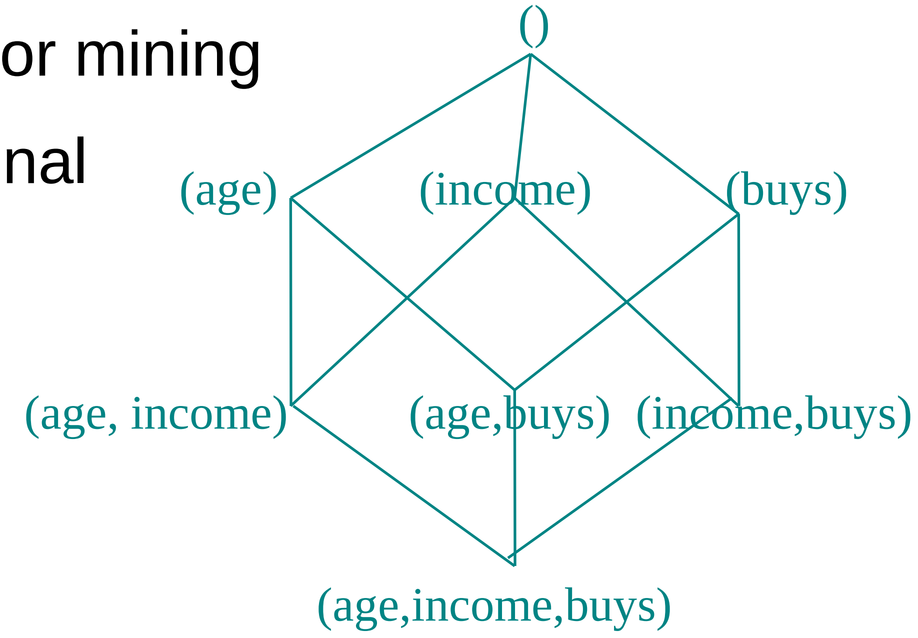
- Techniques can be categorized by how numerical attributes
  - Static discretization based on predefined concept hierarchies
  - Dynamic discretization based on data distribution
  - Clustering: Distance-based association
    - one dimensional clustering then association

Multidimensional Association Rules and 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: $A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$
- 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

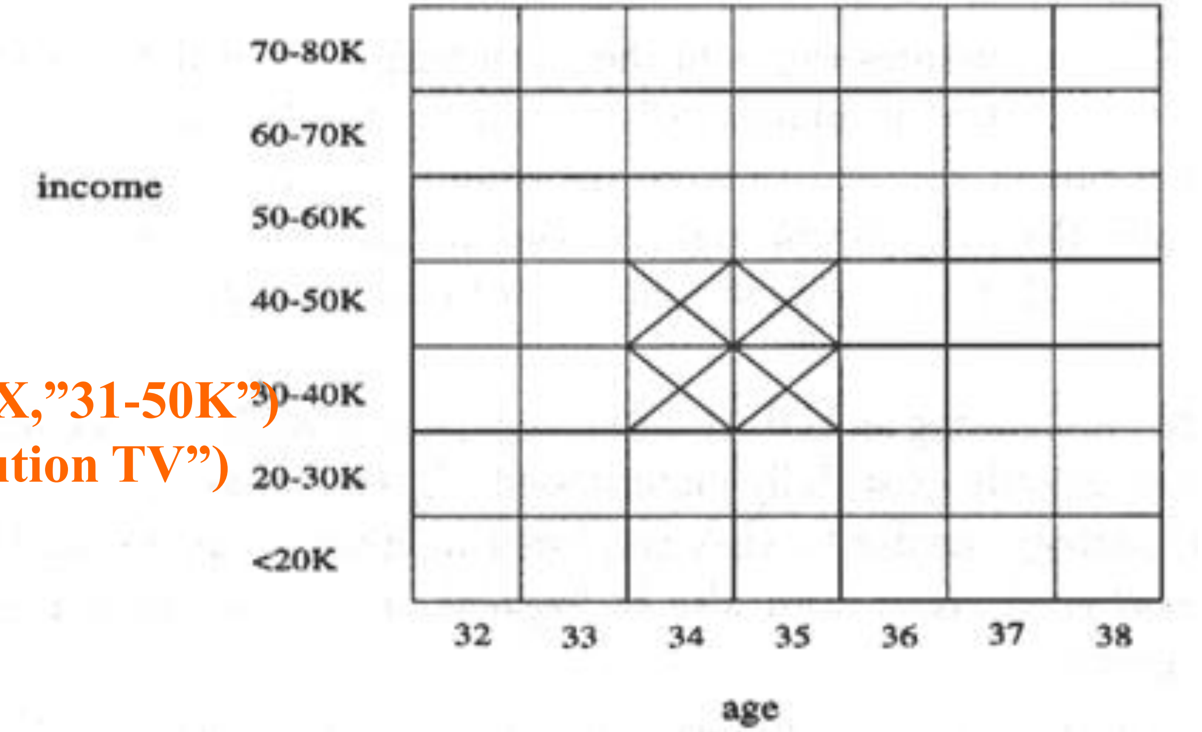
- Example

  age(X,”34”) ∧ income(X,”31-40K”) ⇒ buys(X,”high resolution TV”)

  age(X,”35”) ∧ income(X,”31-40K”) ⇒ buys(X,”high resolution TV”)

  age(X,”35”) ∧ income(X,”41-50K”) ⇒ buys(X,”high resolution TV”)

  age(X,”35”) ∧ income(X,”41-50K”) ⇒ buys(X,”high resolution TV”)



age(X,”34-35”) ∧ income(X,”31-50K”)
⇒ 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