Apriori Algorithm

James Kwok

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Association Rule Mining

1. find all frequent itemsets
   - by definition, all these itemsets satisfy *min_sup*
2. generate strong association rules
   - analyze the frequent itemsets further to extract rules that also satisfy *min_conf*

- the problem of association rule mining reduces to finding the frequent itemsets

how to discover frequent itemsets in large transactional databases?

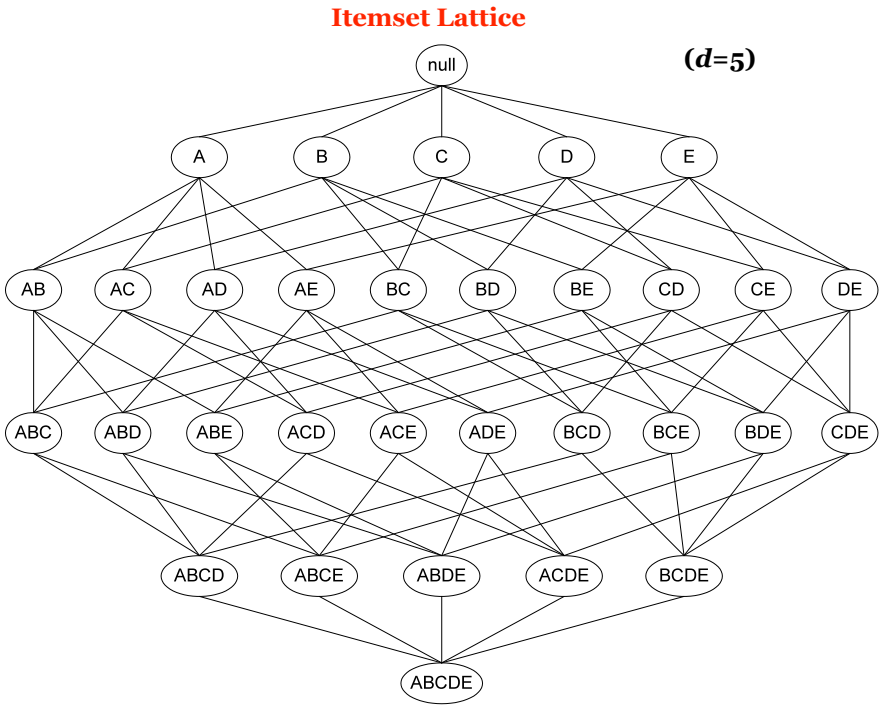
brute-force approach

1. enumerate all candidate itemsets
2. find the support count of each candidate

Brute-Force: Step 1

enumerate all candidate itemsets

- by creating an itemset lattice



- for a dataset that contains d items, the total number of (nonempty) candidate itemsets is $2^d - 1$

Brute-Force: Step 2

find the support count of each candidate

- by scanning the database and matching each transaction against the candidate

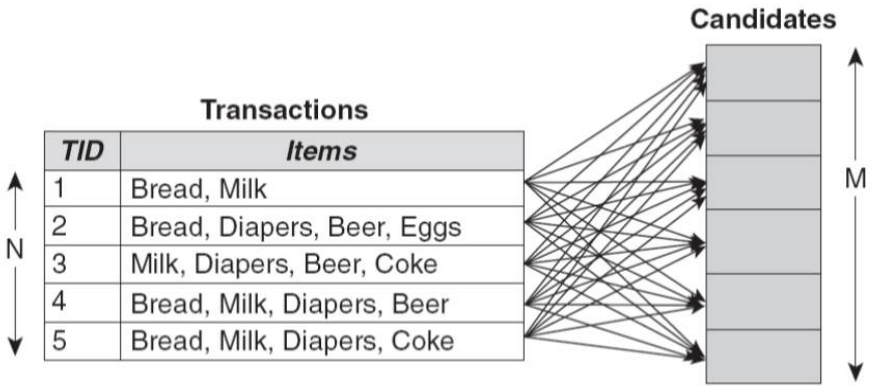


Figure 6.2. Counting the support of candidate itemsets.

for *each transaction t in database* **do**
  for *each candidate contained in t* **do**
    increment the support count;
  end
end

potential problems

- the total number of candidates can be very huge
- one transaction may contain many candidates

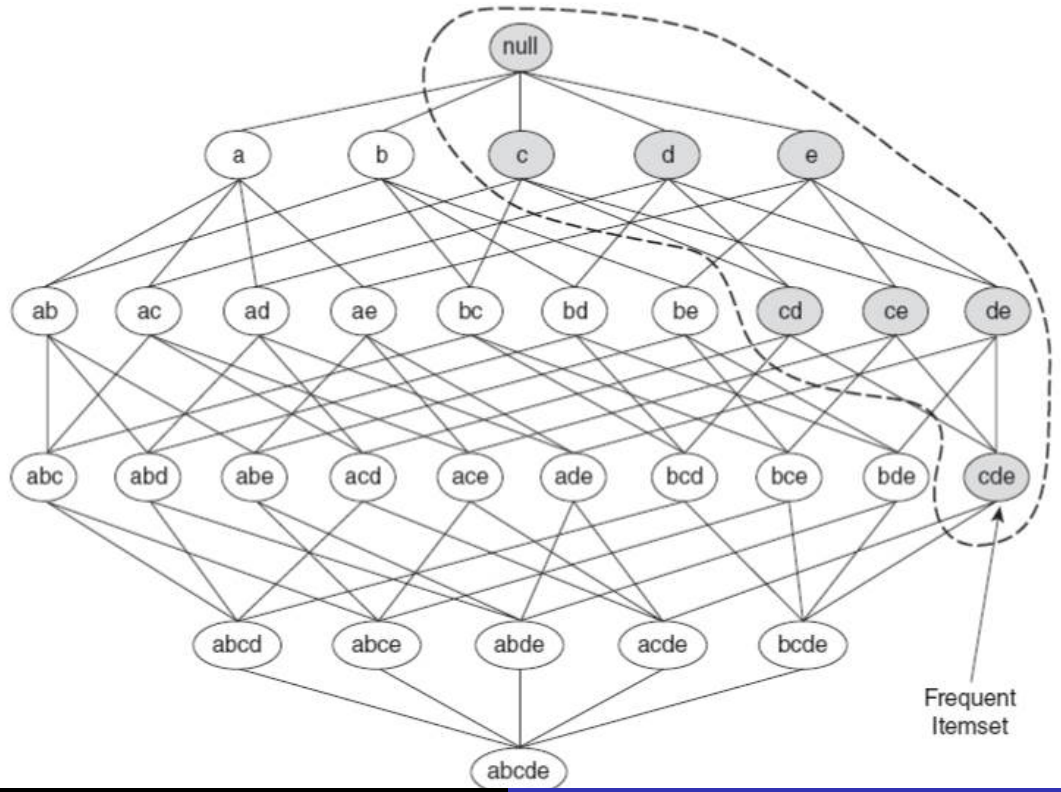
Reduce Number of Candidate Itemsets

Example

if {beer, diaper, nuts} is frequent, so is {beer, diaper}

- every transaction having {beer, diaper, nuts} also contains {beer, diaper}

Any subset of a frequent itemset must be frequent



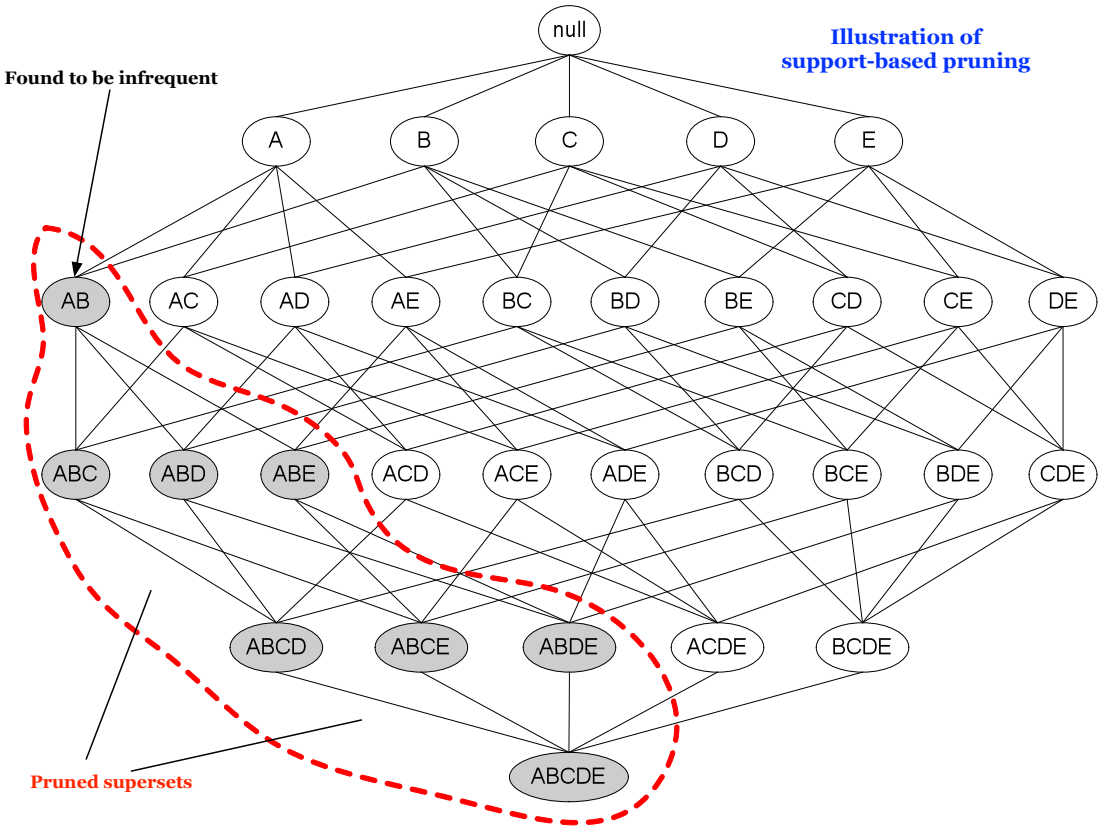
Support-Based Pruning

Antimonotone property of the support measure

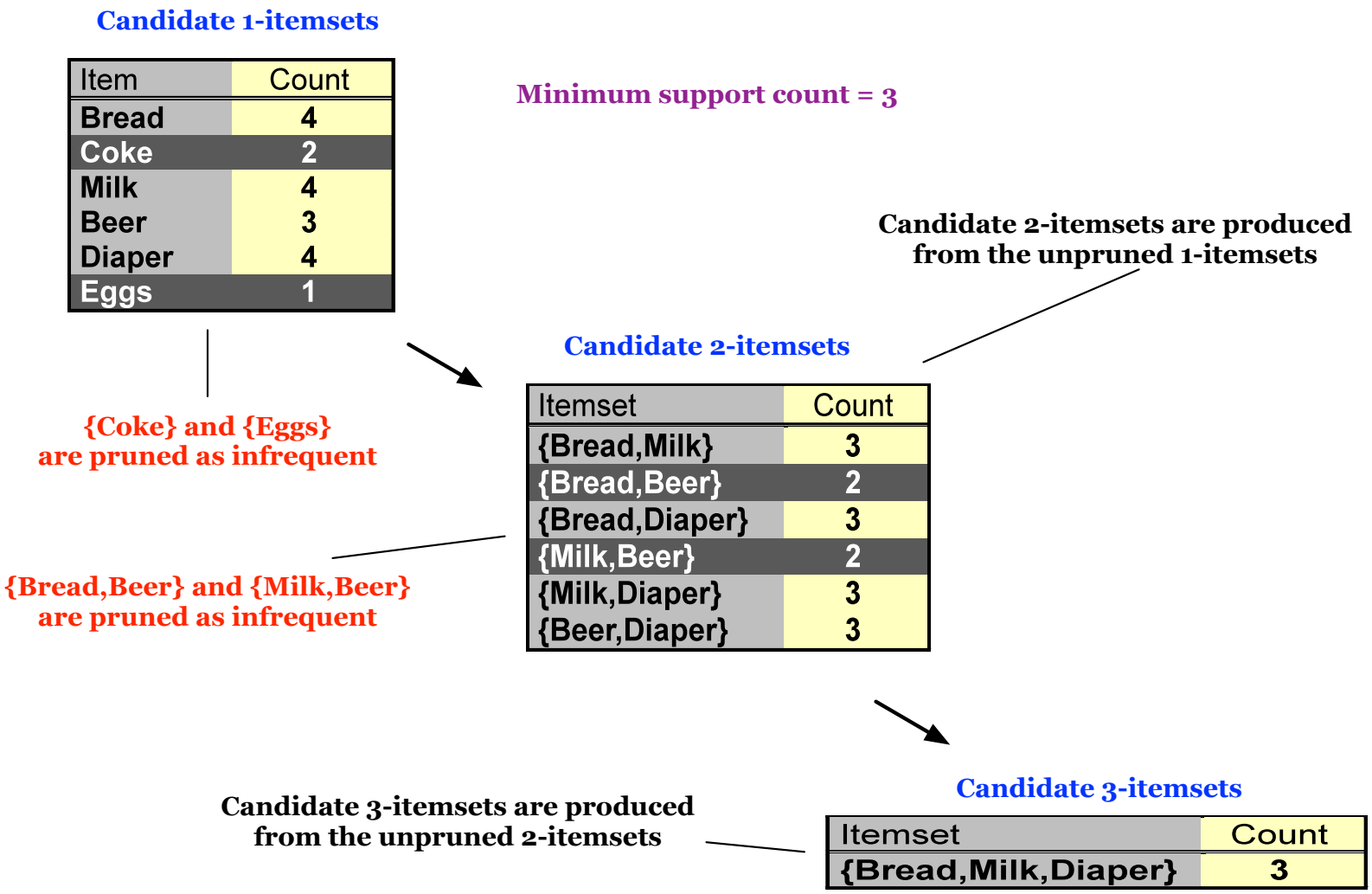
$\forall$ itemsets X, Y,

$X \subseteq Y$ implies support_count$(X) \geq$ support_count(Y)

- if there is any itemset which is infrequent, its superset should not be generated/tested!



Example

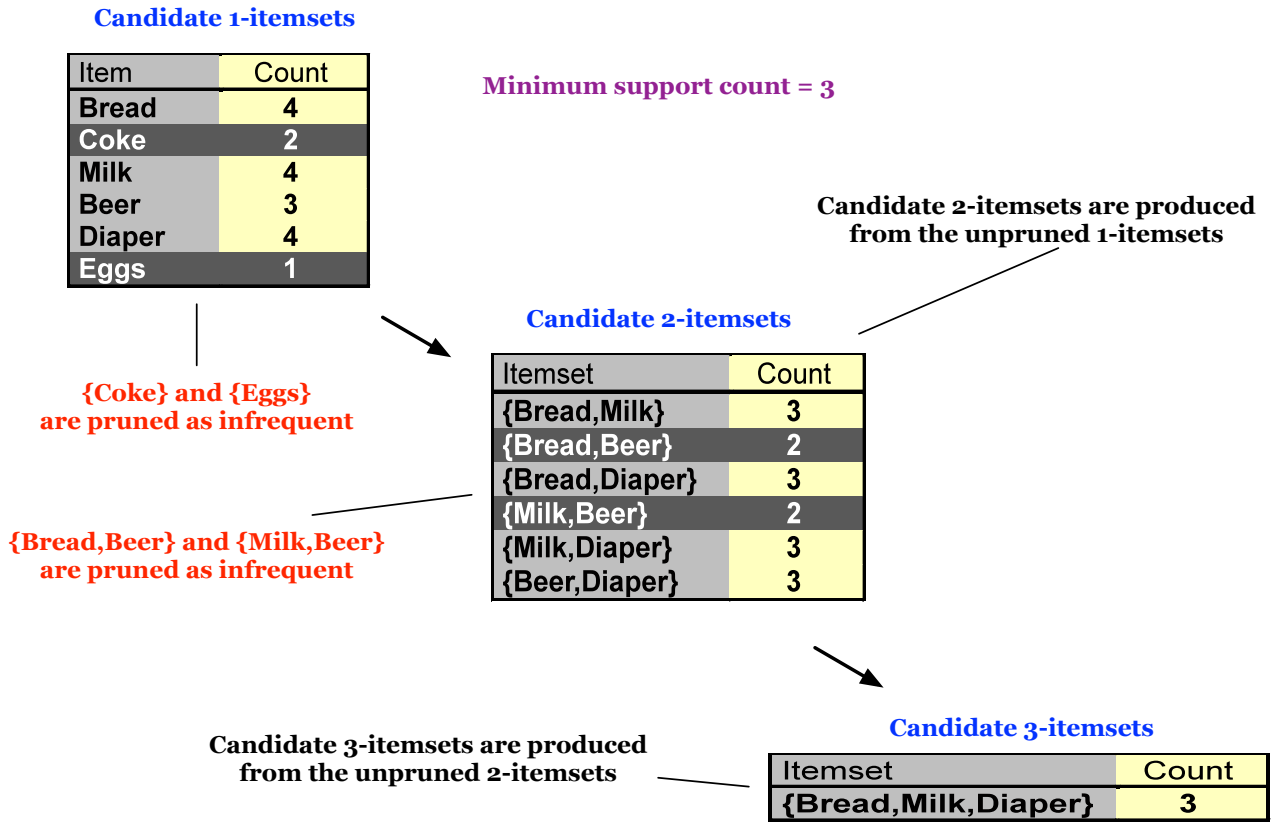

Candidate 1-itemsets
Item
Count
Bread
4
Coke
2
Milk
4
Beer
3
Diaper
4
Eggs
1
Minimum support count = 3
Candidate 2-itemsets are produced from the unpruned 1-itemsets
Candidate 2-itemsets
{Coke} and {Eggs} are pruned as infrequent
Itemset
Count
{Bread,Milk}
3
{Bread,Beer}
2
{Bread,Diaper}
3
{Milk,Beer}
2
{Milk,Diaper}
3
{Beer,Diaper}
3
{Bread,Beer} and {Milk,Beer} are pruned as infrequent
Candidate 3-itemsets
Candidate 3-itemsets are produced from the unpruned 2-itemsets
Itemset
Count
{Bread,Milk,Diaper}
3

Apriori Algorithm: Basic Idea

- Step 1:
  - enumerate the candidate 1-itemsets
  - remove the candidate 1-itemsets that are not frequent by measuring their support counts
- Step 2:
  - generate the candidate 2-itemsets, based on the remaining (frequent) 1-itemsets
  - remove the candidate 2-itemsets that are not frequent by measuring their support counts
- . . . [continue the same process until no other itemsets can be generated] . . .

The final set of frequent itemsets is the union of the itemsets that remained in every step

Effectiveness: Example



Brute-force

- enumerates all itemsets
- produce $\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 41$ candidates of size up to 3

Apriori

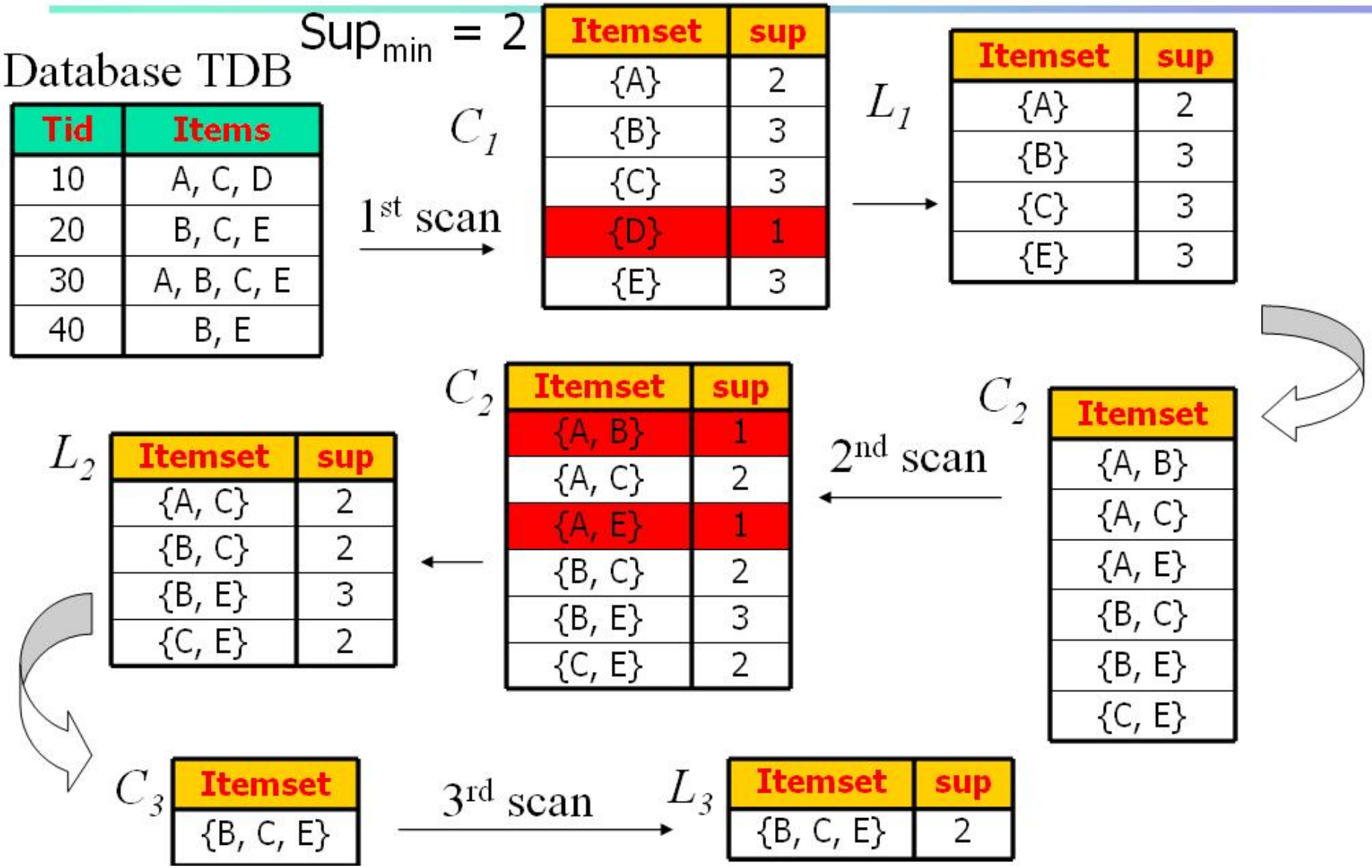
- produces $\binom{6}{1} + \binom{4}{2} + 1 = 13$ candidates of size up to 3

Pseudo-Code

- C_k: candidate itemset of size k
- L_k: frequent itemset of size k

L_1 = frequent items;
for $(k = 1; L_k \neq \emptyset; k++)$ **do**
    C_{k+1} = candidates generated from L_k;
    for *each transaction t in database* **do**
        for *each candidate in* C_{k+1} *that are contained in t* **do**
            increment the support count;
        end
    end
    L_{k+1} = candidates in C_{k+1} with min_support;
end
return $\cup_k L_k$;

Example


Database TDB
Tid | Items
10 | A, C, D
20 | B, C, E
30 | A, B, C, E
40 | B, E
Sup_min = 2
C1
1st scan
Itemset | sup
{A} | 2
{B} | 3
{C} | 3
{D} | 1
{E} | 3
L1
Itemset | sup
{A} | 2
{B} | 3
{C} | 3
{E} | 3
C2
2nd scan
Itemset | sup
{A, B} | 1
{A, C} | 2
{A, E} | 1
{B, C} | 2
{B, E} | 3
{C, E} | 2
C2
Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}
L2
Itemset | sup
{A, C} | 2
{B, C} | 2
{B, E} | 3
{C, E} | 2
C3
Itemset
{B, C, E}
3rd scan
L3
Itemset | sup
{B, C, E} | 2

Generation of Candidates (Simple Method)

recall that

- based on the frequent $(k-1)$-itemsets found in the previous iteration, we generate new candidate k-itemsets

How?

- extend every frequent $(k-1)$-itemset with other frequent items

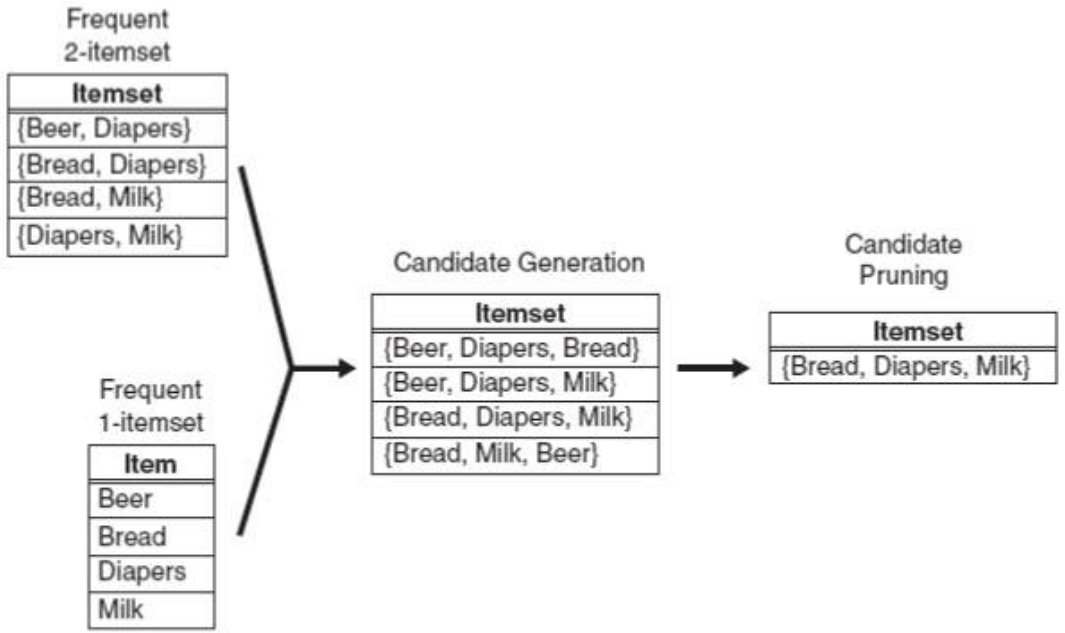


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

Duplicates

Example

$\{bread, diapers, milk\}$ can be generated by merging

- $\{bread, diapers\}$ with $\{milk\}$
- $\{bread, milk\}$ with $\{diapers\}$
- $\{diapers, milk\}$ with $\{bread\}$

How to avoid duplicates?

Avoiding Duplicates

- sort the items in lexicographic order

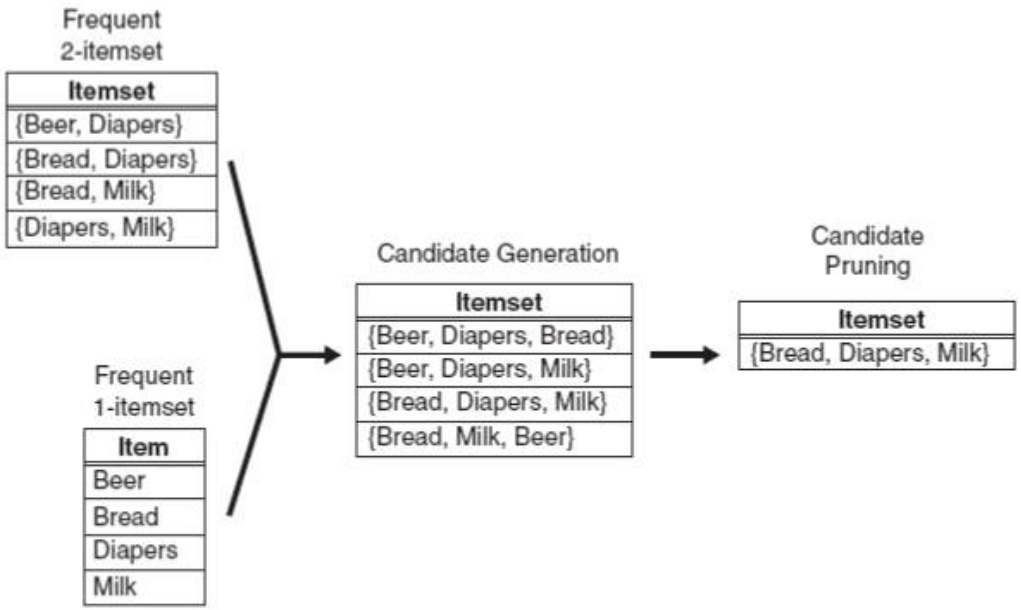


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

- itemset X is extended only by frequent items that are lexicographically larger than the items in X
  - e.g., $\{bread, diapers\}$ can be augmented with $\{milk\}$, but $\{diapers, milk\}$ cannot be augmented with $\{bread\}$

Candidate Pruning

- may still produce a large number of unnecessary candidates

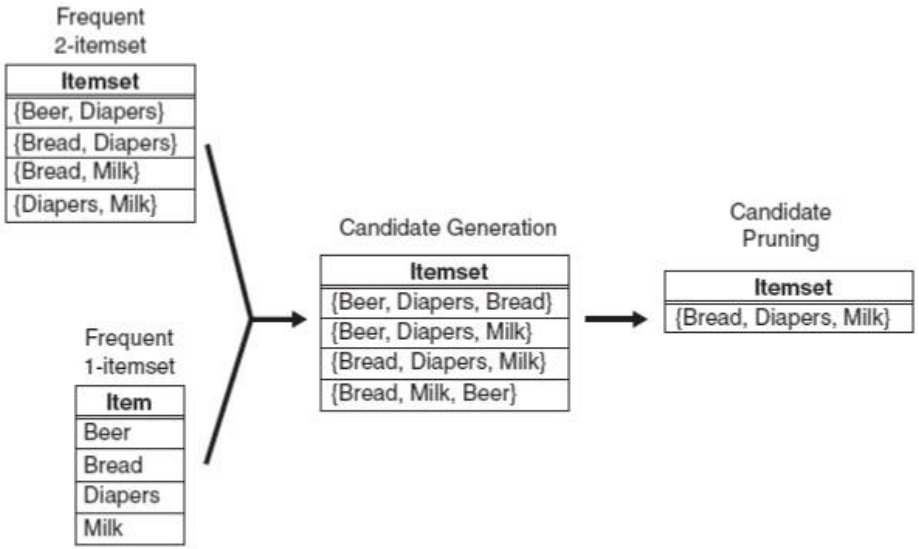


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

from support-based pruning

- for $\{beer, diapers, milk\}$ to be a candidate 3-itemset, $\{beer, diapers\}$, $\{diapers, milk\}$ and $\{beer, milk\}$ should be frequent 2-itemsets
- $\{beer, diapers, milk\}$ cannot be a frequent 3-itemset because $\{beer, milk\}$ is infrequent

Generation of Candidates in Apriori

(self-join) Generate candidate k-itemsets by merging frequent $(k-1)$-itemsets

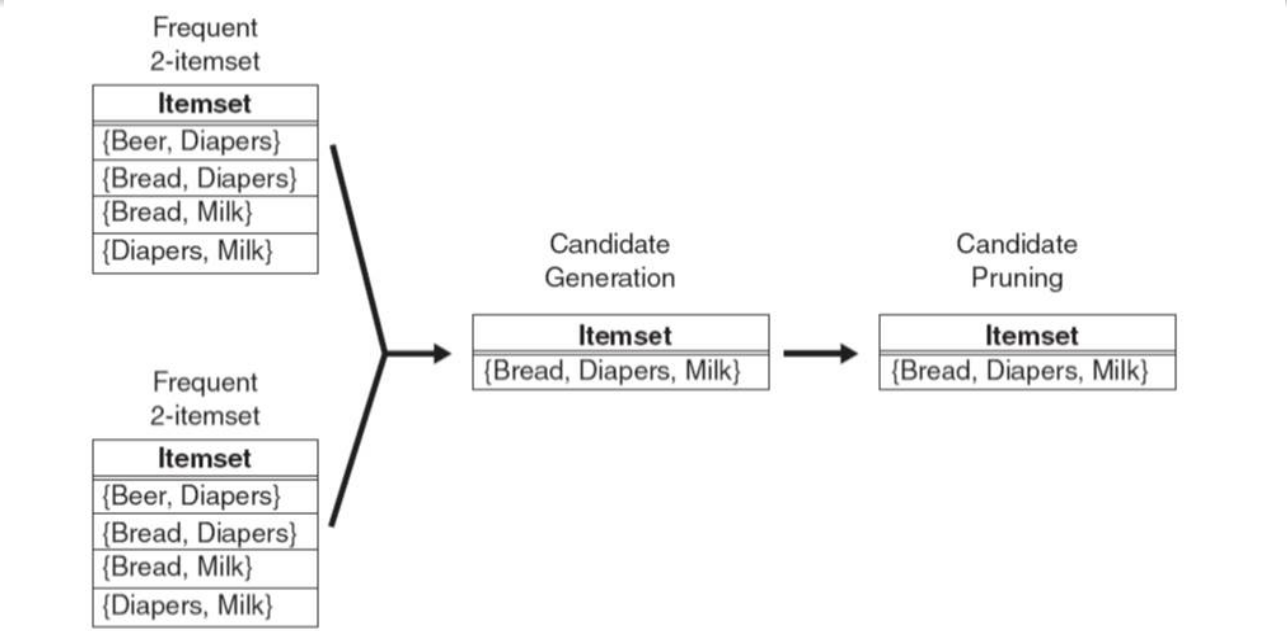


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

- A, B: any pair of $(k-1)$-frequent itemsets
  - $A = \{a_1, a_2, \ldots, a_{k-1}\}$ and $B = \{b_1, b_2, \ldots, b_{k-1}\}$
- merge A and B if and only if their first $(k-2)$ items are identical
  - i.e., if $a_i = b_i$ for $i = 1, 2, \ldots, k-2$ and $a_{k-1} < b_{k-1}$ (avoid duplicates), then form $\{a_1, a_2, \ldots, a_{k-1}, b_{k-1}\}$

Example ($\{beer, diapers, milk\}$)

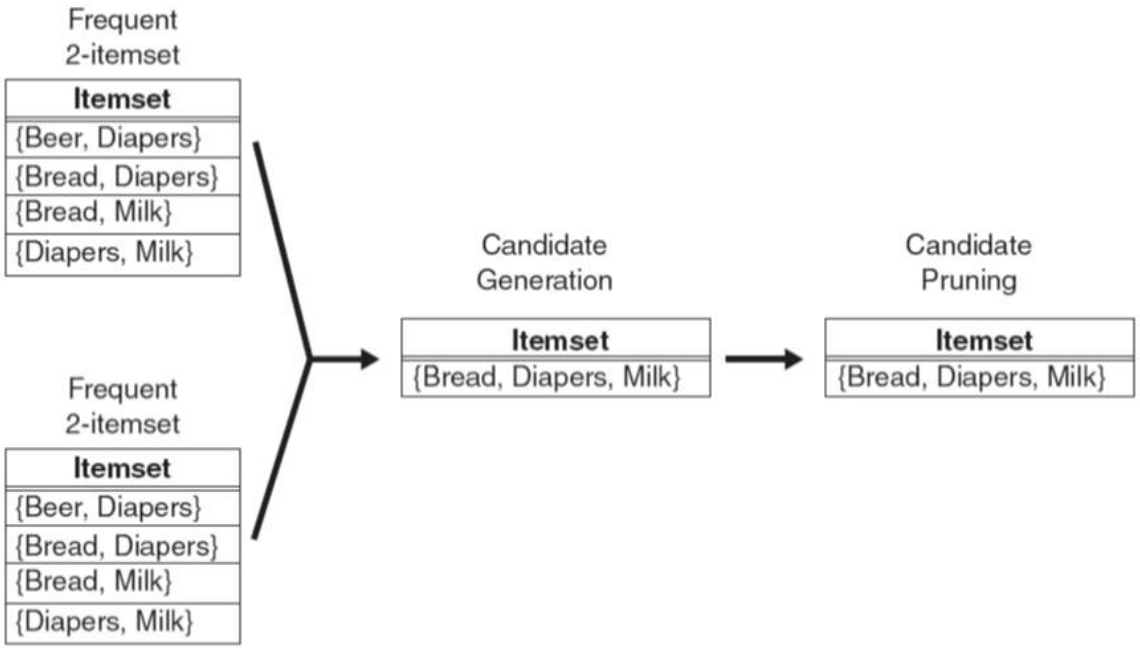


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

- does not have to merge $\{beer, diapers\}$ with $\{diapers, milk\}$ because the first item in both itemsets are different
- if $\{beer, diapers, milk\}$ is a frequent itemset, it would have been obtained by merging $\{beer, diapers\}$ with $\{beer, milk\}$ instead

Candidate Pruning

- remove an itemset if it contains a $(k-1)$-itemset that is not frequent

Example

- frequent 3-itemsets: $\{abc, abd, acd, ace, bcd\}$
- candidate 4-itemset: $\{abcd, acde\}$
  - $abcd$ from abc and abd
  - $acde$ from acd and ace
- candidate pruning $\rightarrow \{abcd\}$
  - $acde$ is removed because ade is not a frequent 3-itemset

Do we need to check if the candidate contains a

- $(k-2)$-itemset,
- $(k-3)$-itemset, ...

that is not frequent?