HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

582364 Data mining, 4 cu
Lecture 3: Association analysis

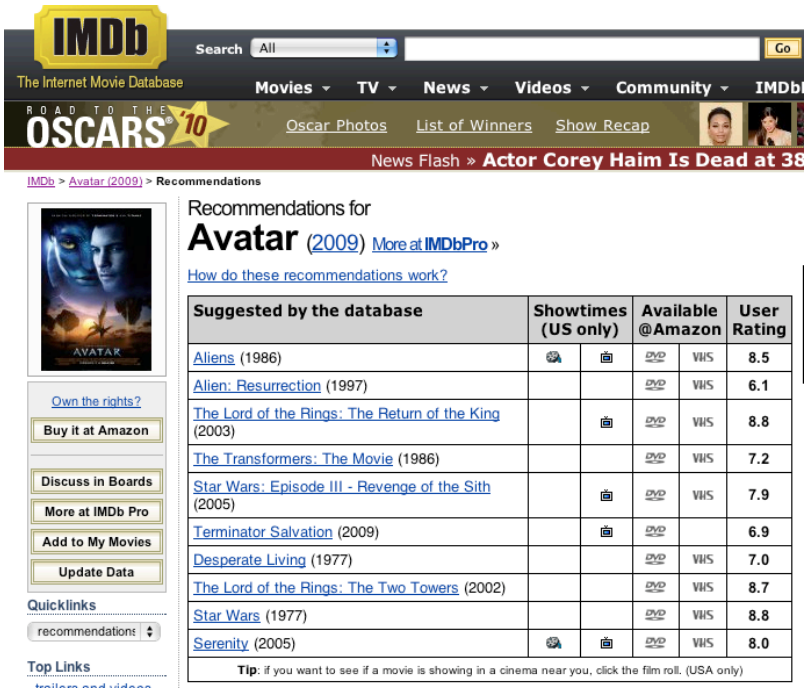
Spring 2010

Lecturer: Juho Rousu

Teaching assistant: Taru Itäpelto

Frequent Pattern Discovery Application: Recommendation systems

- Many recommendation web sites are based on collecting data on frequently co-occurring items
- If you liked the film “Aliens” you probably like “Avatar”
- People that buy book X, frequently buy book Y



Suggested by the database	Showtimes (US only)		Available @Amazon		User Rating
Aliens (1986)					8.5
Alien: Resurrection (1997)					6.1
The Lord of the Rings: The Return of the King (2003)					8.8
The Transformers: The Movie (1986)					7.2
Star Wars: Episode III - Revenge of the Sith (2005)					7.9
Terminator Salvation (2009)					6.9
Desperate Living (1977)					7.0
The Lord of the Rings: The Two Towers (2002)					8.7
Star Wars (1977)					8.8
Serenity (2005)					8.0

Association analysis

Goal: Given a set of transactions, find

- items that occur frequently together (Frequent itemsets)
  - "Introduction to Data Mining" & "Elements of Statistical Learning " are frequently bought together
- rules that will predict the occurrence of an item based on the occurrences of other items in the transaction (Association rules)
  - People that bought "Introduction to Data Mining", often buy "Elements of Statistical Learning " as well

Definition: Itemset

- **Itemset**
  - A collection of one or more items
    - Example: {Milk, Bread, Diaper}
  - k-itemset
    - An itemset that contains k items
- **Support**
  - **Support count (σ):** Count of occurrences of an itemset
    - E.g. σ({Milk, Bread,Diaper}) = 2
  - **Support:** Fraction of transactions that contain an itemset
    - E.g. s({Milk, Bread, Diaper}) = 2/5

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Definition: Frequent Itemset

- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a *minsup* threshold
- Example:
  - We set minsup = 0.5
  - Frequent itemsets:
    - Bread (support = 0.8 > 0.5)
    - Milk (support = 0.8)
    - Diaper (support = 0.8)
    - Beer (support = 0.6)
    - {Bread, Milk} (support = 0.6)
    - {Bread, Diaper} (support = 0.6)
    - {Beer, Diaper} (support = 0.6)
    - {Milk, Diaper} (support = 0.6)

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Definition: Association Rule

- An expression of the form

  $X \rightarrow Y$,

  where X and Y are itemsets
- Semantics: “When X happens, Y frequently happens as well”
- Examples:

  {Milk, Diaper} → {Beer}

  {“Introduction to Data Mining”} → {“Elements of Statistical Learning”}
- Different from
  - logical implication: “When X happens, Y always happens as well”
  - causal relation: “X causes Y to happen”

Definition: Association Rule

- The *strength* of an association rule X → Y
  is measured by its *support* and *confidence*
- Support s(X → Y): Fraction of transactions that contain both the set X and the set Y
  - s(X → Y) = σ(X U Y)/|T|
- Confidence c(X → Y): how often items in Y appear in transactions that contain X
  - c(X → Y) = σ(X U Y)/σ(X)

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|\text{T}|} = \frac{2}{5} = 0.4$$

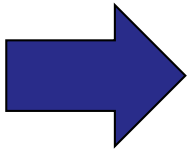
$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

minsup =0.4
minconf = 0.5



{Bread}→{Milk} (s=0.6,c=0.75)
{Bread}→{Diaper} (s=0.6,c=0.75)
…
{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Association Rule Mining: brute-force approach

- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds
- How much time this would take?
  - For d unique items, there are $3^d - 2^{d+1} + 1$ possible rules
    - For d = 100, this gives ca. 5×10^{47} possible rules to check against the database
    - Compare to age of universe ca. 5×10^{17} seconds
- Will not work except for toy examples

Mining Association Rules

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

- All the above rules are binary partitions of the same itemset:
  {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support ≥ minsup
       - e.g. {Milk, Diaper, Beer}
  2. Rule Generation
     - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

- Frequent itemset generation is still computationally expensive
  - There are 2^d itemsets, which is still too much to enumerate

Frequent Itemset Generation Strategies

- Reduce the number of candidate itemsets (M)
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
- Reduce the number of comparisons (NM)
  - Use efficient data structures to store the candidate itemsets or transactions
  - No need to match every candidate against every transaction

Reducing the number of candidate itemsets: Apriori principle

- **Apriori principle:**
  - If an itemset is frequent, then all of its subsets must also be frequent
  - i.e. if {Milk,Diaper,Beer} is frequent, then

    {Milk,Diaper}, {Milk,Beer}, {Diaper,Beer},{Milk},{Diaper},{Beer} must also be frequent
- Why this is true (informally):
  - In *every* transaction, subsets *always* occur if the whole set occurs
  - Support of the itemset is given by the sum of occurrences over the transactions
- Converse does not hold:
  - Even though all subsets are frequent, an itemset may be infrequent
  - Need to check against the transactions to find the true support

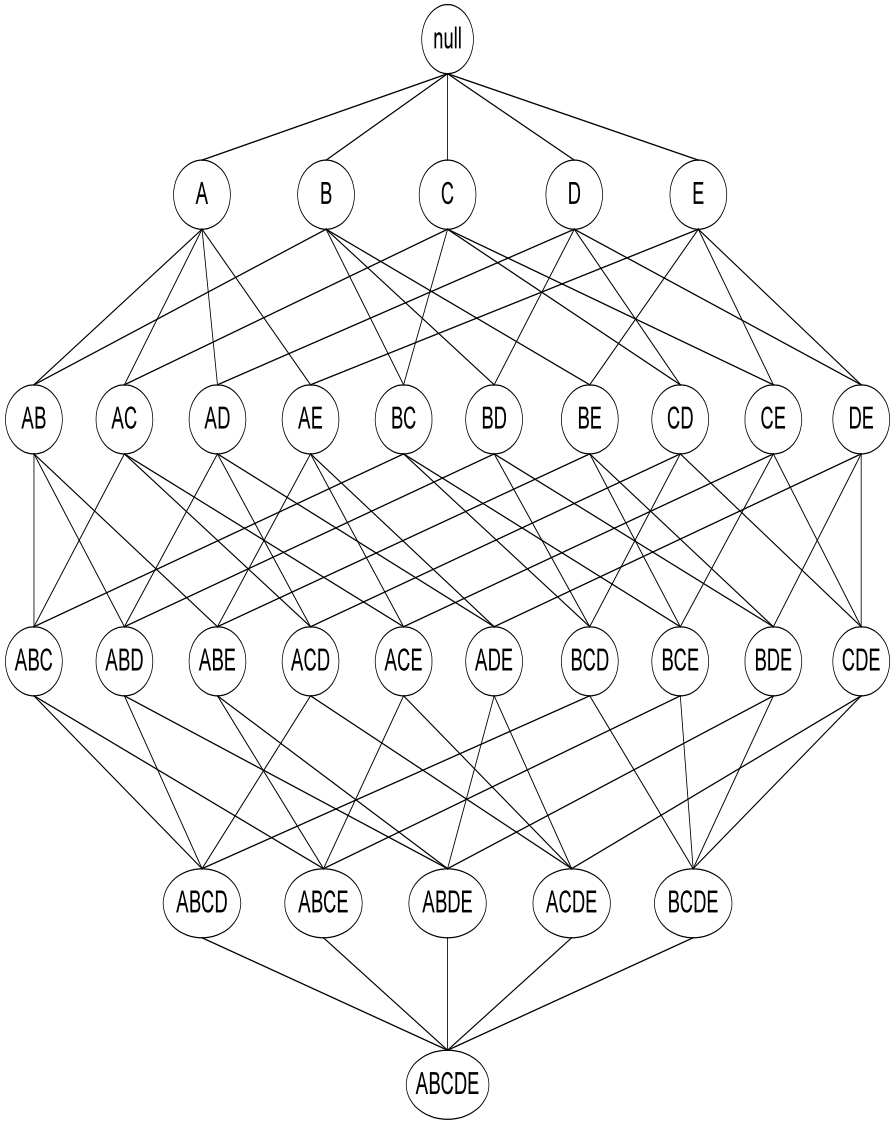
Reducing the number of candidate itemsets: Apriori principle

- **Apriori principle:**
  - If an itemset is frequent, then all of its subsets must also be frequent
- Formally, the principle follows from the anti-monotone property of the support function:
  - When we put more items into the itemset, the support can only decrease

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

Itemset lattice

- Itemsets that can be constructed from a set of items have a *partial order* with respect to the subset operator
  - i.e. a set is larger than its proper subsets
- This induces a lattice where nodes correspond to itemsets and arcs correspond to the subset relation
- The lattice is called the *itemset lattice*
- For d items, the size of the lattice is 2^d



Frequent itemsets on the itemset lattice

- The Apriori principle is illustrated on the itemset lattice
  - The subsets of a frequent itemset are frequent
  - They span a sublattice of the original lattice (the grey area)

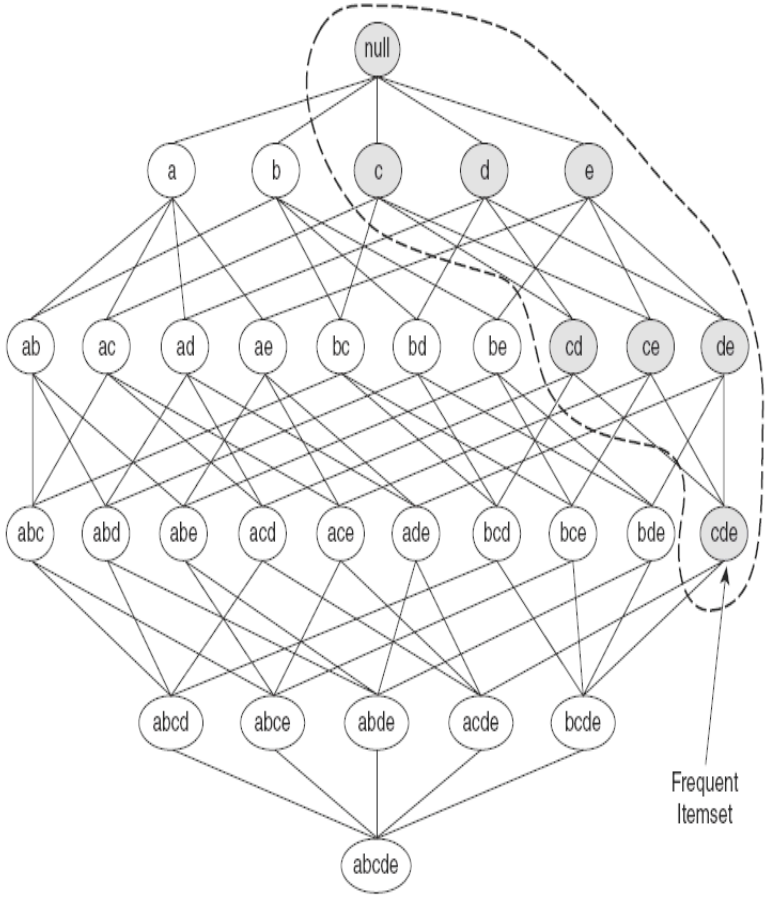


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Frequent itemsets on the itemset lattice

- Conversely
  - The supersets of an infrequent itemset are infrequent
  - They also span a sublattice of the original lattice (the crossed out nodes)
- If we know that {a,b} is infrequent, we never need to check any of the supersets
  - This fact is used in *support-based pruning*

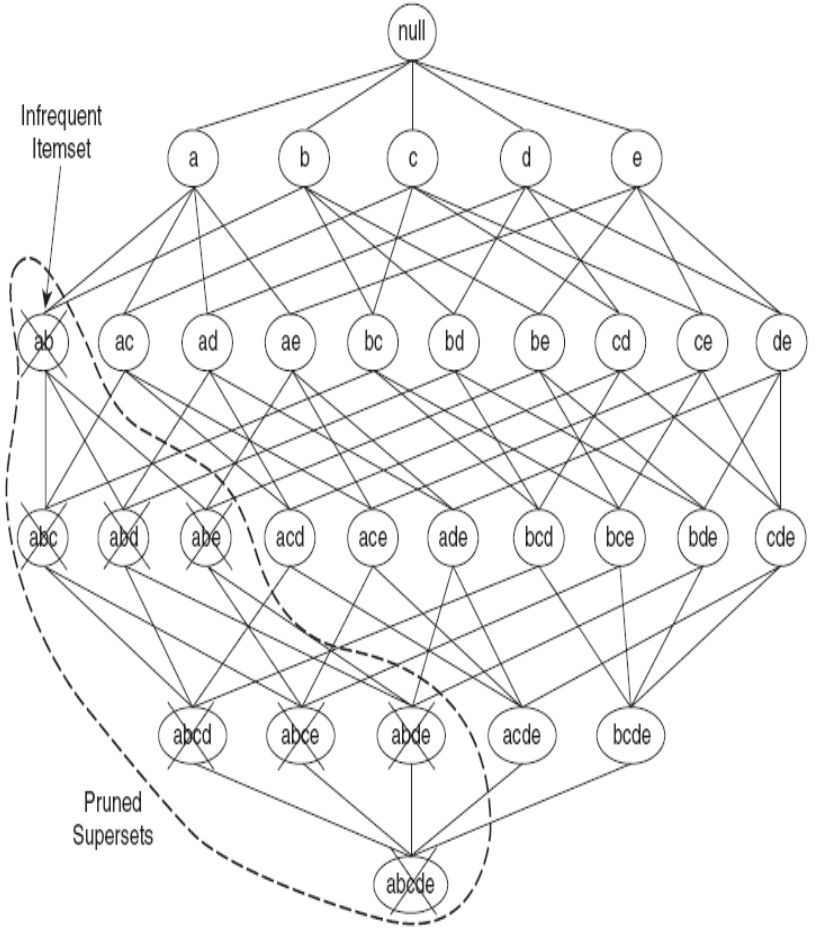


Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

Frequent items set generation in Apriori Algorithm

- Input: set of items I, set of transactions T, number of transactions N, minimum support *minsup*
- Output: frequent k-itemsets F_k, k=1,...
- Method:
  - k=1
  - Compute support for each 1-itemset (item) by scanning the transactions
  - F_1 = items that have support above *minsup*
  - Repeat until no new frequent itemsets are identified
    1. C_{k+1} = candidate *k+1* -itemsets generated from length k frequent itemsets F_k
    2. Compute the support of each candidate in C_{k+1} by scanning the transactions T
    3. F_{k+1} = Candidates in C_{k+1} that have support above *minsup.*

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

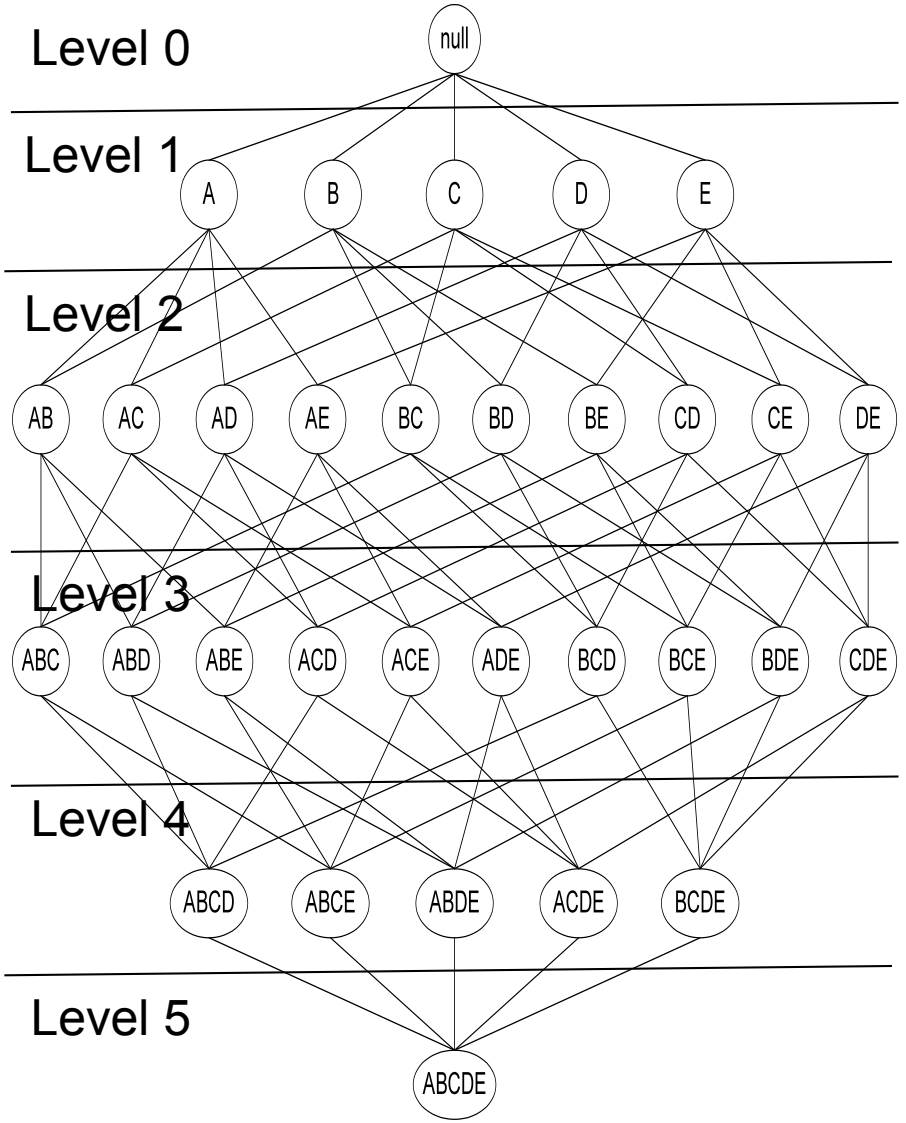
Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 1 = 13$

Characteristics of Apriori algorithm

- Level-wise search algorithm:
  - traverses the itemset lattice level-by-level (1-itemsets, 2-itemsets ...)
- Generate-and-test strategy:
  - new candidate itemsets are generated from smaller frequent itemsets
  - support is tested to weed out infrequent itemsets



Requirements for candidate generation step

1. It should not generate too many unnecessary candidates, i.e. itemsets where at least one of the subsets is infrequent
2. It should ensure the completeness of the candidate set, i.e. no frequent itemset is left out
3. It should not generate the same candidate set more than once
   - e.g. {a,b,c,d} can be generated by merging {a,b,c} with {d} or {b,d} with {a,c}, {a,b} with {c,d}

Candidate generation strategies: F_{k-1} x F_1 method

- F_{k-1} x F_1 method: Combine frequent *k-1* –itemsets with frequent 1-itemsets

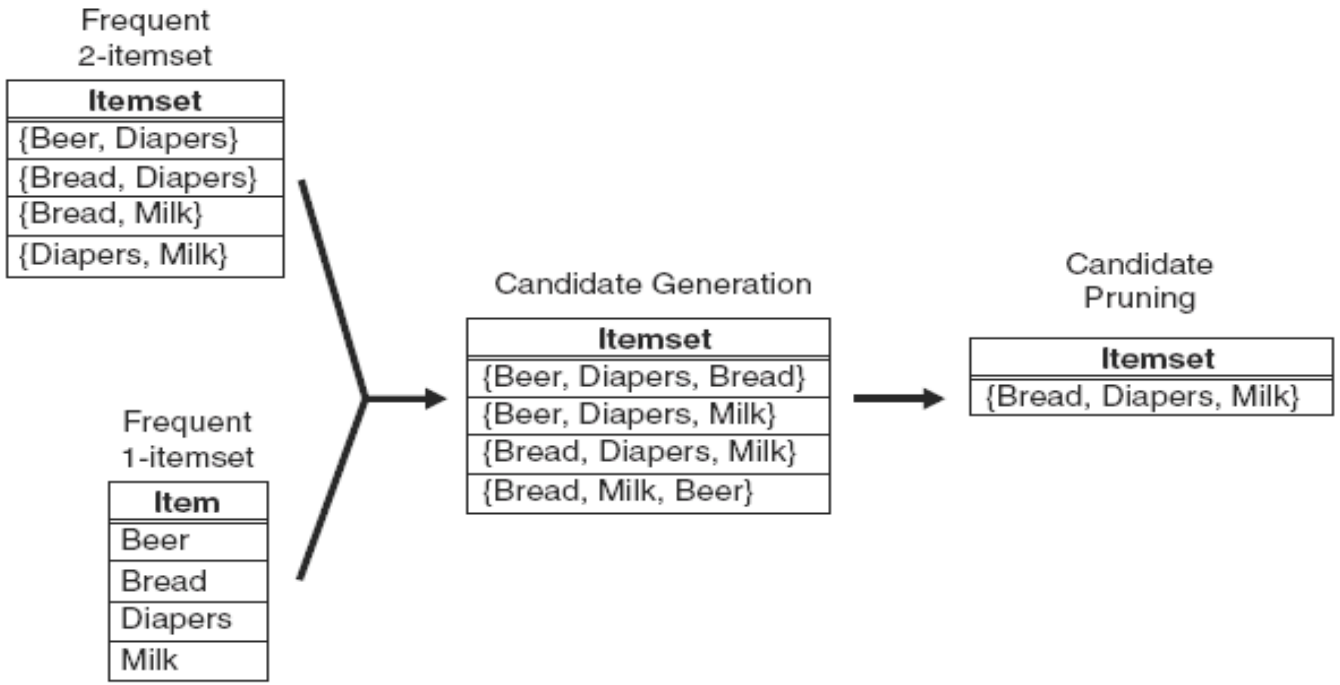


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

Candidate generation strategies: F_{k-1} x F_1 method

- Satisfaction of our requirements (#1-3):

1. while many *k*-itemsets are left ungenerated, can still generate unnecessary candidates ✖
   - e.g. merging {Beer, Diapers} with {Milk} is unnecessary, since {Beer, Milk} is infrequent

2. method is complete: each frequent itemset consists of a frequent *k-1* –itemset and a frequent 1-itemset ✔

Candidate generation strategies: F_{k-1} x F_1 method

3. can generate the same set twice (✖)
   - e.g. {Bread, Diapers, Milk} can be generated by merging {Bread,Diapers} with {Milk} or {Bread,Milk} with {Diapers} or {Diapers, Milk} with {Bread}
   - This can be circumvented by keeping all frequent itemsets in their lexicographical order (✔):
     - e.g. {Bread,Diapers} can be merged with {Milk} as 'Milk' comes after 'Bread' and 'Diapers' in lexicographical order
     - {Diapers, Milk} is not merged with {Bread}, {Bread, Milk} is not merged with {Diapers} as that would violate the lexicographical ordering

Candidate generation strategies: F_{k-1} x F_{k-1} method

- F_{k-1} x F_{k-1} method: Combine a frequent *k-1* –itemset with another frequent *k-1* -itemset

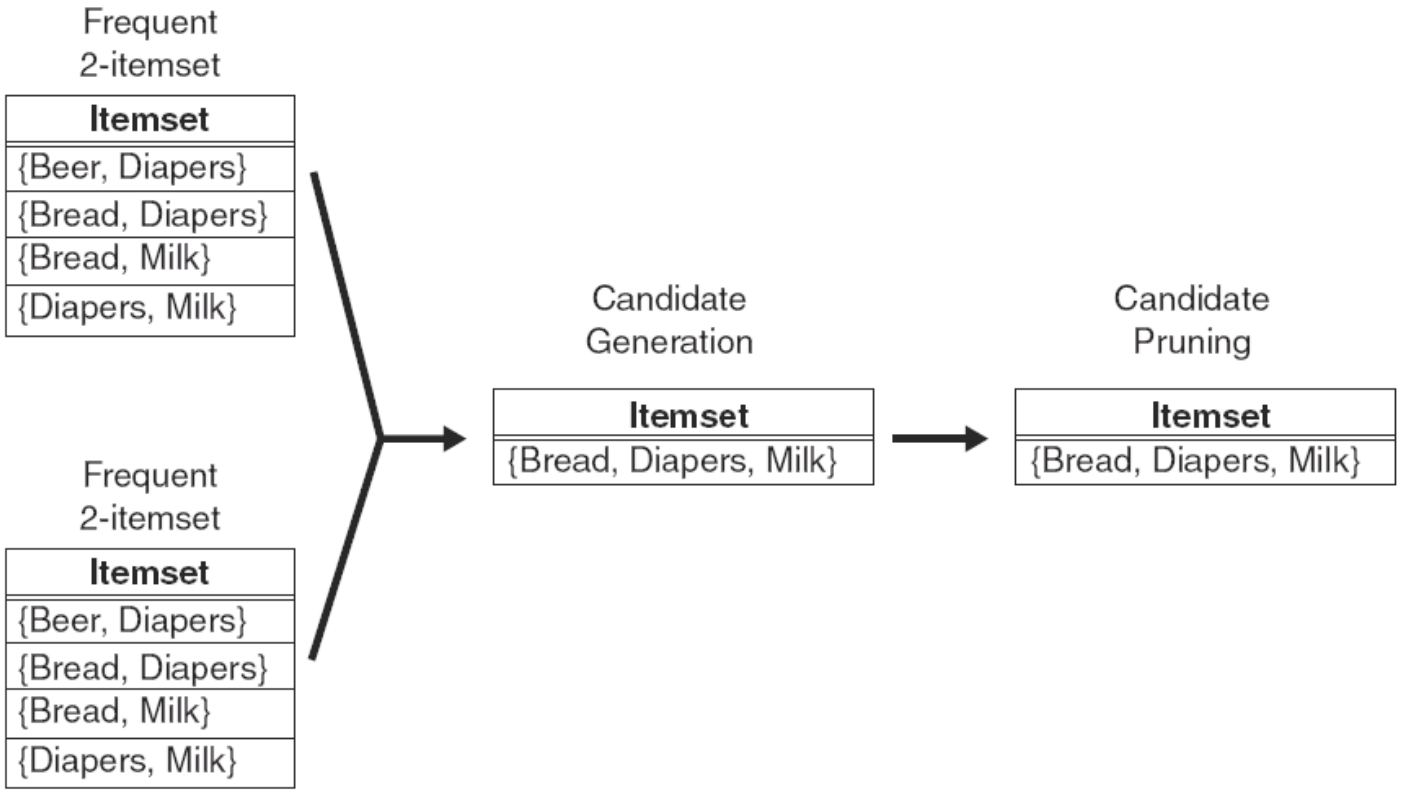


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

Candidate generation strategies: F_{k-1} x F_{k-1} method

- Items are stored in lexicographical order in the itemset
- When considering merging, only pairs that share first k-2 items are considered
  - e.g. {Bread, Diapers} is merged with {Bread,Milk}
  - if the pairs share fewer than k-2 items, the resulting itemset would be larger than k, so we do not need to generate it yet
- The resulting *k*-itemset has *k* subsets of size *k-1*, which will be checked against support threshold
  - The merging ensures that at least two of the subsets are frequent
  - An additional check is made that the remaining *k-2* subsets are frequent as well

Candidate generation strategies: F_{k-1} x F_{k-1} method

- Satisfaction of our requirements (#1-3):

1. avoids the generation of many unnecessary candidates that are generated by the F_{k-1} x F_1 method
   - e.g. will not generate {Beer, Diapers, Milk} as {Beer, Milk} is infrequent
2. method is complete: every frequent k-itemset can be formed of two frequent k-1 –itemsets differing in their last item.
3. each candidate itemset is generated only once

Support counting

- Given the candidate itemsets C_k and the set of transactions T, we need to compute the support counts σ(X) for each itemset X in C_k
- Brute-force algorithm would compare each transaction against each itemset → large amount of comparisons
- An alternative approach
  - divide the candidate itemsets C_k into buckets by using a hash function
  - for each transaction t:
    - hash the itemsets contained in t into buckets using the same hash function
    - compare the corresponding buckets of candidates and the transaction
    - increment the support counts of each matching candidate itemset
  - A hash tree is used to implement the hash function

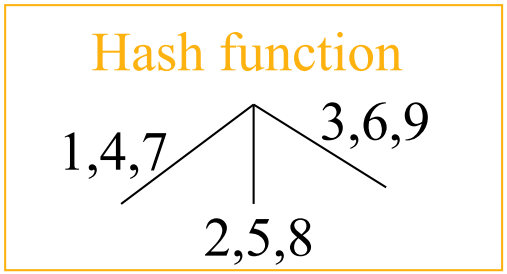
Generate Hash Tree

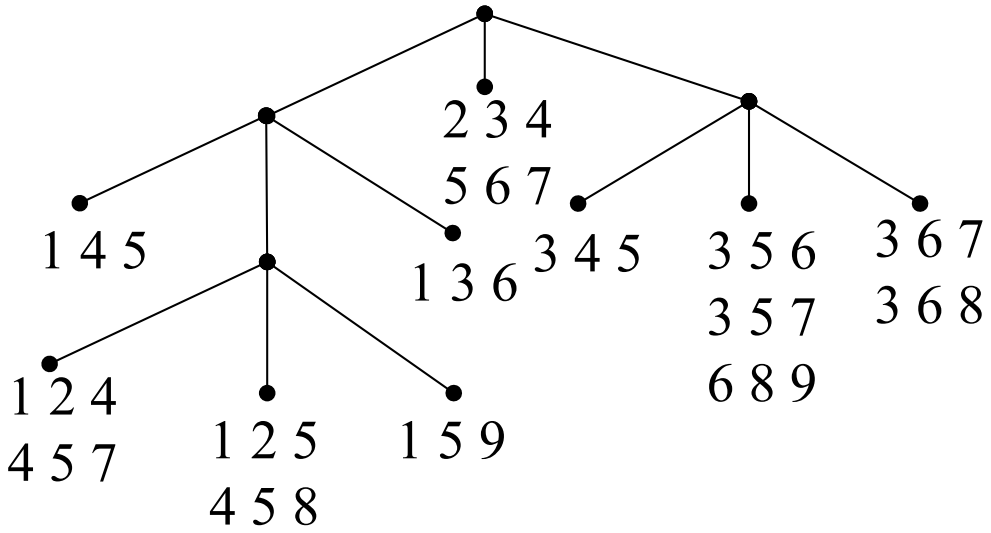
Suppose you have 15 candidate itemsets of length 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

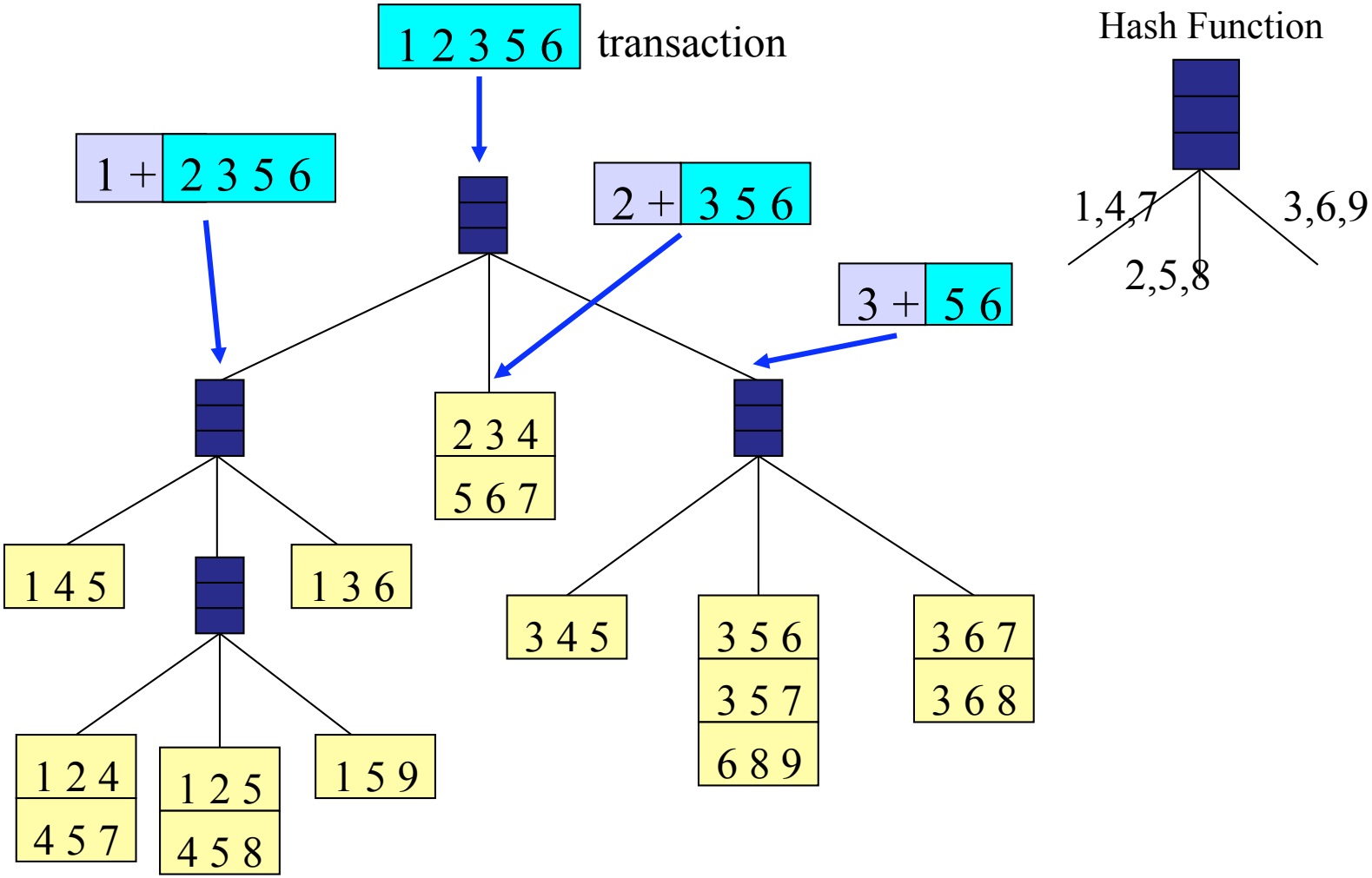
You need:

- Hash function e.g. h(p) = p mod 3
- Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

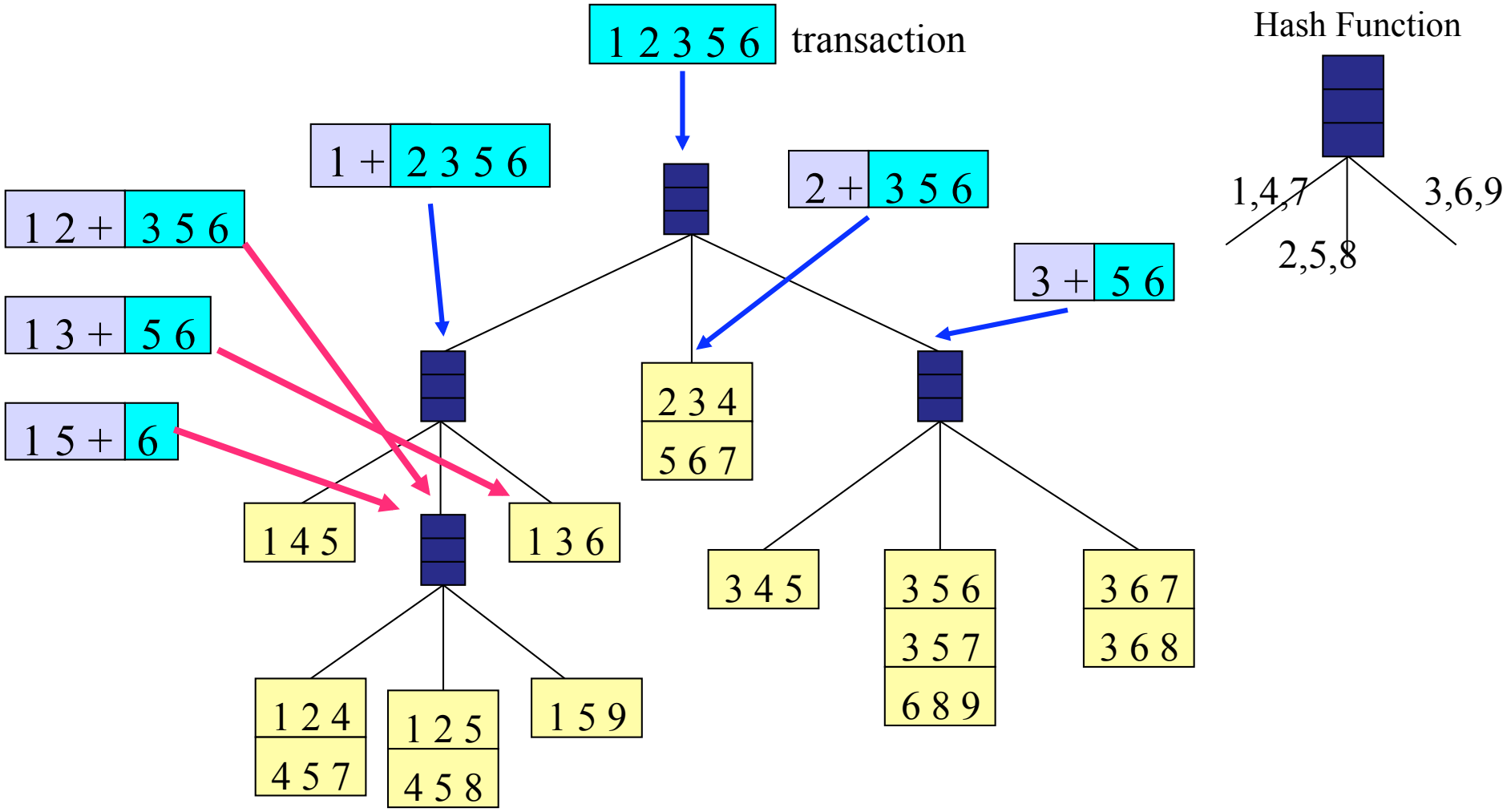




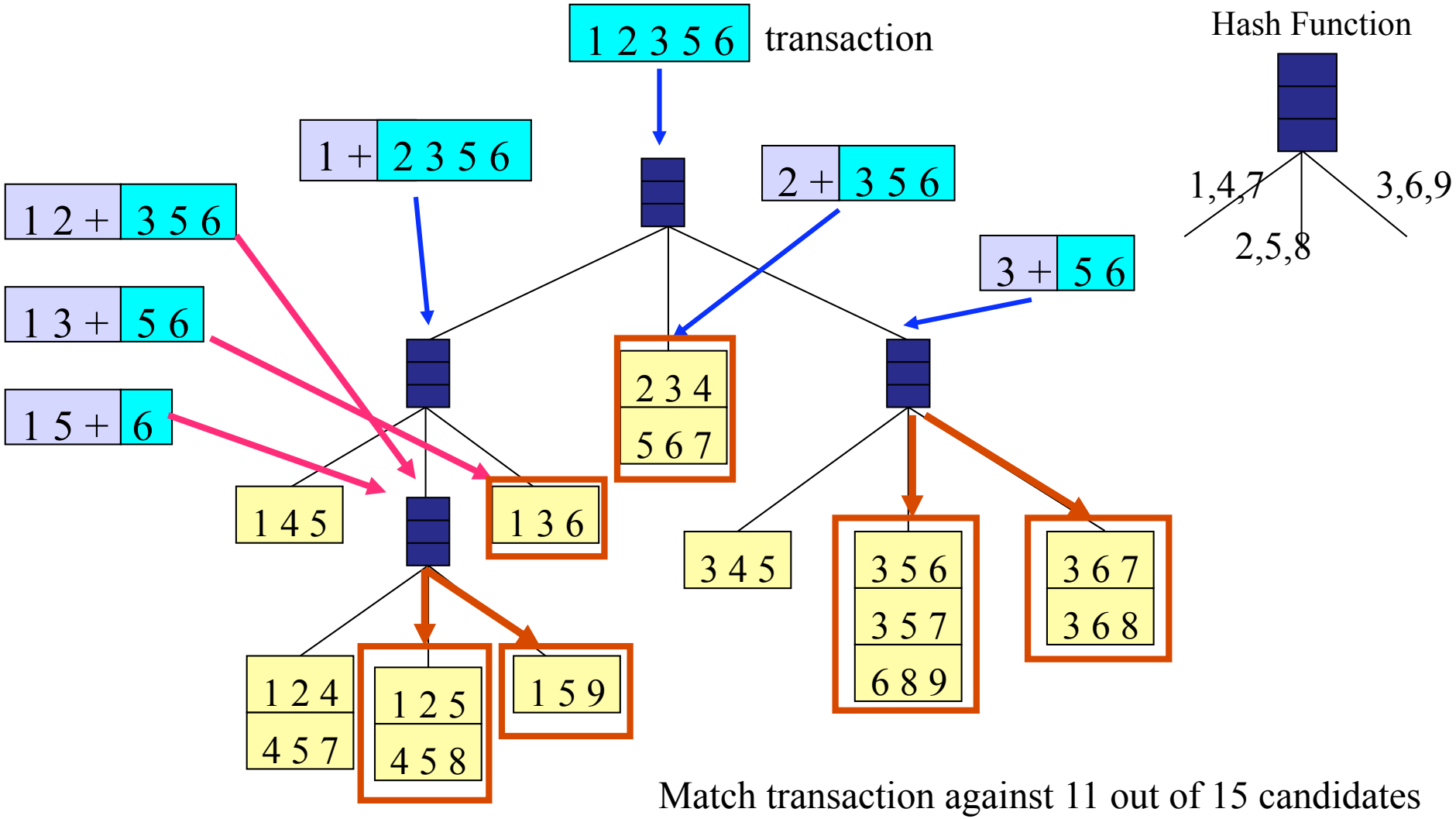
Matching the transaction against candidates



Matching the transaction against candidates



Matching the transaction against candidates



Rule generation in Apriori

- From a frequent itemset, we still need to generate the association rules
- From each k-itemset, one can produce 2^k-2 association rules
  - e.g. {Beer,Bread,Diapers} generates {Beer} → {Bread,Diapers}, {Bread} → {Beer, Diapers}, Diapers → {Beer,Bread}, {Beer,Bread} -> {Diapers}, {Beer,Diapers} → {Bread}, {Bread, Diapers} → {Beer}
- All of the association rules generated from the same frequent itemset have the same support
- The confidence of the rules will be different, however
- We want to find efficiently the rules that have high confidence

Rule generation in Apriori

- The confidence of any association rule can be computed from the support counts of the frequent itemsets:
  - c(Beer -> Bread,Diapers) = σ(Beer,Bread,Diapers)/σ(Beer)
  - We don't need to scan the transactions to find the high-confidence rules
- The confidence does not have a similar anti-monotone property as support has:
  - e.g. if c({Beer,Milk} → {Bread,Diapers}) exceed the confidence threshold *minconf,* it does not follow that {Beer} → {Bread} satisfies the confidence threshold

Rule generation in Apriori

- However, between rules generated from the same frequent itemset we have the following property: *if X → Y-X does not satisfy the confidence threshold then no rule X' → Y-X', where X' is a subset of X, satisfies the confidence threshold*
  - e.g. if {Beer,Bread}→ {Milk} does not satisfy the confidence threshold, {Beer} → {Bread,Milk} and {Bread} →{Beer,Milk} also do not
- This property follows from the anti-monotone property of the support count $\sigma(X') \geq \sigma(X)$, thus

  $c(X \rightarrow Y\text{-}X) = \sigma(X \cup Y\text{-}X)/\sigma(X) \geq \sigma(X' \cup Y\text{-}X')/\sigma(X') = c(X' \rightarrow Y\text{-}X')$

Rule generation in Apriori

- Apriori uses level-wise search for rule generation
- It starts from empty right-hand side and all items in the left-hand side
- To generate a rule in the next level it merges the left-hand sides of two confident rules on the previous level
- When a non-confident rule is found, an entire subgraph (grey area) is pruned

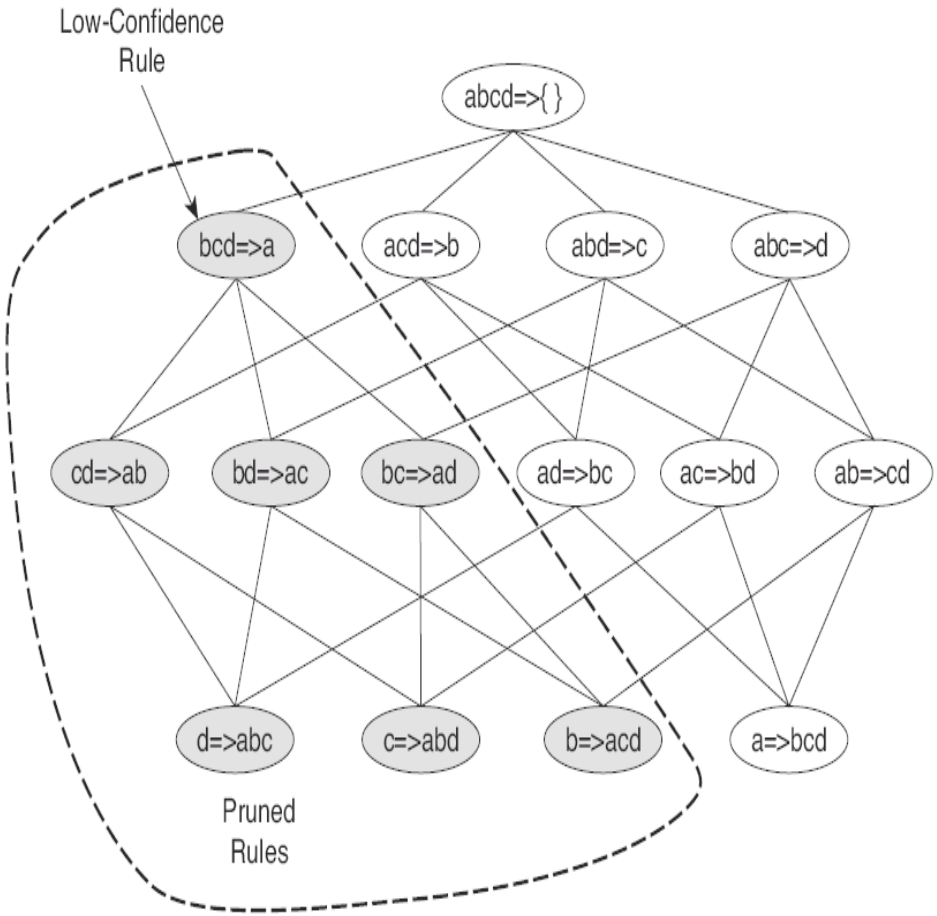


Figure 6.15. Pruning of association rules using the confidence measure.