Data Mining:

2. Assoziationsanalyse

A) Frequent Itemsets

Transaction Data

- A special type of record data, where
  - each record (a transaction) involves a set of items.
  - For example, consider a grocery store.
  - The set of products purchased by a customer during one shopping trip constitute a “transaction” or ”market basket” [Warenkorb], while the individual products that were purchased are the items.

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

Association Rule Mining

- Given a set (a database) of transactions T, find rules that will describe (and hopefully predict) the occurrence of an item based on the occurrences of other items in the transaction.

Market-Basket Transactions

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 Association Rules

$\{Diaper\} \rightarrow \{Beer\}$,
$\{Milk, Bread\} \rightarrow \{Eggs, Coke\}$,
$\{Beer, Bread\} \rightarrow \{Milk\}$

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

- **Itemset**
  - A collection of one or more items
  - Example: {Milk, Bread, Diaper}
- **k-Itemset**
  - An itemset that contains k items
- **Support count (σ)** (of X in T)
  - Frequency of occurrence of an itemset X
  - E.g. σ({Milk, Bread, Diaper}) = 2
- **Support (*s*)** (of X in T)
  - Fraction of transactions that contain an itemset X
  - E.g. *s*({Milk, Bread, Diaper}) = 2/5
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a *minsup* 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

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets
  - Example: {Milk, Diaper} → {Beer}

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

- **Rule Evaluation Metrics**
  - Support s (of $X \rightarrow Y$ in T)
    - Fraction of transactions that contain both X and Y = s(XuY)
  - Confidence c (of $X \rightarrow Y$ in T)
    - Measures how often all items of Y appear in transactions that contain X
    - estimates conditional probability of Y given X (P(Y|X))

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 (interesting rules only)
  - confidence ≥ *minconf* threshold (reliable rules only)

- 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

  ⇒ *Computationally prohibitive!*

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)

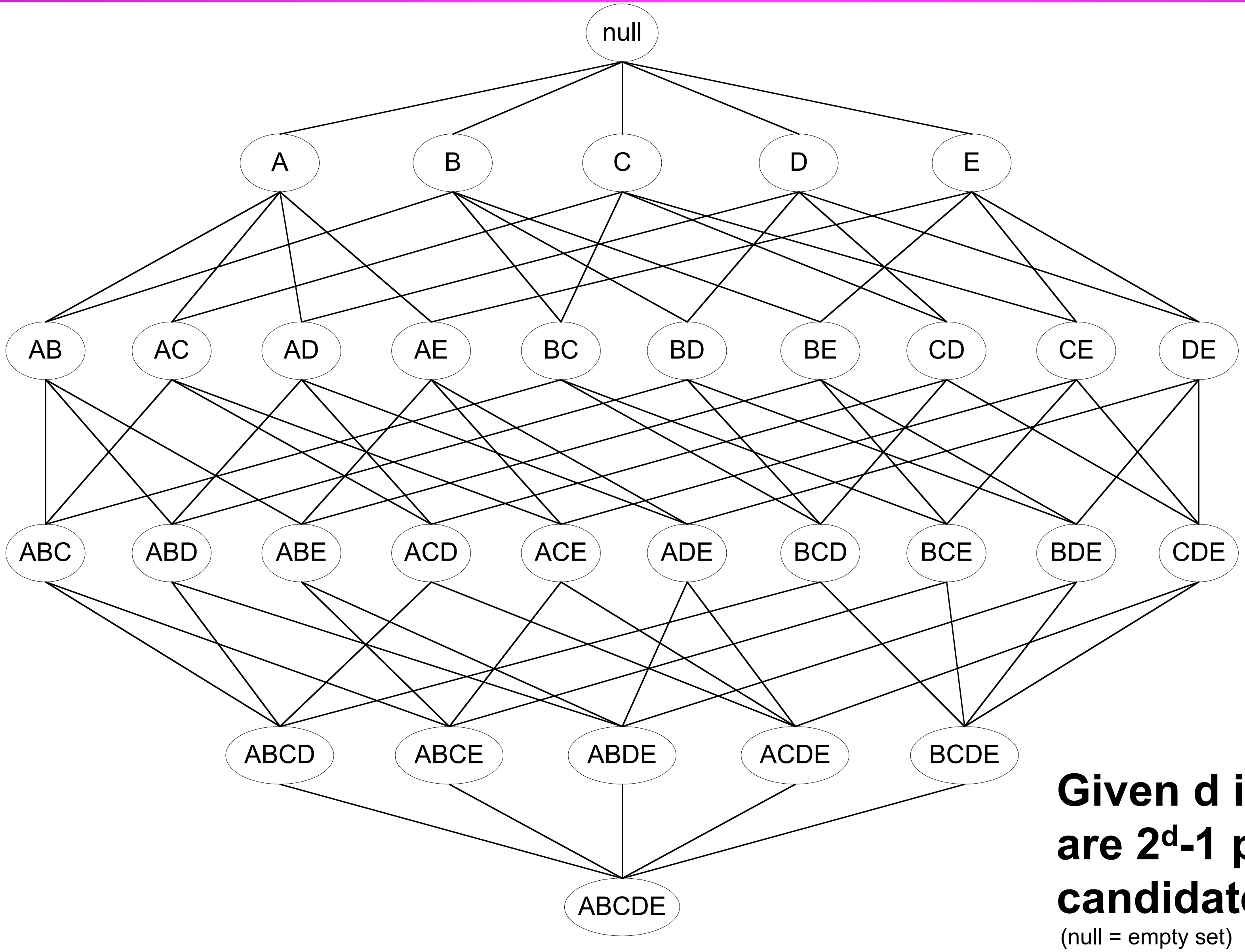
Observations:

- 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
- If the itemset is infrequent, all such rules have low support, and can be pruned without checking confidence

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     – Generate all itemsets whose support ≥ minsup
  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

Frequent Itemset Generation

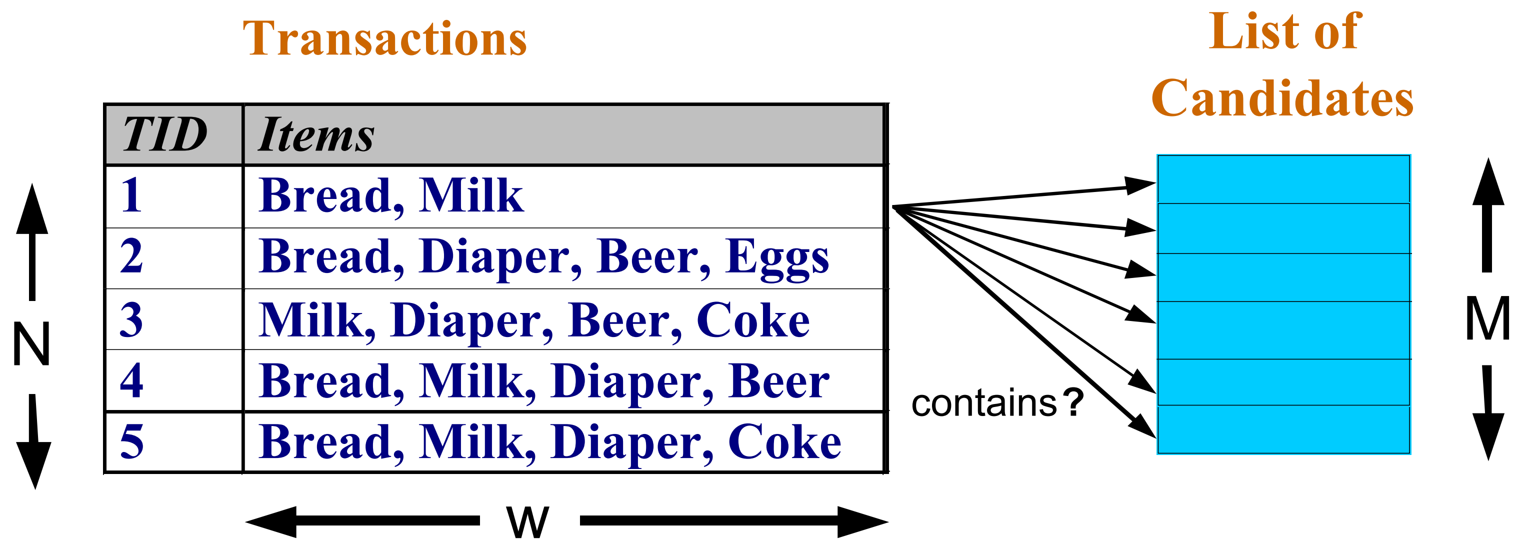


Given d items, there are 2^d-1 possible candidate itemsets

(null = empty set)

Frequent Itemset Generation

- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the transactions database and the list of candidates

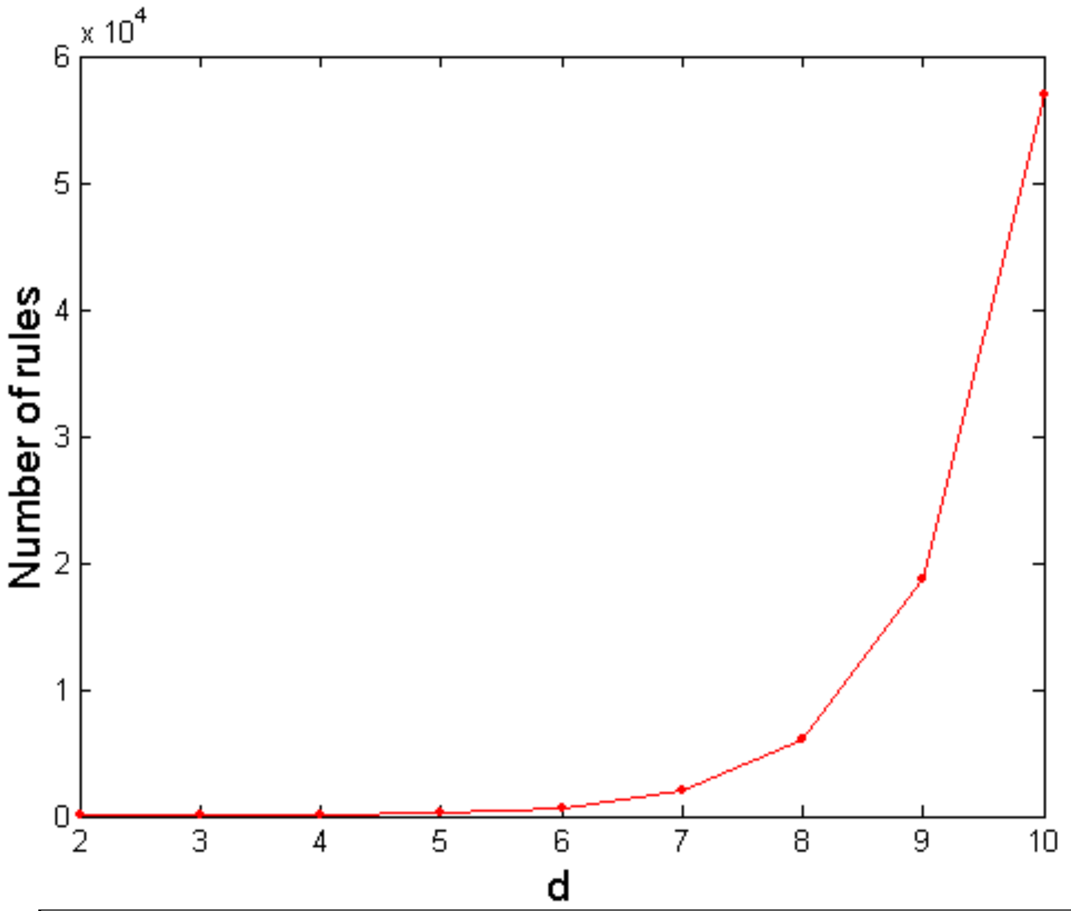


TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

  - Match each transaction against every candidate
    and increment the candidate's support counter if contained
  - Complexity ~ O(NMw) => *expensive* since M = 2^d-1!!!

Computational Complexity

- Given d unique items:
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



k-item lefthandsides

j -item righthandsides

$$R = \sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right]$$

$$= 3^d - 2^{d+1} + 1$$

If d=6, R=3^6-2^7+1=602 rules

Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
  - Complete search: $M=2^d-1$
  - Use pruning techniques to reduce M
- Reduce the number of comparisons (NM)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction
- Reduce the number of transactions (N)
  - Reduce size of N as the size of itemset increases
  - Used by “DHP” and “vertical-based mining” algorithms

Reducing Number of Candidates

- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent

- Apriori principle holds due to the following property of the support measure:

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

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Illustrating Apriori Principle

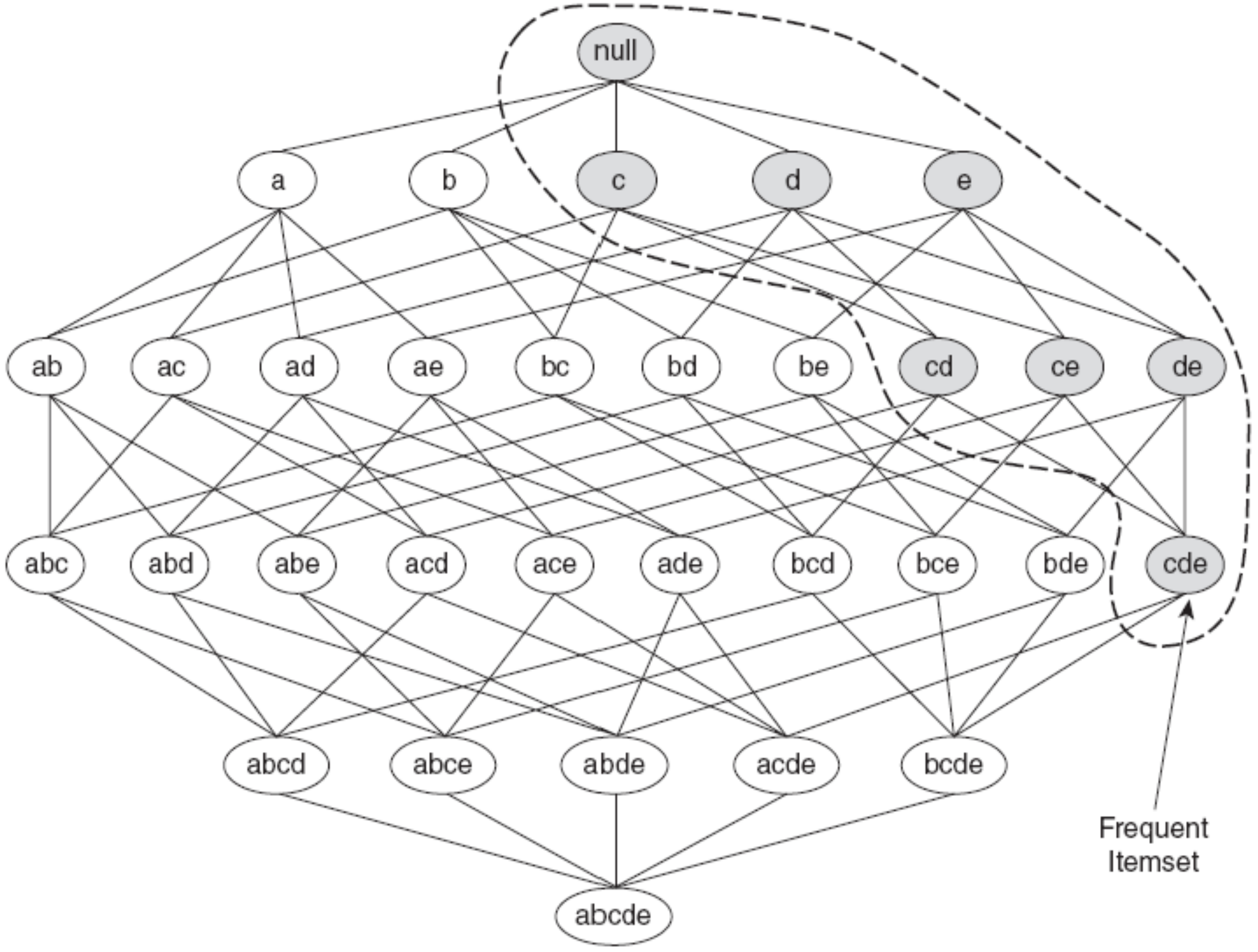
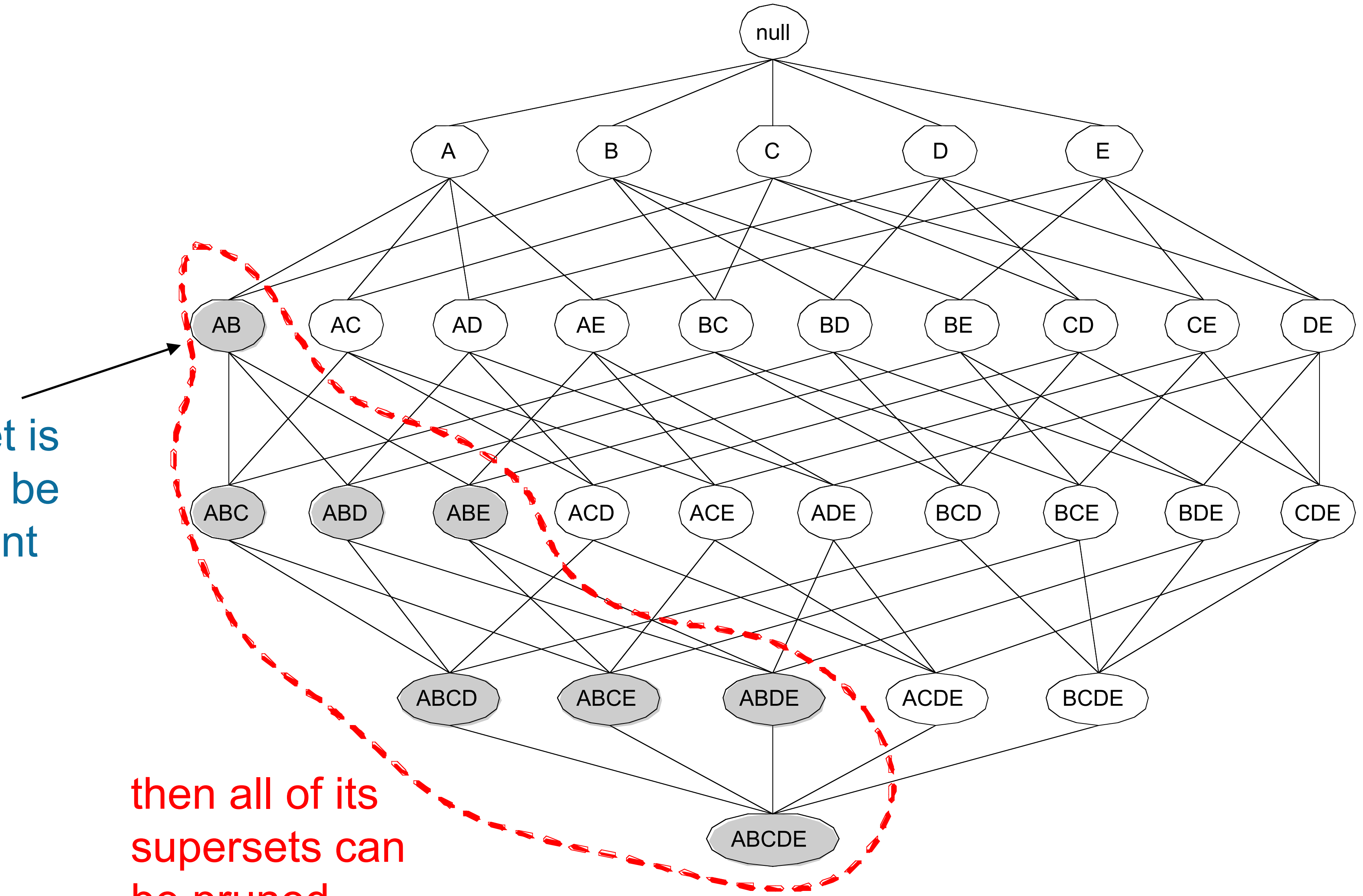


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Illustrating Apriori Principle



If this set is found to be Infrequent

then all of its supersets can be pruned

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 = 3/5

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	**3**

If every subset is considered,

$${}^6\#_1 + {}^6\#_2 + {}^6\#_3 = 6 + 15 + 20 = 41$$

With support-based pruning,

$$6 + 6 + 1 = 13$$

Apriori Algorithm

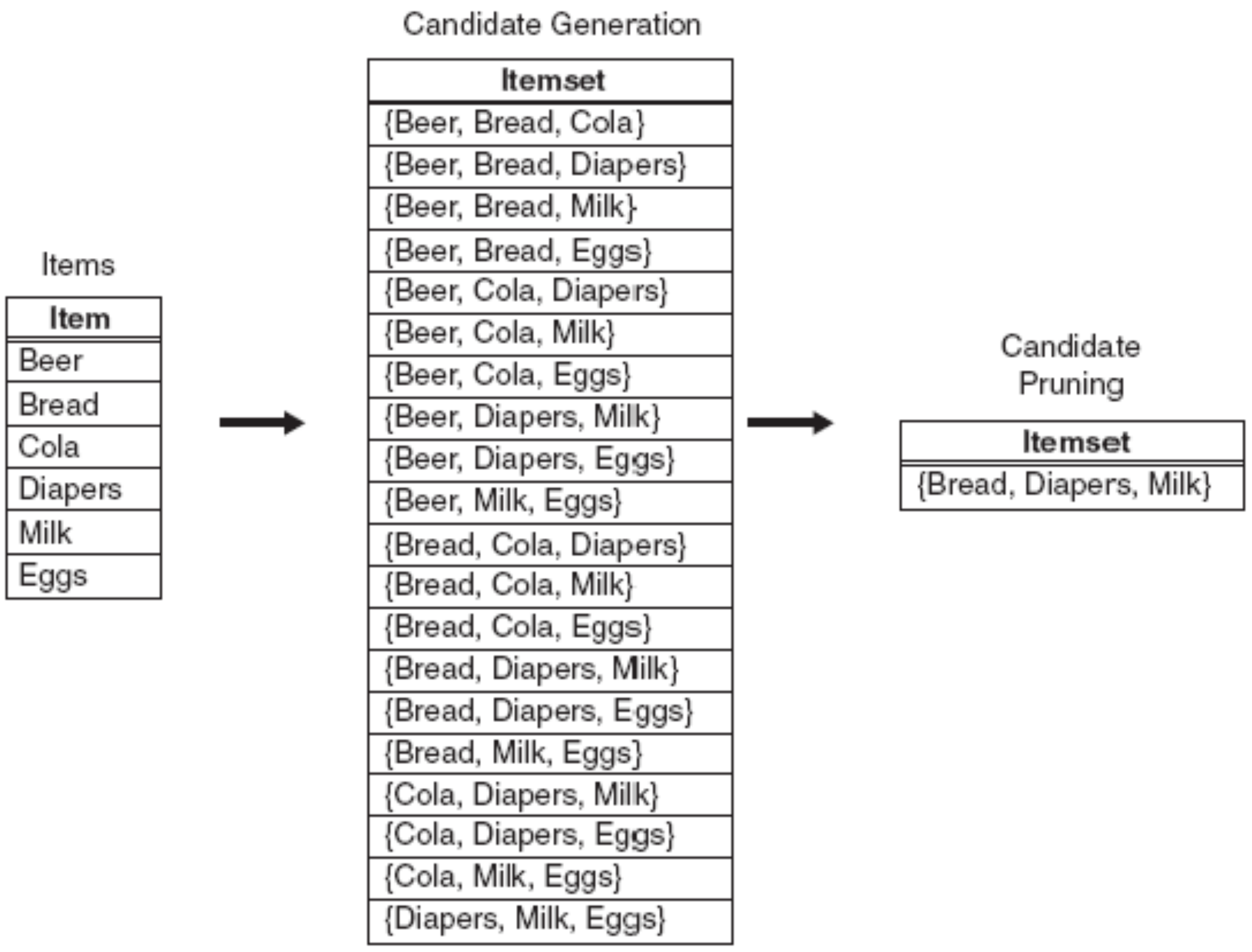
- Method:
  - Let k=1
  - Generate frequent itemsets of length 1
  - Repeat the following steps
    until no new frequent itemsets are identified:
    - k=k+1
    - Generate length k candidate itemsets from length (k-1) frequent itemsets,
    - but prune candidate itemsets containing subsets of length (k-1) that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

Apriori Algorithm

```
1:  $k = 1$.
2:  $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup \}$.    {Find all frequent 1-itemsets}
3:  repeat
4:     $k = k + 1$.
5:     $C_k$ = apriori-gen($F_{k-1}$).    {Generate and prune candidate $k$-itemsets}
6:     for each transaction $t \in T$ do
7:        $C_t$ = subset($C_k$, $t$).    {Identify all candidates that belong to $t$}
8:        for each candidate itemset $c \in C_t$ do
9:           $\sigma(c) = \sigma(c) + 1$.    {Increment support count}
10:       end for
11:    end for
12:    $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup \}$.    {Extract the frequent $k$-itemsets}
13: until $F_k = \emptyset$
14: Result = $\bigcup F_k$.
```

- T given set (database) of transactions, N number of transactions
- C_k set of candidate itemsets of length k, F_k set of frequent item sets of length k
- *Note:* Every - given or constructed - itemset or transaction is represented by an *ordered* nonrepetitive sequence of items

Candidate Generation and Pruning ?



Items

Item
Beer
Bread
Cola
Diapers
Milk
Eggs

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Milk}
{Beer, Bread, Eggs}
{Beer, Cola, Diapers}
{Beer, Cola, Milk}
{Beer, Cola, Eggs}
{Beer, Diapers, Milk}
{Beer, Diapers, Eggs}
{Beer, Milk, Eggs}
{Bread, Cola, Diapers}
{Bread, Cola, Milk}
{Bread, Cola, Eggs}
{Bread, Diapers, Milk}
{Bread, Diapers, Eggs}
{Bread, Milk, Eggs}
{Cola, Diapers, Milk}
{Cola, Diapers, Eggs}
{Cola, Milk, Eggs}
{Diapers, Milk, Eggs}

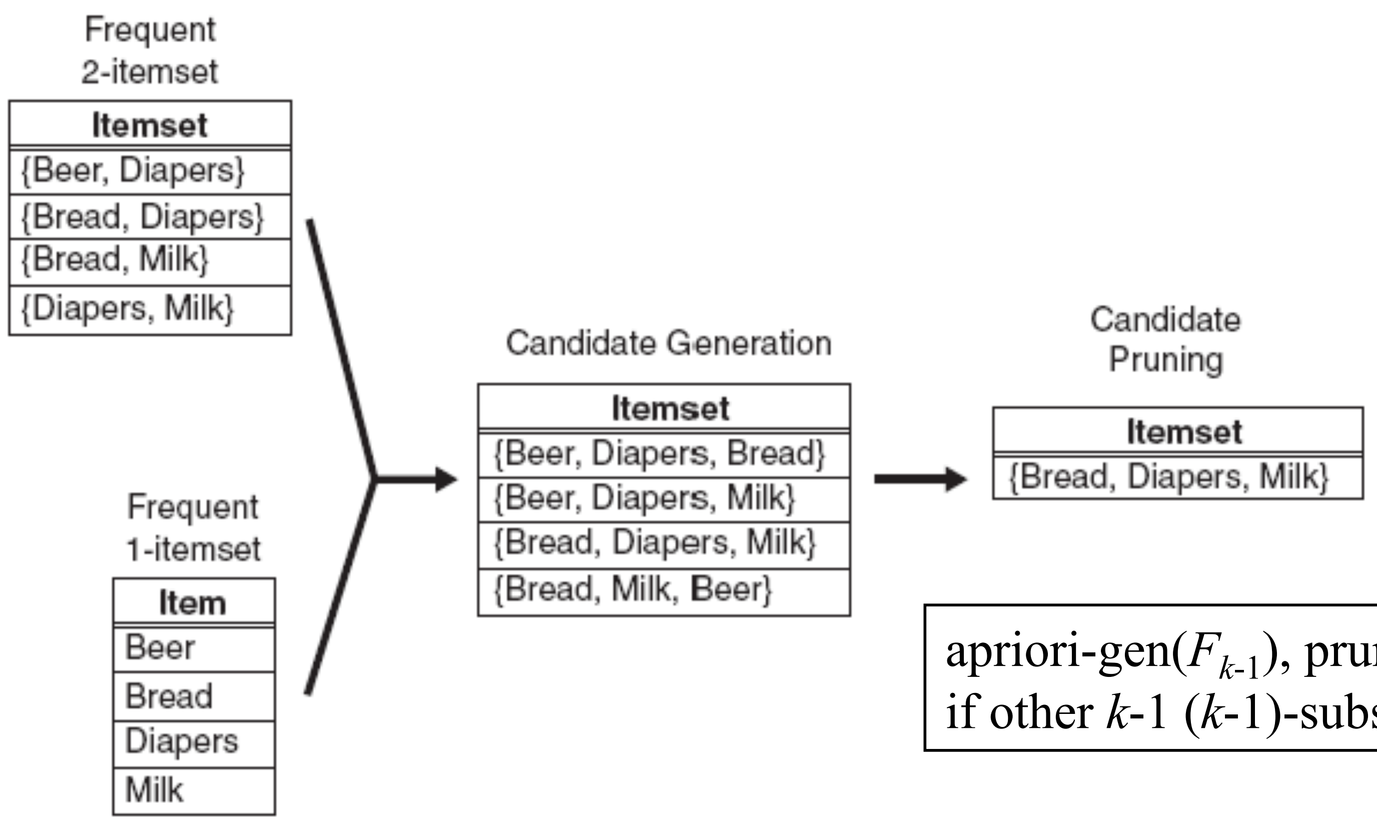
Candidate Pruning

Itemset
{Bread, Diapers, Milk}

Figure 6.6. A brute-force method for generating candidate 3-itemsets.

Apriori Alg.: Candidate Generation and Pruning

1st version



Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 1-itemset

Item
Beer
Bread
Diapers
Milk

Candidate Generation

Itemset
{Beer, Diapers, Bread}
{Beer, Diapers, Milk}
{Bread, Diapers, Milk}
{Bread, Milk, Beer}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

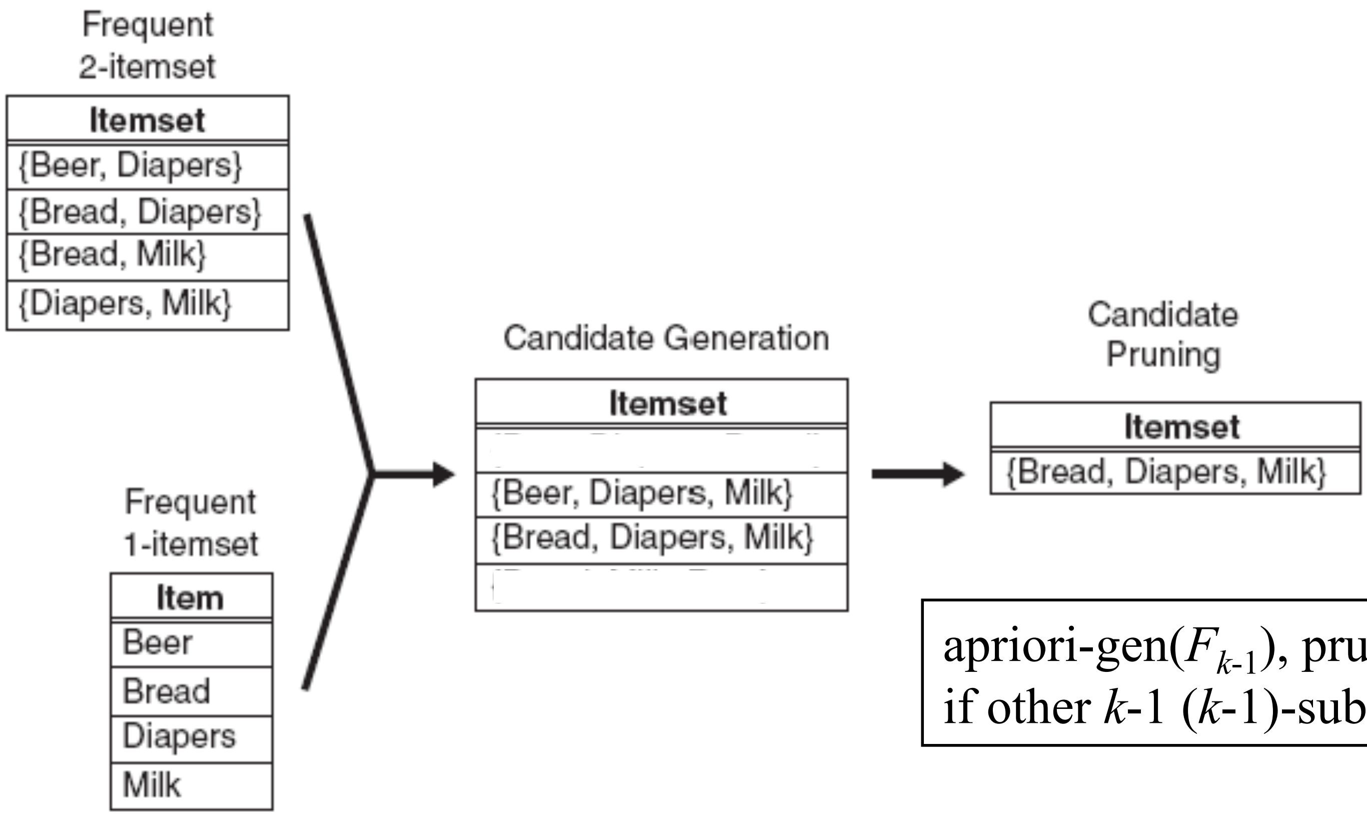
apriori-gen($F_{k\text{-}1}$), pruning: check, if other k-1 (k-1)-subsets are in $F_{k\text{-}1}$

Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item.

apriori-gen($F_{k\text{-}1}$), generation := all k-itemsets in ($F_{k\text{-}1}$ crossjoin F_1) [cartesian product]

Apriori Alg.: Candidate Generation and Pruning

2nd version



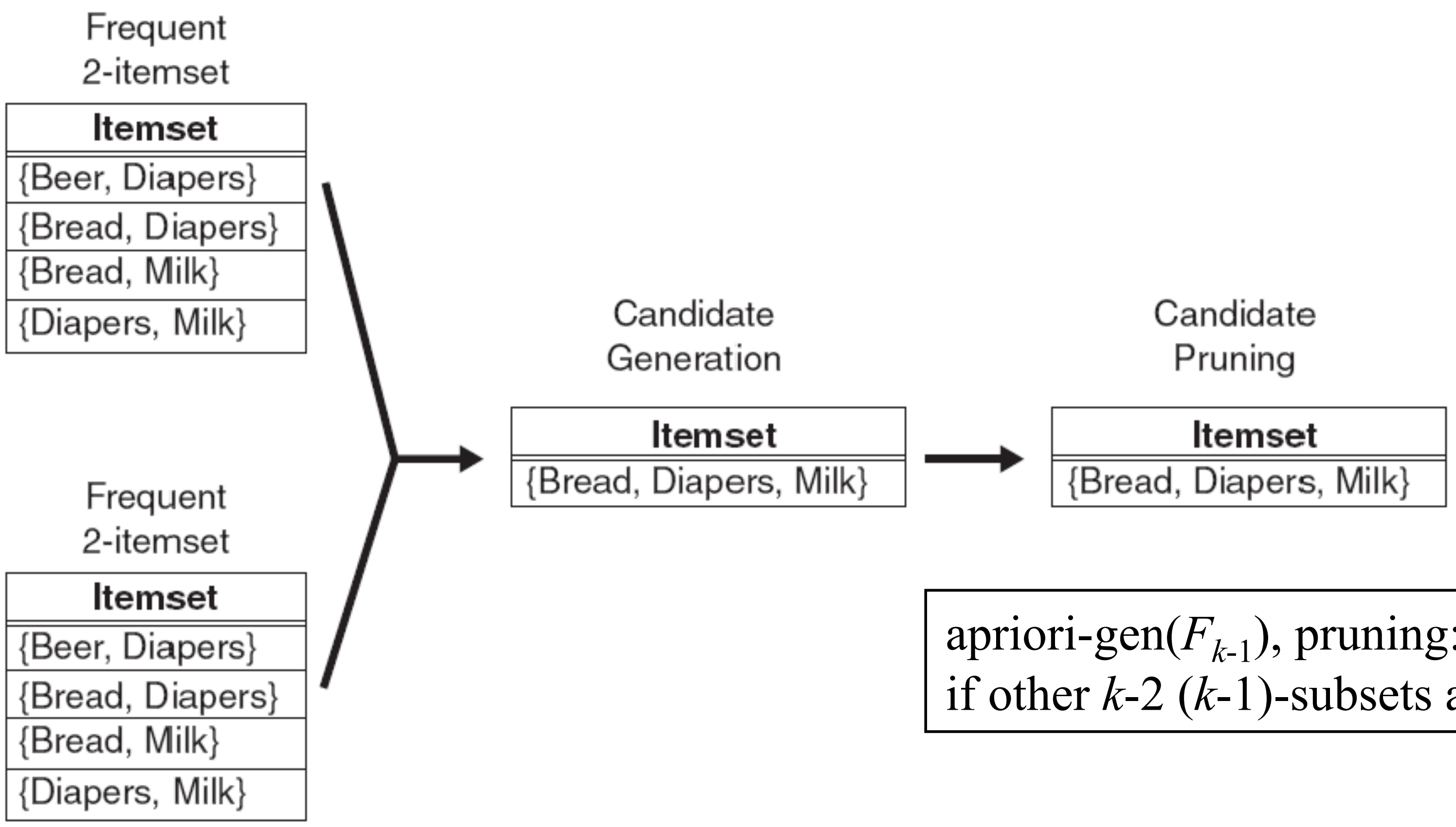
apriori-gen($F_{k\text{-}1}$), pruning: check, if other k-1 (k-1)-subsets are in $F_{k\text{-}1}$

Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item.

apriori-gen($F_{k\text{-}1}$), generation := all k-itemsets in ($F_{k\text{-}1}$ <-join F_1) [join on ordering condition]

Apriori Alg.: Candidate Generation and Pruning

3rd version



apriori-gen($F_{k\text{-}1}$), pruning: check, if other k-2 (k-1)-subsets are in $F_{k\text{-}1}$

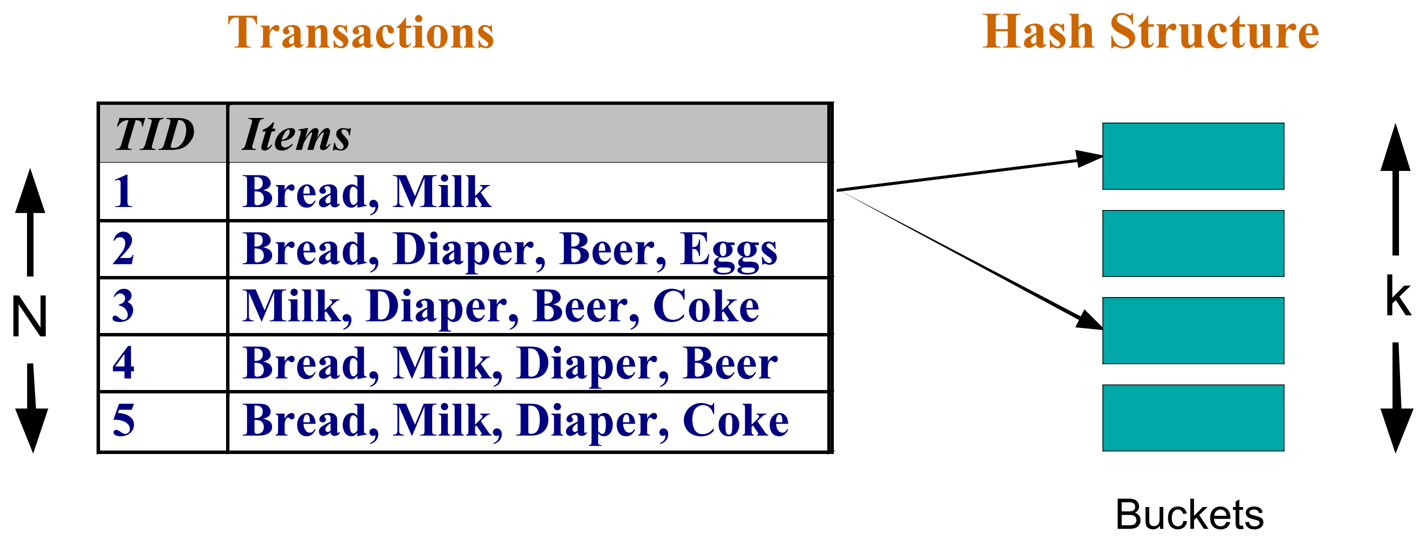
Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

apriori-gen($F_{k\text{-}1}$), generation := all k-itemsets in ($F_{k\text{-}1}$ equijoin $F_{k\text{-}1}$ using the first k-2 positions)

$a_1 \ldots a_{k-2}\, a_{k-1}$ and $b_1 \ldots b_{k-2} b_{k-1}$ are joined to $a_1 \ldots a_{k-2} a_{k-1} b_{k-1}$ if $a_i = b_i$ $(i=1,\ldots k\text{-}2)$ and $a_{k-1} < b_{k-1}$

Reducing Number of Comparisons

- Candidate counting:
  - Determine for each transaction which candidate items are supported by the transaction.
  - To reduce the number of comparisons, store the candidates in a hash structure
    - Instead of matching each transaction against every candidate, match it against candidates corresponding hash buckets.



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

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}

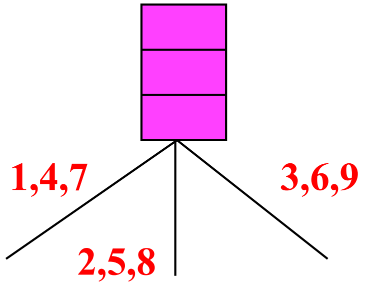
You need:

- **Hash function on items,**
  here h(p) = left|down|right corresponding to 1|2|0 = p mod 3
- Start with hashing the first item position
- **Max leaf size:** max number of itemsets stored in a leaf node,
  here 3
- If number of itemsets exceeds max leaf size, split the node
  by hashing the next item position
- **Max depth k** (no more splitting on this level)

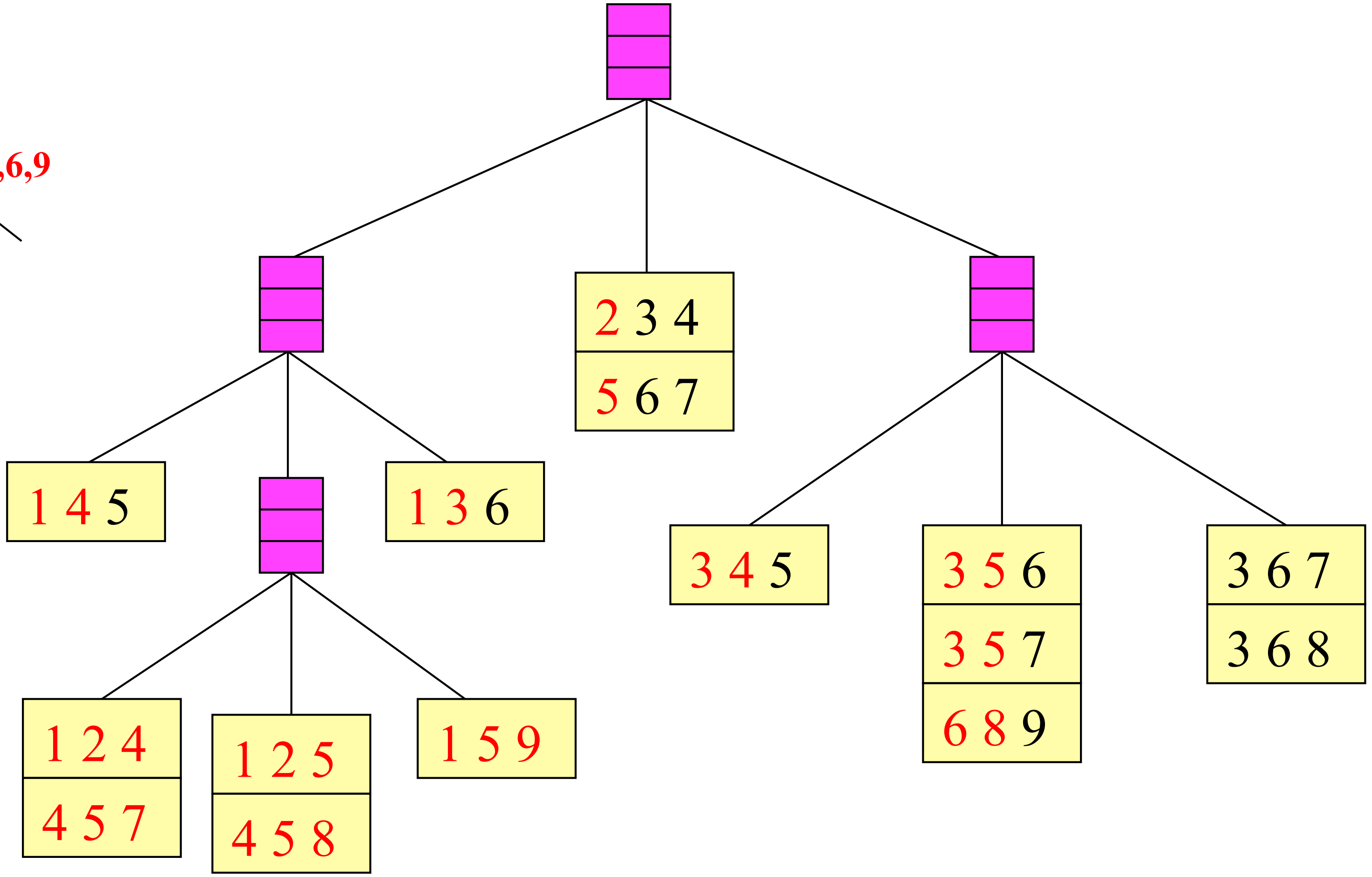
Support Counting: Hash tree

{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}

Hash Function

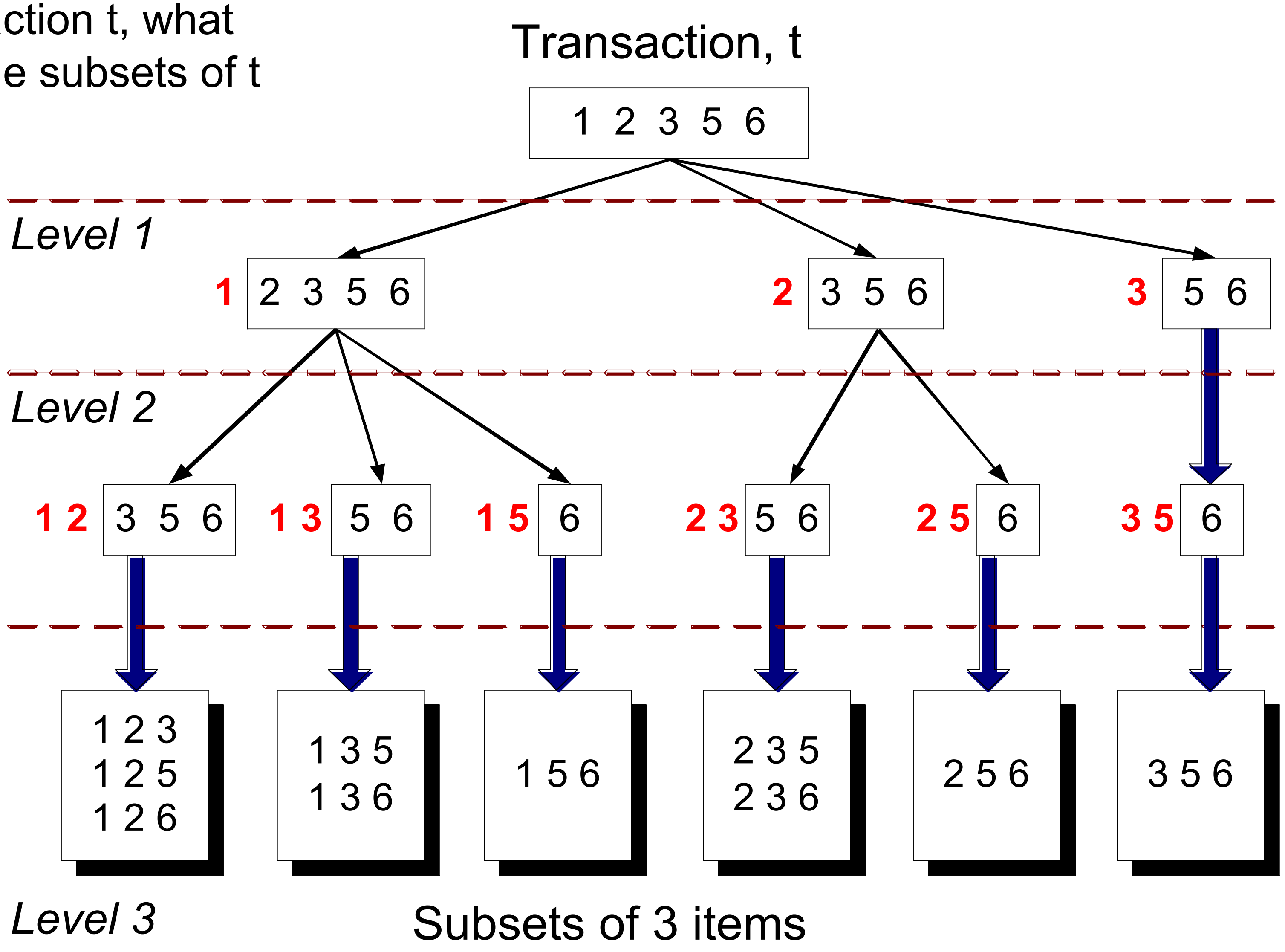


Candidate Hash Tree

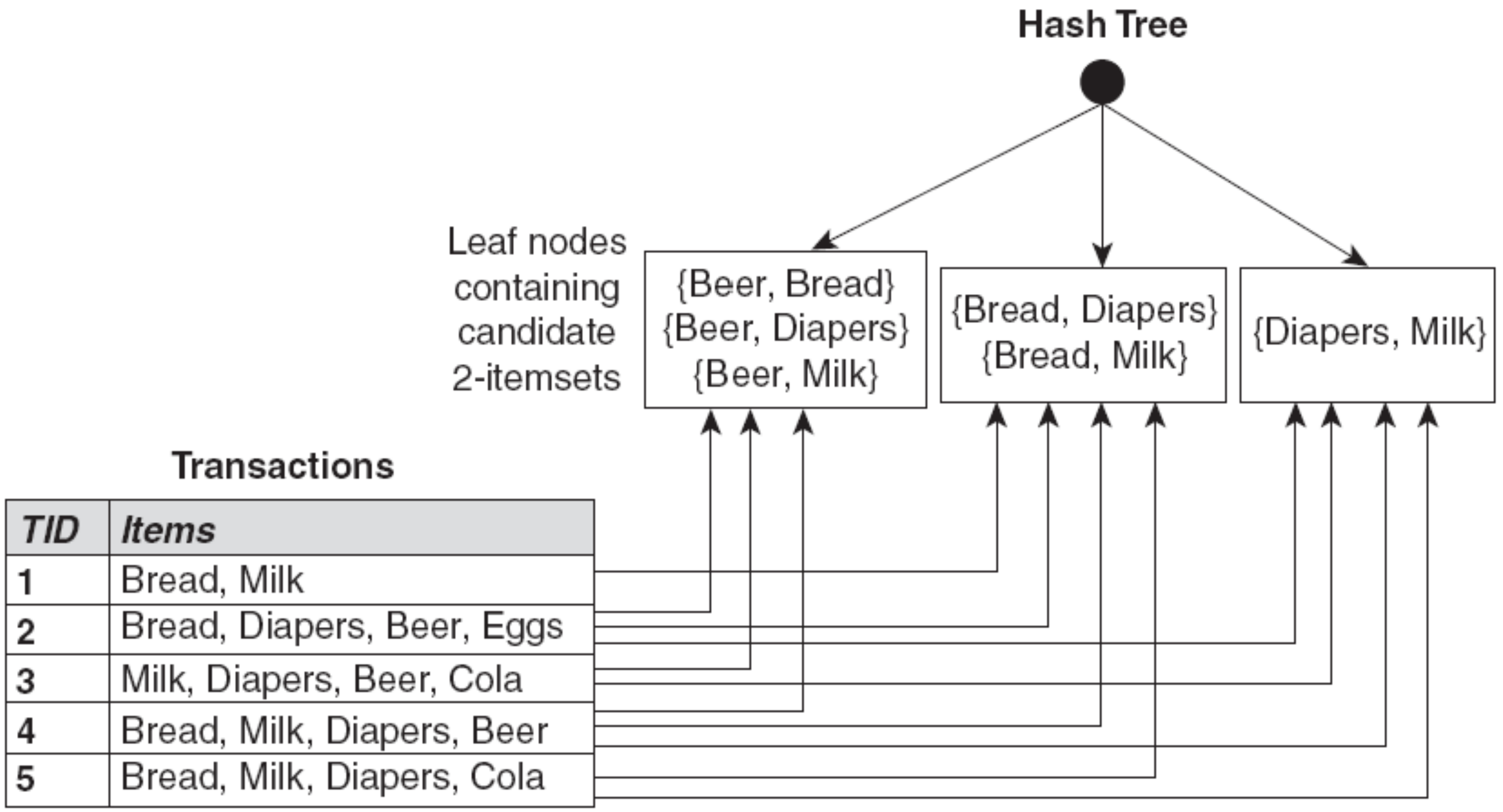


Subset Operation

Given a transaction t, what are the possible subsets of t with size 3?



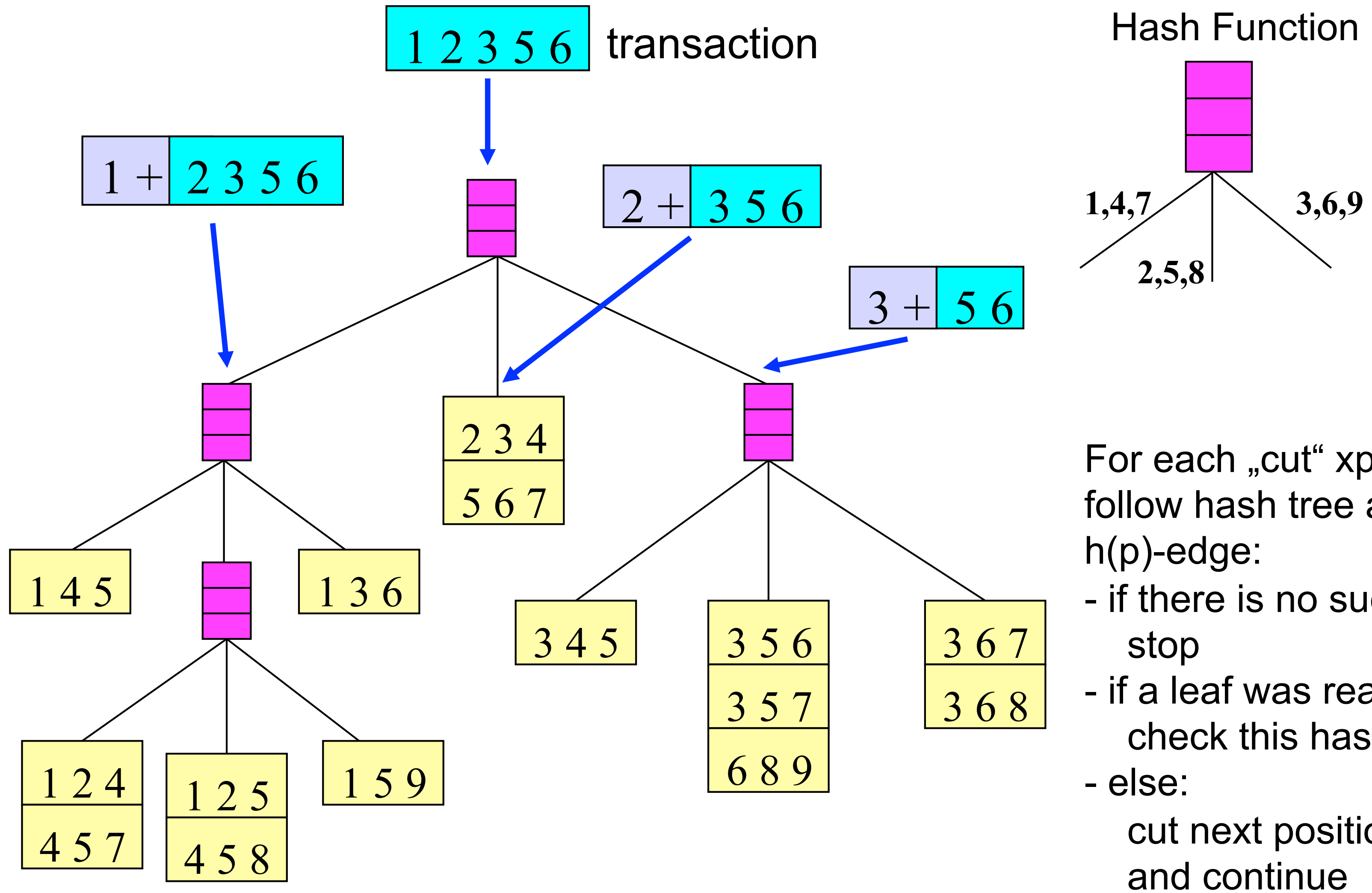
Subset Operation Using Hash Tree



TID	*Items*
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Cola
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Cola

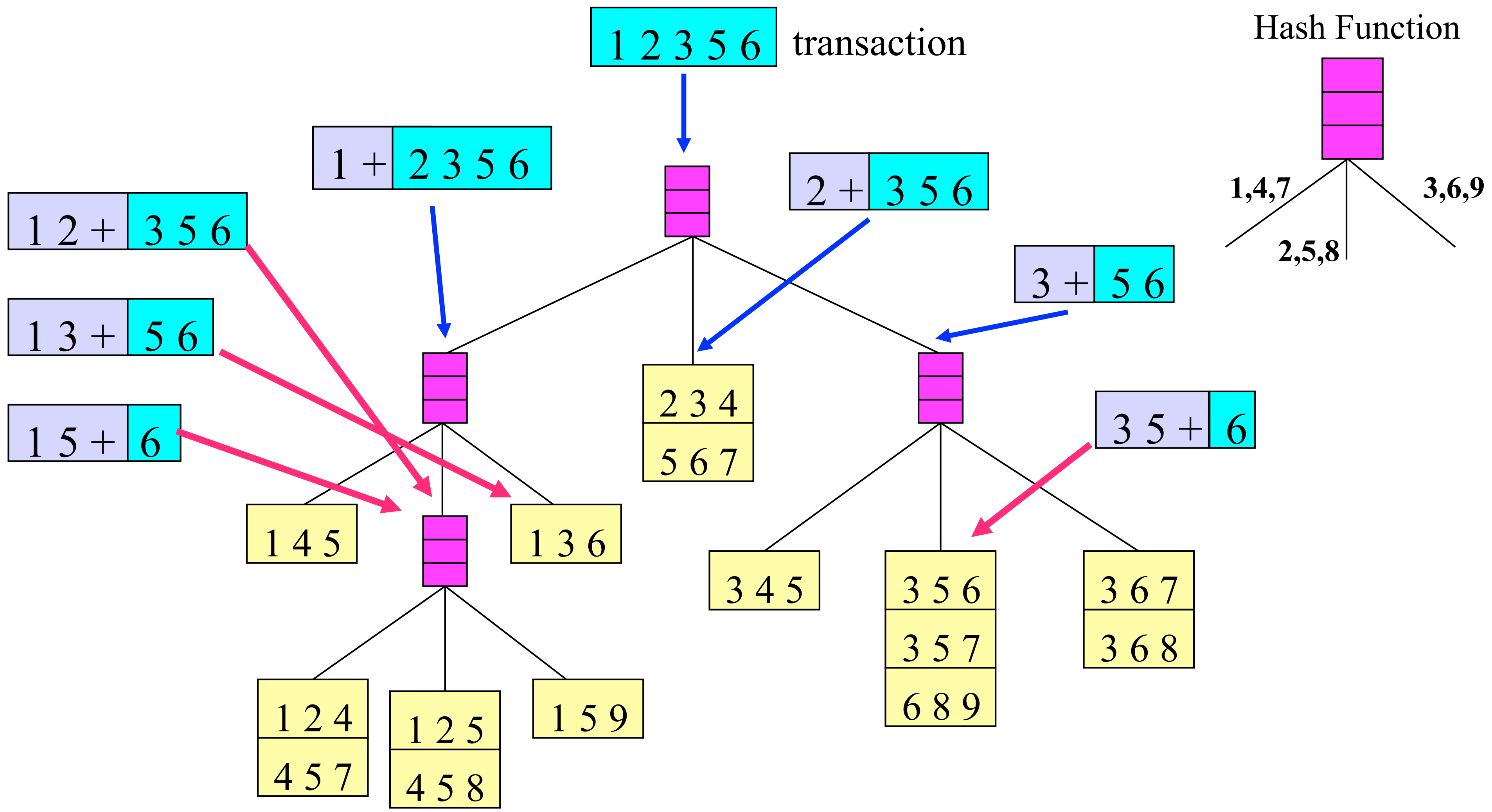
Figure 6.10. Counting the support of itemsets using hash structure.

Subset Operation Using Hash Tree

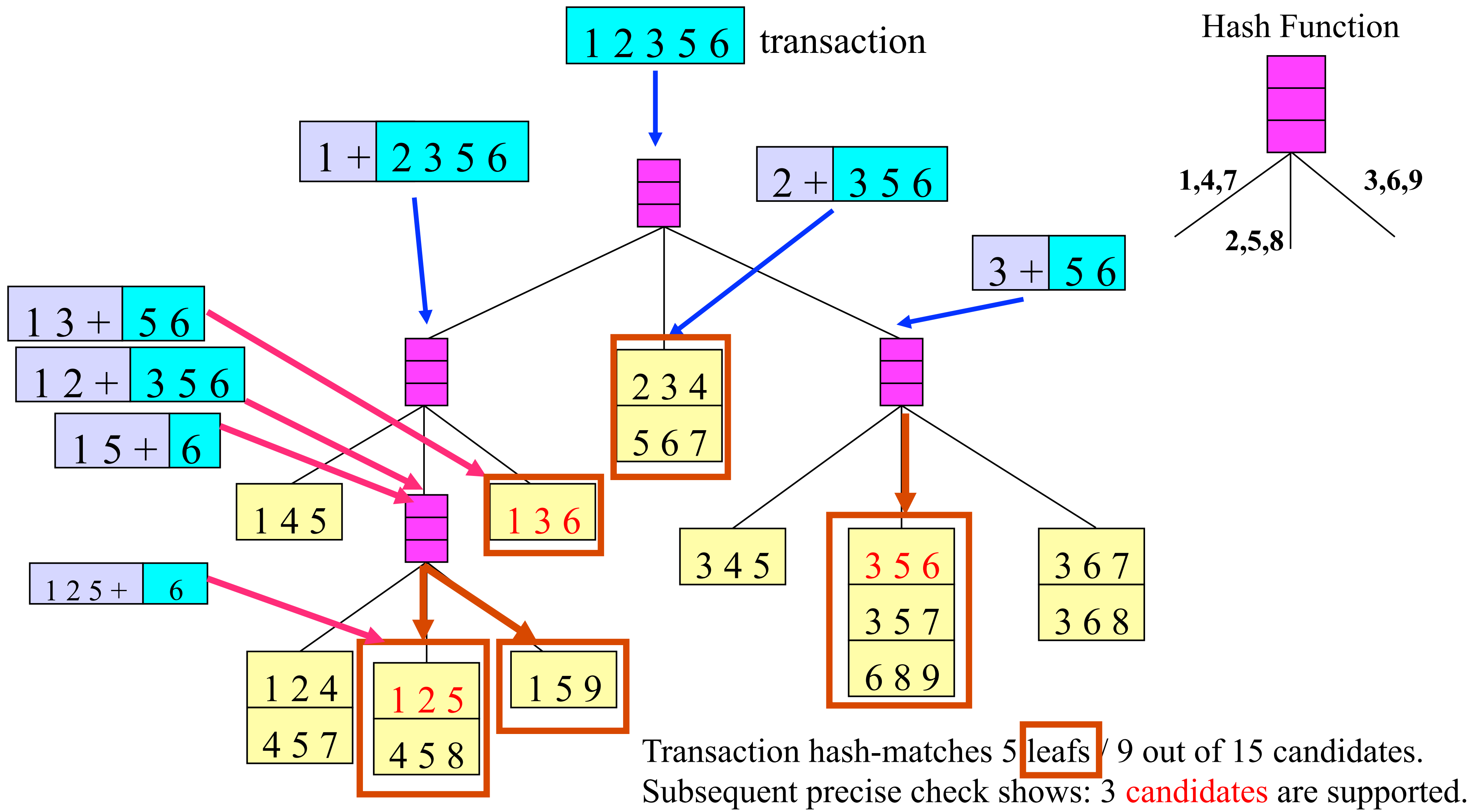


For each „cut“ xp+ , follow hash tree along h(p)-edge:
- if there is no such edge: stop
- if a leaf was reached: check this hash bucket
- else: cut next position xpq+ and continue

Subset Operation Using Hash Tree



Subset Operation Using Hash Tree

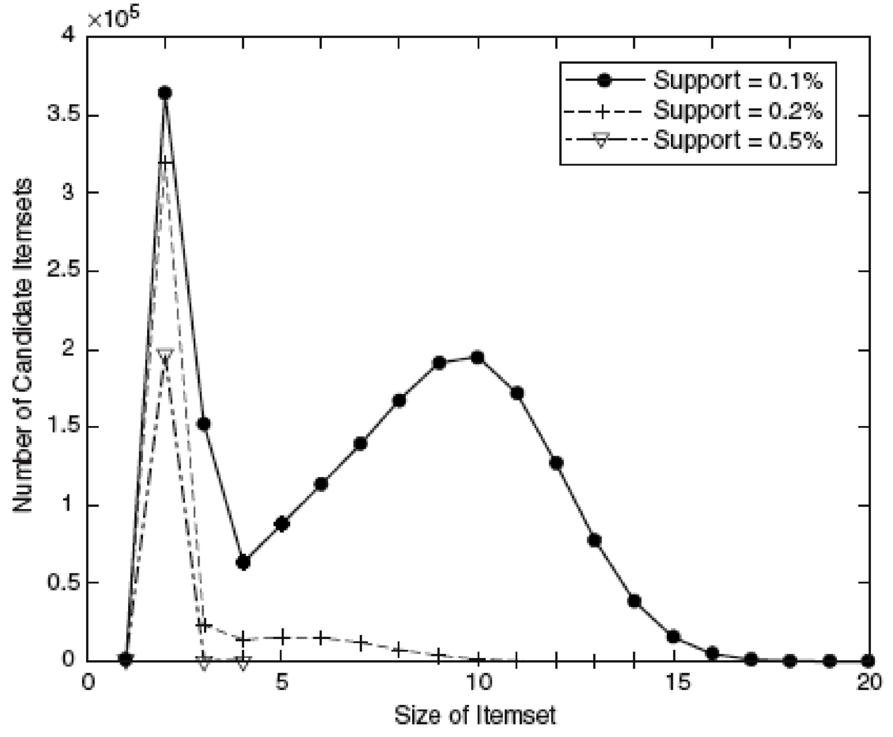


Transaction hash-matches 5 leafs / 9 out of 15 candidates.
Subsequent precise check shows: 3 candidates are supported.

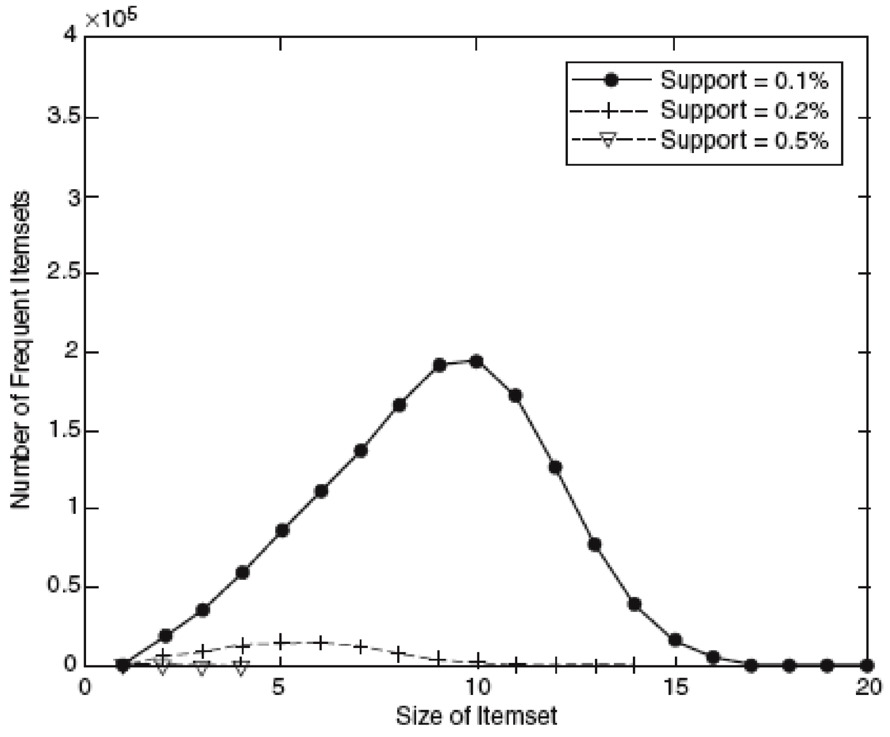
Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Factors Affecting Complexity



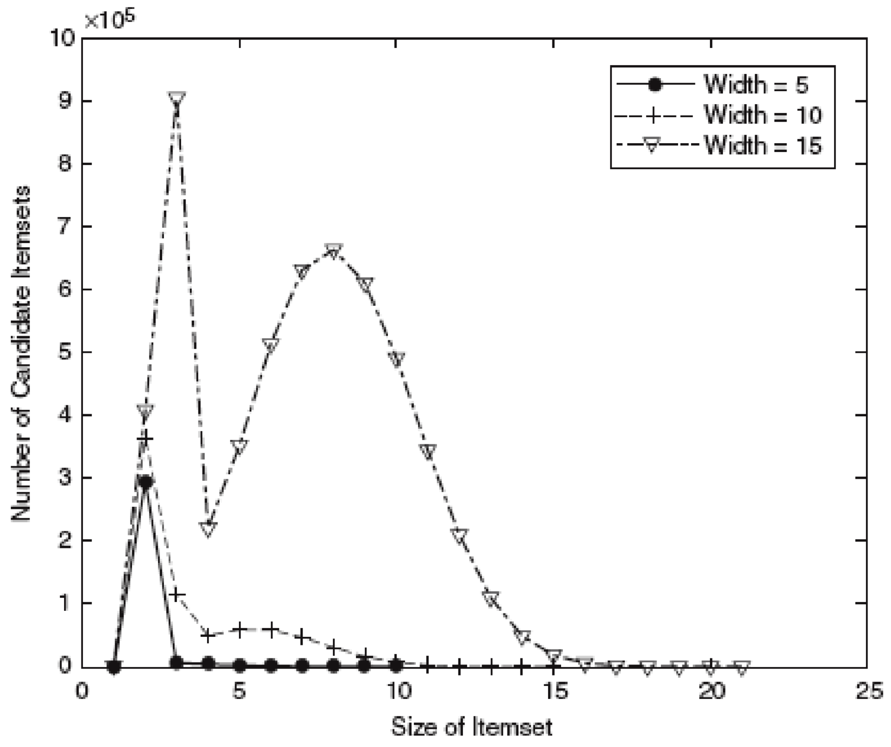
(a) Number of candidate itemsets.



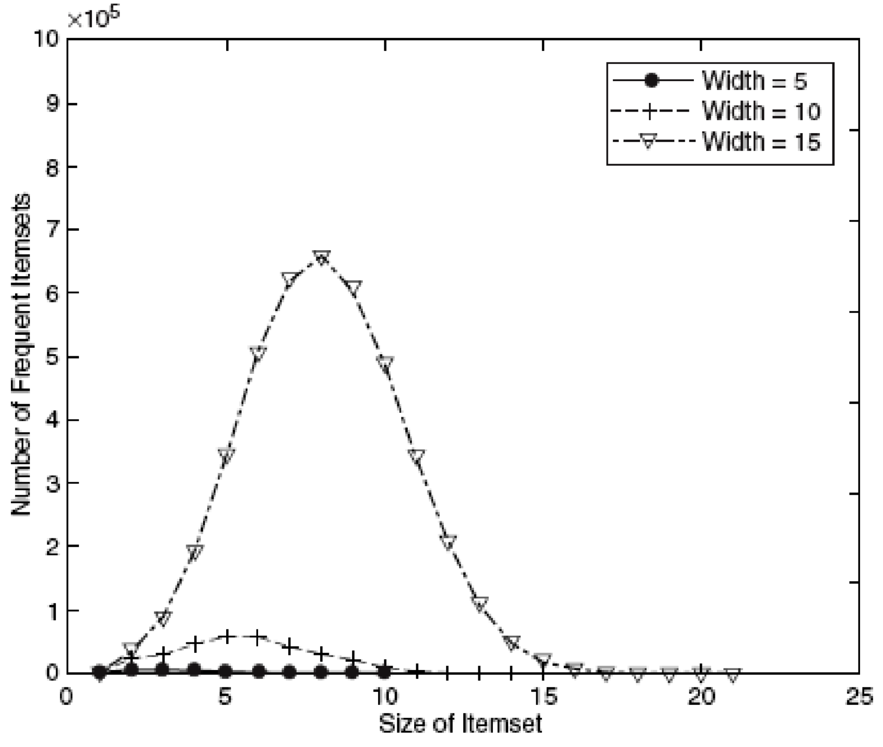
(b) Number of frequent itemsets.

Figure 6.13. Effect of support threshold on the number of candidate and frequent itemsets.

Factors Affecting Complexity



(a) Number of candidate itemsets.



(b) Number of Frequent Itemsets.

Figure 6.14. Effect of average transaction width on the number of candidate and frequent itemsets.

Compact Representation of Frequent Itemsets

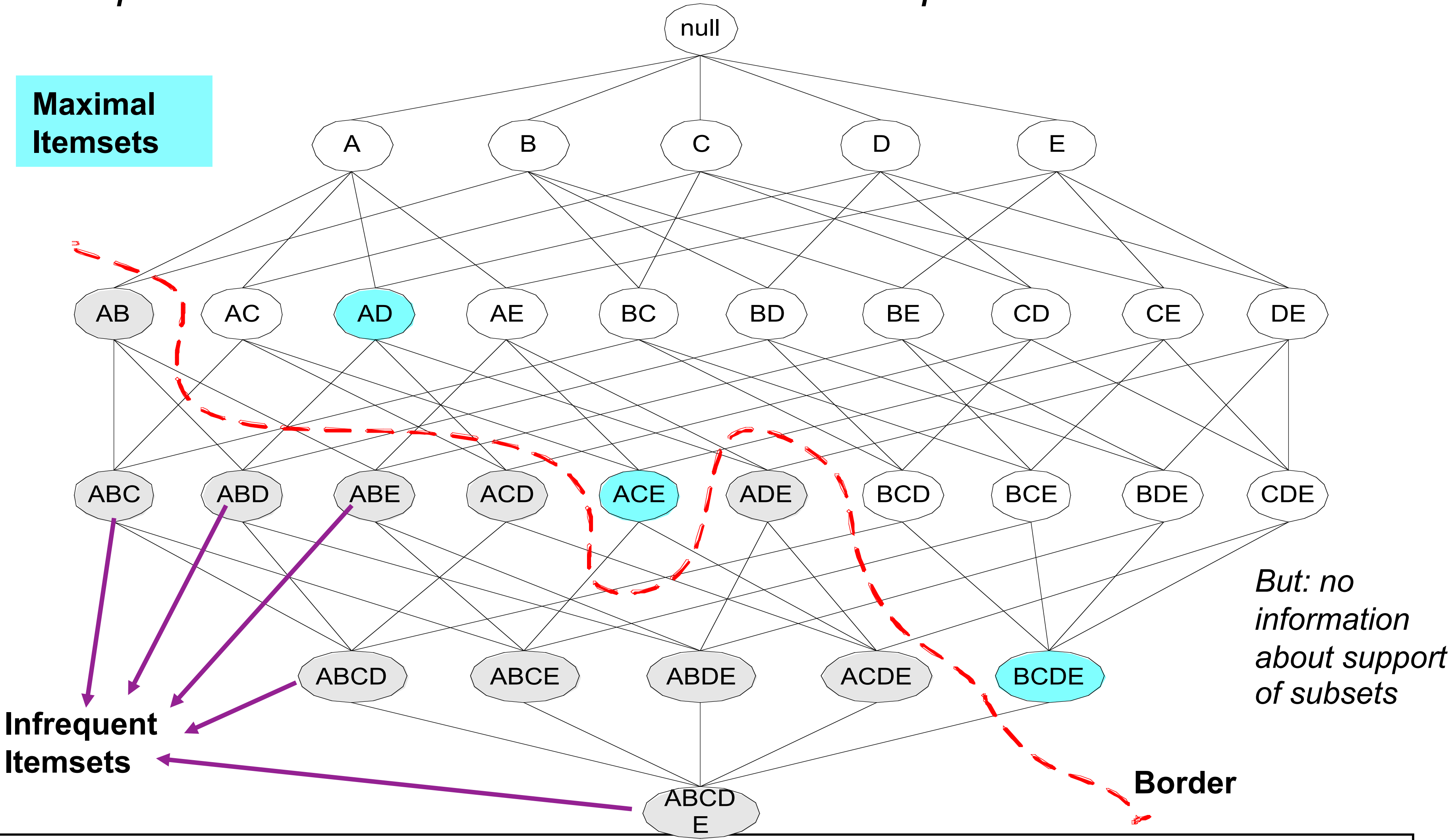
- Some itemsets are redundant because they have identical support as their supersets. Consider an extreme example:

TID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

- Number of frequent itemsets in example $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$
- Need a compact representation; here, 3 would suffice.

Maximal Frequent Itemset

An itemset is maximal frequent if none of its immediate supersets is frequent.
Then: *frequent itemset ⇔ subset of a maximal frequent itemset !*



Closed Itemset

- An itemset is closed if none of its immediate supersets has the same support as the itemset

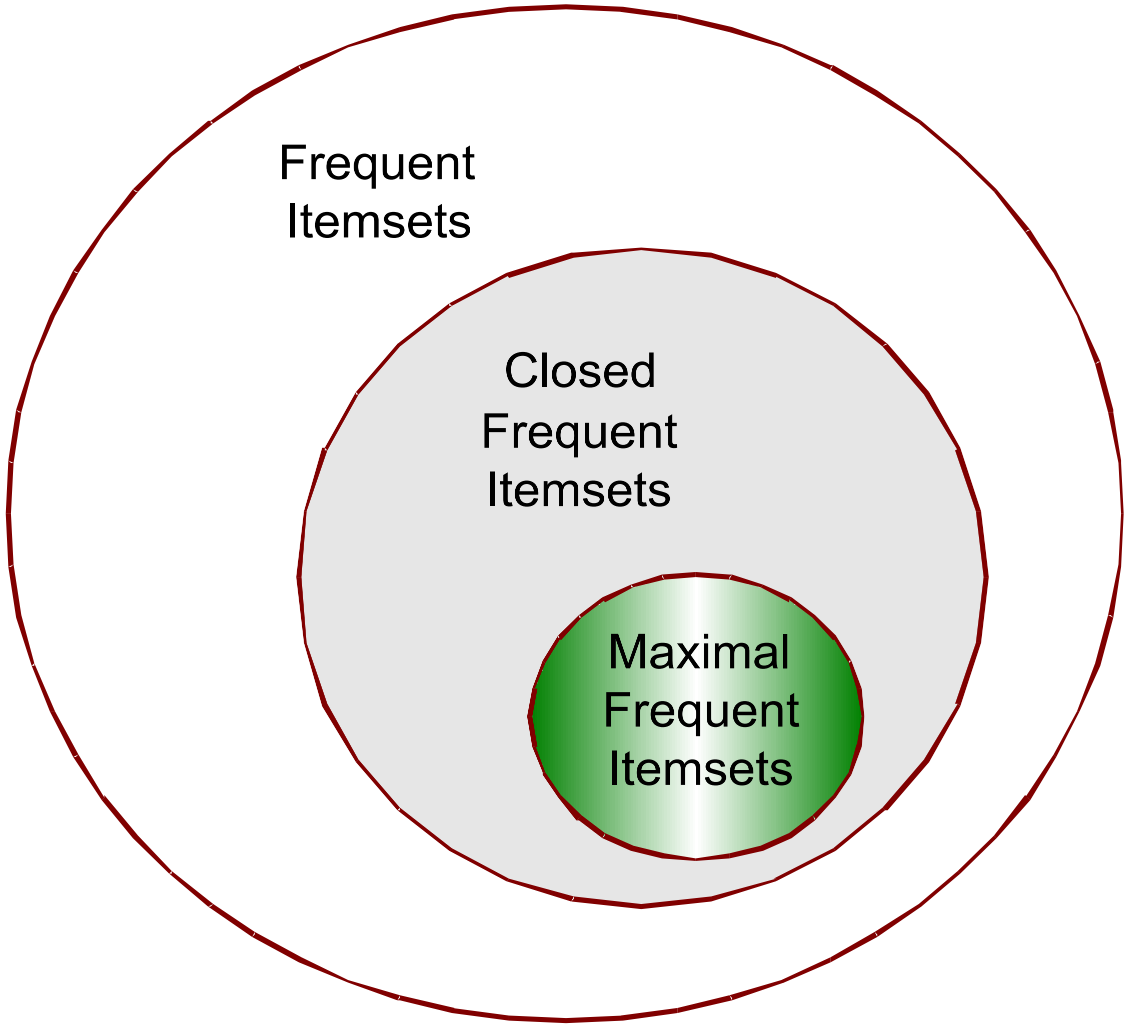
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

Itemset	Support
{A}	4
🟨 {B}	5
{C}	3
{D}	4
🟨 {A,B}	4
{A,C}	2
{A,D}	3
🟨 {B,C}	3
🟨 {B,D}	4
🟨 {C,D}	3

Itemset	Support
{A,B,C}	2
🟨 {A,B,D}	3
{A,C,D}	2
{B,C,D}	3
🟨 {A,B,C,D}	2

🟨 closed itemsets

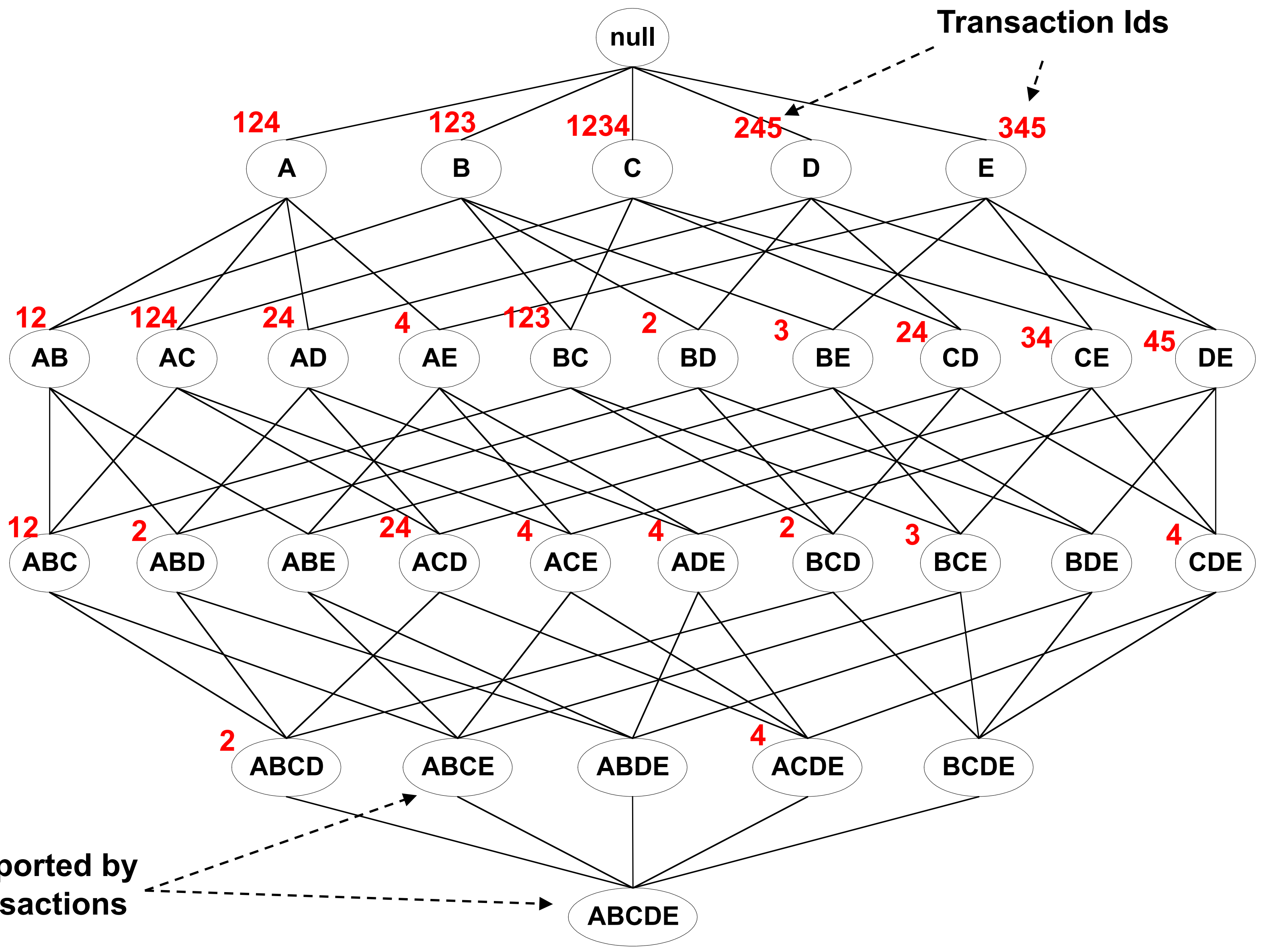
Maximal vs Closed Itemsets



Every maximal frequent itemset is closed !

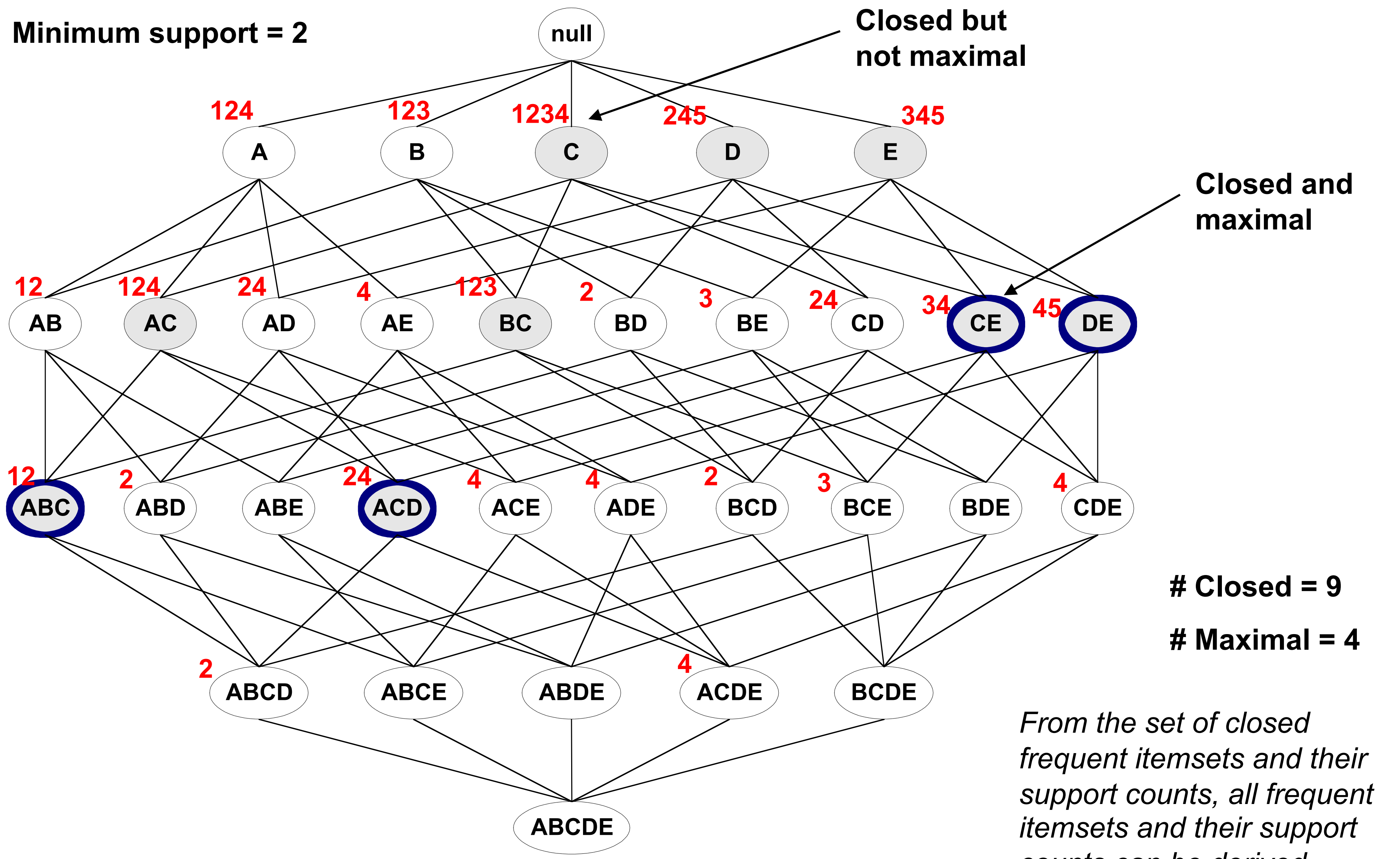
Maximal vs Closed Frequent Itemsets

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Not supported by any transactions

Maximal vs Closed Frequent Itemsets



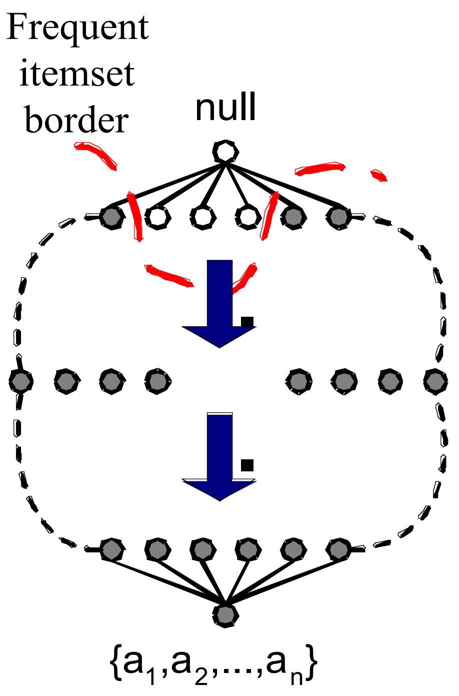
# Closed = 9

# Maximal = 4

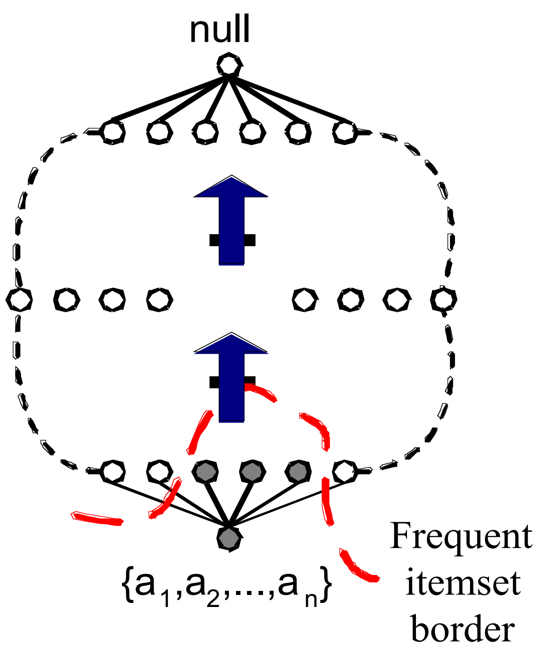
From the set of closed frequent itemsets and their support counts, all frequent itemsets and their support counts can be derived.

Alternative Methods for Frequent Itemset Generation

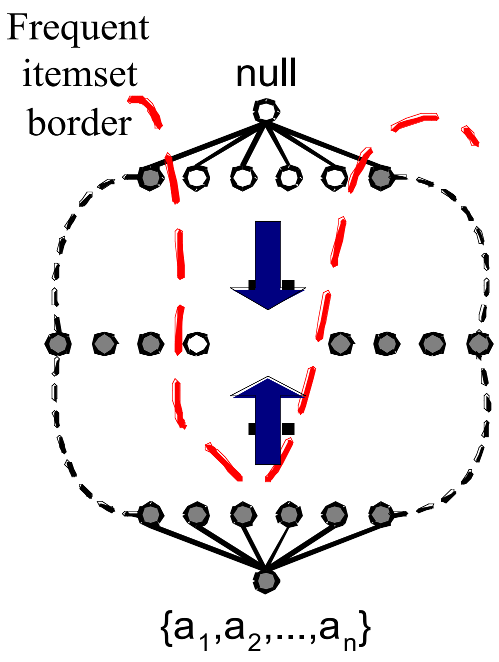
- Traversal of Itemset Lattice
  - General-to-specific vs Specific-to-general



(a) General-to-specific



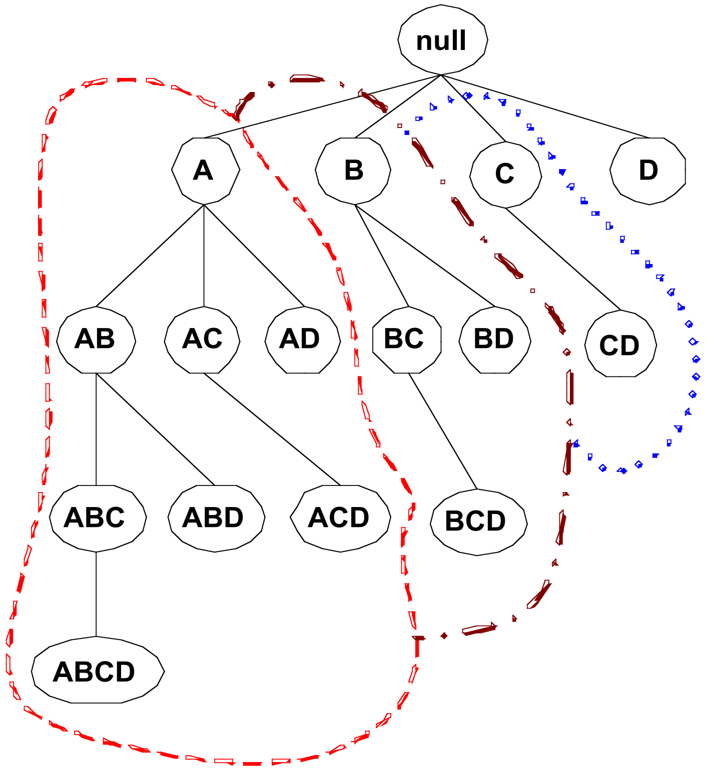
(b) Specific-to-general



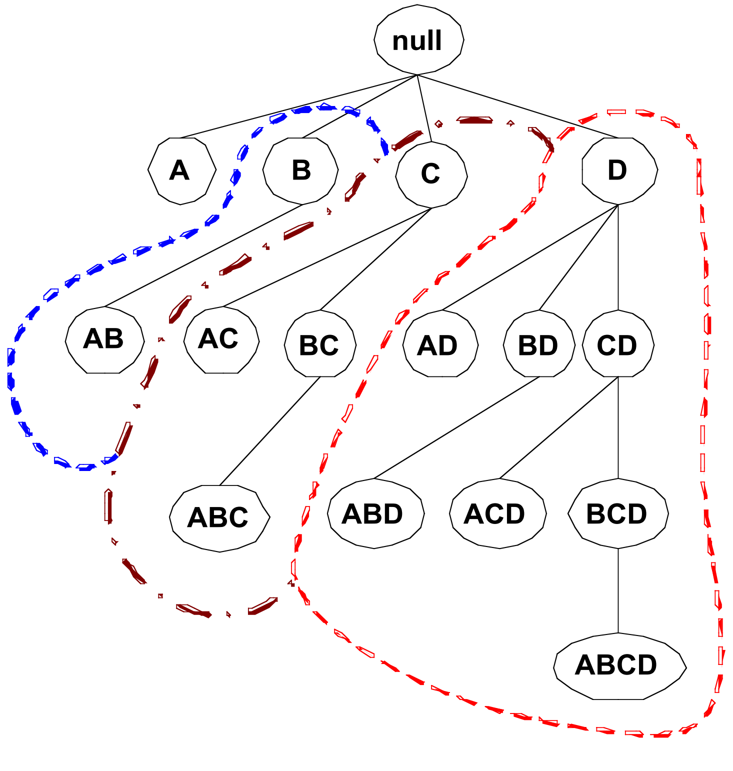
(c) Bidirectional

Alternative Methods for Frequent Itemset Generation

- Traversal of Itemset Lattice
  - Equivalence Classes, e.g. level-wise,
    or based on common prefixes/suffixes:



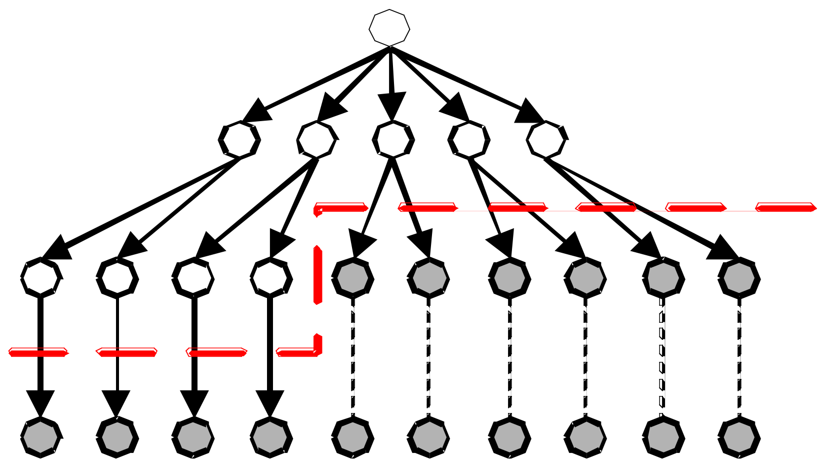
(a) Prefix tree

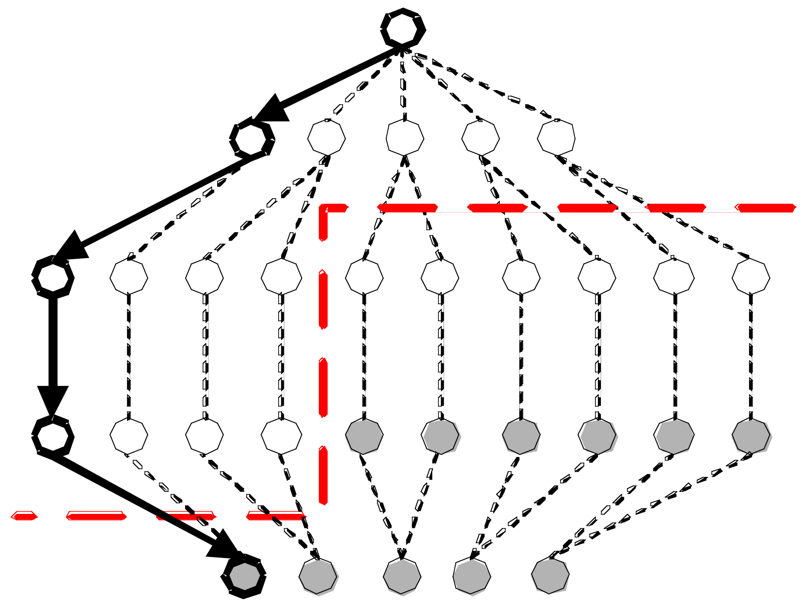


(b) Suffix tree

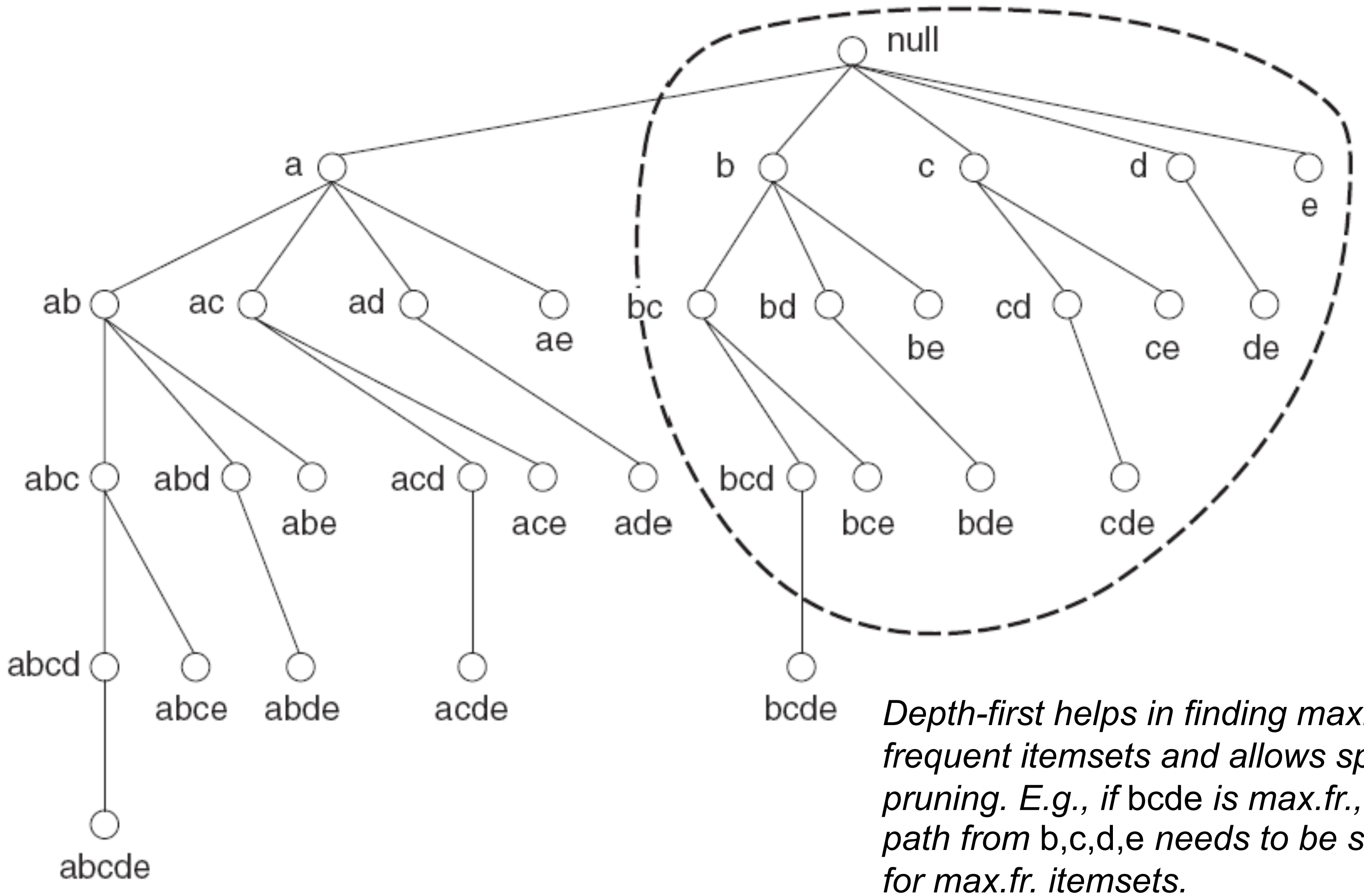
Alternative Methods for Frequent Itemset Generation

- Traversal of Itemset Lattice
  - Breadth-first vs Depth-first


(a) Breadth first


(b) Depth first

Alternative Methods for Frequent Itemset Generation



Depth-first helps in finding maximal frequent itemsets and allows special pruning. E.g., if bcde *is max.fr., no other path from* b,c,d,e *needs to be searched for max.fr. itemsets.*

Figure 6.22. Generating candidate itemsets using the depth-first approach.

Alternative Methods for Frequent Itemset Generation

- Representation of Database
  - horizontal vs vertical data layout

Horizontal Data Layout

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

Vertical Data Layout

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

Item

TIDs

Using vertical layout

- Determine support of any k-itemset by intersecting TID-lists (maybe bit vectors) of two of its (k-1)-subsets.

A
1
4
5
6
7
8
9

ˇ

B
1
2
5
7
8
10

=

AB
1
5
7
8

- Advantage: very fast support counting
- Disadvantage: intermediate TID-lists may become too large for main memory

FP ("Frequent pattern") -Growth Algorithm

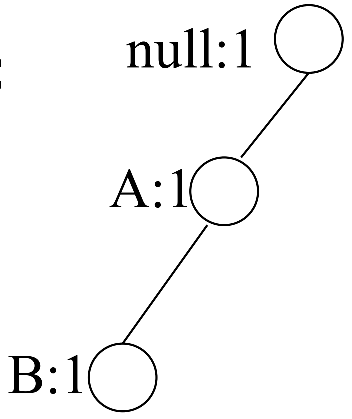
- Uses a compressed representation of the transaction database by means of an FP-tree
- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to extract the frequent itemsets from this tree

- Preliminarily, support counting of items should be done; infrequent items should be ignored and others be sorted by decreasing support counts *(not in the example).*
- Then, transactions are mapped to - overlapping - paths in the FP-tree.

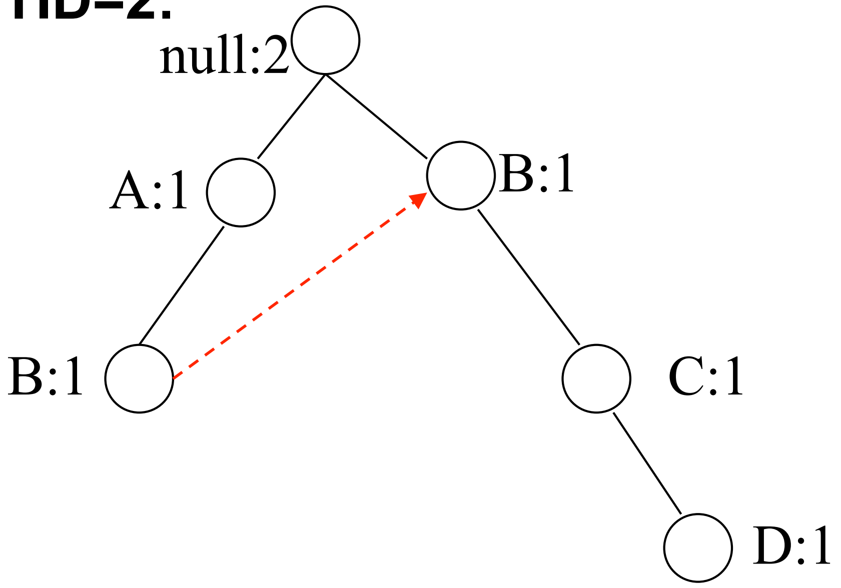
FP-Tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

After reading TID=1:



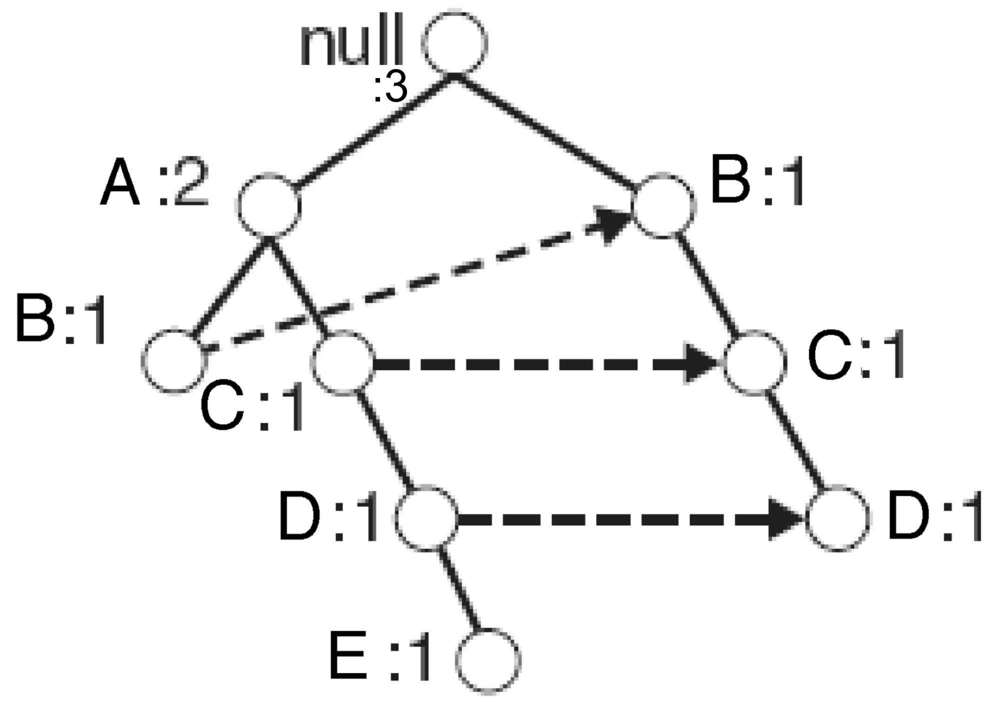
After reading TID=2:



FP-Tree Construction

After reading TID=3:

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



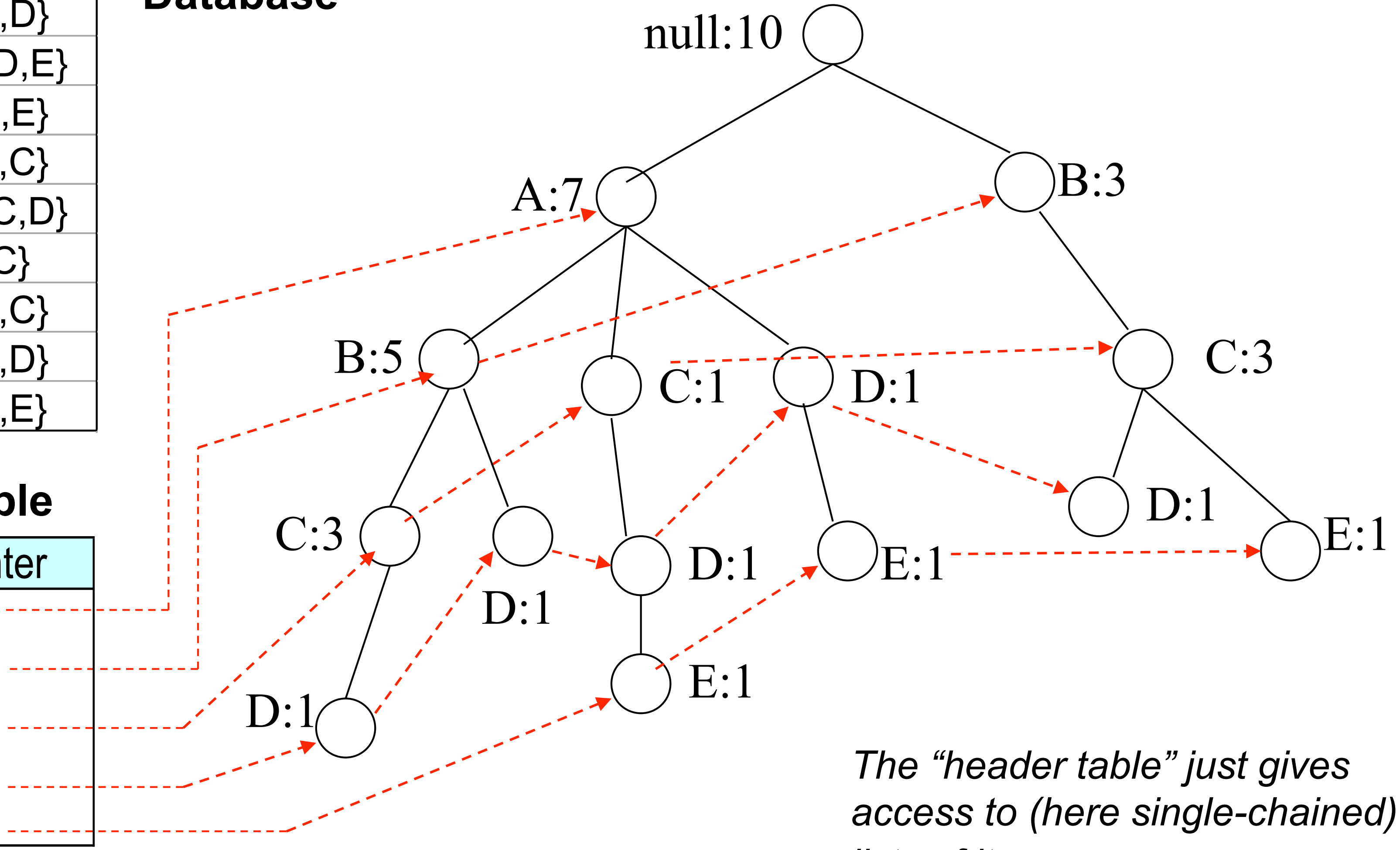
FP-Tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Header table

Item	Pointer
A	
B	
C	
D	
E	



The "header table" just gives access to (here single-chained) lists of item occurrences.

FP-Tree Construction

Transaction Data Set

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

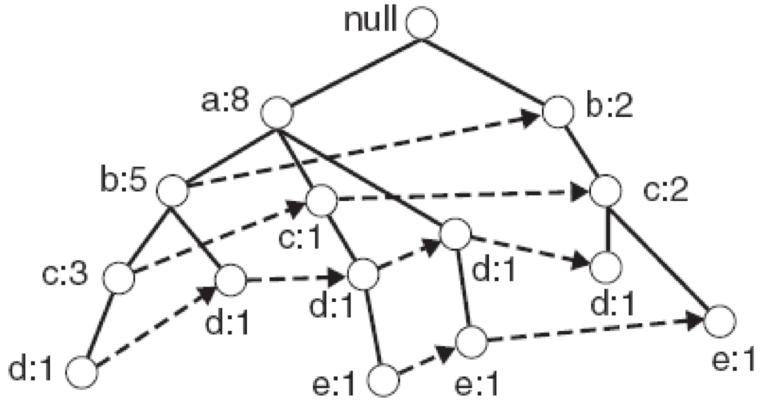


Figure 6.24. Construction of an FP-tree.

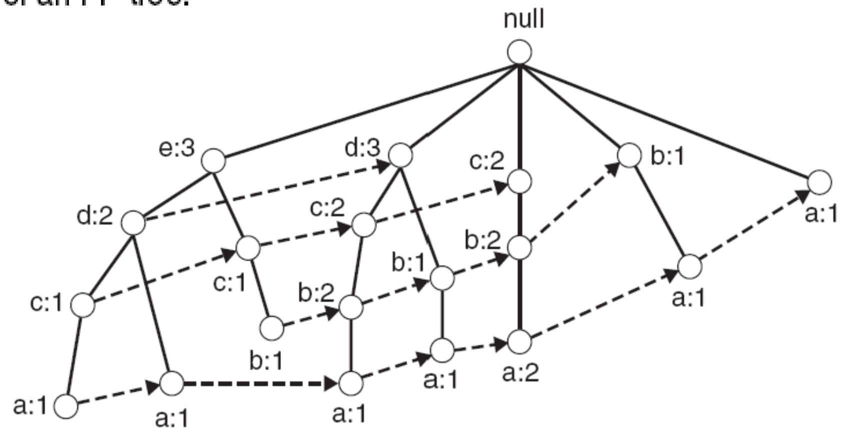
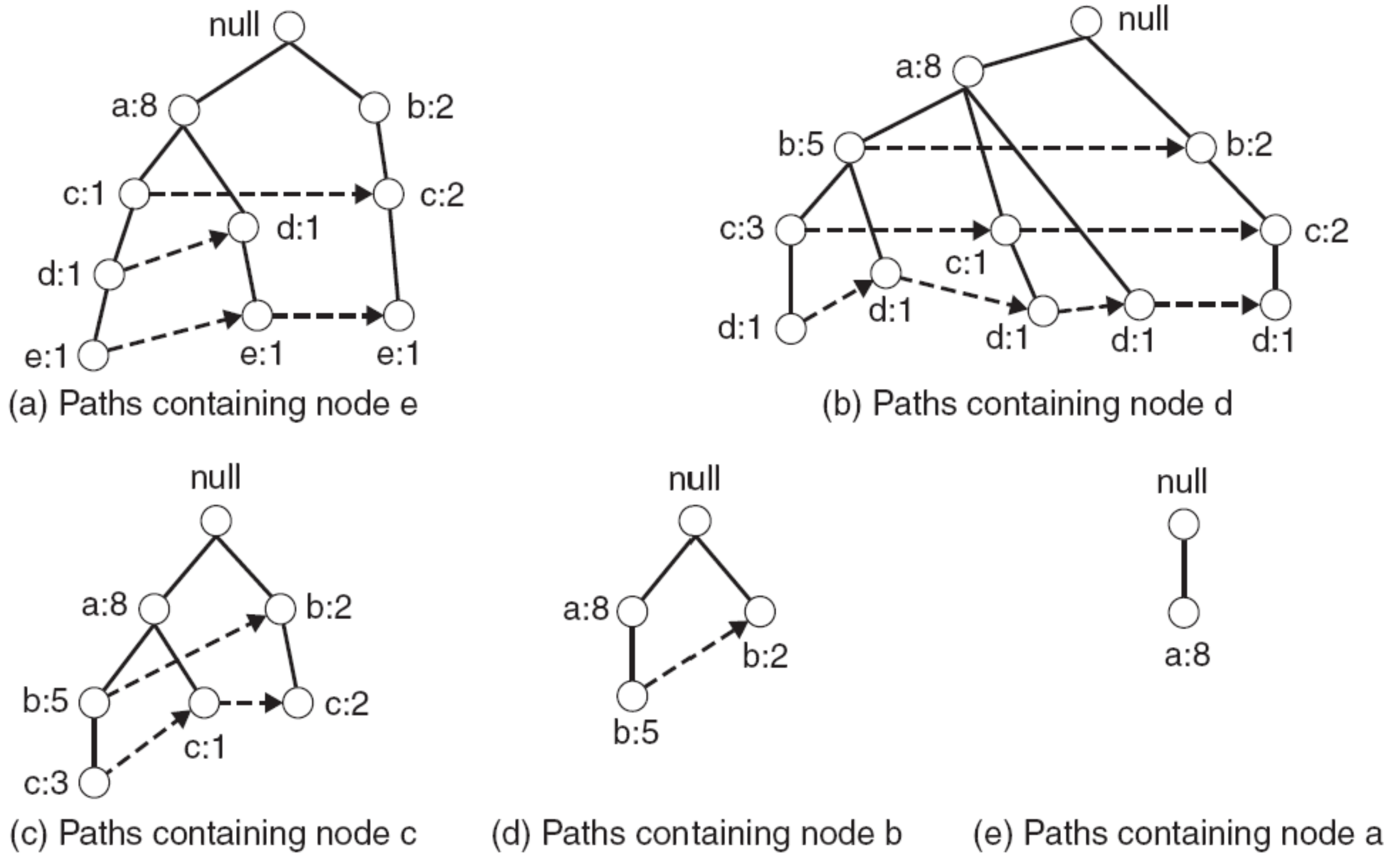


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

FP-Growth Algorithm: Frequent Itemset Generation

bottom-up,
suffix classes,
in inverse
order of items



null
a:8
b:2
c:1
c:2
d:1
d:1
e:1
e:1
e:1
(a) Paths containing node e
null
a:8
b:5
b:2
c:3
c:1
c:2
d:1
d:1
d:1
d:1
d:1
(b) Paths containing node d
null
a:8
b:2
b:5
c:1
c:2
c:3
(c) Paths containing node c
null
a:8
b:2
b:5
(d) Paths containing node b
null
a:8
(e) Paths containing node a

Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.

FP-Growth Algorithm: Frequent Itemset Generation

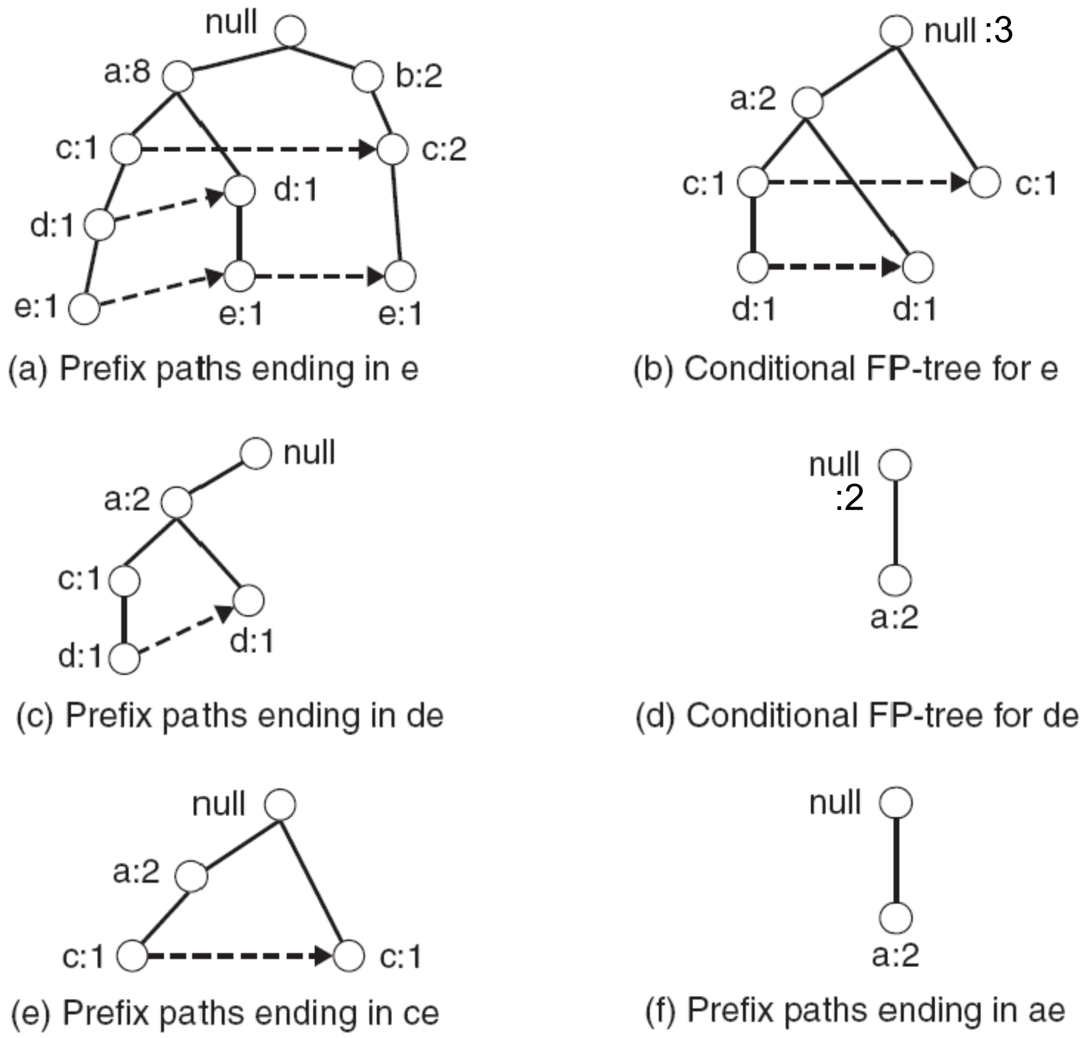

null
a:8
b:2
c:1
c:2
d:1
d:1
e:1
e:1
e:1
(a) Prefix paths ending in e
null :3
a:2
c:1
c:1
d:1
d:1
(b) Conditional FP-tree for e
null
a:2
c:1
d:1
d:1
(c) Prefix paths ending in de
null :2
a:2
(d) Conditional FP-tree for de
null
a:2
c:1
c:1
(e) Prefix paths ending in ce
null
a:2
(f) Prefix paths ending in ae

Figure 6.27. Example of applying the FP-growth algorithm to find frequent itemsets ending in e.

Subproblem:
generate frequent itemsets ending with e from initial (null-conditional) FP-tree

1. Check support_count(e)
2. If {e} is frequent:

   output e;
   new subproblems:
   generate freq.itemsets ending with de,ce,be, or ae from e-conditional FP-tree

Construct e-conditional FP-tree (to represent patterns before e) from null-conditional FP-tree:

0. Traverse paths backwards from all occurrences of e
1. Adapt counts
2. Omit e-conditionally infrequent items