Association analysis

- Many business enterprises accumulate large quantities of data from their daily operations.
- These data are commonly known as market basket transactions.
- Each row in the table corresponds to a transaction, which contains
  - A unique identifier labeled TID
  - A set of items bought by a given customer

Association analysis

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}

Association analysis

- Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers.
- These information can be used to support a variety of business-related applications such as
  - Marketing promotions
  - Inventory management
  - Customer relationship management.

Association analysis

- Association analysis is useful for discovering interesting relationships hidden in large data sets.
- The uncovered relationships can be represented in the form of association rules or sets of frequent items.

Association analysis

- The following rule can be extracted from the previous data set
  - {Diapers}→{Beer}
- The rule suggests that a strong relationship exists between the sale of diapers and beer.
- Retailers can use this type of rules to help them identify new opportunities for cross-selling their products to customers.

Association analysis

- There are two key issues that need to be addressed when applying association analysis to market basket data.
- First, discovering patterns from a large transaction data set can be computationally expensive.
- Second, some of the discovered patterns are potentially spurious because they may happen simply by chance.

Binary representation

- Market basket data can be represented in a binary format as shown in the following table.
- Each row corresponds to a transaction.
- Each column corresponds to an item.
- An item can be treated as a binary variable whose value is
  - One if the item is present in a transaction
  - Zero otherwise

Binary representation

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Binary representation

- The presence of an item in a transaction is often considered more important than its absence.
- This is an example of an asymmetric binary variable.
- The binary representation ignores certain aspects of the data such as
  - The quantity of items sold
  - The price paid to purchase them

Itemset and support count

- Let $I=\{i_1,i_2,\ldots\ldots,i_d\}$ be the set of all items in a market basket data.
- Let $T=\{t_1,t_2,\ldots\ldots,t_N\}$ be the set of all transactions.
- Each transaction t_i contains a subset of items chosen from I.

Itemset and support count

- A collection of zero or more items is called an itemset.
- If an itemset contains k items, it is called a k-itemset.
- The null (or empty) set is an itemset that does not contain any items.

Itemset and support count

- A transaction t_j is said to contain an itemset X if X is a subset of t_j.
- The support count of an itemset is the number of transactions which contain that particular itemset.
- Formally, the support count $\sigma(X)$ for an itemset X can be stated as

$$\sigma(X) = \left|\left\{t_i \mid X \subseteq t_i, t_i \in T\right\}\right|$$

Association rule

- An association rule is an implication expression of the form $X \rightarrow Y$.
- X and Y are disjoint itemsets.
- The strength of an association rule can be measured in terms of its support and confidence.

Support and confidence

- Support determines how often a rule is applicable to a given data set.
- Confidence determines how frequently items in Y appear in transactions that contain X.
- The formal definition of these metrics are

$$Support, s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$Confidence, c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Support and confidence

- Consider the rule {Milk, Diapers}$\rightarrow${Beer}
- From the previous table
  - The support count for {Milk, Diapers, Beer} is 2.
  - The total number of transactions is 5.
- Therefore, the rule’s support is 2/5=0.4.

Support and confidence

- The rule's confidence is obtained by dividing the support count for {Milk, Diapers, Beer} by the support count for {Milk, Diapers}.
- There are 3 transactions that contain milk and diapers.
- Therefore, the confidence for this rule is 2/3=0.67.

Support and confidence

- A rule that has very low support may occur simply by chance.
- A low support rule is likely to be uninteresting from a business perspective.
- For this reason, support is often used to eliminate uninteresting rules.

Support and confidence

- Confidence measures the reliability of the inference made by a rule.
- For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X.
- Confidence also provides an estimate of the conditional probability of Y given X.

Support and confidence

- The inference made by an association rule does not necessarily imply causality.
- Instead, it suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule.
- Causality, on the other hand, requires knowledge about cause and effects in the data.

Association rule mining

- The association rule mining problem can be formally stated as follows:
  - Given a set of transactions T, find all the rules having support $\geq$ minsup and confidence $\geq$ minconf.
  - minsup and minconf are the corresponding support and confidence thresholds.

Association rule mining

- A brute force approach for mining association rules is to compute the support and confidence for every possible rule.
- This approach is prohibitively expensive because there are a large number of rules that can be extracted from a data set.

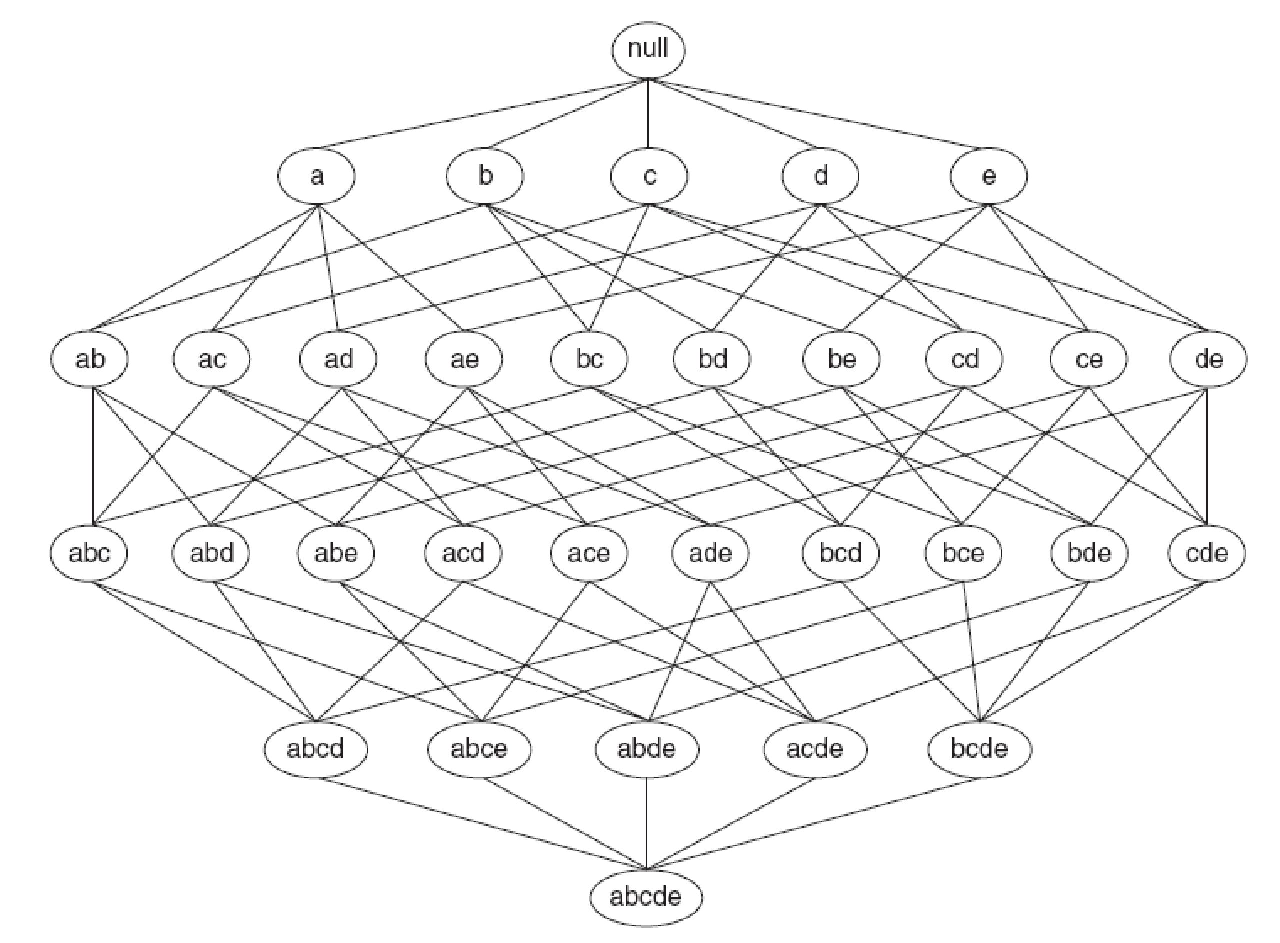
Association rule mining

- A common strategy is to decompose the problem into two major subtasks:
  - Frequent itemset generation
    - The objective of this step is to find all the itemsets that satisfy the minsup threshold.
    - These itemsets are called frequent itemsets.
  - Rule generation
    - The objective of this step is to extract all the high-confidence rules from the frequent itemsets found in the previous step.
    - These rules are called strong rules.

Frequent itemset generation

- A lattice structure can be used to enumerate the list of all possible itemsets.
- The following figure shows an itemset lattice for $I=\{a,b,c,d,e\}$.
- In general, a data set that contains k items can potentially generate up to 2^k-1 frequent itemsets, excluding the null set.
- As a result, the search space of itemsets that need to be explored is exponentially large.

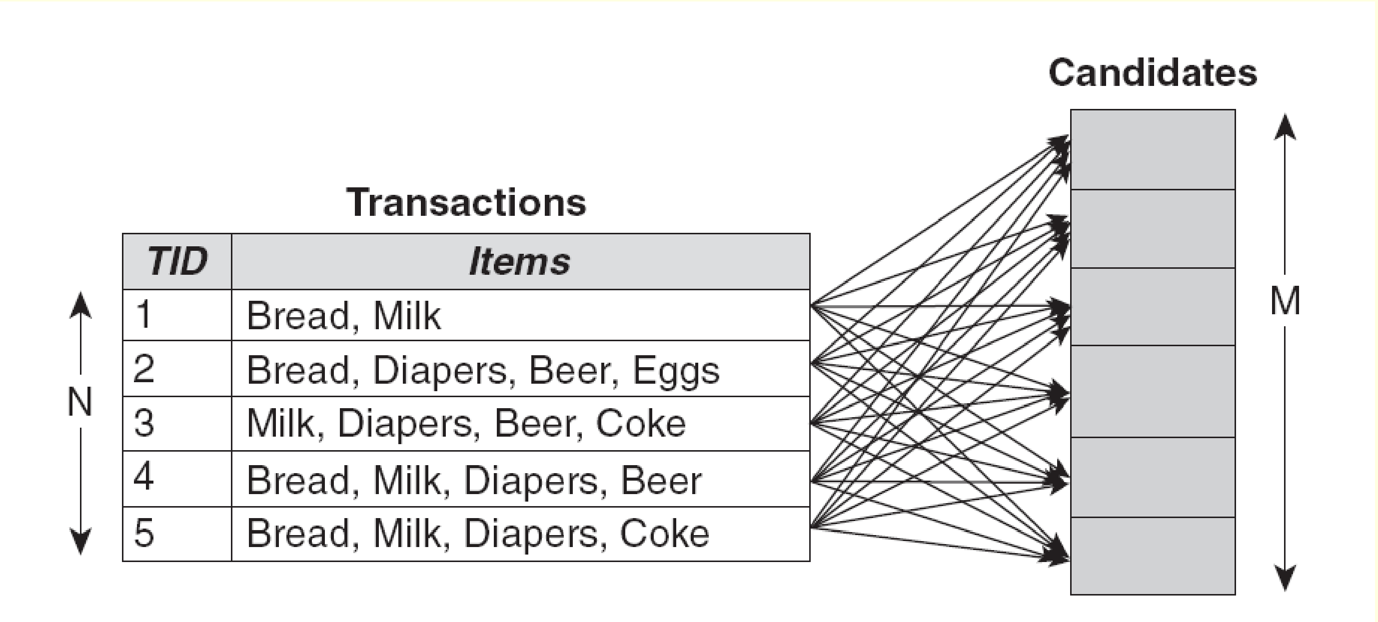
Frequent itemset generation



Frequent itemset generation

- A brute force approach for finding frequent itemsets is to determine the support count of every candidate itemset in the lattice.
- To do this, we need to compare every candidate against every transaction.
- If the candidate is contained in a transaction, its support count will be incremented.
- In the following figure, the support count for {Bread, Milk} is incremented three times because the itemset is contained in transactions 1,4 and 5.

Frequent itemset generation



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

Frequent itemset generation

- There are several ways to reduce the computational complexity of frequent itemset generation
  - Reduce the number of candidate itemsets
    - The Apriori principle is an effective way to eliminate some of the candidate itemsets without counting their support values.
  - Reduce the number of comparisons
    - We can reduce the number of comparisons by using more advanced data structures.

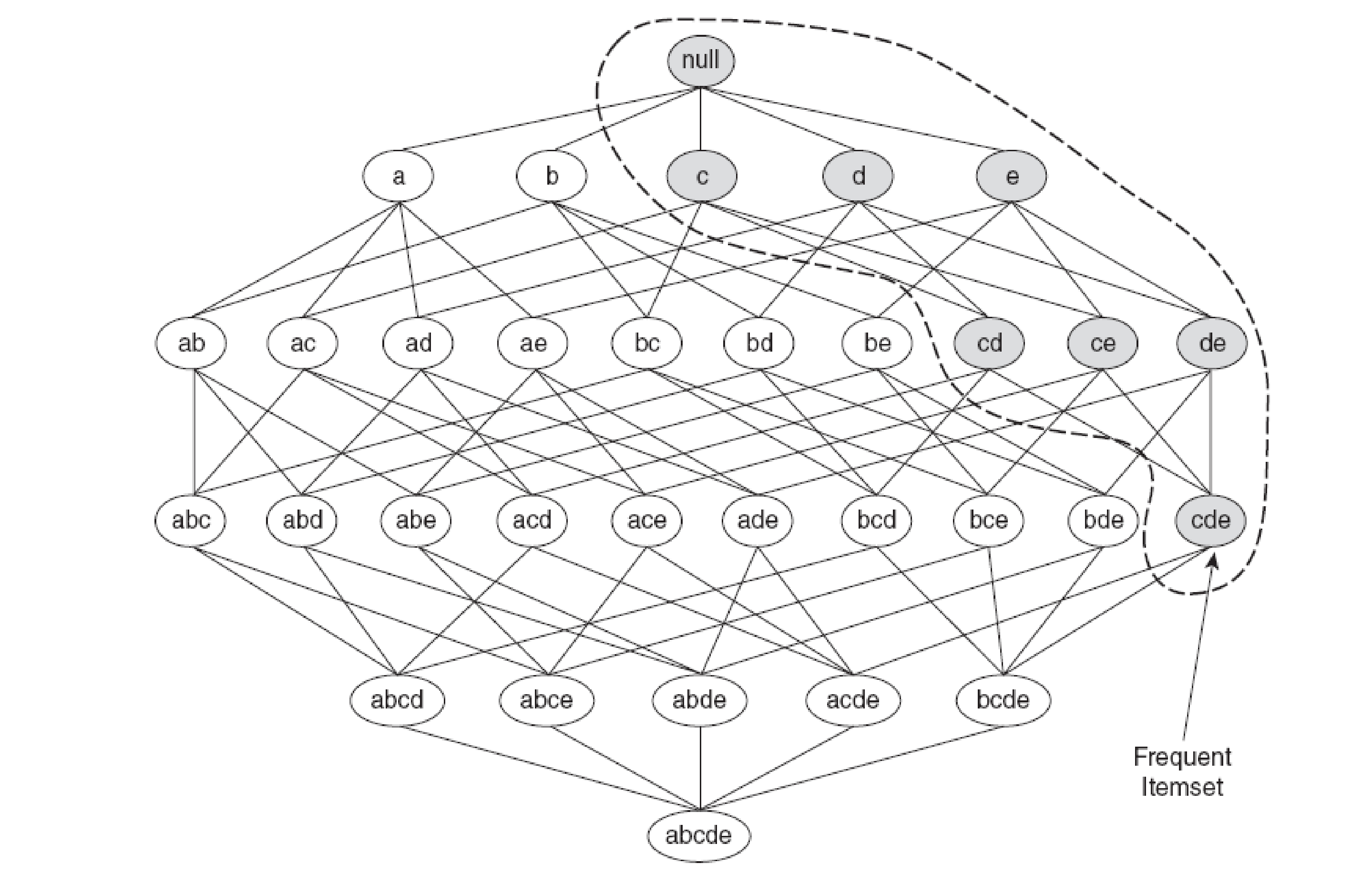
The Apriori principle

- The support measure helps to reduce the number of candidate itemsets explored during frequent itemset generation.
- The use of support for pruning candidate itemsets is guided by the Apriori principle.
- The Apriori principle
  - If an itemset is frequent, then all of its subsets must also be frequent.

The Apriori principle

- We consider the itemset lattice shown in the following figure.
- Suppose {c,d,e} is a frequent itemset.
- Any transaction that contains {c,d,e} must also contain its subsets, {c,d}, {c,e}, {d,e}, {c}, {d} and {e}.
- As a result, if {c,d,e} is frequent, then all subsets of {c,d,e} must also be frequent.

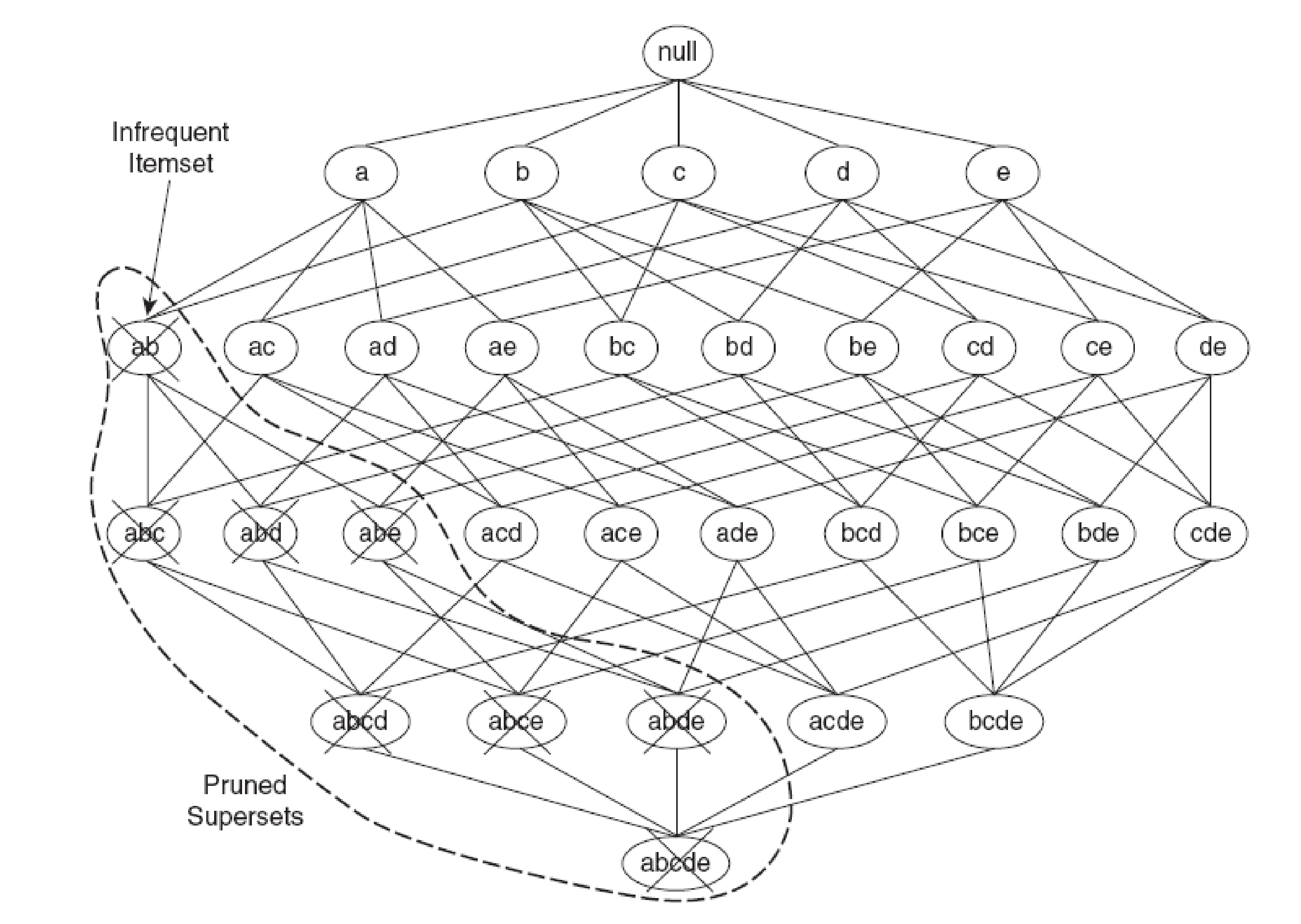
The Apriori principle



The Apriori principle

- Conversely, if an itemset is infrequent, then all of its supersets must be infrequent.
- This strategy of trimming the search space based on the support measure is known as support-based pruning.
- Such a pruning strategy is made possible by the anti-monotone property of the support measure.

The Apriori principle



The Apriori principle

- Let I be a set of items.
- Let J=2^{I} be the power set of I.
- A measure f is monotone if
  - $\forall X, Y \in J : (X \subseteq Y) \rightarrow f(X) \leq f(Y)$
- A measure f is anti-monotone if
  - $\forall X, Y \in J : (X \subseteq Y) \rightarrow f(Y) \leq f(X)$

The Apriori algorithm

- Apriori is the first algorithm that uses support-based pruning to control the exponential growth of candidate itemsets.
- We apply this algorithm to the transactions shown in the previous example.
- We assume that the support threshold is 60%.
- This is equivalent to a minimum support count of 3.

The Apriori algorithm

- Initially, every item is considered as a candidate 1-itemset.
- After counting their supports, the candidate itemsets {Cola} and {Eggs} are discarded.
- This is because they appear in fewer than three transactions.

The Apriori algorithm

Candidate 1-Itemsets

Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Minimum support count = 3

Candidate 2-Itemsets

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

Itemsets removed because of low support

Candidate 3-Itemsets

Itemset	Count
{Bread, Diapers, Milk}	2

The Apriori algorithm

- In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets.
- This is because the Apriori principle ensures that all supersets of the infrequent 1-itemsets must be infrequent.
- There are only four frequent 1-itemsets.
- As a result, the number of candidate 2-itemsets generated is $\binom{4}{2} = 6$

The Apriori algorithm

- Two of these six candidates, {Beer, Bread} and {Beer, Milk}, are subsequently found to be infrequent.
- The remaining four candidates are frequent.
- They will be used to generate candidate 3-itemsets.

The Apriori algorithm

- Without support-based pruning, there are $\binom{6}{3} = 20$ candidate 3-itemsets that can be formed using the six items in this example.
- With the Apriori principle, we only need to keep candidate 3-itemsets whose subsets are frequent.
- The only candidate that has this property is {Bread, Diapers, Milk}.

The Apriori algorithm

- The number of candidates produced based on a brute force strategy of enumerating all itemsets is
  - $$\binom{6}{1}+\binom{6}{2}+\binom{6}{3}+\binom{6}{4}+\binom{6}{5}+\binom{6}{6}=6+15+20+15+6+1=2^6-1=63$$
- With the Apriori principle, this number decreases to
  - $$\binom{6}{1}+\binom{4}{2}+1=6+6+1=13$$
- This represents a significant reduction in the number of candidate itemsets.

The Apriori algorithm

- Let C_k denote the set of candidate k-itemsets.
- Let F_k denote the set of frequent k-itemsets.
- The algorithm initially makes a single pass over the data set to determine the support count of each item.
- Upon completion of this step, the set of all frequent 1-itemsets, F_1, will be known.

The Apriori algorithm

- Next, the algorithm will iteratively generate new candidate k-itemsets.
- These itemsets are generated using the frequent (k-1)-itemsets found in the previous iteration.

The Apriori algorithm

- To count the support of the candidates, the algorithm needs to make an additional pass over the data set.
- After determining their support counts, the algorithm eliminates all candidate itemsets whose support counts are less than the threshold.
- The algorithm terminates when there are no new frequent itemsets generated.

The Apriori algorithm

- The frequent itemset generation part of the Apriori algorithm has two important characteristics.
- First, it is a level-wise algorithm
  - It traverses the itemset lattice one level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets.
- Second, it employs a generate-and-test strategy for finding frequent itemset.
  - At each iteration, new candidate itemsets are generated from the frequent itemsets found in the previous iteration.
  - After a pruning process, the support count of each remaining candidate is then determined and tested against the threshold.

Candidate generation and pruning

- Candidate itemsets are generated by performing the following two operations
  - Candidate generation
    - This operation generates new candidate k-itemsets based on the frequent (k-1)-itemsets found in the previous iteration.
  - Candidate pruning
    - This operation eliminates some of the candidate k-itemsets.

Candidate generation and pruning

- Consider a candidate k-itemset, $X=\{i_1,i_2,\ldots\ldots,i_k\}$.
- The algorithm must determine whether all of the subsets, $X-\{i_j\}$, $j=1,2,\ldots\ldots,k$, are frequent.
- If one of them is infrequent, then X is immediately pruned.
- This approach can effectively reduce the number of candidate itemsets considered during support counting.

Candidate generation and pruning

- We do not have to examine all k subsets of a given candidate itemset.
- Suppose m of the k subsets were used to generate a candidate.
- In this case, we only need to check the remaining k-m subsets during candidate pruning.

Candidate generation and pruning

- There are a number of requirements for an effective candidate generation procedure.
- First, it should avoid generating too many unnecessary candidates.
  - A candidate itemset is unnecessary if at least one of its subsets is infrequent.
  - Such a candidate is guaranteed to be infrequent according to the anti-monotone property of support.

Candidate generation and pruning

- It must ensure that the candidate set is complete.
  - In other words, no frequent itemsets are left out by the candidate generation procedure.
  - To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall k : F_k \subseteq C_k$
- It should not generate the same candidate itemset more than once.
  - Generation of duplicate candidates leads to wasted computations.

Candidate generation and pruning

- We now introduce the candidate generation procedure used in the Apriori algorithm.
- In this algorithm, a pair of frequent (k-1)-itemsets are merged only if their first k-2 items are identical.
- This process is called apriori-gen.

Candidate generation and pruning

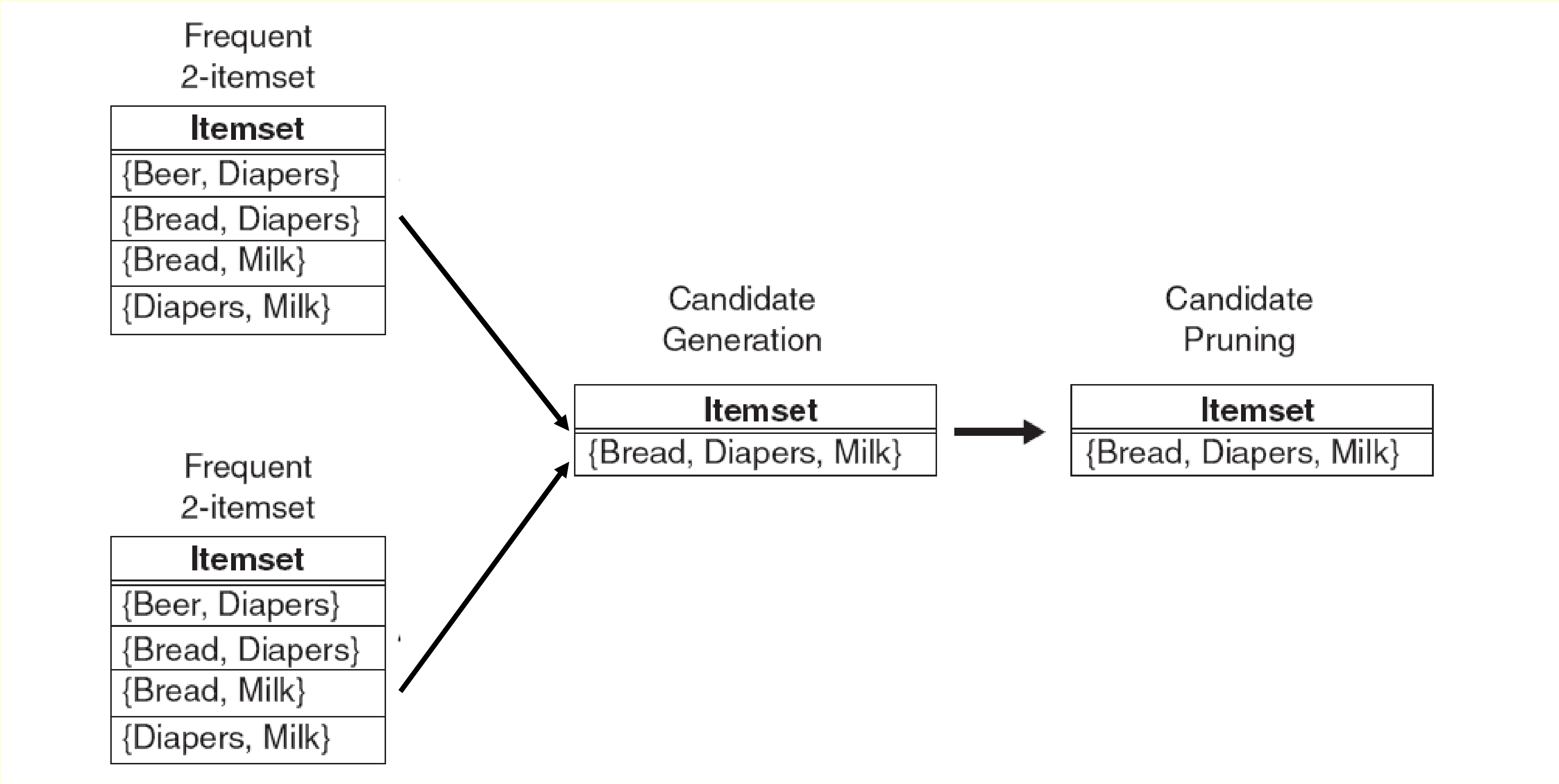
- Let A={a_1,a_2,……,a_{k-1}} and B={b_1,b_2,……,b_{k-1}} be a pair of frequent (k-1)-itemsets.
- A and B are merged if they satisfy the following conditions:

$$a_i = b_i \ \text{(for } i = 1,2,\ldots\ldots,k-2\text{) and } a_{k-1} \neq b_{k-1}$$

Candidate generation and pruning

- In the following example, the frequent itemset {Bread, Diapers} and {Bread, Milk} are merged to form {Bread, Diapers, Milk}.
- An additional candidate pruning step is required to ensure that the remaining k-2 subsets of the candidate are frequent.

Candidate generation and pruning



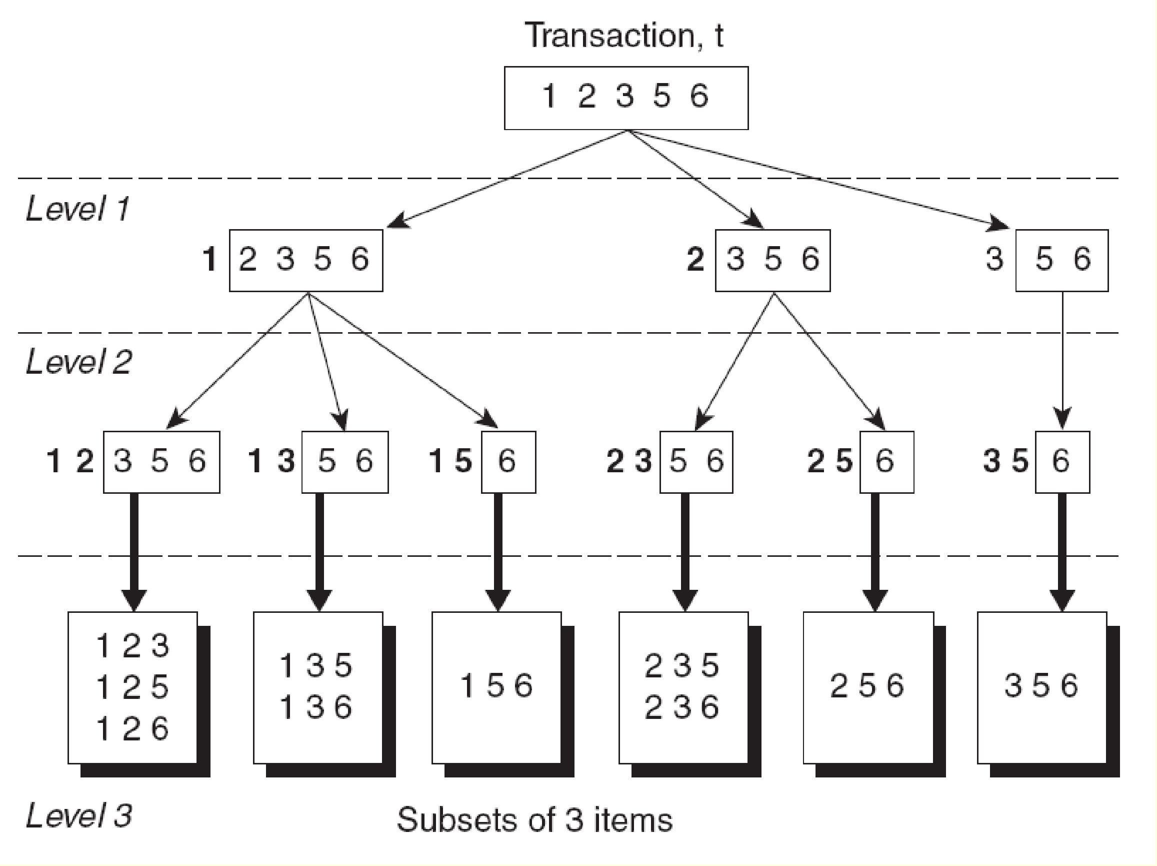
Support counting

- Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step.
- We need to compare a transaction against the candidate itemsets.
- We then update the support counts of candidates contained in the transaction.

Support counting

- Consider a transaction t that contains five items, {1,2,3,5,6}.
- The following example shows a systematic way for enumerating the 3-itemsets contained in t.
- We assume that each itemset keeps its items in increasing lexicographic order.

Support counting



Support counting

- Let t={1,2,3,5,6}
- All the 3 itemsets contained in t must begin with item 1,2 or 3.
- It is not possible to construct a 3-itemset that begins with items 5 or 6.
- This is because there are only two items in t whose labels are greater than or equal to 5.

Support counting

- The number of ways to specify the first item of a 3-itemset in t is illustrated by the Level 1 prefix structures in the previous figure.
- For example, 1 $\boxed{2\ 3\ 5\ 6}$ represents a 3-itemset that
  - Begins with item 1 and
  - Followed by two more items chosen from the set {2,3,5,6}

Support counting

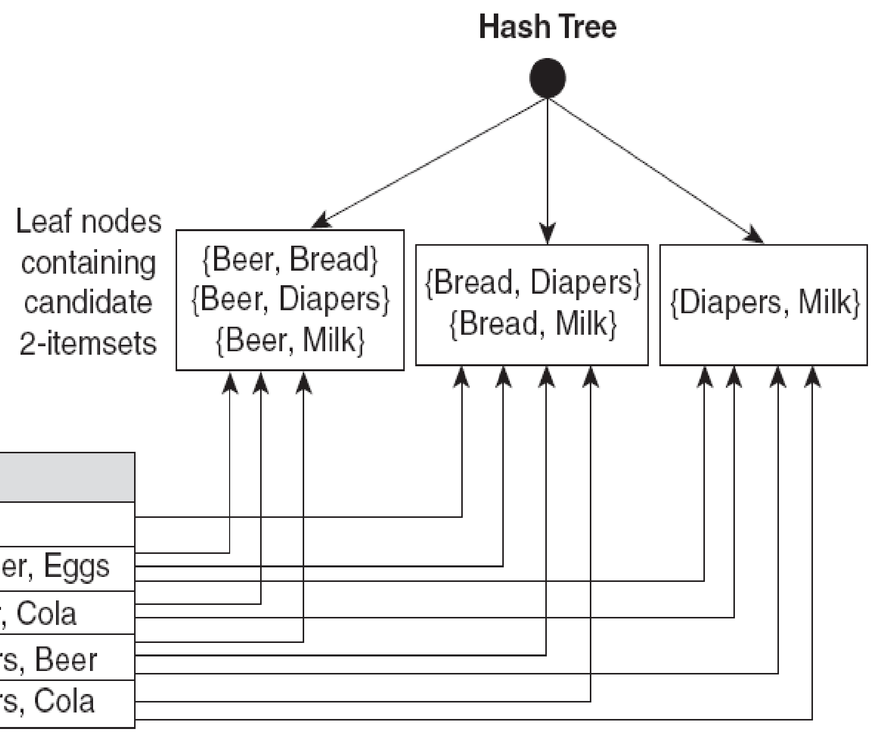
- After fixing the first item, the prefix structures at Level 2 represent the number of ways to select the second item.
- For example, 1 2 $\boxed{3\ 5\ 6}$ corresponds to itemsets that
  - Begin with prefix {1 2} and
  - Followed by items 3,5 or 6.

Support counting

- Finally, the prefix structures at Level 3 represent the complete set of 3-itemsets in t.
- For example, the 3-itemsets that begin with {1,2} are {1,2,3}, {1,2,5} and {1,2,6}.
- On the other hand, those that begin with prefix {2,3} are {2,3,5} and {2,3,6}.

Support counting using a hash tree

- We can partition the candidate itemsets into different buckets and store them in a hash tree.



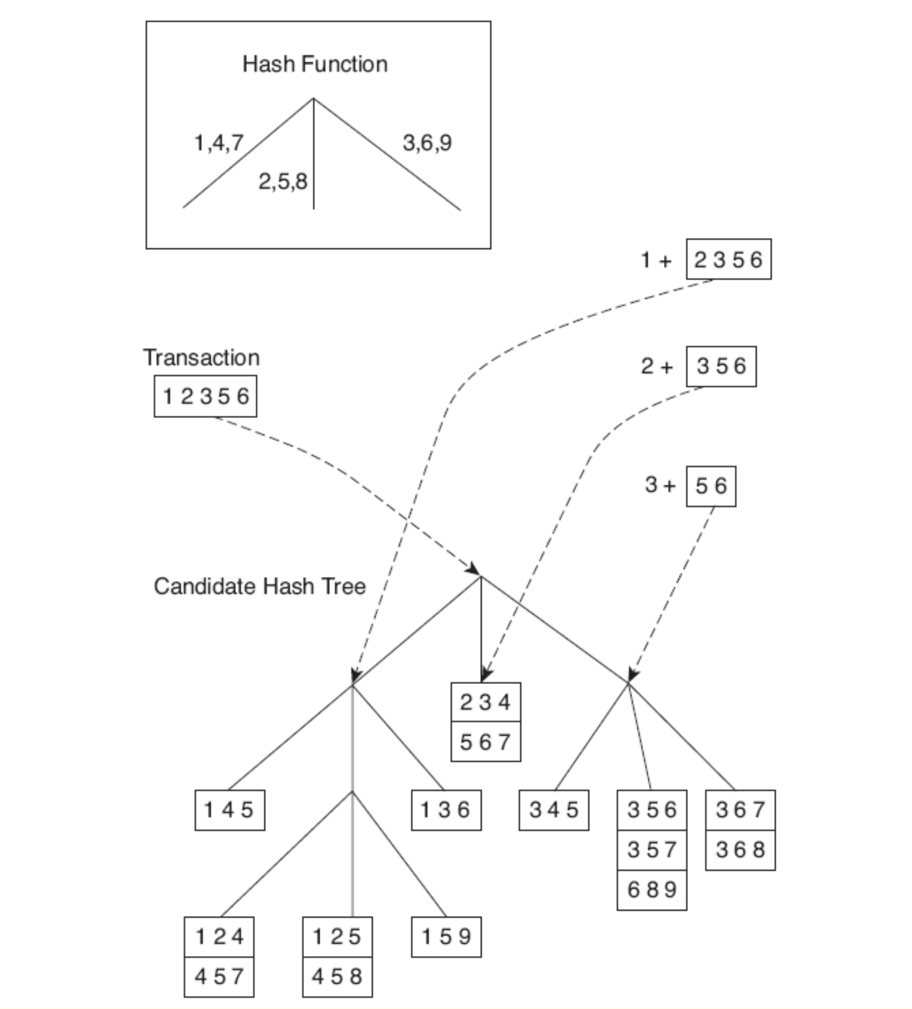
Transactions

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

Support counting using a hash tree

- The following figure shows an example of a hash tree structure.
- Each internal node of the tree implements the hash function, h(p)=p mod 3.
- This function is used to determine which branch of the current node should be followed next.
- For example, items 1,4 and 7 are hashed to the same branch, i.e., the leftmost branch.
- This is because they have the same remainder after diving the number by 3.

Support counting using a hash tree



Support counting using a hash tree

- Consider a transaction t={1,2,3,5,6}
- To update the support counts of the candidate itemsets, we need to traverse the hash tree.

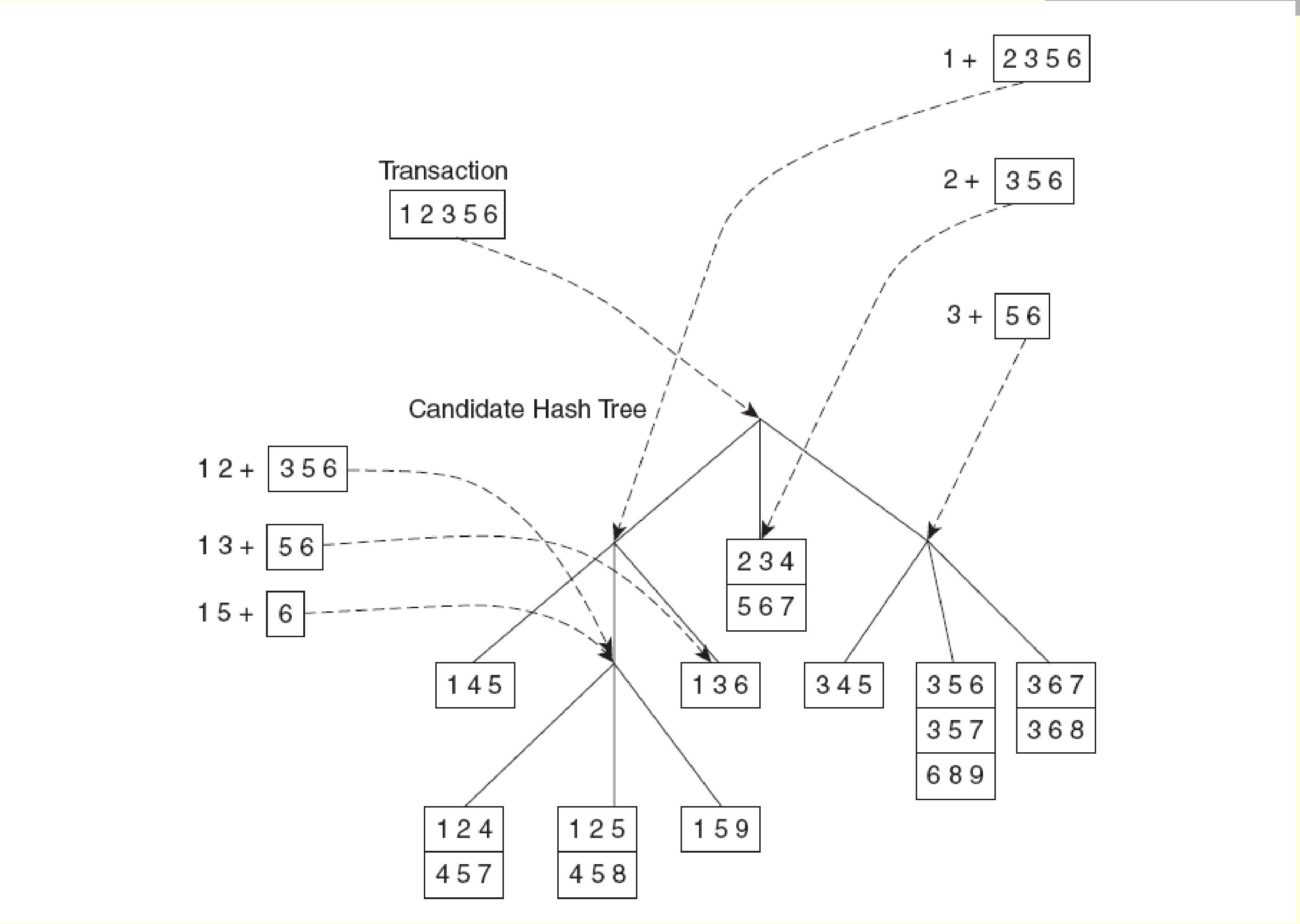
Support counting using a hash tree

- The 3-itemsets in t must begin with items 1,2 or 3, as indicated by the Level 1 prefix structures.
- At the root node of the hash tree, the items 1,2 and 3 are hashed separately
  - Item 1 is hashed to the left child
  - Item 2 is hashed to the middle child
  - Item 3 is hashed to the right child

Support counting using a hash tree

- At the next level, the transaction is hashed on the second item listed in the Level 2 structures.
- For example, after hashing on item 1 at the root node, items 2, 3, and 5 are hashed.
  - Items 2 and 5 are hashed to the middle child.
  - Item 3 is hashed to the right child.

Support counting using a hash tree



Support counting using a hash tree

- This process continues until the leaf nodes of the hash tree are reached.
- If a candidate is a subset of the transaction, its support count is incremented.
- In this example, 5 out of the 9 leaf nodes are visited.

Rule generation

- Each frequent k-itemset Y can produce up to 2^k-2 association rules.
- We ignore rules that have empty antecedents and consequents, e.g. $\emptyset \rightarrow Y$ or $Y \rightarrow \emptyset$.
- An association rule can be extracted by partitioning the itemset Y into two non-empty subsets.
- Specifically, the subsets X and Y-X should form a rule $X \rightarrow Y-X$ that satisfies the confidence threshold.
- All such rules must have already met the support threshold since they are generated from a frequent itemset.

Rule generation

- Let Y={1,2,3} be a frequent itemset.
- There are six candidate association rules that can be generated from Y
  - $\{1,2\} \rightarrow \{3\}$
  - $\{1,3\} \rightarrow \{2\}$
  - $\{2,3\} \rightarrow \{1\}$
  - $\{1\} \rightarrow \{2,3\}$
  - $\{2\} \rightarrow \{1,3\}$
  - $\{3\} \rightarrow \{1,2\}$
- The support of each rule is identical to the support for Y.
- As a result, the rules satisfy the support threshold.

Rule generation

- Consider the rule $\{1,2\} \rightarrow \{3\}$ generated from the frequent itemset $Y=\{1,2,3\}$.
- The confidence of this rule is $\sigma(\{1,2,3\})/\sigma(\{1,2\})$.
- Since {1,2,3} is frequent, the anti-monotone property of support ensures that {1,2} is also frequent.
- The support counts for both itemsets were already found during frequent itemset generation.

Rule generation

- The following property holds for the confidence measure:
  - Suppose a rule $X \rightarrow Y-X$ does not satisfy the confidence threshold.
  - Then any rule $X' \rightarrow Y-X'$, where X' is a subset of X, will not satisfy the confidence threshold.

Rule generation

- To verify this property, consider the following two rules
  - $X \rightarrow Y-X$
  - $X' \rightarrow Y-X'$, where X' is a subset of X.
- The confidence of the rules are $\sigma(Y)/\sigma(X)$ and $\sigma(Y)/\sigma(X')$ respectively.
- Since X' is a subset of X, $\sigma(X') \geq \sigma(X)$.
- As a result, the second rule cannot have a higher confidence than the first rule.

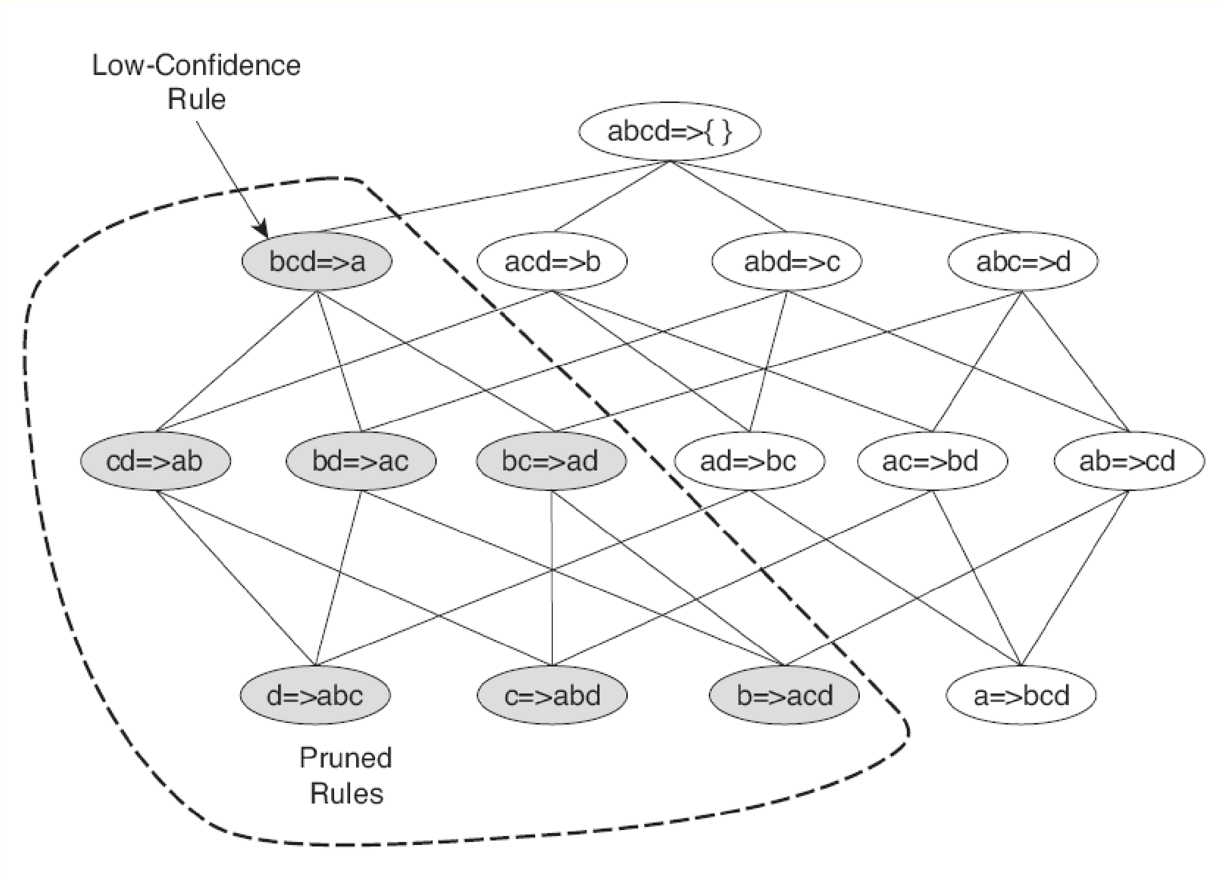
Rule generation

- The Apriori algorithm uses a level-wise approach for generating association rules.
- Each level corresponds to the number of items that belong to the rule consequent.
- Initially, all the high-confidence rules that have only one item in the rule consequent are extracted.
- These rules are then used to generate new candidate rules.

Rule generation

- We consider the frequent itemset {a,b,c,d}.
- The following figure shows a lattice structure for the association rules generated from {a,b,c,d}.

Rule generation



Rule generation

- Suppose the confidence for {b,c,d}→{a} is low.
- All the rules containing item a in its consequent can be discarded.
- In general, if any node in the lattice has low confidence, then the entire subgraph spanned by the node can be pruned immediately.

Rule generation

- An example of rule generation:
  - Suppose $\{a,c,d\} \rightarrow \{b\}$ and $\{a,b,d\} \rightarrow \{c\}$ are high confidence rules.
  - Then the candidate rule $\{a,d\} \rightarrow \{b,c\}$ is generated by merging the consequents of both rules.
- In general, new rules are generated by merging the consequents of two high confidence rules using the apriori-gen step.