Contents of this Chapter

The Frequent Pattern Mining Model [Aggarwal 2015, section 4.2]

Association Rules [sections 4.3 and 4.5]

The Apriori Algorithm [section 4.4.2]

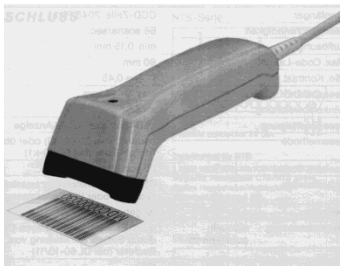
Enumeration-Tree Algorithms [section 4.4.3]

Suffix-based Pattern Growth Methods [section 4.4.4]

Pattern Summarization [section 5.2]

Introduction

Motivation



{butter, bread, milk, sugar}
{butter, flour, milk, sugar}
{butter, eggs, milk, salt}
{eggs}
{butter, flour, milk, salt, sugar}

Database of Sales Transactions

Market basket analysis

- Which products are frequently purchased together?
- Applications
  - Improvement of store layouts
  - Cross marketing
  - Attached mailings/add-on sales

Frequent Patterns

Basic assumption

Frequent patterns are more interesting than infrequent ones.

Major challenge

Efficiently finding (all) frequent patterns

Types of frequent patterns

- Frequent *itemsets* (boolean attributes)
- *Generalized* frequent *itemsets* (boolean attributes with concept hierarchies)
- *Quantitative* frequent *sets* (numerical attributes)
- Frequent *sequences* (sequence data)
- Frequent *subgraphs* . . .

The Frequent Pattern Mining Model

Definitions [Agrawal & Srikant 1994]

- *Items* $I = \{i_1, ..., i_m\}$ a set of literals
- *Itemset* X: set of items $X \subseteq I$
- *Database* D: set of *transactions* T_i where $T_i \subseteq I$
- T *contains* X: $X \subseteq T$
- Items in transactions or itemsets are sorted in lexicographic order:

  Item set $X = (x_1, x_2, ..., x_k)$, where $x_1 \leq x_2 \leq ... \leq x_k$
- *Length of itemset*: number of elements of itemset
- k-itemset: itemset of length k

The Frequent Pattern Mining Model

Definitions

- *Support of itemset X in D*: percentage of transactions in D containing X

$$\sup(X, D) = \frac{|\{T \in D \mid X \subseteq T\}|}{|D|}$$

- *Frequent itemset X in D:* item set X with

$$freq(X, D) :\Leftrightarrow \sup(X, D) \geq \text{minsup}$$

- *Association rule*: implication of the form $X \Rightarrow Y$,

where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$

The Frequent Pattern Mining Model

Definitions

- *Support s of association rule* $X \Rightarrow Y$ *in* D:

  $$s = \frac{|\{T \in D \mid (X \cup Y) \subseteq T\}|}{|D|}$$

  support of $X \cup Y$ in D

- *Confidence c of association rule* $X \Rightarrow Y$ *in* D:

  $$c = \frac{|\{T \in D \mid X \cup Y \subseteq T\}|}{|\{T \in D \mid X \subseteq T\}|}$$

  percentage of transactions containing Y

  in the subset of all transactions in D that contain X

- *Task*: discover all association rules that have support ≥ *minsup*

  and confidence ≥ *minconf* in D.

The Frequent Pattern Mining Model

Example

TransactionID	Items
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

minsup = 50%,
minconf = 50%

Support

(A): 75%, (B), (C): 50%, (D), (E), (F): 25%,

(A, C): 50%, (A, B), (A, D), (B, C), (B, E), (B, F), (E, F): 25%

Association rules

$A \Rightarrow C$ (support = 50%, confidence = 66.6%)

$C \Rightarrow A$ (support = 50%, confidence = 100%)

Association Rules

Mining Association Rules

1. Determine the frequent itemsets in the database

„Naive“ algorithm:

count the frequencies of all k-itemsets $\subseteq I$

inefficient, since $\binom{m}{k}$ such itemsets

2. Generate the association rules from the frequent itemsets

Itemset X frequent and $A \subseteq X$

$A \Rightarrow (X - A)$ satisfies minimum support constraint

→ confidence has to be checked.

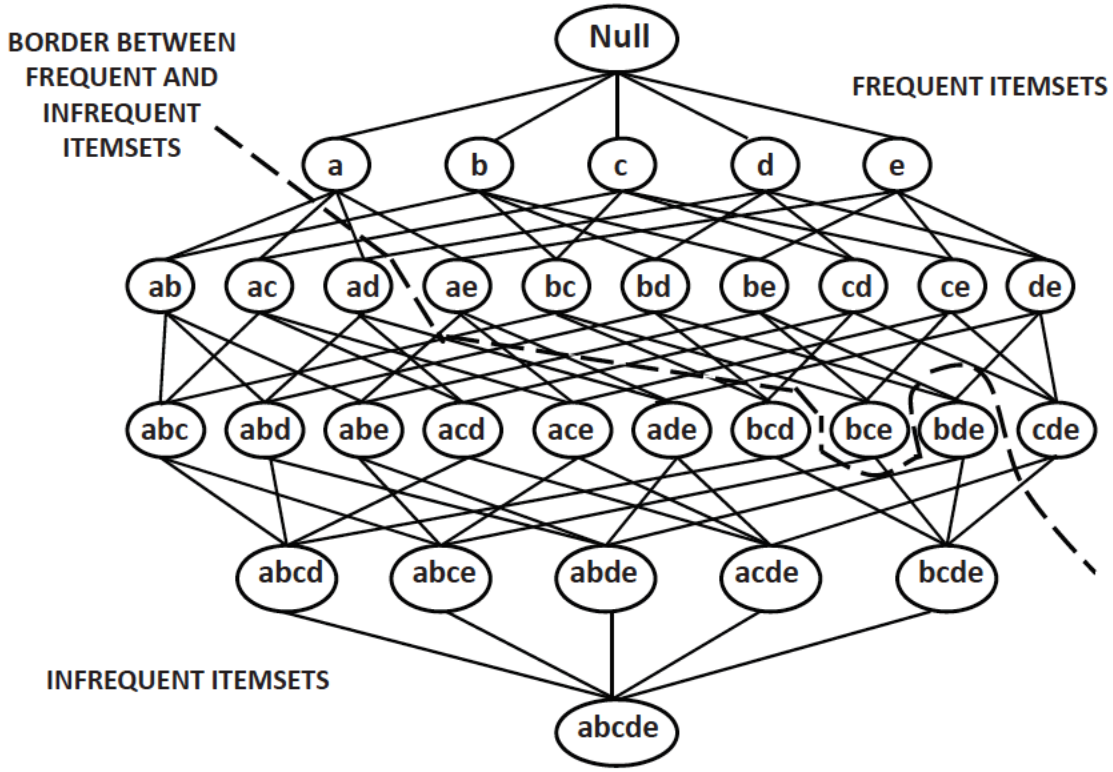
Itemset Lattice

- Lattice: a partially ordered set with unique least upper bound and greatest lower bound.
- Itemset lattice

  - elements: itemsets $X_1 \subseteq I, X_2 \subseteq I, \ldots X_n \subseteq I$
  - partial order: $X_1 < X_2 :\Leftrightarrow X_1 \subset X_2$
  - least upper bound: I
  - greatest lower bound: $\varnothing$

Association Rules

Itemset Lattice



Association Rules

Anti-monotonicity (downward closure) property

Each subset of a frequent itemset is also frequent.

$$\forall T_1 \subseteq I, T_2 \subseteq I: \quad T_1 \subseteq T_2 \wedge freq(T_2, D) \Rightarrow freq(T_1, D)$$

because of

$$\forall T_1 \subseteq I, T_2 \subseteq I: \quad T_1 \subseteq T_2 \Rightarrow \sup(T_1, D) \geq \sup(T_2, D)$$

If one subset is not frequent, then superset cannot be frequent.

This property makes frequent itemset mining efficient, since in practice most itemsets are infrequent.

Association Rules

Computing the Association Rules

- Given a frequent itemset X.
- For each (frequent!) subset A of X, form the rule $A \Rightarrow (X - A)$.
- Compute confidence of the rule $A \Rightarrow (X - A)$

$$confidence(A \Rightarrow (X - A)) = \frac{\sup(X)}{\sup(A)}$$

- Discard rules that do not have minimum confidence.
- Store frequent itemsets with their supports in a hash table.

⇒ no DB accesses (no disk I/O).

Association Rules

Interestingness of Association Rules

Application

- Data about the behavior of students at a school with 5000 students.

Example

- Itemsets with support:

  60% of the students play soccer, 75% of the students eat candy bars

  40% of the students play soccer *and* eat candy bars

- Association rules:

  „play soccer“ ⇒ „eat candy bars“, confidence = 67%

  TRUE ⇒ „eat candy bars“, confidence = 75%

„play soccer“ and „eat candy bars“ are *negatively correlated.*

Interestingness of Association Rules

- Filter out misleading association rules.
- Expected support for rule $A \Rightarrow B$

$$P(A \cup B) = P(A) \cdot P(B)$$

assuming the independence of A and B

- Interestingness measure for rule $A \Rightarrow B$

$$\frac{P(A \cup B)}{P(A)} - P(B)$$

- The larger this measure, the more interesting the discovered association between A and B.

Interestingness of Association Rules

- An alternative interestingness measure is the lift of an association rule.
- If A and B are independent, then $P(A \cup B) = P(A) \cdot P(B)$, i.e.

$$\frac{P(A \cup B)}{P(A) \cdot P(B)} = 1$$

- We define the lift of a rule $A \Rightarrow B$ as follows:

$$lift(A \Rightarrow B) = \frac{P(A \cup B)}{P(A) \cdot P(B)}$$

- The lift can also be formulated as

$$lift(A \Rightarrow B) = \frac{P(A \cup B)/P(A)}{P(B)}$$

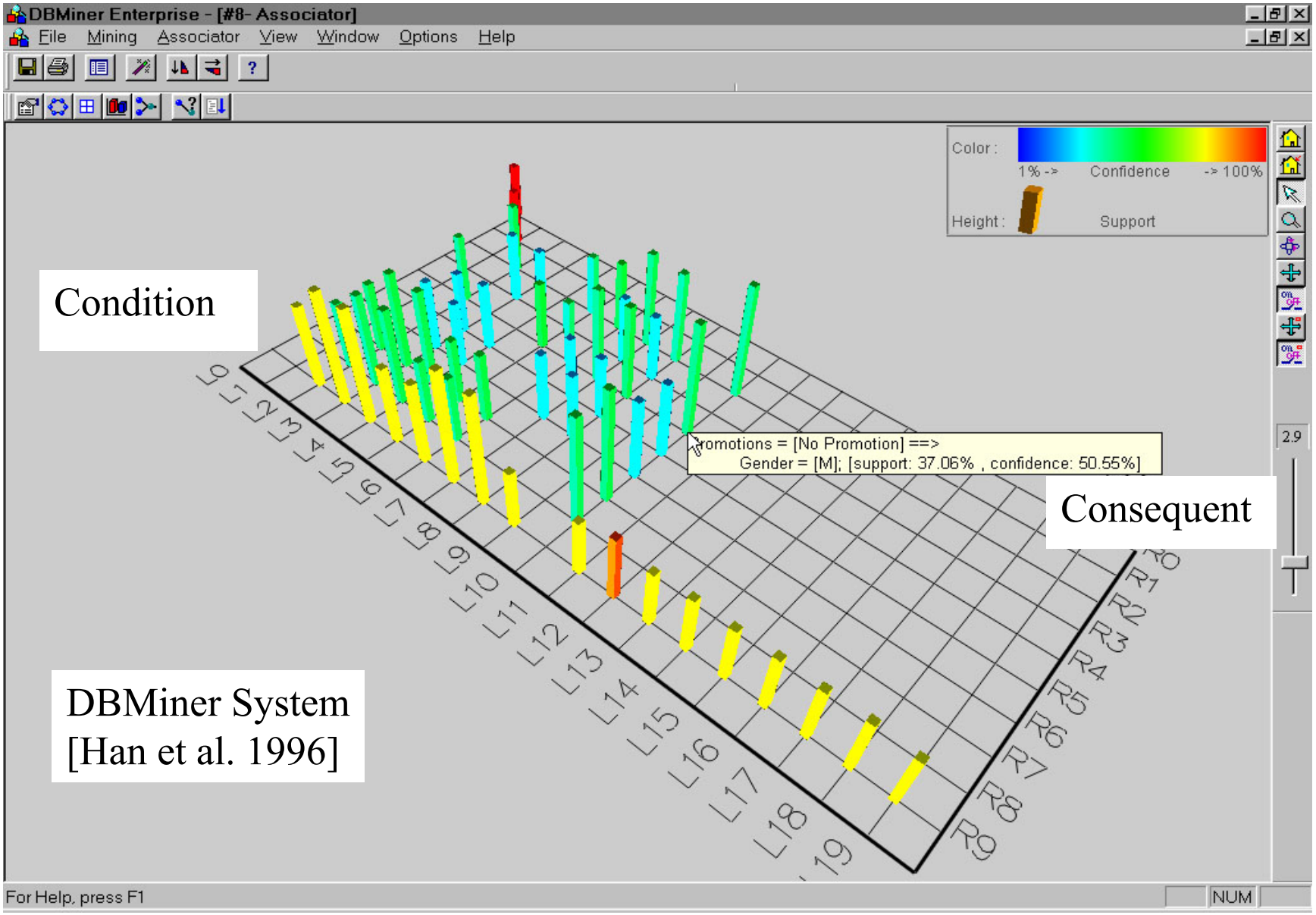
  i.e. as the ratio of the conditional probability P($B|A$) and the unconditional probability P(B).
- A lift >> 1 indicates that the discovered association between A and B is interesting.

Association Rules

	Body	Implies	Head	Supp (%)	Conf (%)	F	G	H	I
1	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00'	28.45	40.4				
2	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00'	20.46	29.05				
3	cost(x) = '0.00~1000.00'	==>	order_qty(x) = '0.00~100.00'	59.17	84.04				
4	cost(x) = '0.00~1000.00'	==>	revenue(x) = '1000.00~1500.00'	10.45	14.84				
5	cost(x) = '0.00~1000.00'	==>	region(x) = 'United States'	22.56	32.04				
6	cost(x) = '1000.00~2000.00'	==>	order_qty(x) = '0.00~100.00'	12.91	69.34				
7	order_qty(x) = '0.00~100.00'	==>	revenue(x) = '0.00~500.00'	28.45	34.54				
8	order_qty(x) = '0.00~100.00'	==>	cost(x) = '1000.00~2000.00'	12.91	15.67				
9	order_qty(x) = '0.00~100.00'	==>	region(x) = 'United States'	25.9	31.45				
10	order_qty(x) = '0.00~100.00'	==>	cost(x) = '0.00~1000.00'	59.17	71.86				
11	order_qty(x) = '0.00~100.00'	==>	product_line(x) = 'Tents'	13.52	16.42				
12	order_qty(x) = '0.00~100.00'	==>	revenue(x) = '500.00~1000.00'	19.67	23.88				
13	product_line(x) = 'Tents'	==>	order_qty(x) = '0.00~100.00'	13.52	98.72				
14	region(x) = 'United States'	==>	order_qty(x) = '0.00~100.00'	25.9	81.94				
15	region(x) = 'United States'	==>	cost(x) = '0.00~1000.00'	22.56	71.39				
16	revenue(x) = '0.00~500.00'	==>	cost(x) = '0.00~1000.00'	28.45	100				
17	revenue(x) = '0.00~500.00'	==>	order_qty(x) = '0.00~100.00'	28.45	100				
18	revenue(x) = '1000.00~1500.00'	==>	cost(x) = '0.00~1000.00'	10.45	96.75				
19	revenue(x) = '500.00~1000.00'	==>	cost(x) = '0.00~1000.00'	20.46	100				
20	revenue(x) = '500.00~1000.00'	==>	order_qty(x) = '0.00~100.00'	19.67	96.14				
21									
22									
23	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00' AND order_qty(x) = '0.00~100.00'	28.45	40.4				
24	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00' AND order_qty(x) = '0.00~100.00'	28.45	40.4				
25	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00' AND order_qty(x) = '0.00~100.00'	19.67	27.93				
26	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00' AND order_qty(x) = '0.00~100.00'	19.67	27.93				
27	cost(x) = '0.00~1000.00' AND order_qty(x) = '0.00~100.00'	==>	revenue(x) = '500.00~1000.00'	19.67	33.23				

Sheet1

Association Rules



The Apriori Algorithm

Approach [Agrawal & Srikant 1994]

- Determine first the frequent 1-itemsets, then the frequent 2-item sets, . . .
- To determine the frequent k+1-itemsets:

  consider only the k+1-itemsets for which all k-subsets are frequent
- Calculation of support:

  one database scan counting the support for all „relevant“ itemsets

The Apriori Algorithm

Algorithm Apriori

C_k: set of *candidate* itemsets of length k
F_k: set of all *frequent* itemsets of length k

```
Apriori(D, minsup)
k=1;
F_1 = {frequent 1-itemsets in D};
while F_k ≠ ∅ do
   Generate C_{k+!} by joining itemset-pairs in F_k;
   Prune itemsets from C_{k+!} that violate anti-monotonicity;
   Determine F_{k+1} by counting support of C_{k+!} in D and retaining
     itemsets from C_{k+!} with support at least minsup;
   k=k+1;
return ∪_k F_k;
```

The Apriori Algorithm

Candidate Generation

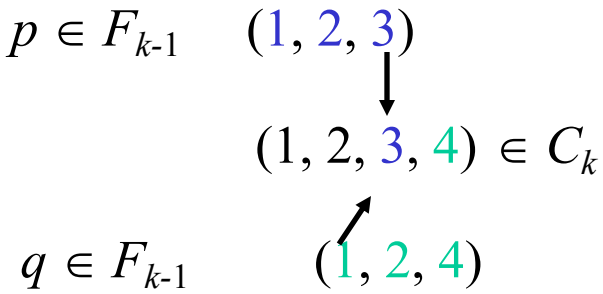
Requirements for set C_k of candidate itemsets

- Superset of F_k
- Significantly smaller than set of all k-subsets of I

Step 1: Join

Frequent k-1-item sets p and q

p and q are joined, if they agree in their first k–2 items



The Apriori Algorithm

Candidate Generation

Step 2: Pruning

Remove all elements from C_k having a $k-1$-subset not contained in F_{k-1}.

Example

$F_3 = \{(1\ 2\ 3), (1\ 2\ 4), (1\ 3\ 4), (1\ 3\ 5), (2\ 3\ 4)\}$

After join step: $C_4 = \{(1\ 2\ 3\ 4), (1\ 3\ 4\ 5)\}$

In pruning step: remove (1 3 4 5)

⇒ $C_4 = \{(1\ 2\ 3\ 4)\}$

The Apriori Algorithm

Example

minsup = 2

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

F_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

F_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D

F_3

itemset	sup
{2 3 5}	2

The Apriori Algorithm

Support Counting

```
for each candidate c ∈ C_k do c.count=0;
for each transaction T ∈ D do
   CT := subset(C_k, T); // all candidates from C_k, that are
                         // contained in transaction T;
   for each candidate c ∈ CT do c.count++;
F_k := {c ∈ C_k | (c.count / |D|) ≥ minsup};
```

- One scan over the database D
- For each transaction T in D, call

Subset(C_k, T)

all candidates from C_k, that are contained in transaction T

One Minute Survey

What is the (worst-case) runtime complexity of Apriori for an itemset *I* of size *m* (in Big O notation)?

Provide an example dataset and parameter minsupp for this worst case.

→ Discuss with your neighbor.

→ Share your ideas.

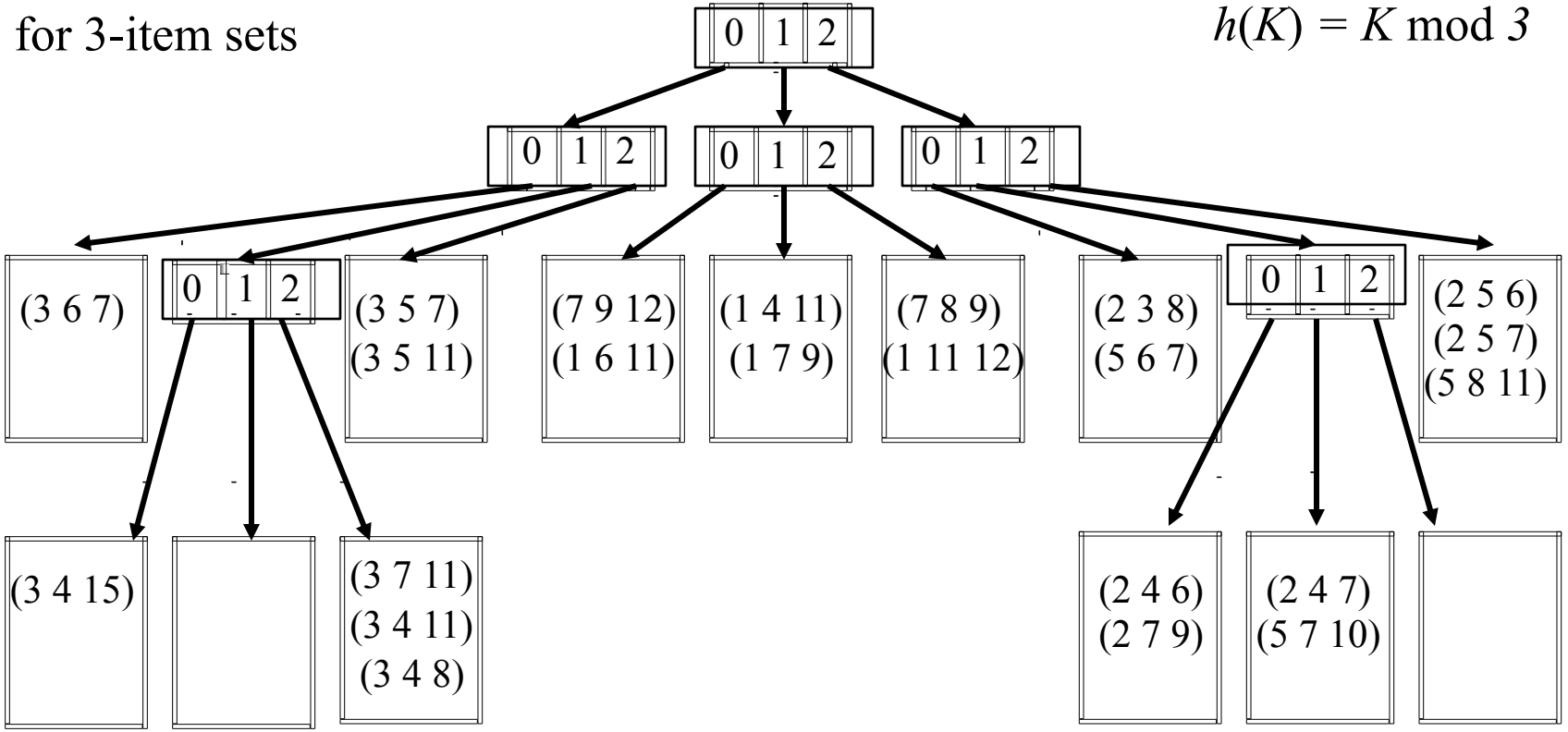
The Apriori Algorithm

Support Counting

- Problems
  - Very large number of candidate itemsets.
  - One transaction may contain many candidates.
- Hash tree as data structure for C_k
  - *Leaf node* records list of itemsets (with frequencies).
  - *Inner node* contains hash table (apply hash function to *d*-th elements)
    each hash bucket at level *d* references son node at level *d*+1 .
  - *Root* has level 1.

The Apriori Algorithm

Example



The Apriori Algorithm

Hash Tree

Finding an itemset

- Start from the root.
- At level *d*: apply hash function *h* to the *d*-th element of the itemset.

Inserting an item set

- Find the corresponding leaf node and insert new itemset.
- In case of overflow:
  - Convert leaf node into inner node and create all its son nodes (new leaves).
  - Distribute all entries over the new leaf nodes according to hash function *h*.

The Apriori Algorithm

Hash Tree

Find all candidates contained in $T = (t_1\ t_2 \ldots t_m)$

- At root

  Determine hash values $h(t_i)$ for each item t_i in T.

  Continue search in all corresponding son nodes.

- At inner node of level d

  Assumption: inner node has been reached by hashing t_i.

  Determine hash values and continue search for all items t_k in T with $k > i$.

- At leaf node

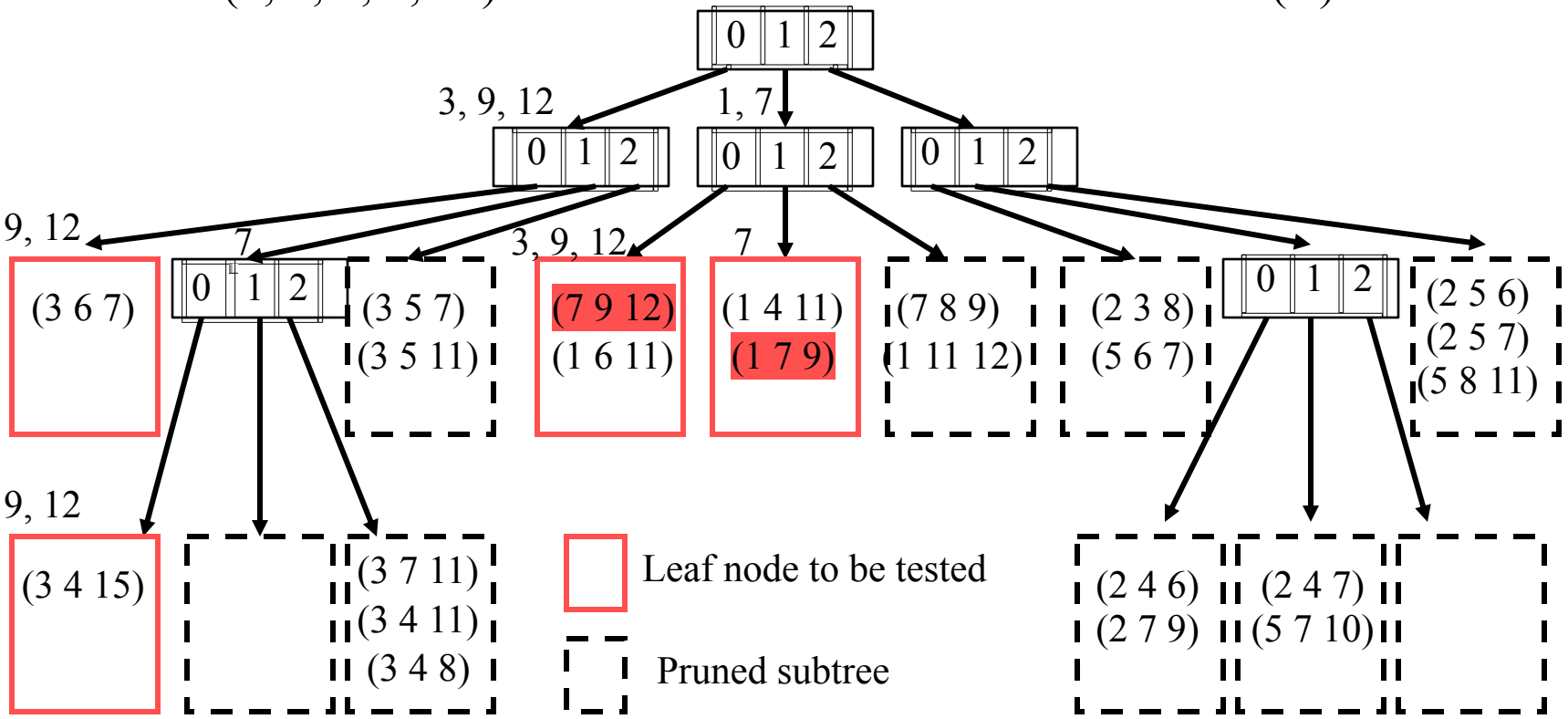
  For each itemset X in this node, test whether $X \subseteq T$.

The Apriori Algorithm

Example

Transaction (1, 3, 7, 9, 12)

$h(K) = K \bmod 3$



The Apriori Algorithm

Methods of Efficency Improvement

Support counting using a hash table [Park, Chen & Yu 1995]

- Hash table instead of hash tree, support counters for hash buckets.
- *k*-itemset with corresponding bucket counter < minsup cannot be frequent

  ⇒ more efficient access to candidates but inaccurate counts.

Reduction of transactions [Agrawal & Srikant 1994]

- Transactions that do not contain any frequent *k*-itemset are irrelevant.
- Remove such transactions for future phases

  ⇒ more efficient database scan, but additional writing of database.

The Apriori Algorithm

Methods of Efficency Improvement

Partitioning of the database [Savasere, Omiecinski & Navathe 1995]

- Itemset is only frequent if frequent in at least one partition.
- Form memory-resident partitions of the database.

⇒ more efficient on partitions, but expensive combination of intermediate results.

Sampling [Toivonen 1996]

- Apply algorithm to sample to find frequent itemsets.
- Count support of these frequent itemsets in the whole database.
- Determine further candidates and support counting on the whole database.

The Apriori Algorithm

Discussion

Pros

Cons

Enumeration-Tree Algorithms

Enumeration Tree

- Frequent itemsets are stored in a tree-like data structure, the enumeration tree, which provides an abstract, hierarchical representation of (part of) the lattice of itemsets.
- Items within a set are ordered lexicographically.

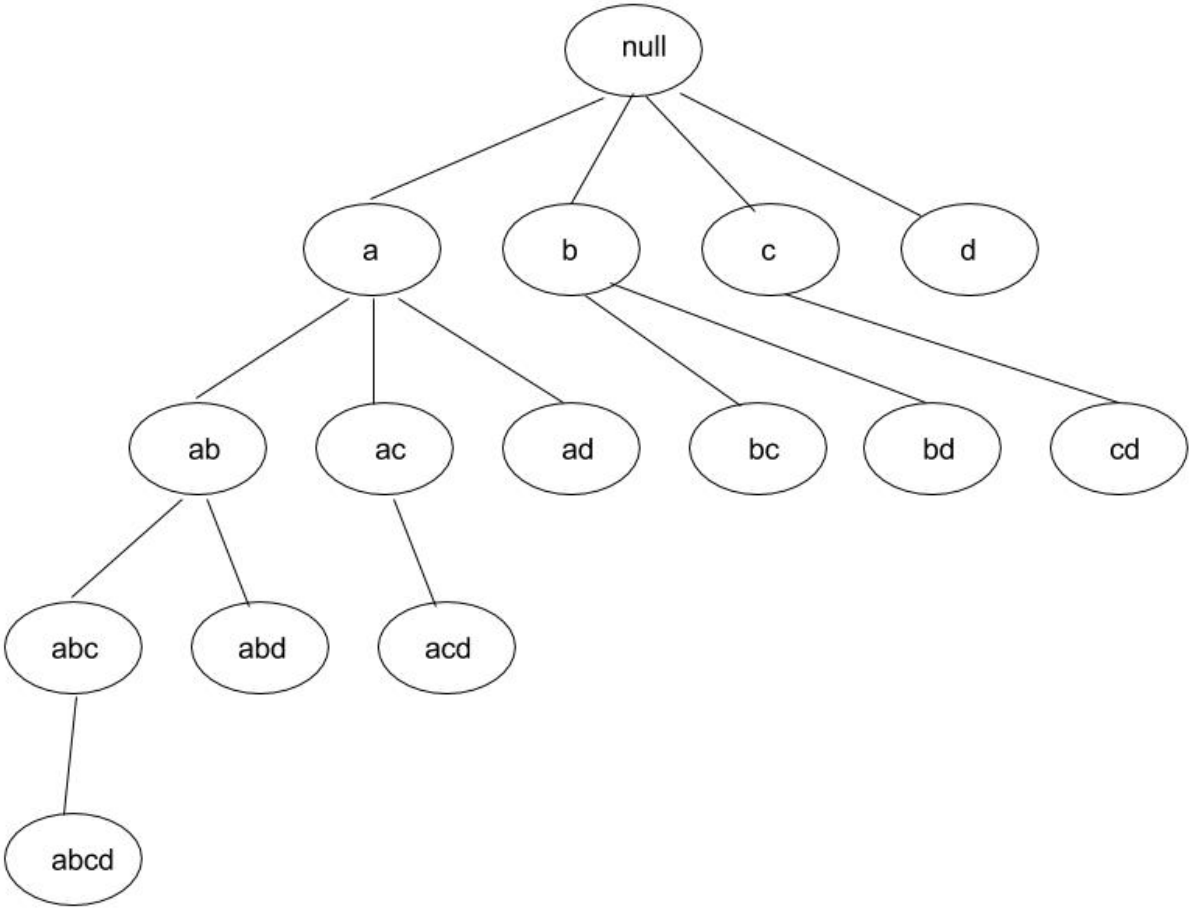
  → lexicographic tree
- Hierarchical structure supports systematic and non-redundant exploration of the lattice of itemsets.

Enumeration-Tree Algorithms

Enumeration Tree

- Nodes represent itemsets.
- Edges represent subset relationships.
- A child node $C = \{i_1,\ldots,i_k\}$ extends the parent node $P = \{i_1,\ldots,i_{k-1}\}$ by one item that is lexicographically larger than all items of the parent node.
- The root represents the empty itemset (null).

Enumeration-Tree Algorithms



Enumeration-Tree Algorithms

Algorithmic Scheme

```
Generic Enumeration Tree (D, minsup)
Initialize enumeration tree ET to Null node;
while any node in ET has not been examined do
   Select one or more unexamined nodes P;
   for each p in P do
     Generate candidate extensions C(p);
   count support in D for all n in any C(p);
   if support of n ≥ minsup then
     extend node p by node n
return ET;
```

Enumeration-Tree Algorithms

Example Algorithms

- Apriori

  candidate generation is level-wise (breadth-first),

  joining siblings,

  single database scan to count support of all candidates of a level.

- FP-growth

  candidate generation is depth-first,

  create projected database of transactions supporting an itemset,

  count support of candidate extensions only in projected database.

  ⇒ Minimize the number of candidate itemsets generated and counted, without missing any frequent itemsets.

Suffix-based Pattern Growth Methods

Recursive Suffix-based Pattern Growth

- In Apriori, have to count support from scratch at every level.
- In order not to waste the computational effort of counting, form projected database for a frequent itemset P:

  all transactions containing itemset P.
- If a transaction does not contain the itemset corresponding to an enumeration-tree node, then this transaction will not be relevant for counting at any descendent (superset itemset) of that node.

Suffix-based Pattern Growth Methods

Recursive Suffix-based Pattern Growth

- Count support of extensions of P only in projected database of P.
- Use absolute minsup, not relative minsup.
- Start with empty pattern (suffix) and complete database D, where D has been filtered to contain only frequent items.
- Recursive calls for all extensions and their projected databases.

Suffix-based Pattern Growth Methods

Algorithm Recursive Suffix Growth

D: transactions in terms of frequent 1-items , i.e. without infrequent items
P: current suffix itemset

reports all frequent itemsets with suffix P

```
Recursive Suffix Growth(D, minsup, P)
For each item i that appears at least once in D do
   report itemset P_i = {i} ∪ P as frequent;
   Form D_i with all transactions from D containing item i;
   Remove all items from D_i that are lexicographically ≥ i;
   Remove all infrequent items from D_i;
   if D_i ≠ ∅ then Recursive Suffix Growth(D_i, minsup, P_i);
```

Suffix-based Pattern Growth Methods

Example

minsup = 2

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Remove infrequent →

D

TID	Items
100	1 3
200	2 3 5
300	1 2 3 5
400	2 5

RSG = Recursive Suffix Growth

RSG(D,2,[])

Report [1]
D1={}

Report [2]
D2={}

Report [3]
D3={{1},{2},{1,2}}
RSG(D3,2,[3])

Report [5]
D5={{2,3},{2,3},{2}}
RSG(D5,2,[5])

Report [1,3]
D31={}

Report [2,3]
D32={}

Report [2,5]
D52={}

Report [3,5]
D53={{2},{2},{2}}
RSG(D53,2,[3,5])

Report [2,3,5]
D532={}

Suffix-based Pattern Growth Methods

FP-Tree [Han, Pei & Yin 2000]

- Space-efficient data structure for projected database.
- Trie structure represents conditional database by consolidating the prefixes.
- Path from the root to a leaf represents a transaction (or a set of identical transactions).
- Path from the root to internal node represents a prefix of a transaction (or a transaction).
- Each node has count (in the original database) of transactions that support that prefix (or transaction).

Suffix-based Pattern Growth Methods

FP-Tree

- Lexicographic ordering of items from most frequent to least frequent.
- Maximizes the effect of prefix-based compression

⇒ balances the size of the different conditional databases

•Recursive Suffix Growth with FP-Tree

Algorithm FP-growth [Han, Pei & Yin 2000]

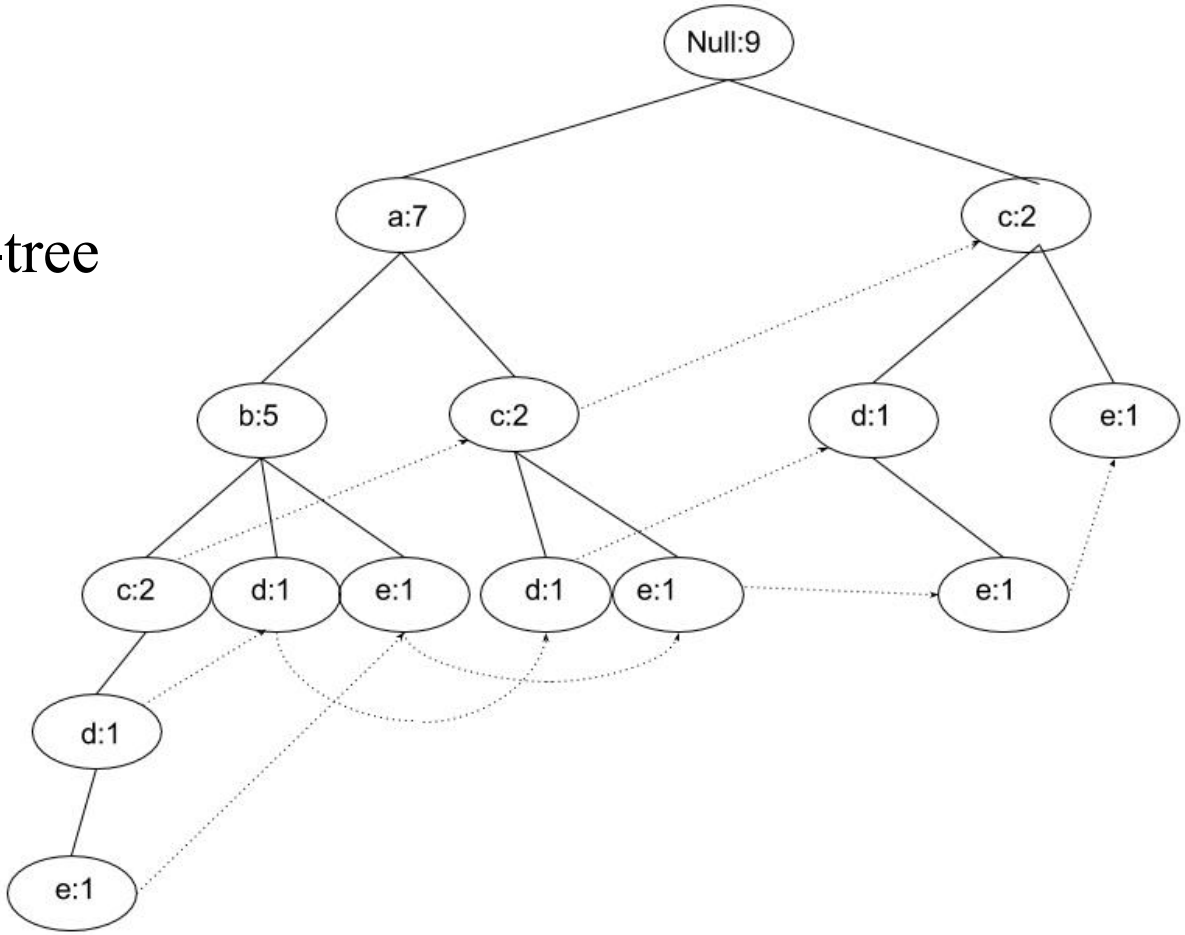
Suffix-based Pattern Growth Methods

Construction of FP-Tree

- Create an empty tree.
- Remove infrequent items from the transactions.
- Sort items within a transaction from most frequent to least frequent.
- Insert the modified transactions into the tree, one by one.
- When the prefix of the transaction overlaps with an existing path, increment the counts of that path by 1.
- For the non-overlapping part of the transaction, create new nodes with a count of 1. If applicable, create pointer to „next“ node with the same item.

Suffix-based Pattern Growth Methods

FP-tree



Suffix-based Pattern Growth Methods

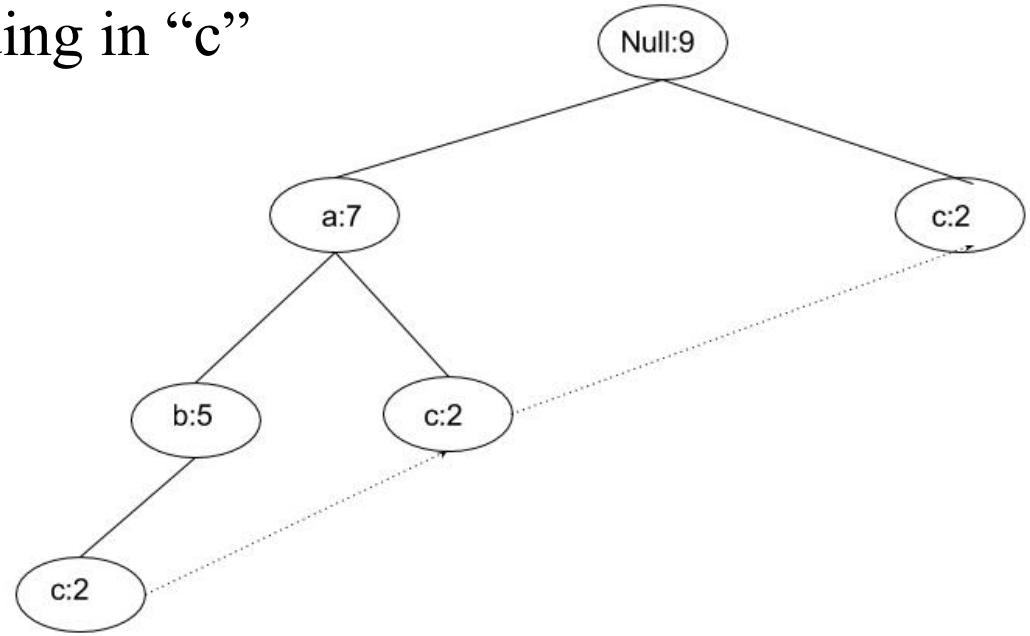
Extraction of conditional FP-Tree of item i

- Chase pointers for item *i* to extract the tree of its conditional prefix paths. Prune remaining branches.
- Adjust counts in the prefix paths to account for the pruned branches.
- Count frequency of each item by aggregating the counts of that item in the tree of prefix paths. Remove infrequent items. Item i is also removed.

⇒ conditional FP-tree may have to be re-created by successive insertion of prefix paths.

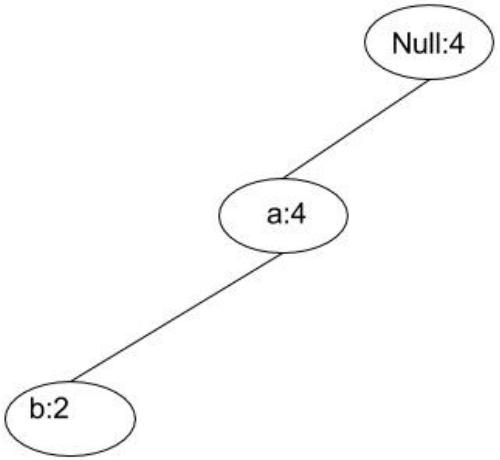
Suffix-based Pattern Growth Methods

Paths ending in “c”

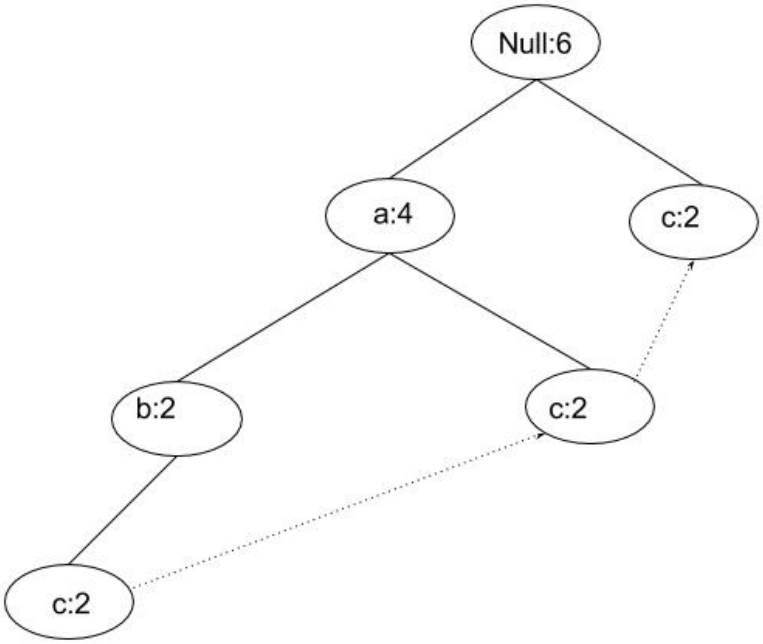


Suffix-based Pattern Growth Methods

Remove item “c”
and infrequent items



Readjust counts



Performance Comparison

Datasets [Zheng, Kohavi &Mason 2001]

Table 1 Dataset characteristics

	Transac-tions	Distinct Items	Maximum Transaction Size	Average Transaction Size
IBM-Artificial	100,000	870	29	10.1
BMS-POS	515,597	1,657	164	6.5
BMS-WebView-1	59,602	497	267	2.5
BMS-WebView-2	77,512	3,340	161	5.0

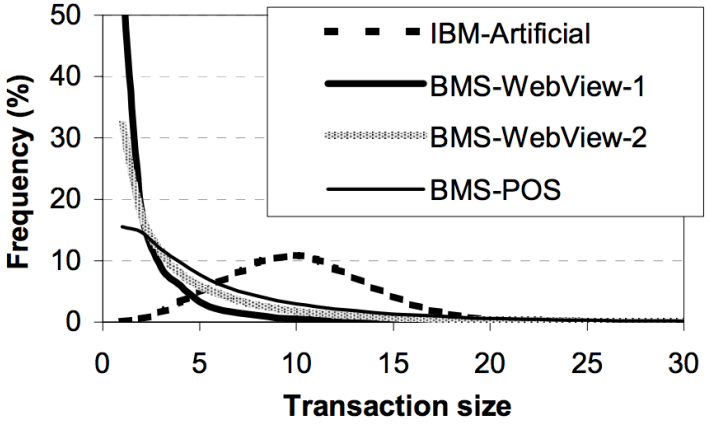


Figure 1. Dataset transaction size distribution.

Performance Comparison

On IBM Artificial

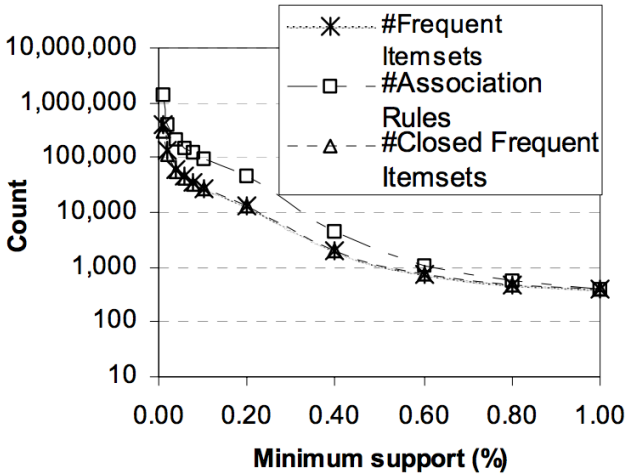


Figure 2. The number of frequent itemsets and association rules at different minimum support levels on IBM-Artificial.

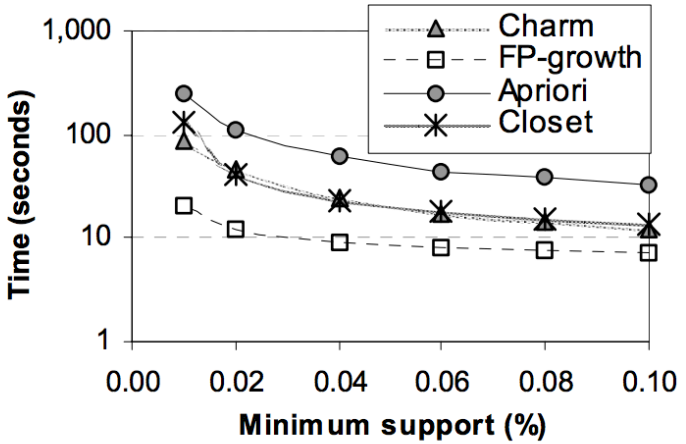


Figure 4. Running time of the algorithms for generating frequent itemsets for IBM-Artificial.

Performance Comparison

On BMS-POS

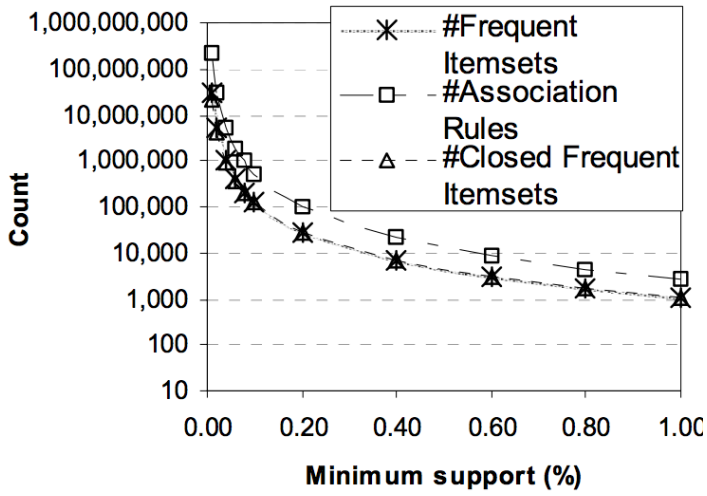


Figure 3. The number of frequent itemsets and association rules at different minimum support levels on BMS-POS. The

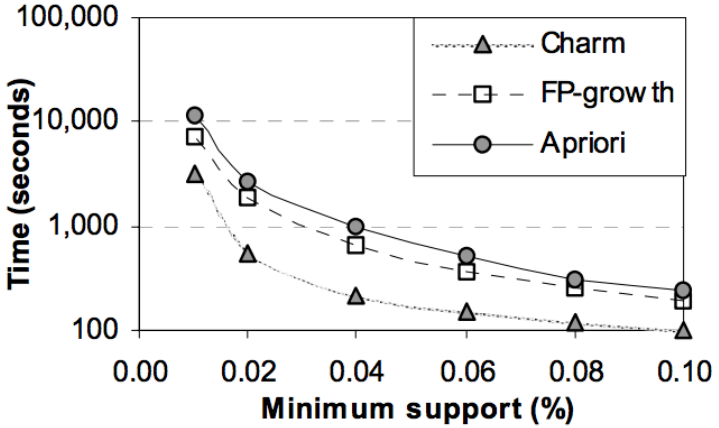


Figure 5. Running time of the algorithms for creating frequent itemsets on BMS-POS. Closet does not show up

Performance Comparison

Interpretation of Results

- Relative performance of the algorithms different on the artificial dataset and real datasets.
- On real dataset, the number of frequent itemsets and the runtime of all algorithms increases exponentially with decreasing min-support.
- The maximum number of association rules for manual inspection is ~ 1000, and ~ 1,000,000 for use by a recommendation algorithm.
- For such numbers of association rules, Apriori and FP-Growth show similar performance.

FP-Growth

Discussion

Pros

Cons

Pattern Summarization

Overview

- In many applications, there are (too) many frequent itemsets (patterns).
- Leads to inefficient mining.
- Makes it hard for user to analyze the patterns in a meaningful way.
- The vast majority of the generated patterns are typically redundant.

⇒ Summarize sets of patterns

Pattern Summarization

Overview

- Closed frequent itemsets
  lossless compression: can reconstruct all frequent itemsets and their support.
- Maximal frequent itemsets
  lossy compression: can reconstruct all frequent itemsets but not their support.
- Approximate representations

  lossy compression: can reconstruct neither all frequent itemsets nor their support.

Pattern Summarization

Closed Frequent Itemsets [Pei, Han & Mao 2000]

- An itemset X is closed, if none of its supersets has the same support as X.
- S(X): set of subsets of X, which have the same support as X.
- Every itemset in S(X) is supported by the same set of transactions, and therefore, it suffices to keep X as the single representative itemset.
- Goal is to produce all closed, frequent itemsets.

Two approaches:

- post-process the results of frequent itemset mining,
- directly find the closed frequent patterns during the process of frequent pattern discovery.

Pattern Summarization

Mining Closed Frequent Itemsets

- Partition the set F of frequent itemsets into equi-support groups.
- The maximal itemsets from each equi-support group are closed frequent itemsets and are to be determined.
- For that goal, the patterns in an equi-support group are processed in decreasing order of length and either ruled in or ruled out, depending on whether or not they are closed.
- This order ensures that closed patterns are encountered before their redundant subsets.

Pattern Summarization

Mining Closed Frequent Itemsets

- Initially, all patterns are unmarked.
- When an unmarked pattern $X \in F$ is processed, it is added to the frequent closed set CF .
- The proper subsets of X with the same support are marked. To achieve this goal, the subset of the itemset lattice representing F can be traversed in depth-first or breadth-first order starting at X , and exploring subsets of X . Itemsets that are subsets of X are marked when they have the same support as X .
- The traversal process backtracks when an itemset is reached with strictly larger support, or the itemset has already been marked.
- After the traversal is complete, the next unmarked node is selected for further exploration and added to CF .

Pattern Summarization

Example

F_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

F_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

F_3

itemset	sup
{2 3 5}	2

Group Sup = 2: {2 3 5} {1 3} {2 3} {3 5} {1}
Group Sup = 3: {2 5} {2} {3} {5}
Return {2 3 5}, mark {2 3} and {3 5}
Return {1 3}, mark {1}
Return {2 5}, mark {2} and {5}
Return {3}

Pattern Summarization

Maximal Frequent Itemsets [Gouda & Zaki 2001]

- A frequent itemset X is maximal, if none of its supersets is frequent.
- Maximal frequent itemsets are closed.
- The number of maximal frequent itemsets may be orders of magnitude smaller than the number of all frequent itemsets.
- Naïve algorithm: determine all frequent itemsets, and post-process them to remove all non-maximal ones.
- Direct algorithm: most tree-enumeration algorithms can be modified with the concept of look-aheads to directly determine only (mostly) maximal patterns.

Pattern Summarization

Naïve Algorithm

- Apply any algorithm to determine all frequent itemsets.
- Examine itemsets in decreasing order of length, removing all proper subsets of a given itemset.
- Continue until all itemsets have either been examined or have been removed.
- The itemsets that have not been removed at termination are the maximal ones.

Pattern Summarization

Example

F_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

F_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

F_3

itemset	sup
{2 3 5}	2

{2 3 5} {1 3} {2 3} {2 5} {3 5} {1} {2} {3} {5}

Remove proper subsets of {2 3 5}

{2 3 5} {1 3} {1}

Remove proper subsets of {1 3}

{2 3 5} {1 3}

Pattern Summarization

Direct Algorithm

- Let C(P) be the set of candidate extensions of node (frequent itemset) P.
- Before support counting, test whether P ∪C (P) is a subset of a frequent pattern that has already been found.
- If yes, P ∪ C(P) is a non-maximal frequent pattern, and the entire subtree (of the enumeration tree) rooted at P can be pruned.
- If P cannot be pruned, the supports of its candidate extensions need to be determined. During this support counting, the support of P ∪ C(P) is counted along with the individual item extensions of P.
- If P ∪ C(P) is frequent, then it eliminates any further work of counting the support of nodes in the subtree rooted at node P .

Pattern Summarization

Direct Algorithm

- This subset-based pruning approach is particularly effective with depth-first methods.
- Maximal patterns are found much earlier with a depth-first strategy than with a breadth-first strategy.
- For a maximal pattern of length k, the depth-first approach discovers it after exploring only (k − 1) of its prefixes, rather than the 2^k subsets. This maximal pattern then becomes available for subset-based pruning.
- The pruning approach provides a smaller set of patterns that includes all maximal patterns but may also include some non-maximal patterns despite the pruning.

Pattern Summarization

Discussion

Pros

Cons

One Minute Survey

Do you see any connection between association rule mining and classification?

→ Discuss with your neighbor.

→ Share your ideas.

References

[Agrawal & Srikant 1994]
Rakesh Agrawal, Ramakrishnan Srikant: Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499
[Gouda & Zaki 2001]
Karam Gouda, Mohammed Javeed Zaki: Efficiently Mining Maximal Frequent Itemsets. ICDM 2001: 163-170
[Han, Pei & Yin 2000]
Jiawei Han, Jian Pei, Yiwen Yin: Mining Frequent Patterns without Candidate Generation. SIGMOD 2000: 1-12
[Park, Chen & Yu 1995]
Jong Soo Park, Ming-Syan Chen, Philip S. Yu: An Effective Hash Based Algorithm for Mining Association Rules. SIGMOD 1995:175-186
[Pei, Han & Mao 2000]
Jian Pei, Jiawei Han, Runying Mao: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000: 21-30

References

[Savasere, Omiecinski & Navathe 1995]
Ashok Savasere, Edward Omiecinski, Shamkant B. Navathe: An Efficient Algorithm for Mining Association Rules in Large Databases. VLDB 1995: 432-444
[Toivonen 1996]
Hannu Toivonen: Sampling Large Databases for Association Rules. VLDB 1996: 134-145
[Zheng, Kohavi & Mason 2001]
Zijian Zheng, Ron Kohavi, Llew Mason: Real world performance of association rule algorithms. KDD 2001: 401-406