Knowledge Discovery in Databases

WS 2017/18

Kapitel 3: Frequent Itemset Mining

Vorlesung: Prof. Dr. Peer Kröger

Übungen: Anna Beer, Florian Richter

1) Introduction
   – Transaction databases, market basket data analysis
2) Mining Frequent Itemsets
   – Apriori algorithm, hash trees, FP-tree
3) Simple Association Rules
   – Basic notions, rule generation, interestingness measures
4) Further Topics
5) Extensions and Summary

Was ist Frequent Itemset Mining?

Frequent Itemset Mining:

Finde häufige Muster, Assoziationen, Korrelationen, … zwischen Mengen von Items oder Objekten in einer Datenbank.

- Gegeben:
  - Eine Menge von Items $I = \{i_1, i_2, \dots, i_m\}$
  - Eine Datenbank D von Transaktionen $T \subseteq I$ (= Menge von Items, Itemsets)

- Task 1 (Freuqent Itemset Mining): Finde alle Teilmengen von Items (Itemsets), die zusammen in vielen Transaktionen vorkommen.
  - Z.B.: 85% aller Transaktionen enthalten das Itemset {milk, bread, butter}

=> Zählproblem; was kommt so häufig zusammen vor, dass es ein interessantes Muster ist

Was ist Frequent Itemset Mining?

- Task 2 (Association Rule Mining): Finde Regeln, die das Vorkommen eines Itemsets mit dem Vorkommen eines anderen Itemsets korreliert.
  - Z.B.: 98% der Kunden, die Räder und Autozubehör kaufen, lassen auch den Service machen

- Anwendungen:
  - Basket data analysis
  - Cross-marketing
  - Catalog design
  - Loss-leader analysis
  - Clustering
  - Classification
  - Recommendation systems

  etc.

Beispiel: Basket Data Analysis

- Transaktionsdatenbank

  D= {{butter, bread, milk, sugar};
  {butter, flour, milk, sugar};
  {butter, eggs, milk, salt};
  {eggs};
  {butter, flour, milk, salt, sugar}}

- Fragestellung:
  - Welche Items werden häufig miteinander gekauft?

items	frequency
{butter}	4
{milk}	4
{butter, milk}	4
{sugar}	3
{butter, sugar}	3
{milk, sugar}	3
{butter, milk, sugar}	3
{eggs}	2
…	

- Anwendung
  - Ladenlayout-Optmierung
  - Cross marketing
  - Focused attached mailings / add-on sales
  - ** ⇒ Maintenance Agreement*
    (What the store should do to boost Maintenance Agreement sales)
  - *Home Electronics ⇒ ** (What other products should the store stock up?)

Beispiel: Basket Data Analysis

- Und das kommt dann dabei raus …

1) Introduction
   - Transaction databases, market basket data analysis
2) Mining Frequent Itemsets
   - Apriori algorithm, hash trees, FP-tree
3) Simple Association Rules
   - Basic notions, rule generation, interestingness measures
4) Further Topics
   - Hierarchical Association Rules
     - Motivation, notions, algorithms, interestingness
   - Quantitative Association Rules
     - Motivation, basic idea, partitioning numerical attributes, adaptation of apriori algorithm, interestingness
5) Extensions and Summary

Mining Frequent Itemsets: Basics

- *Items* $I = \{i_1, i_2, \dots, i_m\}$: a set of literals (denoting items)
- *Itemset* X*:* Set of items $X \subseteq I$
- *Database* D: Set of *transactions* T, each being a set of items $T \subseteq I$
- Transaction T *contains* an itemset X: $X \subseteq T$
- The items in transactions and itemsets are sorted lexicographically:
  - itemset $X = (x_1, x_2, \dots, x_k)$, where $x_1 \leq x_2 \leq \dots \leq x_k$
- *Length* of an itemset: number of elements in the itemset
- *k-itemset:* itemset of length k
- The *support* of an itemset X is defined as: $support(X) = |\{T \in D | X \subseteq T\}|$
- *Frequent itemset: an itemset* X is called frequent for database D iff it is contained in more than $minSup$ many transactions: $support(X) \geq minSup$

- Goal 1: Given a database D and a threshold $minSup$, find all frequent itemsets $X \in Pot(I)$.

Mining Frequent Itemsets: Basic Idea

- Naïve Algorithm
  - count the frequency of all possible subsets of I in the database

  → *too expensive* since there are 2^m such itemsets for $|I| = m$ items

  cardinality of power set

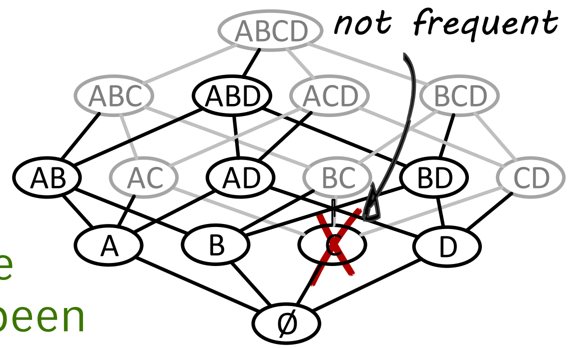
- The *Apriori* principle (anti-monotonicity):

  Any non-empty subset of a frequent itemset is frequent, too!

  $\mathrm{A} \subseteq \mathrm{I}$ with $\mathrm{support(A)} \geq \mathrm{minSup} \ \Rightarrow \forall \mathrm{A}' \subset \mathrm{A} \wedge \mathrm{A}' \neq \emptyset: \mathrm{support(A')} \geq \mathrm{minSup}$

  Any superset of a non-frequent itemset is non-frequent, too!

  $\mathrm{A} \subseteq \mathrm{I}$ with $\mathrm{support(A)} < \mathrm{minSup} \ \Rightarrow \forall \mathrm{A}' \supset \mathrm{A}: \mathrm{support(A')} < \mathrm{minSup}$



- Method based on the Apriori principle
  - First count the 1-itemsets, then the 2-itemsets, then the 3-itemsets, and so on
  - When counting (k+1)-itemsets, only consider those (k+1)-itemsets where all subsets of length k have been determined as frequent in the previous step

The Apriori Algorithm

variable C_k: candidate itemsets of size k
variable L_k: frequent itemsets of size k

L_1 = {frequent items}
for (k = 1; L_k !=∅; k++) **do begin**

produce candidates

// JOIN STEP: join L_k with itself to produce C_{k+1}
// PRUNE STEP: discard (k+1)-itemsets from C_{k+1} that contain non-frequent k-itemsets as subsets
C_{k+1} = candidates generated from L_k

prove candidates

for each transaction t in database do
Increment the count of all candidates in C_{k+1} that are contained in t
L_{k+1} = candidates in C_{k+1} with min_support

return $\cup_k L_k$

Generating Candidates (Join Step)

- Requirements for set of all candidate $(k+1)$-itemsets C_{k+1}
  - *Completeness*:
    Must contain all frequent $(k+1)$-itemsets (superset property $C_{k+1} \supseteq L_{k+1}$)
  - *Selectiveness*:
    Significantly smaller than the set of all $(k+1)$-subsets
  - Suppose the items are sorted by any order (e.g., lexicograph.)
- Step 1: Joining ($C_{k+1} = L_k \bowtie L_k$)
  - Consider frequent k-itemsets p and q
  - p and q are joined if they share the same first $k-1$ items

insert into C_{k+1}
select $p.i_1, p.i_2, \ldots, p.i_{k-1}, p.i_k, q.i_k$
from $L_k : p, L_k : q$
where $p.i_1 = q.i_1, \ldots, p.i_{k-1} = q.i_{k-1}, p.i_k < q.i_k$

$p \in L_{k=3}$ (A, C, F)

(A, C, F, G) $\in C_{k+1=4}$

$q \in L_{k=3}$ (A, C, G)

Generating Candidates (Prune Step)

- Step 2: Pruning ($L_{k+1} = \{X \in C_{k+1} | support(X) \geq minSup\}$)
  - Naïve: Check support of every itemset in C_{k+1} ← inefficient for huge C_{k+1}
  - Instead, apply Apriori principle first: Remove candidate *(k+1)* -itemsets which contain a non-frequent *k*-subset *s*, i.e., $s \notin L_k$

    forall *itemsets c* ***in*** C_{k+1} **do**
        forall *k-subsets s* ***of*** *c* **do**
            if *(s is not in* L_k*)* **then delete** *c* **from** C_{k+1}

- Example 1
  - L_3 = {(ACF), (ACG), (AFG), (AFH), (CFG)}
  - Candidates after the join step: {(ACFG), (AFGH)}
  - In the pruning step: delete (AFGH) because (FGH) $\notin L_3$, i.e., (FGH) is not a frequent 3-itemset; also (AGH) $\notin L_3$

  → C_4 = {(ACFG)} → check the support to generate L_4

Apriori Algorithm – Full Example

minSup=0.5

database D

TID	items
100	1 3 4 6
200	2 3 5
300	1 2 3 5
400	1 5 6

scan D

C_1

itemset	count
{1}	3
{2}	2
{3}	3
{4}	1
{5}	3
{6}	2

L_1

itemset	count
{1}	3
{2}	2
{3}	3
{5}	3
{6}	2

$L_1 \bowtie L_1$

C_2

itemset
{1 2}
{1 3}
{1 5}
{1 6}
{2 3}
{2 5}
{2 6}
{3 5}
{3 6}
{5 6}

prune C_1

C_2

itemset
{1 2}
{1 3}
{1 5}
{1 6}
{2 3}
{2 5}
{2 6}
{3 5}
{3 6}
{5 6}

scan D

C_2

itemset	count
{1 2}	1
{1 3}	2
{1 5}	2
{1 6}	2
{2 3}	2
{2 5}	2
{2 6}	0
{3 5}	2
{3 6}	1
{5 6}	1

L_2

itemset	count
{1 3}	2
{1 5}	2
{1 6}	2
{2 3}	2
{2 5}	2
{3 5}	2

$L_2 \bowtie L_2$

C_3

itemset
{1 3 5}
{1 3 6}
{1 5 6}
{2 3 5}

prune C_2

C_3

itemset
{1 3 5}
{1 3 6} ✗
{1 5 6} ✗
{2 3 5}

scan D

C_3

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

L_3

itemset	count
{2 3 5}	2

$L_3 \bowtie L_3$

C_4 is empty

Performance?

- First obvious problem: the check if a candidate from C_{k+1} is frequent
- Why? This is simple counting!?!
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Solutuion: Hash-Tree
  - Candidate itemsets and their support are stored in a hash-tree that efficiently supports
    - Insertion of new itemsets
    - Search for itemsets (and their support)
  - Sketch of the data structure
    - Leaf nodes of hash-tree contain lists of itemsets and their support (i.e., counts)
    - Interior nodes contain hash tables
    - Subset function finds all the candidates contained in a transaction

Performance?

- The core of the Apriori algorithm:
  - Use frequent ($k-1$)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - 10^4 frequent 1-itemsets will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs n or $n+1$ scans, n is the length of the longest pattern

→ Is it possible to mine the complete set of frequent itemsets without candidate generation?

Mining Frequent Patterns Without Candidate Generation

- Compress a large database into a compact, *Frequent-Pattern tree* (*FP-tree*) structure
  - highly condensed, but complete for frequent pattern mining
  - avoid costly database scans

- Develop an efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: sub-database test only!

- Idea:
  - Compress database into FP-tree, retaining the itemset association information
  - Divide the compressed database into conditional databases, each associated with one frequent item and mine each such database separately.

Construct FP-tree from a Transaction DB

Steps for compressing the database into a FP-tree:

1. Scan DB once, find frequent 1-itemsets (single items)
2. Order frequent items in frequency descending order

TID	items bought
100	{*f, a, c, d, g, i, m, p*}
200	{*a, b, c, f, l, m, o*}
300	{*b, f, h, j, o*}
400	{*b, c, k, s, p*}
500	{*a, f, c, e, l, p, m, n*}

1&2

minSup=0.5

header table:

item	frequency
f	4
c	4
a	3
b	3
m	3
p	3

sort items in the order of descending support

DATABASE SYSTEMS GROUP

Construct FP-tree from a Transaction DB

LMU

Steps for compressing the database into a FP-tree:

1. Scan DB once, find frequent 1-itemsets (single items)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree starting with most frequent item per transaction

TID	items bought	(ordered) frequent items
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

for each transaction only keep its frequent items sorted in descending order of their frequencies

1&2

header table:

item	frequency
f	4
c	4
a	3
b	3
m	3
p	3

3a

for each transaction build a path in the FP-tree:

- If a path with common prefix exists: increment frequency of nodes on this path and append suffix
- Otherwise: create a new branch

Construct FP-tree from a Transaction DB

Steps for compressing the database into a FP-tree:

1. Scan DB once, find frequent 1-itemsets (single items)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree starting with most frequent item per transaction

TID	items bought	(ordered) frequent items
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

1&2

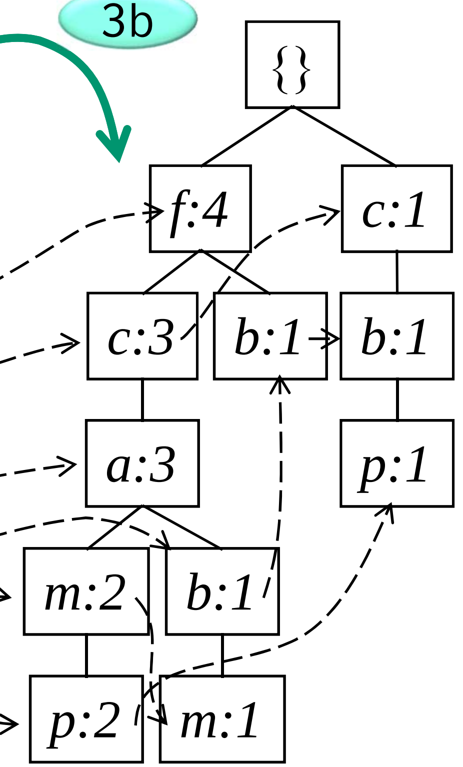
3a

header table:

item	frequency	head
f	4	•
c	4	•
a	3	•
b	3	•
m	3	•
p	3	•

header table references the occurrences of the frequent items in the FP-tree

3b



Benefits of the FP-tree Structure

- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining
- Compactness
  - reduce irrelevant information—infrequent items are gone
  - frequency descending ordering: more frequent items are more likely to be shared
  - never be larger than the original database (if not count node-links and counts)
  - Experiments demonstrate compression ratios over 100

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each item, construct its conditional pattern-base (*prefix paths*), and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree …
  - …until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

Major Steps to Mine FP-tree

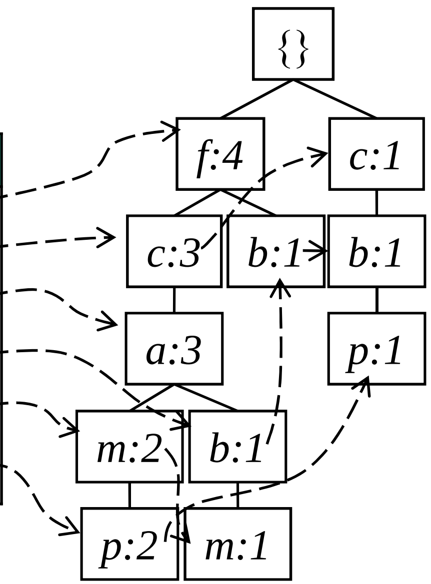
1) Construct conditional pattern base for each node in the FP-tree
2) Construct conditional FP-tree from each conditional pattern-base
3) Recursively mine conditional FP-trees and grow frequent patterns obtained so far
   - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Major Steps to Mine FP-tree: Conditional Pattern Base

1) Construct conditional pattern base for each node in the FP-tree
   - Starting at the frequent header table in the FP-tree
   - Traverse FP-tree by following the link of each frequent item (dashed lines)
   - Accumulate all of transformed prefix paths of that item to form a conditional pattern base
     - For each item its prefixes are regarded as condition for it being a suffix. These prefixes form the conditional pattern base. The frequency of the prefixes can be read in the node of the item.

header table:

item	frequency	head
f	4	•
c	4	•
a	3	•
b	3	•
m	3	•
p	3	•



conditional pattern base:

item	cond. pattern base
f	*{}*
c	*f:3, {}*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

Properties of FP-tree for Conditional Pattern Bases

- Node-link property
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header
- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P needs to be accumulated, and its frequency count should carry the same count as node a_i.

Major Steps to Mine FP-tree: Conditional FP-tree

1) Construct conditional pattern base for each node in the FP-tree
2) Construct conditional FP-tree from each conditional pattern-base
   - The prefix paths of a suffix represent the conditional basis.
     → They can be regarded as transactions of a database.
   - Those prefix paths whose support ≥ minSup, induce a conditional FP-tree
   - For each pattern-base
     - Accumulate the count for each item in the base
     - Construct the FP-tree for the frequent items of the pattern base

conditional pattern base:

item	cond. pattern base
f	*{}*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

e.g. item m

item	frequency
f	3
c	3
a	3
b	~~1~~ X

m-conditional FP-tree

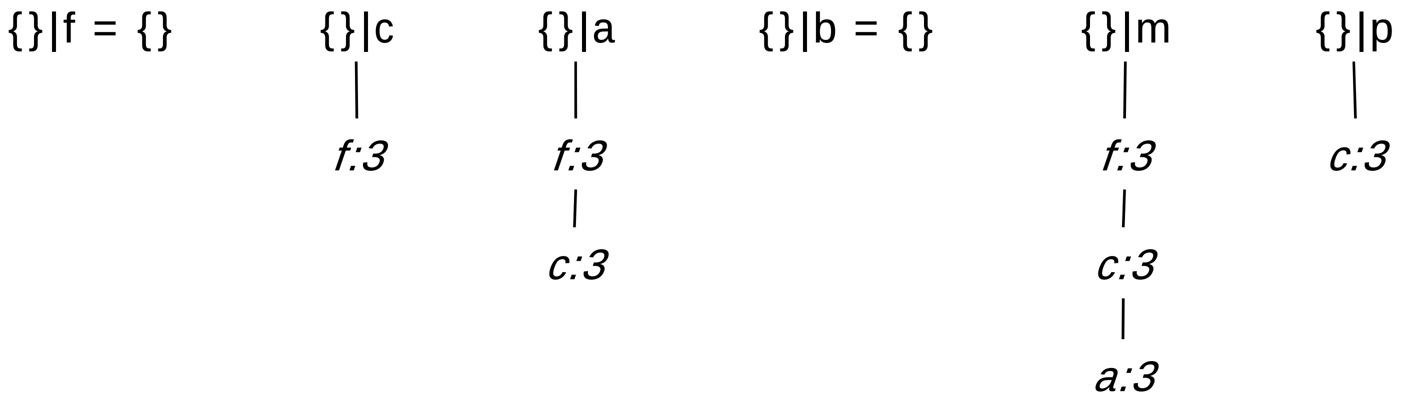
{}|m
|
f:3
|
c:3
|
a:3

Major Steps to Mine FP-tree: Conditional FP-tree

1) Construct conditional pattern base for each node in the FP-tree
2) Construct conditional FP-tree from each conditional pattern-base

conditional pattern base:

item	cond. pattern base
f	*{}*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*



Major Steps to Mine FP-tree

1) Construct conditional pattern base for each node in the FP-tree
2) Construct conditional FP-tree from each conditional pattern-base
3) Recursively mine conditional FP-trees and grow frequent patterns obtained so far
   - If the conditional FP-tree contains a single path, simply enumerate all the patterns (enumerate all combinations of sub-paths)

example:
m-conditional FP-tree

{}|m
|
f:3
|
c:3
|
a:3

just a single path →

All frequent patterns concerning *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

FP-tree: Full Example

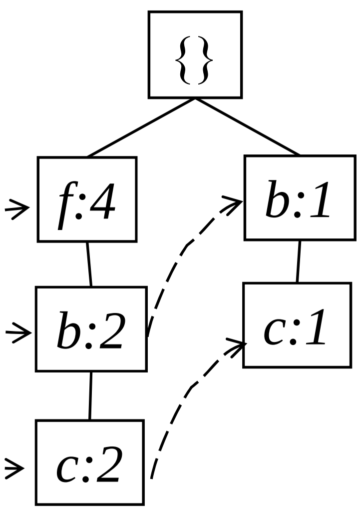
database:

TID	items bought	(ordered) frequent items
100	{*b*, *c*, *f*}	{*f*, *b*, *c*}
200	{*a*, *b*, *c*}	{*b*, *c*}
300	{*d*, *f*}	{*f*}
400	{*b*, *c*, *e*, *f*}	{*f*, *b*, *c*}
500	{*f*, *g*}	{*f*}

minSup=0.4

header table:

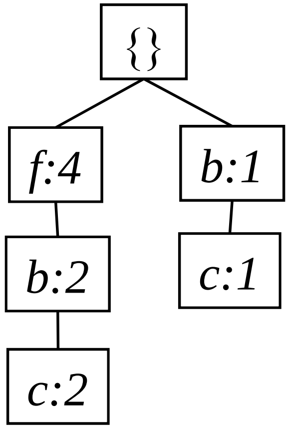
item	frequency	head
f	4	•
b	3	•
c	3	•



conditional pattern base:

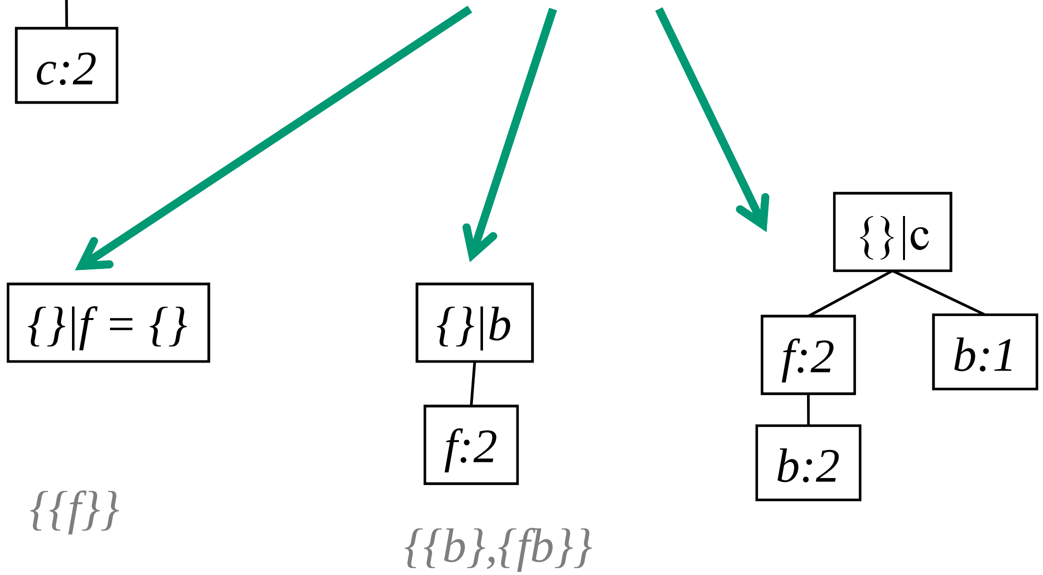
item	cond. pattern base
f	{}
b	*f*:2, {}
c	*fb*:2, *b*:1

FP-tree: Full Example



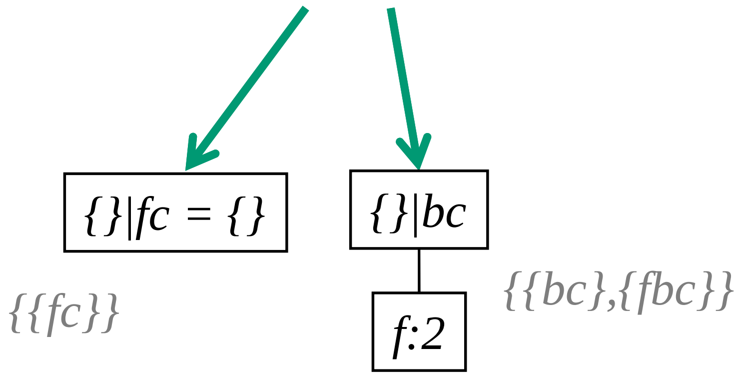
conditional pattern base 1:

item	cond. pattern base
f	*{}*
b	*f:2*
c	*fb:2, b:1*



conditional pattern base 2:

item	cond. pattern base
b	*f:2*
f	*{}*



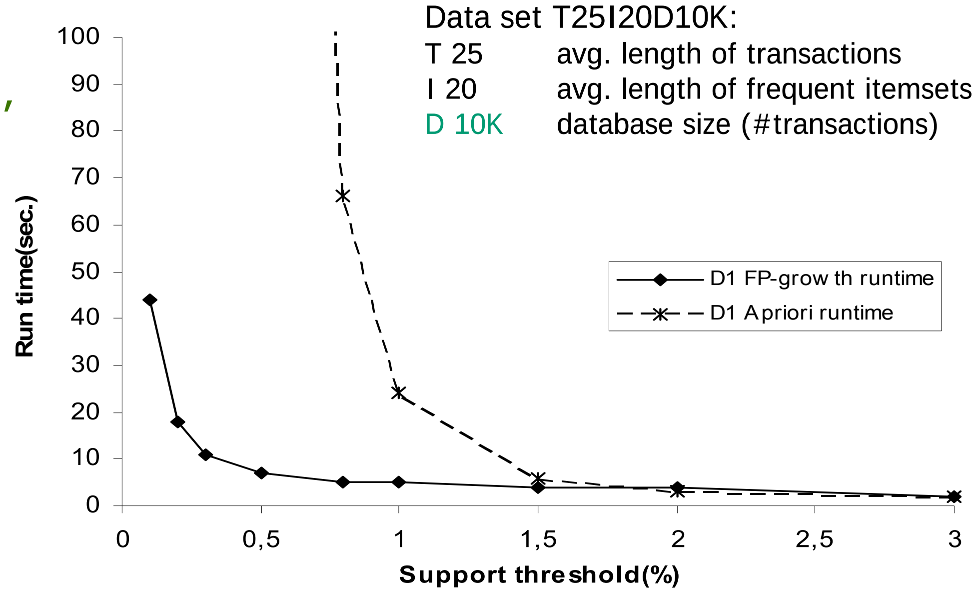
Principles of Frequent Pattern Growth

- Pattern growth property
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.

- "*abcdef*" is a frequent pattern, if and only if
  - "*abcde*" is a frequent pattern, and
  - "*f*" is frequent in the set of transactions containing "*abcde*"

Why Is Frequent Pattern Growth Fast?

- Performance study in [Han, Pei&Yin '00] shows
  - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection



- Reasoning
  - No candidate generation, no candidate test
    - Apriori algorithm has to proceed breadth-first
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

Maximal or Closed Frequent Itemsets

- Big challenge: database contains potentially a huge number of frequent itemsets (especially if minSup is set too low).
  - A frequent itemset of length 100 contains 2^{100}-1 many frequent subsets
- *Closed frequent itemset*:
  An itemset X is *closed* in a data set D if there exists no proper super-itemset Y such that $support(X) = support(Y)$ in D.
  - The set of closed frequent itemsets contains complete information regarding its corresponding frequent itemsets.
- *Maximal frequent itemset*:
  An itemset X is *maximal* in a data set D if there exists no proper super-itemset Y such that $support(Y) \geq minSup$ in D.
  - The set of maximal itemsets does not contain the complete support information
  - More compact representation

1) Introduction
   – Transaction databases, market basket data analysis
2) Mining Frequent Itemsets
   – Apriori algorithm, hash trees, FP-tree
3) Simple Association Rules
   – Basic notions, rule generation, interestingness measures
4) Further Topics
   – Hierarchical Association Rules
      • Motivation, notions, algorithms, interestingness
   – Quantitative Association Rules
      • Motivation, basic idea, partitioning numerical attributes, adaptation of apriori algorithm, interestingness
5) Extensions and Summary

Simple Association Rules: Introduction

- Transaction database:

  D= {{butter, bread, milk, sugar};
  {butter, flour, milk, sugar};
  {butter, eggs, milk, salt};
  {eggs};
  {butter, flour, milk, salt, sugar}}

- Frequent itemsets:

items	support
{butter}	4
{milk}	4
{butter, milk}	4
{sugar}	3
{butter, sugar}	3
{milk, sugar}	3
{butter, milk, sugar}	3

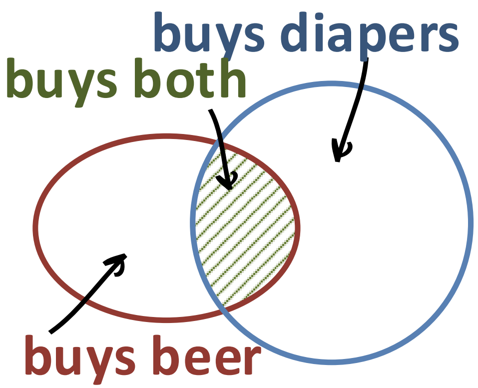
- Question of interest:
  - If milk and sugar are bought, will the customer always buy butter as well? $milk, sugar \Rightarrow butter$?
  - In this case, what would be the probability of buying butter?

Simple Association Rules: Basic Notions

- *Items* $I = \{i_1, i_2, \dots, i_m\}$: a set of literals (denoting items)
- *Itemset* X*:* Set of items $X \subseteq I$
- *Database* D: Set of *transactions* T, each transaction is a set of items $\mathrm{T} \subseteq I$
- Transaction T *contains* an itemset X: $X \subseteq T$
- The items in transactions and itemsets are sorted lexicographically:
  - itemset $X = (x_1, x_2, \dots, x_k)$, where $x_1 \le x_2 \le \dots \le x_k$
- *Length* of an itemset: cardinality of the itemset (*k-itemset:* itemset of length *k)*
- The *support* of an itemset X is defined as: $support(X) = |\{T \in D | X \subseteq T\}|$
- *Frequent itemset: an itemset* X is called frequent iff $support(X) \ge minSup$
- *Association rule:* An association rule is an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ are two itemsets with $X \cap Y = \emptyset$.
- Note: simply enumerating all possible association rules is not reasonable!
  → What are the interesting association rules w.r.t. D?

Interestingness of Association Rules

- *Interestingness of an association rule:*
  Quantify the interestingness of an association rule with respect to a transaction database D:
  - Support: frequency (probability) of the entire rule with respect to D
    $$support(X \Rightarrow Y) = P(X \cup Y) = \frac{|\{T \in D | X \cup Y \subseteq T\}|}{|D|} = support(X \cup Y)/|D|$$
    "probability that a transaction in D contains the itemset $X \cup Y$"
  - Confidence: indicates the strength of implication in the rule
    $$confidence(X \Rightarrow Y) = P(Y|X) = \frac{|\{T \in D | X \cup Y \subseteq T\}|}{|\{T \in D | X \subseteq T\}|} = \frac{support(X \cup Y)}{support(X)}$$
    "conditional probability that a transaction in D containing the itemset X also contains itemset Y"
  - Rule form: "$Body \Rightarrow Head\ [support, confidence]$"
- Association rule examples:
  - buys diapers ⇒ buys beers [0.5%, 60%]
  - major in CS ∧ takes DB ⇒ avg. grade A [1%, 75%]



Mining of Association Rules

- *Task of mining association rules:*
  Given a database D, determine all association rules having a $support \geq minSup$ and a $confidence \geq minConf$ (so-called *strong association rules*).
- Key steps of mining association rules:
  e.g. Apriori, FP-growth
  1) Find *frequent itemsets*, i.e., itemsets that have at least support $= minSup$
  2) Use the frequent itemsets to generate association rules
     - For each itemset X and every nonempty subset $\mathrm{Y} \subset X$ generate rule $\mathrm{Y} \Rightarrow (X - Y)$ if $minSup$ and $minConf$ are fulfilled
     - we have $2^{|X|} - 2$ many association rule candidates for each itemset X
- Example
  frequent itemsets

1-itemset	count	2-itemset	count	3-itemset	count
{A}	3	{A, B}	3	{A, B, C}	2
{B}	4	{A, C}	2		
{C}	5	{B, C}	4		

rule candidates: $\mathrm{A} \Rightarrow B; B \Rightarrow A;\ \mathrm{A} \Rightarrow C;\ C \Rightarrow \mathrm{A};\ B \Rightarrow C;\ \mathrm{C} \Rightarrow B;$
$A, B \Rightarrow C; A, C \Rightarrow B; C, B \Rightarrow A; A \Rightarrow B, C; B \Rightarrow A, C; C \Rightarrow A, B$

DATABASE SYSTEMS GROUP

Generating Rules from Frequent Itemsets

LMU

- For each frequent itemset X
  - For each nonempty subset Y of X, form a rule $Y \Rightarrow (X - Y)$
  - Delete those rules that do not have minimum confidence
    Note: 1) support always exceeds $minSup$
    2) the support values of the frequent itemsets suffice to calculate the confidence
- Example: $X = \{A, B, C\}$, $minConf = 60\%$
  - conf (A ⇒ B) = 3/3; ✓
  - conf (B ⇒ A) = 3/4; ✓
  - conf (A ⇒ C) = 2/3; ✓
  - conf (C ⇒ A) = 2/5; ✗
  - conf (B ⇒ C) = 4/4; ✓
  - conf (C ⇒ B) = 4/5; ✓
  - conf (A ⇒ B, C) = 2/3; ✓ conf (B, C ⇒ A) = ½ ✗
  - conf (B ⇒ A, C) = 2/4; ✗ conf (A, C ⇒ B) = 1 ✓
  - conf (C ⇒ A, B) = 2/5; ✗ conf (A, B ⇒ C) = 2/3 ✓

itemset	count
{A}	3
{B}	4
{C}	5
{A, B}	3
{A, C}	2
{B, C}	4
{A, B, C}	2

- Exploit anti-monotonicity for generating candidates for strong association rules!