Association Rule Mining

Tamás Horváth

University of Bonn &
Fraunhofer IAIS, Sankt Augustin, Germany
tamas.horvath@iais.fraunhofer.de

universitätbonn
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer
IAIS

Association Rules: Example

market basket transactions:

analysis of purchase "basket" data (items purchased together) in a department store

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

Examples of Association Rules:

{Diaper} → {Beer}
{Milk, Bread} → {Eggs,Coke}
{Beer, Bread} → {Milk}

- Implication means co-occurrence, not causality!

Association Rules: Example

- discovery of interesting relations between binary attributes, called *items*, in large databases

example of an association rule extracted from supermarket sales**:**

"Customers who buy milk and diaper also ***tend*** *to buy beer."*

- only rules with support and confidence above some minimal thresholds are extracted
    - support: proportion of customers who bought the three items among **all** customers
    - confidence: proportion of customers who bought beer among the customers who bought milk and diaper

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**

Application Example

market basket analysis

- marketing plan
- advertising strategies
- catalog design
- store layout

Notions and Notations

- $I = \{I_1, \ldots, I_m\}$: set of items
- itemset: collection of one or more items
- k-itemset: itemset of cardinality k
- transaction: itemset

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

- transaction database D: multiset of transactions
  - each transaction is associated with an identifier, called TID

Notions and Notations

support set $D[X]$ of an itemset X:

$$D[X] = \{T : T \in D \text{ and } X \subseteq T\}$$

– multiset of sets

support : fraction of transactions that contain an itemset, i.e., for $X \subseteq I$

$$support(X) = \frac{|D[X]|}{|D|}$$

frequent itemset: itemset with support greater than or equal to a threshold $minsup$

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:

$$support(\{Milk, Bread, Diaper\}) = \frac{2}{5}$$

Association Rules

- **association rule**
  - implication expression of the form $X \rightarrow Y$, where X and Y are disjoint non-empty itemsets
    - **example:** {Milk, Diaper} → {Bread}

- **rule evaluation metrics**
  - support (s): fraction of transactions that contain both X and Y
  - confidence (c): fraction of transactions that contain both X and Y relative to the transactions that contain X

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

example: $R = \{Milk, Diaper\} \rightarrow \{Bread\}$

$$s(R) = \frac{|D[\{Milk, Bread, Diaper\}]|}{|D|} = \frac{2}{5}$$

$$c(R) = \frac{|D[\{Milk, Bread, Diaper\}]|}{|D[\{Milk, Diaper\}]|} = \frac{2}{3}$$

Mining Association Rules

Given

- a *transaction database* D over a set I of items,
- *minimum support threshold* min_sup, and
- *minimum confidence threshold* min_conf

find all association rules $X \rightarrow Y$ satisfying

$$s(X \rightarrow Y) \geq min\_sup \text{ and } c(X \rightarrow Y) \geq min\_conf$$

Brute-Force Approach

1. list all possible association rules
2. compute the support and confidence for each rule
3. prune rules that fail the *min_sup* and *min_conf* thresholds

computationally prohibitive

- total number of *possible* association rules is exponential in the cardinality of the set of all items

⇨ **exponential delay** in worst case

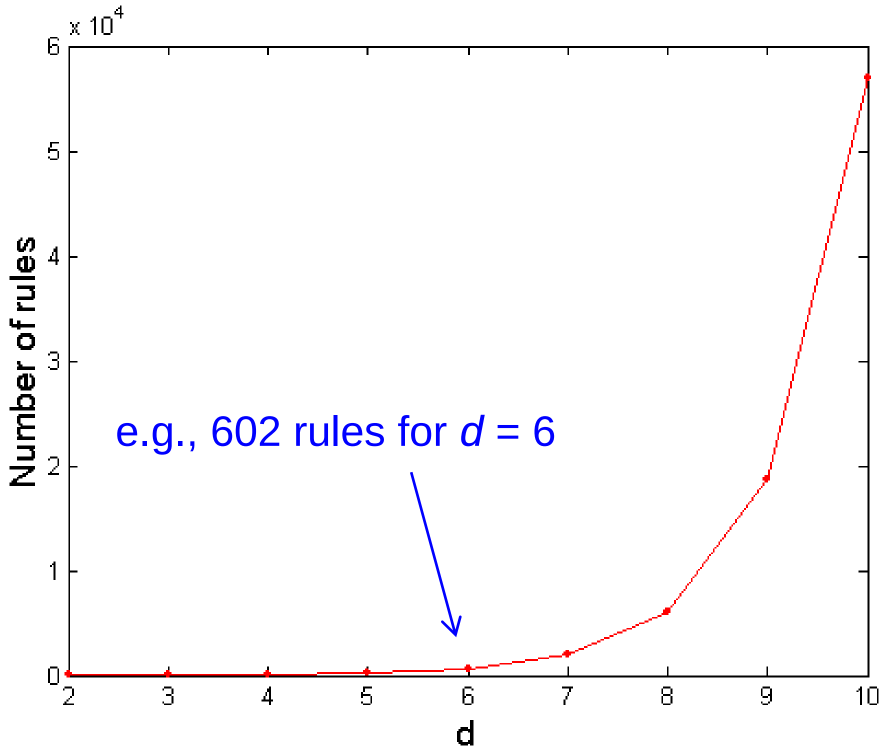
Upper Bound on the Number of Association Rules

let $d = |I|$

$\Rightarrow$ total number of (non-empty) itemsets is $2^d - 1$

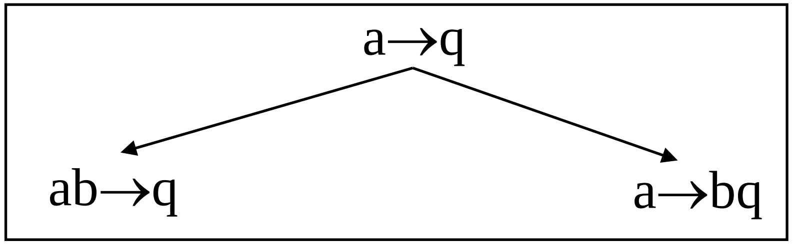
$\Rightarrow$ total number of possible association rules is $3^d - 2^{d+1} + 1$

Proof: *exercise*



Observations about the problem (I)

confidence	support
|*D*[*aq*]|/|*D*[*a*]|	|*D*[*aq*]|/|*D*|
??	IV
|*D*[*abq*]|/|*D*[*ab*]|	|*D*[*abq*]|/|*D*|



confidence	support
|*D*[*aq*]|/|*D*[a]|	|*D*[*aq*]|/|*D*|
IV	IV
|*D*[*abq*]|/|*D*[a]|	|*D*[*abq*]|/|*D*|

- confidence can both rise or fall, while support can only fall **as rules get longer**
  - ⇨ support can be used for pruning
- support depends only on *set* of items, not on exact rule
  - ⇨ do not search in space of rules, but in space of itemsets

universitätbonn
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT
Fraunhofer IAIS

Mining Association Rules

two-step approach:

1. **frequent itemset** generation
   - generate all itemsets whose support ≥ *min_sup*

2. **rule** generation
   - generate association rules of confidence ≥ *min_conf* from each frequent itemset *X* by binary partitioning of *X*

Step 1: Frequent Itemset Mining – Problem Definition

Given

- a *transaction database* D over a set I of items and
- an integer *frequency threshold* $t \geq 0$ (i.e., $t = \lceil min\_sup \cdot |D| \rceil$)

find all itemsets $X \subseteq I$ satisfying

$$|D[X]| \geq t$$

– X is referred to as frequent (or t-frequent) itemset

Remark on the Problem Setting

the transaction database D can be regarded as a

- $0/1$ (or Boolean) matrix,
- set system over I, where each element (i.e., transaction) is associated with its multiplicity in D (i.e., number of occurrences)
- vertices of the $|I|$-dimensional unit hypercube where each vertex is associated with the corresponding multiplicity
- hypergraph over the vertex set I such that each edge is associated with its multiplicity
- bipartite graph (V_1, V_2, E) such that $V_1 = I$, V_2 is the set of transactions, and there is an edge $\{u, v\}$ ($u \in V_1$ and $v \in V_2$) if and only if u is an element of transaction corresponding to v

Frequent Itemset Mining (recap)

- **brute-force** approach:
  - each itemset in the power set of $\mathcal{I}$ is a candidate frequent itemset
  - count the support of each candidate by scanning the database
  - match each transaction against every candidate
    - complexity ~ $O(NMw)$ ⇨ expensive since $M = 2^d - 1$ ($d = |\mathcal{I}|$)
      - N: number of transactions
      - M: number of candidate itemsets
      - w: maximum cardinality of the transactions

Frequent Itemset Mining Strategies

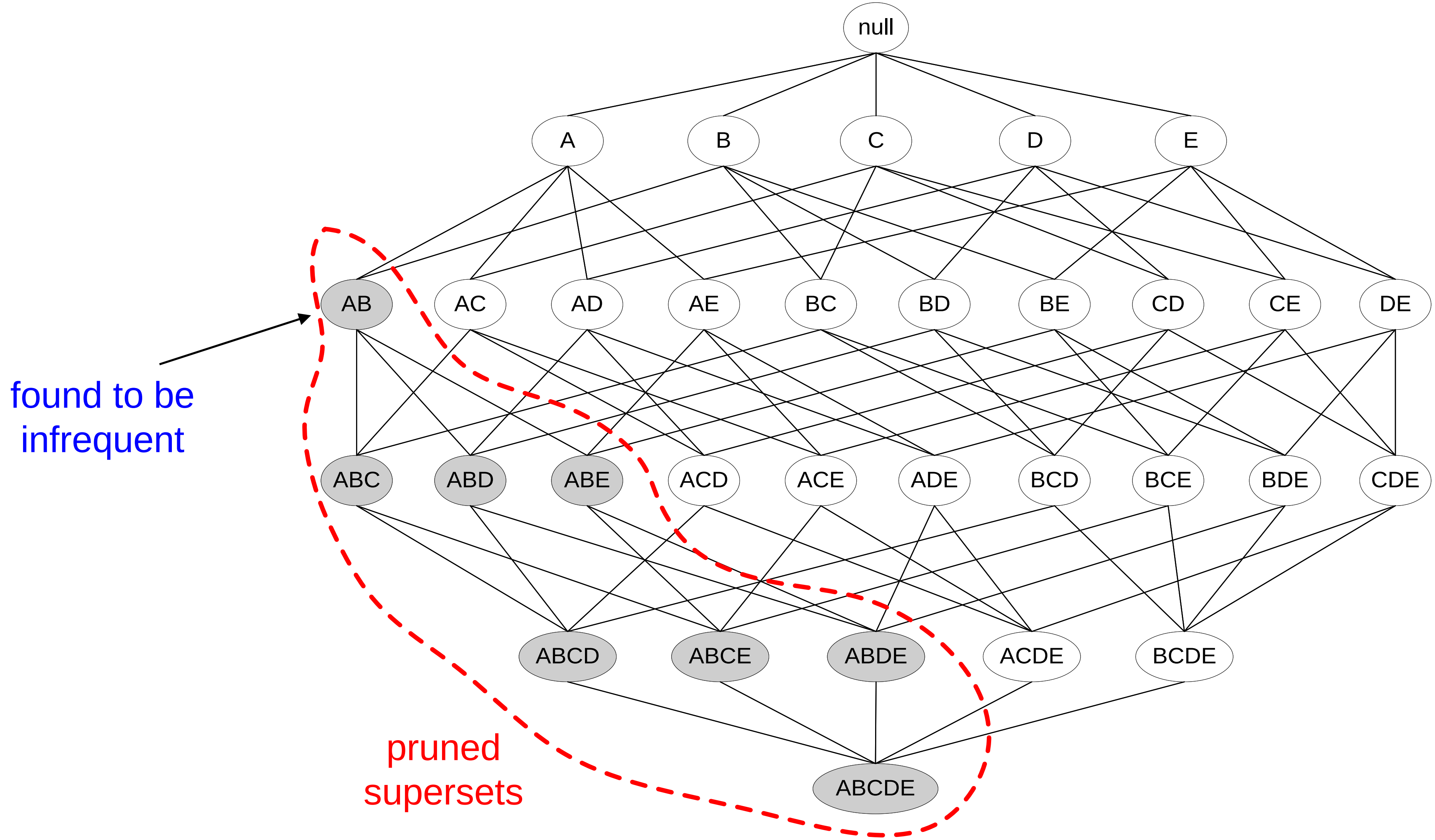
- reduce the number of candidates (*M*)
  - complete search: $M=2^d-1$
  - use **pruning** techniques to reduce *M*
- reduce the number of transactions (*N*)
  - reduce size of *N* as the number of transactions increases
  - use a subset of the *N* transactions by **sampling**
- reduce the number of comparisons (*NM*)
  - use **efficient data structures** to store the candidates or transactions
  - no need to match every candidate against every transaction

Frequent Itemset MiningStrategies

- Apriori principle:
  - if an itemset is frequent then all of its subsets must also be frequent
    - i.e., support set is **anti-monotone** with respect to the subset relation

$$\forall X, Y (X \subseteq Y \implies D[X] \supseteq D[Y])$$

Utilization of the Apriori Principle



Utilization of the 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)

- t = 3 (frequency threshold)

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

- triplets (3-itemsets)

if every subset is considered:

$$^6C_1 + {}^6C_2 + {}^6C_3 = 41$$

with support-based pruning:

$$6 + 6 + 1 = 13$$

The Apriori Algorithm

Input : $0/1$ matrix D with column set I and integer frequency threshold $t \geq 0$
Output: set of t-frequent itemsets

$\mathcal{C}_1 := I$
$i := 1$
while $\mathcal{C}_i \neq \emptyset$ **do**
    $\mathcal{F}_i := \{X \in \mathcal{C}_i : |D[X]| \geq t\}$ // candidate counting
    print $\mathcal{F}_i$
    $\mathcal{C}_{i+1} :=$ CandidateGeneration$(\mathcal{F}_i)$
    $i := i + 1$
endwhile

- [Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996]
- levelwise (breadth-first) search algorithm

Gaining Efficiency I: Generation of Candidates

Approach:

- generate new candidates by combining current frequent itemsets by utilizing that all $(k-1)$-itemsets of a frequent k-itemset are also frequent
- define a total order on I and consider an itemset as an ordered sequence

CANDIDATEGENERATION:

Input : set $\mathcal{F}_k$ of frequent k-itemsets
Output: set $\mathcal{C}_{k+1}$ of candidate $(k+1)$-itemsets

1: $\mathcal{C}_{k+1} = \emptyset$
2: **for all** $X, Y \in \mathcal{F}_k$ such that they differ only in their last elements
3: make a $(k+1)$-element set Z by concatenating the common $(k-1)$-prefix with the two differing elements according to the order
4: **if** all k-subsets of Z are in $\mathcal{F}_k$ **then** add Z to C_{k+1}
5: **return** $\mathcal{C}_{k+1}$

Example

candidate generation:

$$\mathcal{F}_3 = \{uvw, uvx, uwx, uwy, vwx\}$$
$$\mathcal{C}_4 = \{uvwx, uwxy\} \setminus \{uwxy\}$$

Apriori Algorithm for frequency threshold 2

$$\mathcal{C}_1 = \{a, b, c, d, e\}$$
$$\mathcal{F}_1 = \{a, b, c, e\}$$
$$\mathcal{C}_2 = \{ab, ac, ae, bc, be, ce\}$$
$$\mathcal{F}_2 = \{ac, bc, be, ce\}$$
$$\mathcal{C}_3 = \{bce\}$$
$$\mathcal{F}_3 = \{bce\}$$

database

Tid	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Complexity of the Apriori Algorithm

remarks on the frequent itemset mining problem

- enumeration problem
- size of the problem is defined by the *size of the input* database D
- size of the output can be exponentially large in the size of the input
  - e.g., for $D = \{I\}$ with $I = \{1, \ldots, n\}$ and frequency threshold 1, the number of frequent itemsets is exponential in n

  $\Rightarrow$ **hopeless** to compute the set of frequent itemsets in time polynomial in the input parameter

  $\Rightarrow$ the size of the output is also taken into account in the analyses of the time and space complexity

Enumeration Complexities

the **size** of the output (theory) can be exponential in the size of the input *D*

⇨ the output cannot be computed in time polynomial in the size of *D*

enumeration complexities:

a set of *S* with *N* elements, say $s_1,\ldots, s_N$, are listed with

- **polynomial delay** if the time before printing s_1, the time between printing s_i and s_{i+1} for every $i=1,\ldots,N-1$, and the termination time after printing s_N is bounded by a polynomial of the size of the input,
- **incremental polynomial time** if s_1 is printed with polynomial delay, the time between printing s_i and s_{i+1} for every $i=1,\ldots,N-1$ (resp. the termination time after printing s_N) is bounded by a polynomial of the combined size of the input and the set $s_1,\ldots, s_i$ (resp. *S*),
- **output polynomial time** if *S* is printed in the combined size of the input and the entire set *S*

Correctness and Complexity of the Apriori Algorithm

Proposition:

(i) The Apriori algorithm correctly and irredundantly enumerates all frequent itemsets.

(ii) The Apriori algorithm enumerates the set of frequent itemsets in incremental polynomial time.

Proof: *exercise*

Gaining Efficiency II: Candidate Counting

Why is counting supports of candidates a problem?

- the total number of candidates can be very huge
- one transaction may contain many candidates

Method:

- store candidate itemsets in a hash-tree
  - leaf nodes of hash-tree contain lists of itemsets and their support
  - interior nodes contain hash tables
- use subset function to find all the candidates contained in a transaction

Hash Tree - Construction

searching for an itemset $i_1, i_2, \ldots, i_d, \ldots, i_k$

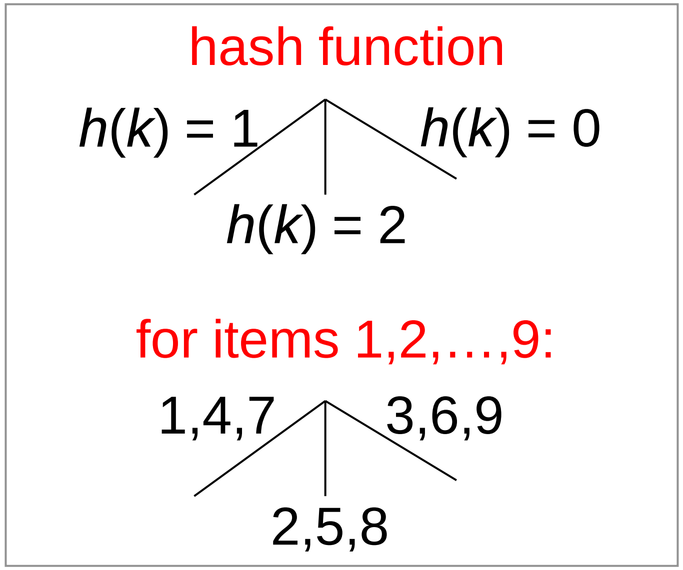
- start at the root
- at level d: apply the hash function h to i_d

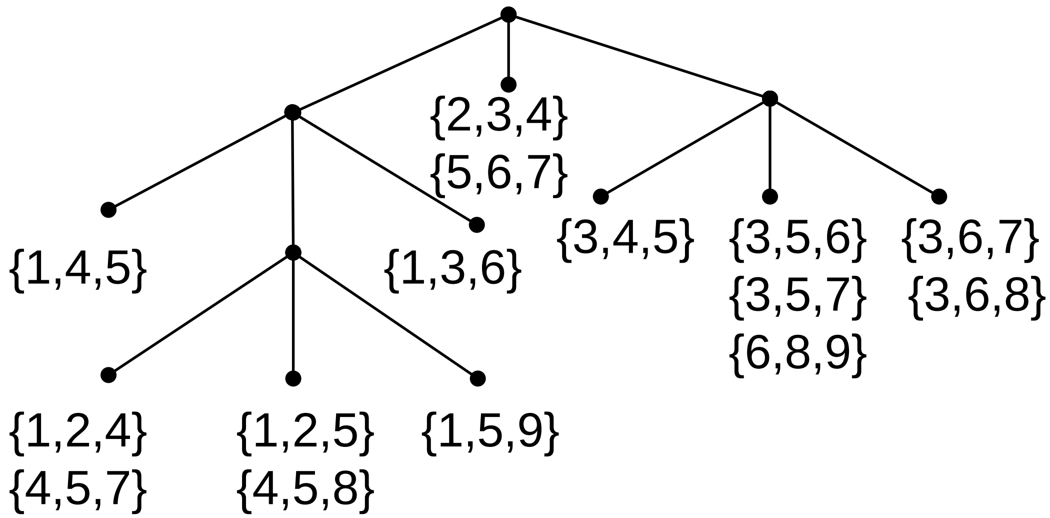
insertion of an itemset

- search for the corresponding leaf node, and insert the itemset into that leaf
- if an overflow occurs:
  - transform the leaf node into an internal node
  - distribute the entries to the new leaf nodes according to the hash function

Hash Tree Construction - Example

- candidate 3-itemsets:
  - {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: $h(k) = k \bmod 3$
- split nodes with more than 3 elements if possible



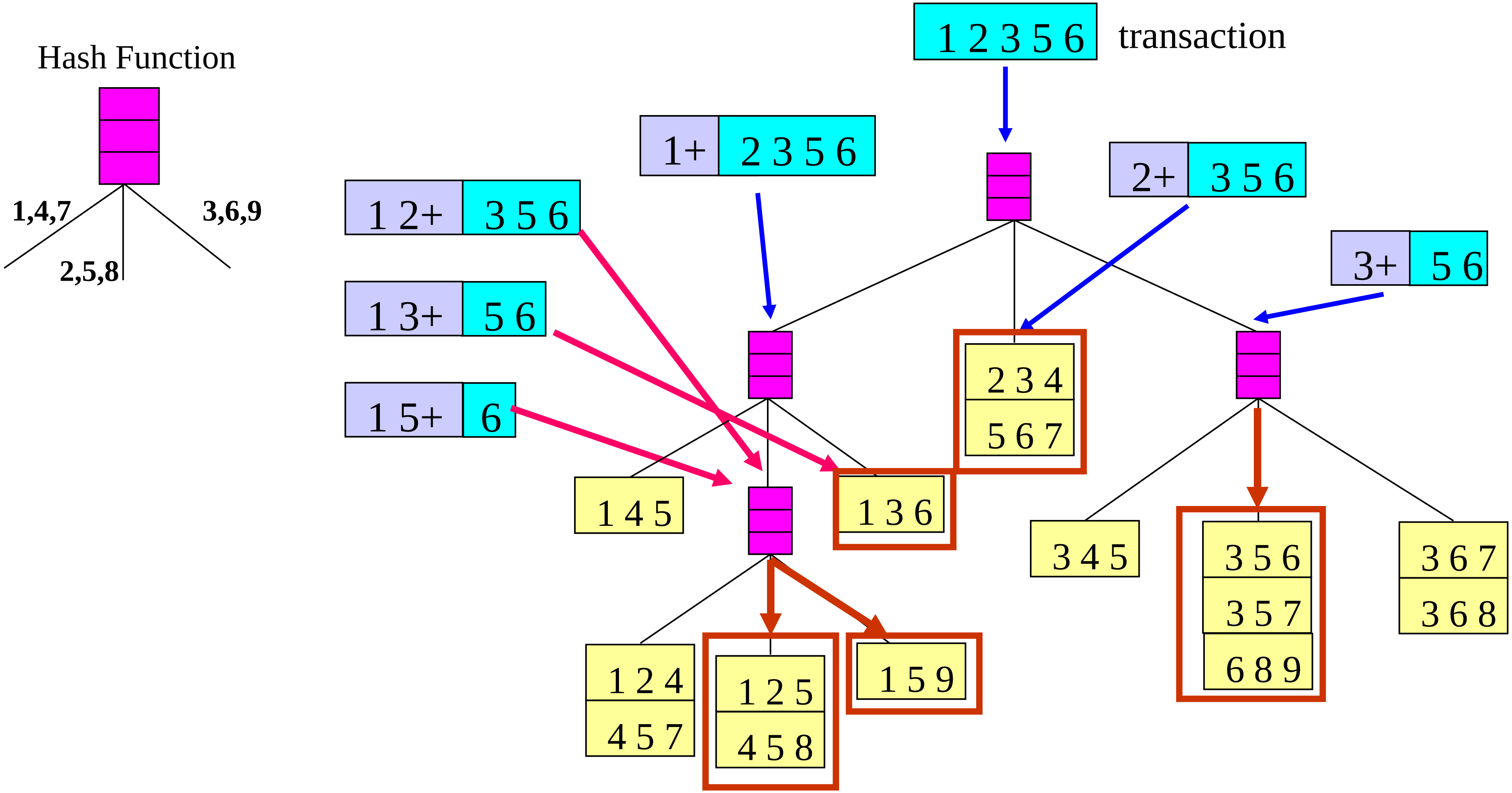


Hash Tree – Subset Function for Counting

search all candidate k-itemsets contained in a transaction $T = (t_1,t_2,\ldots,t_n)$

- at the root:
  - determine the hash values for each item $t_1,t_2,\ldots,t_{n-k+1}$ in T
  - continue the search in the resulting child nodes
- at an internal node at level d (reached after hashing of item t_i):
  - determine the hash values and continue the search for each item t_j with $j > i$ and $j <= n–k+d$
- at a leaf node:
  - check whether the itemsets in the leaf node are contained in transaction T

Subset Function for Counting - Example



- match transaction against 9 out of 15 candidates!

Mining Association Rules

two-step approach:

1. **frequent itemset** generation ✓
   – generate all itemsets whose support ≥ *minsup*

2. **rule** generation
   – generate association rules of confidence ≥ *minconf* from each frequent itemset *X* by binary partitioning of *X*

Observations about the Problem (II)

What happens when we create rules from a frequent itemset?

$$c=|D[abc]|/|D[ab]| \qquad s=|D[abc]|/|D|$$
$$\geq \qquad\qquad\qquad\qquad =$$
$$c=|D[abc]|/|D[a]| \qquad s=|D[abc]|/|D|$$

$ab \rightarrow c$
$\downarrow$
$a \rightarrow bc$

- the more items we put in the conclusion, the smaller the confidence
  - ⇨ search top-down breadth-first from smallest conclusions, prune
- confidence can be expressed in terms of support
  - ⇨ No DB accesses necessary when all supports of frequent itemsets are known!

Rule Generation

GENERATERULES:

Input : frequent k-itemset l_k, family $\mathcal{H}_m \subseteq 2^{l_k}$ of m-itemset consequents
Output: all association rules $l_k \setminus X \rightarrow X$ of confidence at least min_conf such that $|X| = m + 1$

```
if k > m + 1 then
    H_{m+1} = CANDIDATEGENERATION(H_m)        // same function as in Apriori
    forall h_{m+1} ∈ H_{m+1} do
        c = support(l_k)/support(l_k \ h_{m+1})
        if c ≥ min_conf then
            print rule (l_k \ h_{m+1}) → h_{m+1}
        else
            delete h_{m+1} from H_{m+1}
    GENERATERULES(l_k, H_{m+1})
```

Example

D:

- 1 2 3 4
- 1 2 6
- 1 2 3 5
- 1 2 3 8
- 1 3 9
- 2 3 9
- 3 7 8
- 4 5

min_conf = 0.8

min_sup = 3/8

C_1:	1	2	3	4	5	6	7	8	9
s:	5	5	6	2	2	1	1	2	2
F_1:	**1**	**2**	**3**						

C_2:	12	13	23
s:	4	4	4
F_2:	**12**	**13**	**23**

C_3:	123
s:	3
F_3:	**123**

Result:

$1 \rightarrow 2$

$2 \rightarrow 1$

$1 \rightarrow 3$

$2 \rightarrow 3$

Rule Generation:

12: $H_1 = \{\{1\},\{2\}\}$

$c(1 \rightarrow 2)=s(12)/s(1)=4/5=0.8$

$c(2 \rightarrow 1)=s(12)/s(2)=4/5=0.8$

13: $H_1 = \{\{1\},\{3\}\}$

$c(1 \rightarrow 3)=s(13)/s(1)=4/5=0.8$

$c(3 \rightarrow 1)=s(13)/s(3)=4/6=0.66$

23: $H_1 = \{\{2\},\{3\}\}$

$c(2 \rightarrow 3)=s(23)/s(2)=4/5=0.8$

$c(3 \rightarrow 2)=s(23)/s(3)=4/6=0.66$

123: $H_1 = \{\{1\},\{2\},\{3\}\}$

$c(12 \rightarrow 3)=s(123)/s(12)=3/4=0.75$

$c(13 \rightarrow 2)=s(123)/s(13)=3/4=0.75$

$c(23 \rightarrow 1)=s(123)/s(23)=3/4=0.75$

$H_2 = \emptyset$

Performance

evaluation on synthetic data (100.000 transactions based on 1000 items, with frequent set sizes distributed around 4 items and transaction size distributed around 10 items. D size 4.4 MB on an IBM RS6000 534H)

Minimum Support (%):	2.0	1.5	1.0	0.75	0.5
Run time (secs)	3.8	4.8	11.2	17.4	19.3

- [Agrawal et.al 96] found linear scaleup (slope 1) for transaction sets of up to 10 Million transactions (up to 838 MB of data)
- This is due to sparsity of data: in the worst case, all itemsets can be frequent, causing exponential behavior.

Summary of the Apriori Algorithm

1. find all itemsets with sufficient support (called “frequent” or “large” itemsets):
   - search top-down from one-element itemsets
   - breadth-first search, generate candidates of length *k* from those of length *k*-1
   - prune all sets that do not reach min support
2. for each frequent itemset from step 1, build all rules and return those with sufficient confidence
   - search top-down from one-element to longer conclusions
   - breadth-first search, generate conclusions of length *k* from those of length *k*-1
   - prune all rules that do not reach min confidence

Frequent Itemset Mining – Some Issues

1. Apriori is not suited for generating long frequent itemsets (e.g., of length 100)
   - we need alternative algorithms enabling the discovery of **long** patterns
2. it would be useful to know in advance the cardinality of the family of frequent itemsets
   - complexity of counting frequent itemsets
3. length of frequent itemsets
   - complexity of deciding the existence of a frequent itemset of a given length

Bottleneck of the Apriori Algorithm

Observation:

- to discover a frequent itemset of size *k*, one needs to generate at least 2^k-2 candidate itemsets
    - e.g., if k = **100** then about $\mathbf{10^{30}}$ itemsets
    - **hopeless** to find long frequent itemsets

How can we avoid this bottleneck of Apriori?

⇨ use **depth-first** search

Mining Frequent Itemsets Without Candidate Generation

idea: grow **long** itemsets from **short** ones using local frequent items

example:

suppose *abc* is a frequent itemset

1. get all transactions in the database *D* containing *abc*
   - *D*[*abc*]
2. let *d* be a **local** frequent item in *D*[*abc*]
   - ⇨ *abcd* is a frequent itemset in *D*

Depth-First Search Frequent Itemset Mining Algorithm

DFS_LISTING:

Input : transaction database $\mathcal{D}$, itemset F, and frequency threshold $t \geq 0$
Output: $\{F' \supseteq F : F' \setminus F$ is t-frequent in $\mathcal{D}\}$

```
print F
remove all infrequent items from D
define a linear (total) order ≤ on the items in D
forall items i in D such that i ∉ F do
    let D_i = {proj(T, i) : T ∈ D satisfying i ∈ T}, where

              proj(T, i) = {i' ∈ T : i < i'}

    DFS_LISTING(F ∪ {i}, D_i)
```

print F
remove all infrequent items from $\mathcal{D}$
define a linear (total) order $\leq$ on the items in $\mathcal{D}$
forall items i in $\mathcal{D}$ such that $i \notin F$ **do**
  let $\mathcal{D}_i = \{\text{proj}(T, i) : T \in \mathcal{D}$ satisfying $i \in T\}$, where

$$\text{proj}(T, i) = \{i' \in T : i < i'\}$$

  DFS_LISTING$(F \cup \{i\}, \mathcal{D}_i)$

initial call: DFS_LISTING$(\emptyset, \mathcal{D})$

Depth-First Frequent Itemset Mining Algorithm

Prop.: the previous algorithm *correctly* and *irredundantly* enumerates all frequent itemsets with **polynomial delay**

- **correct:** sound and complete
  - **sound:** all itemsets outputted are frequent and
  - **complete:** all frequent itemsets are generated

Proof: *exercise*

How to store projected databases?

Frequent Pattern Trees (FP-Trees)

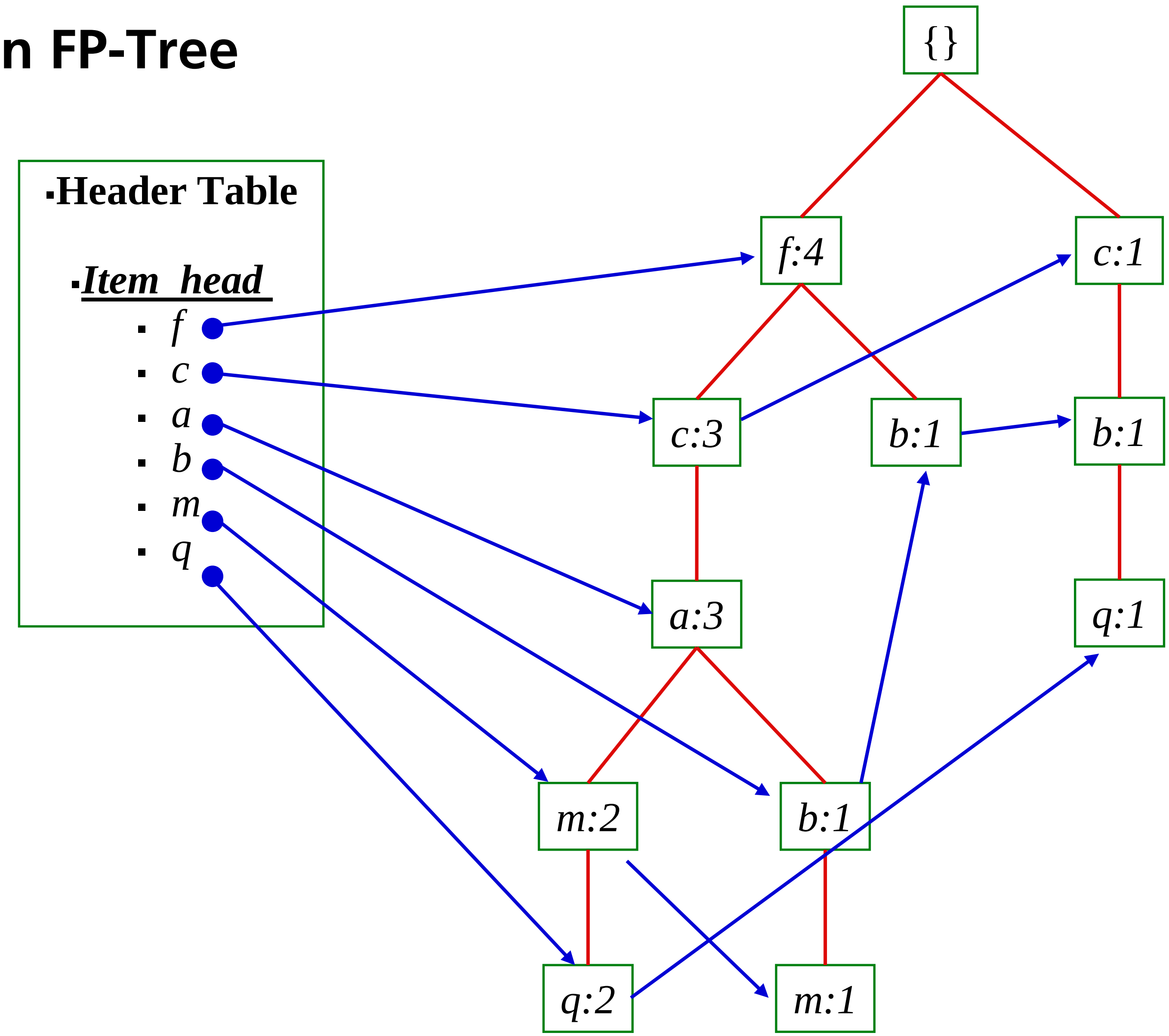
- [Han, Pei, Yin, & Mao, 2004]

FP-tree consists of

1. an **item-prefix tree** with *nodes* consisting of
   - item-name: name of the item represented by the node,
   - count: number of transactions represented by the portion of the path reaching the node,
   - node-link: links to the next node in the item-prefix tree having the same item name (or null if there is no such node)
2. a **frequent item header table** with *entries* consisting of
   - item-name,
   - head of node link: points to the first node in the item-prefix tree having the item name

Provides a compact representation of transaction databases!

Example of an FP-Tree



Algorithm: FP-Tree Construction

Input : transaction database D and frequency threshold t
Output: frequent-pattern tree T of D w.r.t. t

1: compute the set I' of frequent items and their support

2: sort I' in support descending order

3: create the root of an FP-tree T with label null

4: **forall** transaction $X \in D$ **do**

5: select the frequent items in X and sort them according to the order of I'; let the sorted frequent-item list in X be $[p|P]$, where
- p is the first element and
- P is the remaining list

6: $\textsc{InsertTree}([p|P], T)$

Function InsertTree

INSERTTREE($[p|P], T$):

```
if T has a child N such that N.item-name = p.item-name then
    ++N.count
else
    create a new child N of T
    N.name := p.item-name
    N.count := 1
    N.node_link = NULL
    set the node-link of the last element in the node_link chain of p to N
if P is nonempty then
    INSERTTREE(P, N)
```

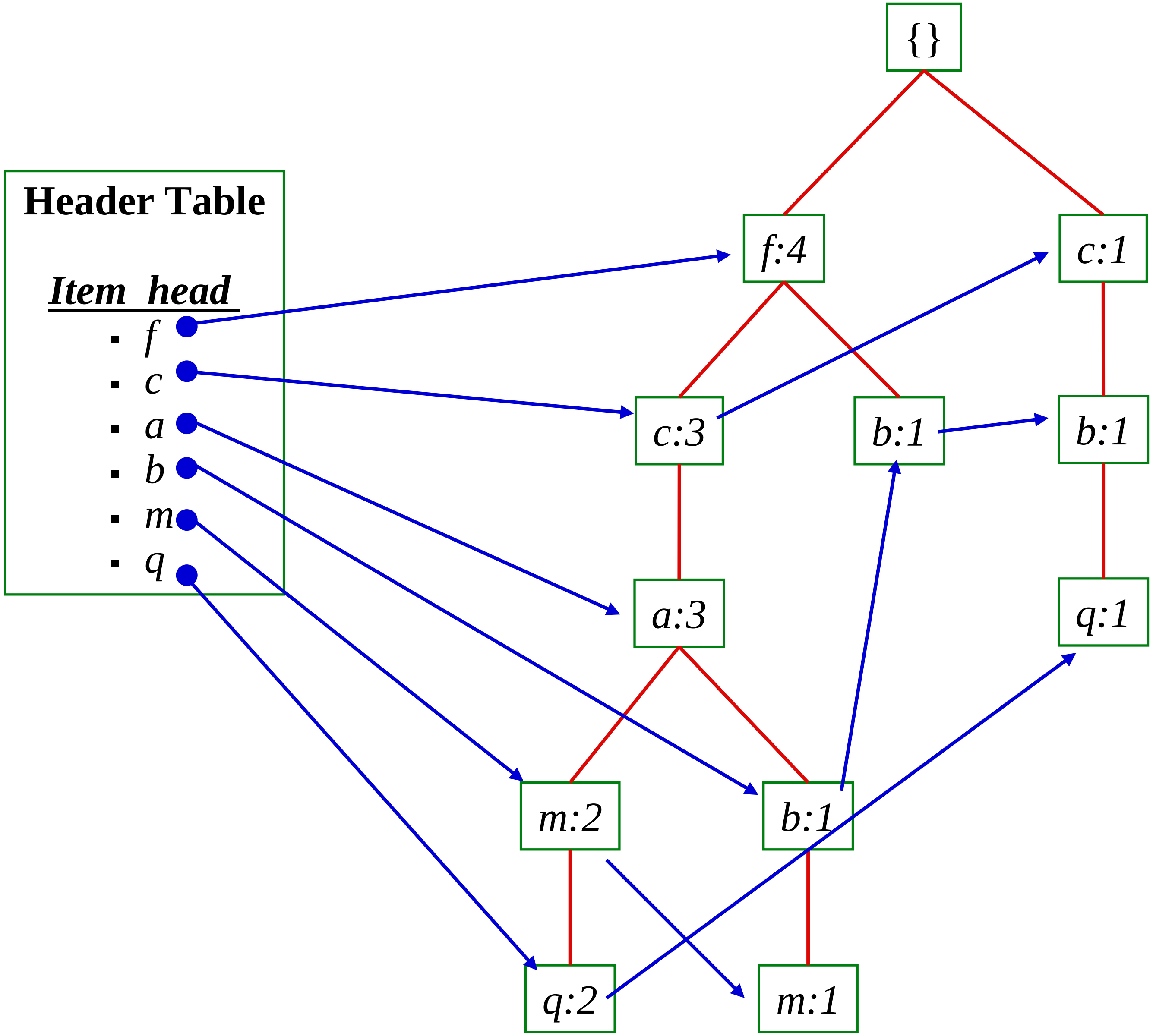
Example (FP-tree)

TID	Items
1	f, a, c, d, g, i, m, q
2	a, b, c, f, l, m, o
3	b, f, h, j, o, w
4	b, c, k, s, q
5	a, f, c, e, l, q, m, n

frequency threshold t = 3

I' = {f:4,c:4,a:3,b:3,m:3,q:3}

TID	Ordered Items
1	f, c, a, m, q
2	f, c, a, b, m
3	f, b
4	c, b, q
5	f, c, a, m, q



Benefits of FP-trees

- **completeness**
  - preserve complete information for frequent pattern mining
  - never break a long pattern of any transaction
- **compactness**
  - reduce irrelevant info
    - infrequent items are removed
  - items in frequency descending order
    - the more frequently occurring, the more likely to be shared
  - never larger than the original database
    - *node-links* and the *count* field not counted!
  - empirically justified
    - *Connect-4* (dataset): **67,557** transactions with **43** items/transaction; t = **33779**
    - size of the input database: **2,219,609**; size of the FP-tree **13,449**

    ⇨ compression ratio = **165.04**

Properties of FP-trees

1. **completeness:**

   Given a transaction database *D* and a frequency threshold *t*, the **complete** set of frequent item projections of transactions in the database can be derived from the FP-tree of *D*.

2. **compactness:**

   Given a transaction database *D* and a frequency threshold *t*, then, without considering the root,

   - the size of *D*'s FP-tree is bounded by

     $$\Sigma_{T \in D} \; |\text{freq}(T)|$$

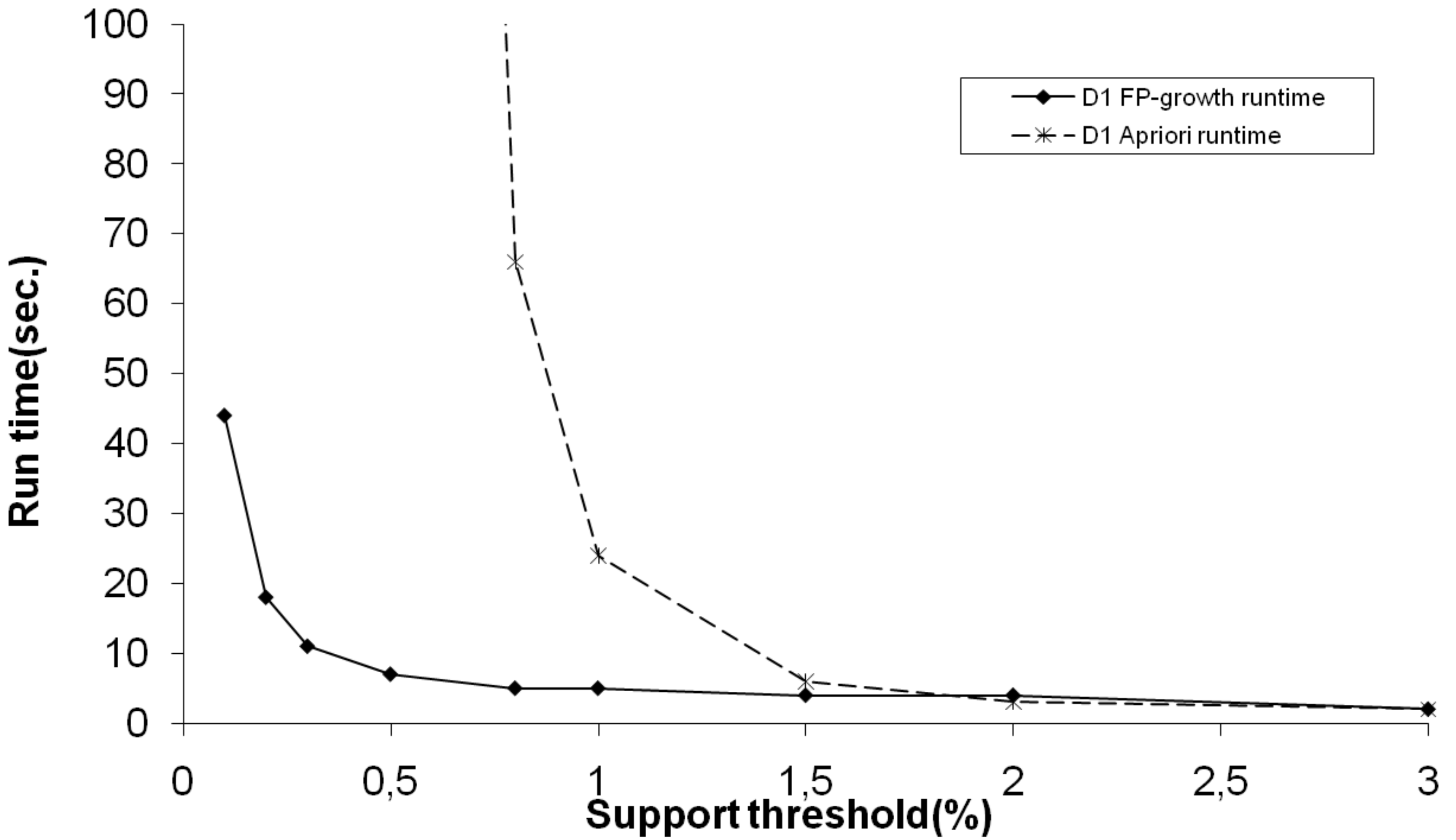
     - freq(T) = { $x \in T$: x is frequent }

   - and the height of DB's FP-tree is bounded by

     $$\max_{T \in D}\{ \; |\text{freq}(T)| \; \}$$

FP-Growth vs. Apriori: Scalability With the Support Threshold

Data set T25I20D10K



universitätbonn

RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer IAIS

Summary of the FP-Growth Algorithm

- **depth-first frequent itemset mining algorithm:**
  - decompose both the mining task and D according to the frequent patterns obtained so far
  - leads to focused search of smaller databases
- **other factors**
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire database
  - basic operations: counting and FP-tree building
    - no pattern search and pattern matching
- **winner** of FIMI 2003 (**F**requent **I**temset **M**ining **I**mplementations)

Frequent Itemset Mining – Some Issues

1. Apriori is not suited for generating long frequent itemsets (e.g., of length 100)
   - an alternative algorithm not excluding the discovery of long patterns ✓
2. it would be useful to know in advance the cardinality of the family of frequent itemsets
   - complexity of counting frequent itemsets
3. length of the itemsets
   - complexity of deciding the existence of a frequent itemset of a given length

Counting Frequent Itemsets

Thm.: Given a transaction database *D* and an integer frequency threshold *t*, the problem of finding the number of *t*-frequent itemsets is #P-hard.

- #P: class of functions *f* such that there is a *nondeterministic polynomial-time* Turing machine *M* with the property that *f*(*x*) is the number of accepting computation paths of *M* on input *x*
  - L. Valiant, 1979
- some functions in #P are at least as difficult to *compute* as some NP-complete problems are to *decide*
  - e.g., #3CNF

⇨ Unless P=NP, frequent itemsets cannot be counted in polynomial time!

Proof

reduction from the **#SAT for monotone 2CNF formulas**

- #SAT: number of satisfying assignments
- monotone 2CNF formulas: CNF in which every clause *has at most two* literals and every literal is *positive* (i.e., unnegated)
- #P-hard problem [Valiant, 1979]

Proof (cont'd)

- let f be a monotone 2CNF formula with m clauses and n variables
  - say, $x_1, \ldots, x_n$
  - see also the next slide for an example
- construct an $m \times n$ binary matrix (i.e., transaction database) D with

$$D_{ij} = \begin{cases} 0 & \text{if } x_j \text{ is present in the } i\text{-th clause} \\ 1 & \text{o/w} \end{cases}$$

⇒ an assignment falsifies f if and only if the set of items corresponding to the variables with value 1 forms a 1-frequent itemset (i.e., abs. freq. $t = 1$)

⇒ number of 1-frequent sets = $2^n -$number of the satisfying assignments of f

q.e.d.

Construction in the Proof: Example

$f = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \quad \Rightarrow$

	x_1	x_2	x_3	x_4
$x_1 \vee x_2$	0	0	1	1
$x_2 \vee x_3$	1	0	0	1
$x_1 \vee x_4$	0	1	1	0

for frequency threshold $t = 1$:

- $\{x_3\}$ is t-frequent because it occurs in $2(> t = 1)$ lines (transactions)

$\Rightarrow$ variable assignment $(0, 0, 1, 0)$ corresponding to $\{x_3\}$ falsifies f

- falsifying assignments: $\{x_1, x_2, x_3, x_4, x_1x_4, x_2x_3, x_3x_4\}$

$\Rightarrow$ number of satisfying assignments: $2^4 - 7 = 9$

Frequent Itemset Mining – Some Issues

1. Apriori is not suited for generating long frequent itemsets (e.g., of length 100)
   - an alternative algorithm not excluding the discovery of long patterns ✓
2. it would be useful to know in advance the cardinality of the family of frequent itemsets
   - complexity of counting frequent itemsets ✓
3. length of frequent itemsets
   - complexity of deciding the existence of a frequent itemset of a given length

Frequent Itemsets of Given Length

Thm.: Given a transaction database D, an integer frequency threshold $t > 0$, and an integer $k > 0$, the problem of deciding if there is a t-frequent itemset consisting of at least k items is **NP-complete**.

Proof:

1. the problem is in NP: *trivial*
2. NP-hardness: reduction from the **Balanced Bipartite Clique** problem
   - (V_1, V_2, E): bipartite graph; a **balanced** bipartite clique of size k is a complete bipartite clique with k vertices from each of V_1 and V_2
   - the **problem**: given a bipartite graph G and a positive integer k, decide whether G has a balanced bipartite clique of size k
     - NP-complete (Garey & Johnson, 1979)

Proof of NP-Hardness (cont'd)

reduction from the **Balanced Bipartite Clique** problem:

- let $G = (V_1, V_2, E)$ be a bipartite graph with $|V_1| = n_1$ and $|V_2| = n_2$
- construct an $n_1 \times n_2$ $0/1$ matrix D (i.e., transaction database) with

$$D_{i,j} = \begin{cases} 1 & \text{if vertex } i \text{ is connected with vertex } j \\ 0 & \text{o/w} \end{cases}$$

$\Rightarrow$ G has a balanced bipartite clique of size k if and only if D has a k-frequent set of cardinality at least k

q.e.d.

Summary

- FP-Growth algorithm: no candidate generation
  - ☺ **polynomial delay** listing
  - ☺ in contrast to Apriori: **able** to generate long frequent itemsets
- sometimes it would be useful to know in advance the **number** of frequent itemsets, but
  - ☹ counting the number of frequent itemsets is computationally **intractable**
- … and/or the **length** of frequent itemsets, but
  - ☹ deciding the existence of a frequent itemset of a given length is computationally **intractable**

Condensed Representations of Frequent Itemsets

1. maximal frequent itemsets

- **the Pincer Search algorithm**
  - (Lin & Kedem, 2002)
- **the Dualize and Advance Algorithm**
  - **(**Gunopulos, Khardon, Mannila, Saluja, Toivonen, & Sharma, 2003)
- **complexity of mining maximal frequent itemsets**

Finding the Positive Border: One-Way Searches

- **bottom-up** search (e.g., Apriori):
  - good performance, if all elements in the positive border are expected to be short
- **top-down** search
  - good performance, if all elements in the positive border are expected to be long

⇨ if some elements in the border are long and some are short, then both are inefficient

☹ **Problem**: deciding if there is a frequent itemset with at least *k* attributes is NP-complete

- see Slides 57-58

Finding the Positive Border with Bidirectional Search

Pincer-Search [Lin & Kedem, 1998, 2002]**:**

- computes the **positive border** (i.e., maximal frequent itemsets)
  - represents the set of frequent itemsets
  - can be exponentially smaller than the set of frequent itemsets
- bidirectional search (i.e., both bottom-up and top-down)
  - **bottom-up:** go up **one** level in each pass (similar to Apriori)
  - **top-down:** can go down **many** levels in one pass
- during the search it prunes by the properties:

Property 1: if an itemset is infrequent, all its supersets must be infrequent

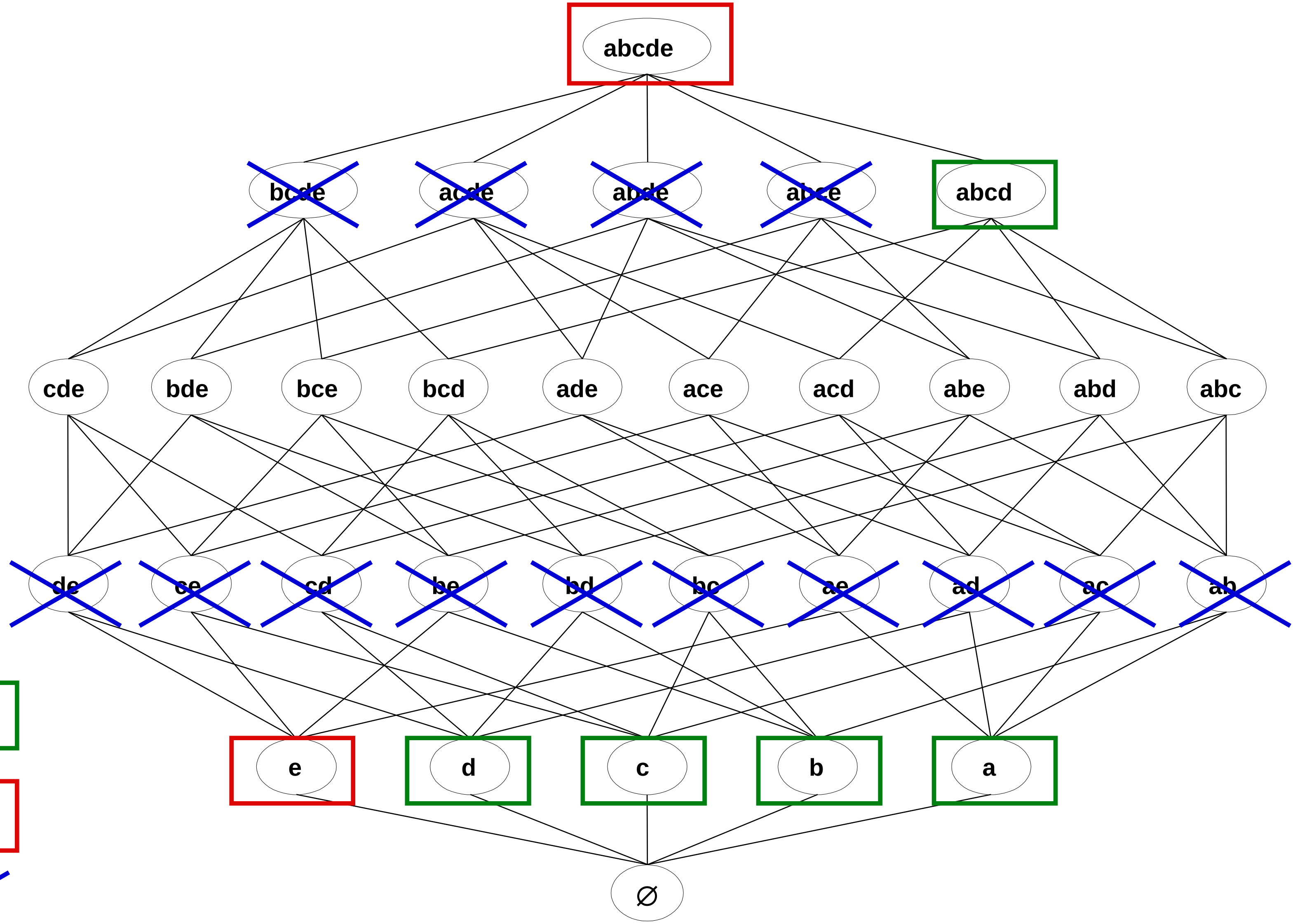
Property 2: if an itemset is frequent, all its subsets must be frequent

Example

Transactions

1: abcde
2: ac
3: ab
4: abcd

freq. threshold: 2



frequent:

infrequent:

prunable:

Maximal Frequent Candidate Set (MFCS)

At some point of the algorithm, let

- **FREQUENT**: set of *known* frequent itemsets
- **INFREQUENT**: set of *known* infrequent itemsets

MFS: set of *known* maximal frequent itemsets

MFCS (auxiliary data structure): set of all candidate **maximal itemsets** satisfying

$$\mathrm{FREQUENT} \subseteq \bigcup\{2^X : X \in \mathrm{MFS} \cup \mathrm{MFCS}\}$$

$$\mathrm{INFREQUENT} \cap \left(\bigcup\{2^X : X \in \mathrm{MFS} \cup \mathrm{MFCS}\}\right) = \emptyset$$

- **not known** to be frequent **at this state** of the algorithm

The Pincer-Search Algorithm

Input : transaction database over $I = \{1, 2, \ldots, n\}$ and frequency threshold
Output: set of all maximal frequent itemsets

1: $k := 1; \mathcal{C}_k := \{\{i\} : i \in I\}$
2: MFCS $:= \{I\}$; MFS $:= \emptyset$
3: **while** $\mathcal{C}_k \neq \emptyset$ **do**
4: read database and count supports for $\mathcal{C}_k$ and MFCS
5: remove frequent itemsets from MFCS and add them to MFS
6: $\mathcal{L}_k := \{X \in \mathcal{C}_k$: (i) X is frequent and $X \notin \mathcal{P}(\text{MFS})$ or // $\mathcal{P}(\text{MFS}) = \bigcup_{M \in \text{MFS}} 2^M$
(ii) $\exists X' \in \mathcal{C}_k$ s.t. X, X' are joinable, $X, X' \in \mathcal{P}(\text{MFS})$, and $\nexists M \in$ MFS with $X, X' \subseteq M\}$
7: $\mathcal{S}_k := \{X \in \mathcal{C}_k : X$ is infrequent$\}$
8: **if** $\mathcal{S}_k \neq \emptyset$ **then** MFCS = MFCS-gen$(\text{MFCS}, \mathcal{S}_k)$ // updates MFCS; Slides 66–67
9: $\mathcal{C}_{k+1} = \text{CandidateGeneration}(\mathcal{L}_k)$ // Apriori; Slide 21
10: **if** any frequent itemset in $\mathcal{C}_k$ has been removed in line 6 **then**
11: call the recovery procedure to recover missing candidates to $\mathcal{C}_{k+1}$ // Slides 68–69
12: call the new pruning procedure to prune candidates in $\mathcal{C}_{k+1}$ // Slide 70
13: $k := k + 1$
14: **return** MFS

Updating MFCS: Algorithm MFCS-gen (Line 8 in Slide 65)

Input : old MFCS and family $\mathcal{S}_k$ of infrequent sets found in pass k

Output: new MFCS

forall itemsets $S \in \mathcal{S}_k$ **do**
    forall itemsets $M \in$ MFCS **do**
        if $S \subseteq M$ **then**
            remove M from MFCS
            forall items $e \in S$ **do**
                if $M \setminus \{e\}$ is not a subset of any itemset in MFCS **then**
                    add the itemset $M \setminus \{e\}$ to MFCS
return MFCS

Algorithm MFCS-gen (Line 8 on Slide 65)

example:

- old MFCS $= \{abcdef\}$
- infrequent sets: $\mathcal{S}_k = \{af, cf\}$

1. $af \subseteq abcdef$

   $\Rightarrow$ MFCS $=$ MFCS $\setminus \{abcdef\} \cup \{bcdef, abcde\} = \{abcde, bcdef\}$

2. $cf \subseteq bcdef$

   $\Rightarrow$ MFCS $=$ MFCS $\setminus \{bcdef\} \cup \{bdef\}$ // $bcde \subseteq abcde$

   $= \{abcde, bdef\}$

Lemma: Algorithm MFCS-gen correctly updates MFCS.

Proof: *exercise*

Candidate Generation in Pincer-Search

- same candidate generation procedure as in Apriori

problem:

- some of the needed itemsets could be missing from the preliminary candidate set

example: suppose *MFS* is empty

- *abcde* ∈ *MFCS* is frequent ⇨ *abcde* is deleted from *MFCS* and added to *MFS*
- L_3 = {*abc,abd,abe,acd,ace,ade,bcd,bce,bde,bdf,bef,cde,def* }

are removed by Pincer-Search in Line 6

set of new candidates is empty, although it should be { *bdef* } !

Missing candidates must be **recovered! (Lines 10-11)**

universitätbonn

RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer IAIS

The Recovery Procedure (Lines 10-11 on Slide 65)

Input:
- current MFS
- $\mathcal{L}_k$ computed in Line 6
- $\mathcal{C}_{k+1}$ obtained by Apriori candidate generation from $\mathcal{L}_k$ in Line 9

Output: a complete set $\mathcal{C}_{k+1}$ of candidate $(k+1)$-itemsets

forall itemsets $X \in \mathcal{L}_k$ **do**
  forall itemsets $M \in$ MFS **do**
    if the first $k-1$ items in X are also in M **then**
      // suppose $M[j] = X[k-1]$
      // $M[j]$: j-th item of M w.r.t. linear order on the items
      forall $i = j+1$ **to** $|M|$ **do**
        $\mathcal{C}_{k+1} = \mathcal{C}_{k+1} \cup \{\{X[1], X[2], \ldots, X[k-1], X[k], M[i]\}\}$
return $\mathcal{C}_{k+1}$

Pruning (Line 12 on Slide 65)

Apriori: Check if all k-subsets of a candidate itemset X in $\mathcal{C}_{k+1}$ are in $\mathcal{L}_k$!

Pincer-Search: Check if X is a subset of an itemset in the current MFCS!

- One fewer loop!

new pruning procedure:

Input : current MFCS and $\mathcal{C}_{k+1}$ after candidate generation and the recovery proc.

Output: final candidate set $\mathcal{C}_{k+1}$

1: **forall** itemsets $X \in \mathcal{C}_{k+1}$ **do**
2:     **if** there exists no $Y \in$ MFCS such that $X \subseteq Y$ **then**
3:         delete X from $\mathcal{C}_{k+1}$
4: **return** $\mathcal{C}_{k+1}$

Pincer-Search Algorithm: Example

dataset: $\mathcal{D} = \{abcde, ac, ab, abcd\}$, (absolute) frequency threshold: $t = 2$

MFCS $= \{abcde\}$, MFS $= \emptyset$

$k = 1$:

- $\mathcal{C}_1 = \{a, b, c, d, e\}$
- $|\mathcal{D}[a]| = 4,\ |\mathcal{D}[b]| = |\mathcal{D}[c]| = 3,\ |\mathcal{D}[d]| = 2,\ |\mathcal{D}[e]| = 1;\quad |\mathcal{D}[abcde]| = 1$ // line 4
- MFCS $= \{abcde\}$ and MFS $= \emptyset$ // line 5
- $\mathcal{L}_1 = \{a, b, c, d\}$ and $\mathcal{S}_1 = \{e\}$ // because MFS $= \emptyset$; lines 6–7
- MFCS-gen $\Rightarrow$ MFCS $= \{abcd\}$ // line 8

$k = 2$:

- $\mathcal{C}_2 = \{ab, ac, ad, bc, bd, cd\}$ // because MFS $= \emptyset$; lines 9–12
- $|\mathcal{D}[ab]| = |\mathcal{D}[ac]| = 3,\ |\mathcal{D}[ad]| = |\mathcal{D}[bc]| = |\mathcal{D}[bd]| = |\mathcal{D}[cd]| = 2;\quad |\mathcal{D}[abcd]| = 2$
- MFCS $= \emptyset$ and MFS $= \{abcd\}$
- $\mathcal{L}_2 = \emptyset$ and $\mathcal{S}_2 = \emptyset$ // because $ab, ac, ad, bc, bd, cd \subseteq abcd$

return MFS $= \{abcd\}$ because $\mathcal{C}_3 = \emptyset$

Pincer Search Algorithm

Thm: The Pincer-Search algorithm correctly generates the family of maximal frequent itemsets.

Proof: *omitted*

Performance evaluation:

- experiments with large datasets of various properties
  - Lin & Kedem, 2002
- outperforms Apriori

Pincer-Search Algorithm: A Remark

Line 6 of the algorithm on slide 65: the original paper requires only condition (i)

- See D.I. Lin and Z.M. Kedem: *Pincer-Search: An Efficient Algorithm for Discovering the Maximum Frequent Set*. IEEE Transactions on Knowledge and Data Engineering, **14**(3):553-566, 2002.
- **however**, there is a remark in Case 4 of Lemma 2 in the paper above: if frequent *k*-itemsets X and X´ are joinable, both are subsets of MFS, but there is no single element of MFS containing X and X´, then their join must also be recovered
  - this is what we ensure with condition (ii) in Line 6
  - it is an interesting question, whether the algorithm remains complete if only condition (i) is used
  - adding condition (ii) to Line 6 does not change the worst-case complexity of the algorithm

Condensed Representations of Frequent Itemsets I

maximal frequent itemsets

- the Pincer Search algorithm ✓
  - (Lin & Kedem, 2002)
- the Dualize and Advance Algorithm
  - (Gunopulos, Khardon, Mannila, Saluja, Toivonen, & Sharma, 2003)
- complexity of mining maximal frequent itemsets

Hypergraph Transversals

hypergraph $H = (V, E)$:

- V: finite set of vertices
- $E \subseteq 2^V \setminus \{\emptyset\}$: set of (hyper)edges of H
  - ordinary undirected graphs are special hypergraphs

some notions:

- H is **simple** (or **Sperner**): none of its edges is contained by any other edge
- **transversal** of H: subset of V that intersects all edges of H
- **minimal** transversal: does not contain properly any other transversal
  - Tr(H): collection of all minimal transversals of H (also a hypergraph)

Hypergraph Transversals

Problem: *Given* a hypergraph H, *compute* $\mathrm{Tr}(H)$.

- listing problem
- can be solved in incremental *subexponential* time
  - subexponential: $k^{O(\log k)}$
  - (Fredman & Khachiyan, 1996)
- open problem whether it can be solved in incremental polynomial time

Hypergraph Transversals: Example

hypergraph $H = (V, E)$:

- $V = \{a, b, c, d\}$
- $E = \{abc, d\}$

- H is **simple**
- **transversals** of H: $\{ad, bd, cd, abd, acd, bcd, abcd\}$
- **minimal transversals** of H: $\text{Tr}(H) = \{ad, bd, cd\}$

Borders of Theories and Hypergraph Transversals

notions: for a family $\mathcal{F}$ of frequent itemsets, let

- $\text{cl}(\mathcal{F}) = \{Y : Y \subseteq X \text{ for some } X \in \mathcal{F}\}$ // downward closure of $\mathcal{F}$
  - $\text{cl}(\mathcal{F})$: family of frequent itemsets represented by $\mathcal{F}$
- $Bd^+(\text{cl}(\mathcal{F}))$: family of maximal frequent itemsets in $\text{cl}(\mathcal{F})$
  - **positive border** of $\text{cl}(\mathcal{F})$
- $Bd^-(\text{cl}(\mathcal{F})) = \{X \subseteq I : X \text{ is infrequent and } 2^X \setminus \{X\} \subseteq \text{cl}(\mathcal{F})\}$
  - i.e., X is infrequent and all proper subsets of X are in $\text{cl}(\mathcal{F})$
  - **negative border** of $\text{cl}(\mathcal{F})$
- $H(\mathcal{F}) = \{I \setminus X : X \in Bd^+(\text{cl}(\mathcal{F}))\}$

  $\Rightarrow$ $\text{Tr}(H(\mathcal{F}))$ is also a hypergraph on I

Borders of Theories and Hypergraph Transversals

Thm.: Let $\mathcal{S}$ be a family of frequent itemsets. Then $\text{Tr}(H(\mathcal{S})) = Bd^-(\text{cl}(\mathcal{S}))$

- folklore; see, e.g., (Mannila & Toivonen, 1997)

Proof:

Step 1. We first show that $X \subseteq I$ is a transversal of $H(\mathcal{S}) \iff X \notin \text{cl}(\mathcal{S})$

$X \subseteq I$ is a transversal of $H(\mathcal{S})$

$\iff$ for every $Y \in H(\mathcal{S})$: $X \cap Y \neq \emptyset$

$\iff$ for every $Z \in Bd^+(\text{cl}(\mathcal{S}))$: $X \cap (I \setminus Z) \neq \emptyset$

$\iff$ for every $Z \in Bd^+(\text{cl}(\mathcal{S}))$: $X \not\subseteq Z$

$\iff$ $X \notin \text{cl}(\mathcal{S})$

Borders of Theories and Hypergraph Transversals

Proof (cont'd) :

Step 1.: $X \subseteq I$ is a transversal of $H(\mathcal{S}) \iff X \notin \mathsf{cl}(\mathcal{S})$ // prev. slide

Step 2.:

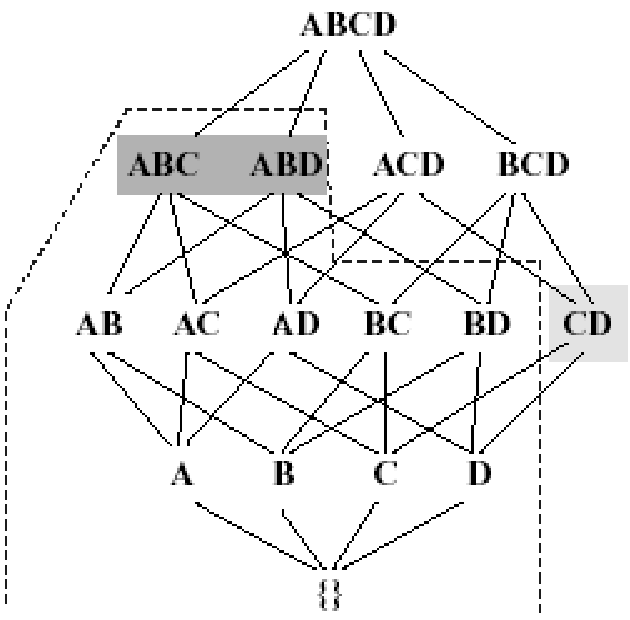
$$
\begin{aligned}
\mathsf{Tr}(H(\mathcal{S})) &= \{X : X \text{ is a minimal transversal of } H(\mathcal{S})\} \\
&= \{X : X \text{ is a minimal set such that } X \notin \mathsf{cl}(\mathcal{S})\} \quad \text{// step 1} \\
&= \{X : X \notin \mathsf{cl}(\mathcal{S}) \text{ and } Y \in \mathsf{cl}(\mathcal{S}) \text{ for every } Y \subsetneq X\} \\
&= Bd^{-}(\mathsf{cl}(\mathcal{S}))
\end{aligned}
$$

q.e.d.

Borders of Theories and Hypergraph Transversals

Example:

- $I = \{a, b, c, d\}$
- $\mathcal{S} = \{abc, abd\}$
- $\mathsf{cl}(\mathcal{S}) = \{abc, abd, ab, ac, ad, bc, bd, a, b, c, d, \emptyset\}$
  - $Bd^{+}(\mathsf{cl}(\mathcal{S})) = \{abc, abd\}$
  - $H(\mathcal{S}) = \{d, c\}$
  - $\mathsf{Tr}(H(\mathcal{S})) = \{cd\}$
  - $Bd^{-}(\mathsf{cl}(\mathcal{S})) = \{cd\}$



$Bd^{-}(\mathsf{cl}(\mathcal{S}))$ can be computed without using $2^{I} \setminus \mathsf{cl}(\mathcal{S})$, which is usually large!

Dualize and Advance Algorithm

idea:

- let $\mathcal{M}$ be the set of *all* maximal frequent itemsets and $\mathcal{S} \subseteq \mathcal{M}$

  $\Rightarrow$ any maximal frequent itemset $X \in \mathcal{M} \setminus \mathcal{S}$ cannot be a subset of any itemset in $\mathcal{S}$

  $\Rightarrow$ for all $Y \in \mathcal{S}$: $X \cap (I \setminus Y) \neq \emptyset$

  $\Rightarrow$ X is a transversal of the hypergraph formed by the complements of the sets in $\mathcal{S}$ // step 1 of the prev. theorem

  1. find a minimal transversal of the above hypergraph that is frequent
  2. extend it to a maximal frequent itemset

  $\Rightarrow$ if all minimal transversals are infrequent then all maximal frequent itemsets have been generated

The Dualize and Advance Algorithm

Input : transaction database $\mathcal{D}$ over set I of items and a frequency threshold

Output: set of all maximal frequent itemsets

1: $i := 1; \mathcal{S}_1 := \emptyset; \overline{\mathcal{S}}_1 := \{I\}$
2: generate a minimal transversal X of $\overline{\mathcal{S}}_i$ // use some listing subroutine
3: **if** no minimal transversal has been generated **then return** $\mathcal{S}_i$ // $\mathcal{S}_i = \mathcal{M}$
4: **if** X is frequent **then**
5:     **forall** $i \in I \setminus X$ **do** // lines 5–6: extend X to a maximal frequent itemset
6:         **if** $X \cup \{i\}$ is frequent **then** $X := X \cup \{i\}$
7:     $\mathcal{S}_{i+1} := \mathcal{S}_i \cup \{X\}$
8:     $\overline{\mathcal{S}}_{i+1} := \{I \setminus Y : Y \in \mathcal{S}_{i+1}\}$
9:     $i := i + 1$
10: **endif**
11: **go to** 2

Dualize and Advance Algorithm

Lemma: For any iteration i of the algorithm, if $\mathcal{S}_i \subsetneq \mathcal{M}$ then at least one of the elements of $\mathsf{Tr}(\overline{\mathcal{S}}_i)$ is frequent.

Proof: suppose $\mathcal{S}_i \subsetneq \mathcal{M}$

$\Rightarrow$ there exists a frequent itemset X such that $X \notin \mathsf{cl}(\mathcal{S}_i)$

$\Rightarrow$ there exists a minimal frequent itemset $X' \subseteq X$ such that $X' \notin \mathsf{cl}(\mathcal{S}_i)$ and all proper subsets of X' are in $\mathsf{cl}(\mathcal{S}_i)$

$\Rightarrow$ $X' \in \mathsf{Bd}^-(\mathsf{cl}(\mathcal{S}_i))$

$\Rightarrow$ $X' \in \mathsf{Tr}(H(\mathcal{S}_i))$ // as $\mathsf{Bd}^-(\mathsf{cl}(\mathcal{S}_i)) = \mathsf{Tr}(H(\mathcal{S}_i))$

$\Rightarrow$ $X' \in \mathsf{Tr}(\overline{\mathcal{S}}_i)$ // because $\mathcal{S}_i \subsetneq \mathcal{M}$

q.e.d.

Dualize and Advance Algorithm

Thm.: The Dualize and Advance algorithm is correct.

Proof:

soundness: Automatic by lines 5–7 of the algorithm.

completeness: By construction, $\mathcal{S}_i \subseteq \mathcal{M}$ for all i. The proof then follows from the previous lemma.

Condensed Representations of Frequent Itemsets I

maximal frequent itemsets

- the Pincer Search algorithm ✓
  - (Lin & Kedem, 2002)
- the Dualize and Advance Algorithm ✓
  - (Gunopulos, Khardon, Mannila, Saluja, Toivonen, & Sharma, 2003)
- complexity of mining maximal frequent itemsets

On the Complexity of Mining Maximal Frequent Itemsets

Theorem (Boros, Gurvich, Khachiyan, & Makino, 2002): Let

- $\mathcal{D}$ be a transactional database over a set I of items with $|I| = n$,
- $t \in \mathbb{N}$ be an absolute frequency threshold, and
- $\mathcal{S} \subseteq \mathcal{M}$ be a family of maximal frequent itemsets of $\mathcal{D}$.

Then it is **NP-hard** to decide if $\mathcal{S} \neq \mathcal{M}$.

Corollary: If P $\neq$ NP then maximal frequent itemsets **cannot** be generated in **output polynomial** time.

On the Complexity of Mining Maximal Frequent Itemsets

Proof: reduction from the NP-complete *independent vertex set* problem

independent vertex set problem: *Given* a graph $G = (V, E)$ and a positive integer t, *decide* if G contains an independent vertex set of size at least t.

– independent vertex set: $V' \subseteq V$ such that no two vertices of V' are connected by an edge

reduction: for G and t, construct a binary matrix (transaction database) $\mathcal{D}$ with $|V|$ columns as follows:

- $\forall u \in V$**:** add 1 row to $\mathcal{D}$ with 0 for the column corresponding to u; 1 for all other columns
- $\forall \{u, v\} \in E$**:** add $t - 2$ identical rows to $\mathcal{D}$ with 0 for the columns corresponding to u and v; 1 for all other columns

On the Complexity of Mining Maximal Frequent Itemsets

Proof (cont'd): $\forall\{u,v\} \in E$: $C_{uv} = V \setminus \{u,v\}$ is maximal t-frequent in $\mathcal{D}$

- let $\mathcal{S} = \{C_{uv} : \{u,v\} \in E\}$
- the theorem follows from the claim below

Claim: $\mathcal{S} \neq \mathcal{M} \iff G$ has an independent set V' of size $|V'| \geq t$.

Proof of the claim:

($\Rightarrow$) $\exists C \in \mathcal{M} \setminus \mathcal{S}$

$\implies C$ cannot be contained by a row introduced for an edge

$\implies V' = V \setminus C$ is an independent set and $|V'| \geq t$

($\Leftarrow$) let V' be an independent set of size t

$\implies V \setminus V'$ is frequent and it cannot be the subset of any member in $\mathcal{S}$

$\implies \mathcal{S} \neq \mathcal{M}$ q.e.d.

On the Complexity of Mining Maximal Frequent Itemsets

Proof of the Corollary: suppose there exists an output-polynomial time algorithm $\mathfrak{A}$ generating all maximal frequent itemsets

$\Rightarrow$ $\exists$ a polynomial $\psi(\cdot, \cdot)$ s.t. $\forall \mathcal{D}$ over n items and $\forall t \in \mathbb{N}$, $\mathfrak{A}$ generates the family $\mathcal{M}$ of all maximal frequent itemsets in time $\psi(\text{size}(\mathcal{D}), |\mathcal{M}|)$

$\Rightarrow$ for any graph G and integer $t > 0$, $\mathfrak{A}$ could be used to decide the independent vertex set problem in **polynomial time** as follows:

1. construct $\mathcal{D}$ and $\mathcal{S}$ for G and t as in the proof of the theorem
2. run $\mathfrak{A}$ on $\mathcal{D}$ with frequency threshold t

   (α) if $\mathfrak{A}$ **terminates** in time $\psi(\text{size}(\mathcal{D}), |\mathcal{S}|)$ with output $\mathcal{M}$ then just check whether $\mathcal{S} = \mathcal{M}$ // claim on the prev. slide

   (β) if $\mathfrak{A}$ does **not** terminate in time $\psi(\text{size}(\mathcal{D}), |\mathcal{S}|)$ then G has an independent vertex set of size t q.e.d.

Maximal Frequent Itemsets: Summary

maximal interesting sentences

- **positive border** of the family of frequent itemsets
- **compact** representation of frequent itemsets
- Pincer search: **bidirectional** search
  - one level up, possibly many levels down
  - good performance in practice
- Dualize and Advance algorithm
  - based on **minimal hypergraph transversals**
  - works in **incremental subexponential** time

☹ listing maximal frequent itemsets is computationally **intractable**

⇨ What about other compact representations of frequent itemsets?

Condensed Representations of Frequent Itemsets II

closed frequent itemsets

- notions and basic properties
- relative cardinalities of maximal frequent, closed frequent, and frequent itemsets
- a divide-and-conquer closed frequent itemset mining algorithm
  - (folklore; see, e.g., Gély, 2005)

Closed Frequent Itemsets: Notions

- I: set of items; $\mathcal{D}$: transaction database over I
  - each transaction in $\mathcal{D}$ has a unique identifier (tid)
  - T: set of all tids
- $it : 2^I \rightarrow 2^T$

  $it(X)$: set of tids of the transactions that contain X as a subset, i.e.,

$$it(X) = \bigcap_{x \in X} it(x)$$

- $ti : 2^T \rightarrow 2^I$

  $ti(Y)$: set of all items common to all the transactions with tids in Y, i.e.,

$$ti(Y) = \bigcap_{y \in Y} ti(y)$$

Closed Frequent Itemsets: Notions

$c : 2^I \to 2^I$ is defined by $c : X \mapsto ti(it(X))$ for every itemset X

Prop: c is a **closure operator**, i.e., for every itemsets X and Y it satisfies

- $X \subseteq c(X)$ (extensivity)
- if $X \subseteq Y$ then $c(X) \subseteq c(Y)$ (monotonicity)
- $c(c(X)) = c(X)$ (idempotency)

Proof: *exercise*

Closed Frequent Itemsets: Notions

$c : 2^I \to 2^I$ is defined by $c : X \mapsto ti(it(X))$ for every itemset X

Def.: An itemset X is

- **closed:** if $c(X) = X$ and
- **closed frequent** if it is closed and frequent

$\mathcal{C}$: family of closed frequent itemsets

Properties:

- X is closed if and only if $|\mathcal{D}[Y]| < |\mathcal{D}[X]|$ for every $Y \supsetneq X$
- all maximal frequent itemsets are closed

Closed Itemsets

Example: let

- $I = \{a, b, c, d, e\}$,
- $T = \{1, 2, 3, 4, 5, 6\}$,
- $\mathcal{D}\{(1, abde), (2, bce), (3, abde), (4, abce), (5, abcde), (6, bcd)\}$

– ae is **not closed** because

$$\begin{aligned} c(ae) &= ti(it(ae)) = ti(it(a) \cap it(e)) = ti(1345 \cap 12345) = ti(1345) \\ &= ti(1) \cap ti(3) \cap ti(4) \cap ti(5) = abde \cap abde \cap abce \cap abcde \\ &= abe \end{aligned}$$

– abe is **closed** because $c(abe) = ti(it(abe)) = ti(1345) = abe$

Closed Frequent Itemsets: Property I

Prop.: for every itemset X, $\mathcal{D}[X] = \mathcal{D}[c(X)]$

- i.e., the support of X is equal to the support of the smallest closed itemset containing X

Proof: *exercise*

Corollary: closed frequent itemsets provide a complete representation of frequent itemsets

- **complete**: support of a frequent itemset can be derived from that of its closure
  - this property does **not** hold for maximal frequent itemsets

algorithm on next slide: generates frequent itemsets with support from closed frequent itemsets without database access

Closed Frequent Itemsets: Property I

Input : $\mathcal{C}$: family of closed frequent itemsets
Output: $\mathcal{F}$: family of frequent itemsets

let $k = 0$ and $\mathcal{F}_i$ be the empty list for every $i \geq 0$
forall closed frequent itemset $C \in \mathcal{C}$ **do**
    append C to $\mathcal{F}_{|C|}$
    if $k < |C|$ **then** $k = |C|$
for $(i = k; i > 1; i = i - 1)$ **do**
    forall itemset $C \in \mathcal{F}_i$ in the order of the elements in $\mathcal{F}_i$ **do**
        forall $(i-1)$-subsets S of C **do**
            if $S \notin \mathcal{F}_{i-1}$ **then**
                $S.support = C.support$
                append S to $\mathcal{F}_{i-1}$
return $\bigcup_{i=1,\ldots,k} \mathcal{F}_i$

Example

database $\mathcal{D} = \{(1, abde), (2, bce), (3, abde), (4, abce), (5, abcde), (6, bcd)\}$

frequency threshold: $t = 4$

closed frequent itemsets: $\{abe, bc, bd, be, b\}$

$\mathcal{F}_3 = [abe_{\underline{4}}],\ \mathcal{F}_2 = [bc_{\underline{4}}, bd_{\underline{4}}, be_{\underline{5}}],\ \mathcal{F}_1 = [b_{\underline{6}}]$

$i = 3$: for $\mathcal{F}_3 = [abe_{\underline{4}}]$ we get

$$\begin{aligned} \mathcal{F}_2 &= \mathcal{F}_2 \oplus ab_{\underline{4}} \oplus ae_{\underline{4}} && \text{// for } abe_{\underline{4}} \\ &= [bc_{\underline{4}}, bd_{\underline{4}}, be_{\underline{5}}, ab_{\underline{4}}, ae_{\underline{4}}] \end{aligned}$$

$i = 2$: for $\mathcal{F}_2 = [bc_{\underline{4}}, bd_{\underline{4}}, be_{\underline{5}}, ab_{\underline{4}}, ae_{\underline{4}}]$ we get

$$\begin{aligned} \mathcal{F}_1 = \mathcal{F}_1 &\oplus [c_{\underline{4}}] && \text{// for } bc_{\underline{4}} \\ &\oplus [d_{\underline{4}}] && \text{// for } bd_{\underline{4}} \\ &\oplus [e_{\underline{5}}] && \text{// for } be_{\underline{5}} \\ &\oplus [a_{\underline{4}}] && \text{// for } ab_{\underline{4}} \end{aligned}$$

return $[abe_{\underline{4}}, bc_{\underline{4}}, bd_{\underline{4}}, be_{\underline{5}}, ab_{\underline{4}}, ae_{\underline{4}}, b_{\underline{6}}, c_{\underline{4}}, d_{\underline{4}}, e_{\underline{5}}, a_{\underline{4}}]$

Condensed Representations of Frequent Itemsets II

closed frequent itemsets

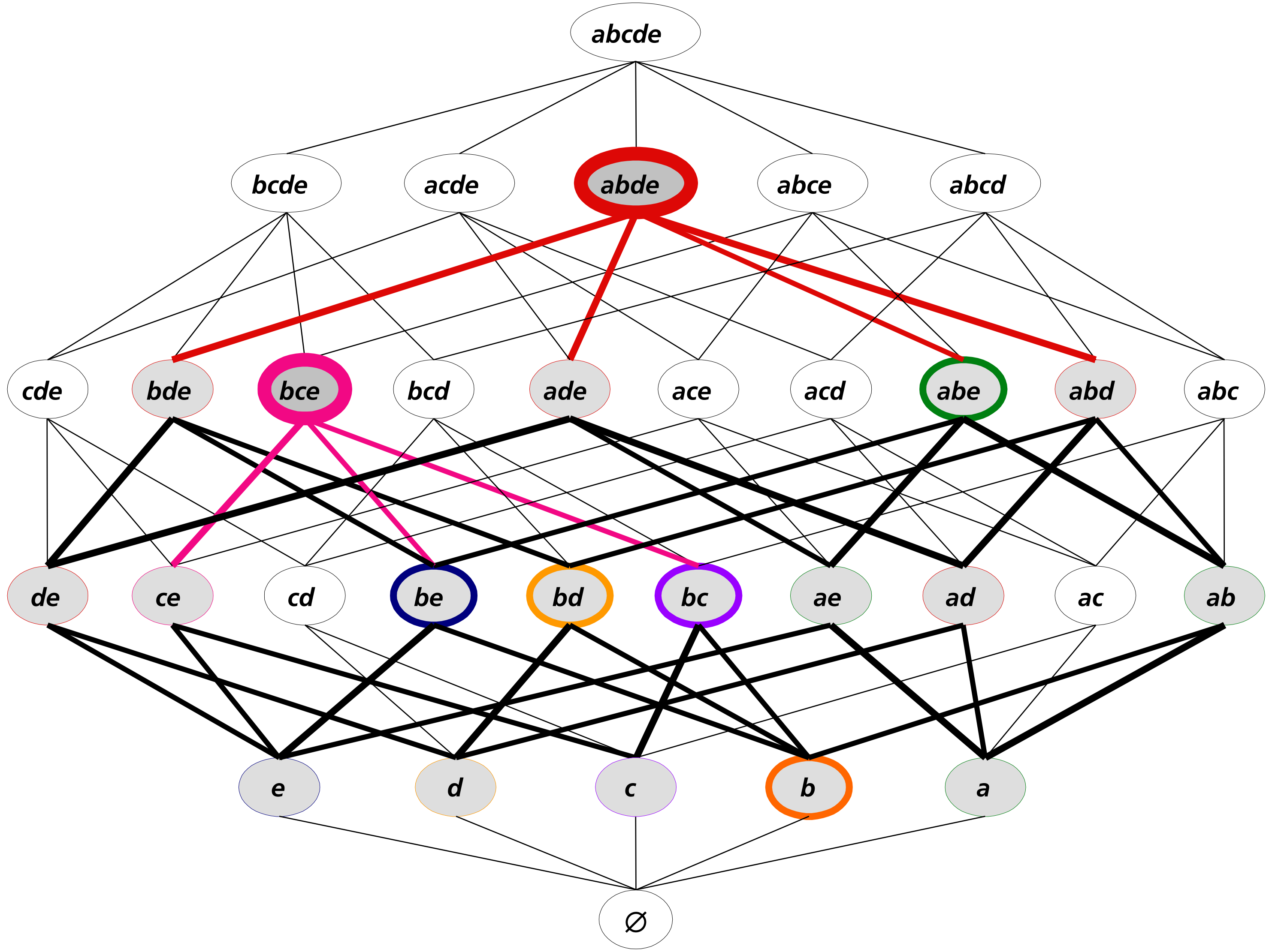
- notions and basic properties ✓
- relative cardinalities of maximal frequent, closed frequent, and frequent itemsets
- a divide-and-conquer closed frequent itemset mining algorithm
  - (folklore; see, e.g., Gély, 2005)

Frequent vs. Closed vs. Maximal Itemsets: Example

Transactions

1. *abde*
2. *bce*
3. *abde*
4. *abce*
5. *abcde*
6. *bcd*

freq. threshold: 3



#frequent: 19
#closed: 7
#maximal: 2

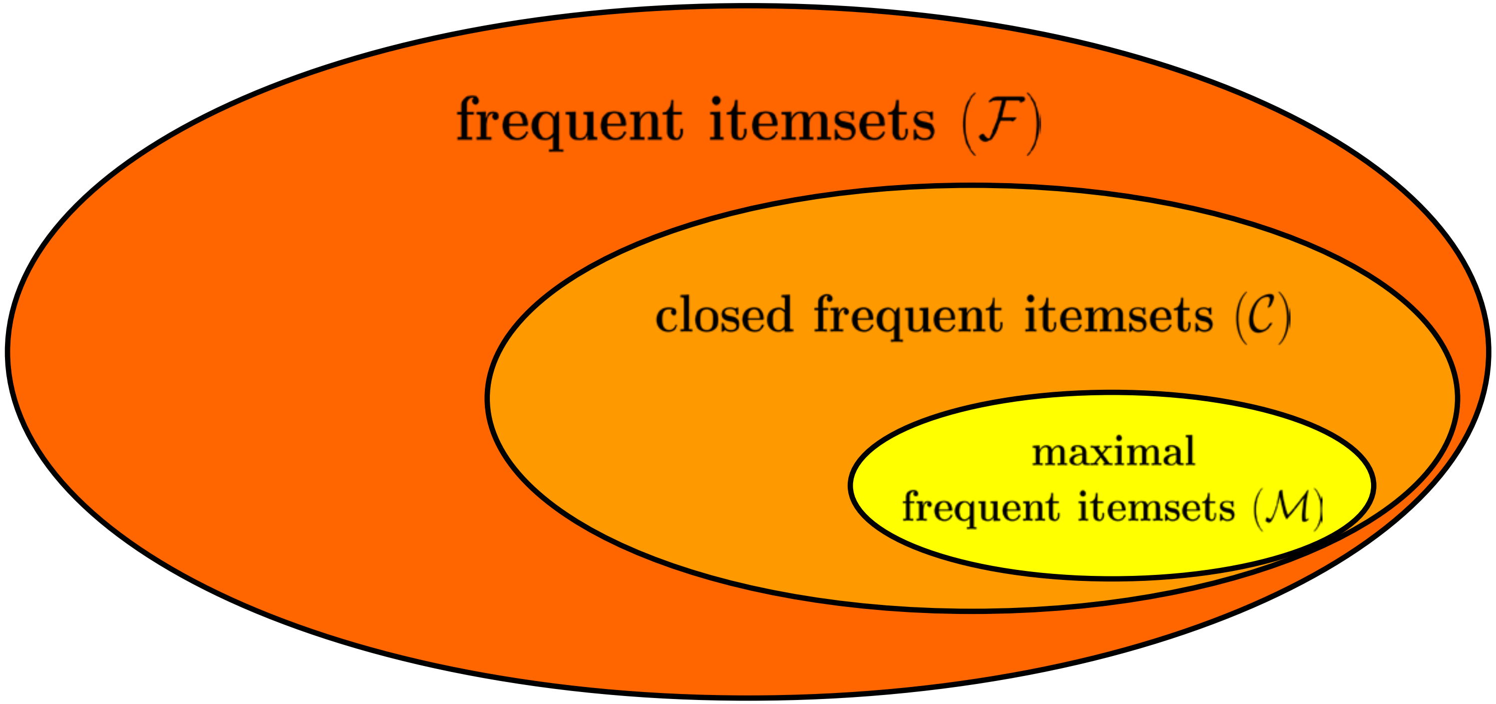
Closed Frequent Itemsets: Property II

Thm. (Boros, Gurvich, Khachiyan, & Makino, 2002):

(i) $|\mathcal{F}|$ can be exponentially larger than $|\mathcal{C}|$ and

(ii) $|\mathcal{C}|$ can be exponentially larger than $|\mathcal{M}|$

$\Rightarrow$ closed frequent itemsets: **compact** representation of frequent itemsets



universitätbonn

RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT

Fraunhofer IAIS

Frequent vs. Closed Freq. vs. Maximal Freq. Itemsets

Thm.: $|\mathcal{F}|$ can be exponentially larger than $|\mathcal{C}|$ and
$|\mathcal{C}|$ can be exponentially larger than $|\mathcal{M}|$

Proof:

$$
\begin{array}{c|ccccc}
 & \overbrace{}^{p} & \overbrace{}^{p} & \overbrace{}^{p} & & \overbrace{}^{p} \; \overbrace{}^{p} \\
t \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. & \begin{array}{c} 0 \ldots 0 \\ \vdots \\ 0 \ldots 0 \end{array} & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \cdots & \begin{array}{cc} 1 \ldots 1 & 1 \ldots 1 \\ \vdots & \vdots \\ 1 \ldots 1 & 1 \ldots 1 \end{array} \\
t \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \begin{array}{c} 0 \ldots 0 \\ \vdots \\ 0 \ldots 0 \end{array} & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \cdots & \begin{array}{cc} 1 \ldots 1 & 1 \ldots 1 \\ \vdots & \vdots \\ 1 \ldots 1 & 1 \ldots 1 \end{array} \\
t \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \begin{array}{c} 0 \ldots 0 \\ \vdots \\ 0 \ldots 0 \end{array} & \cdots & \begin{array}{cc} 1 \ldots 1 & 1 \ldots 1 \\ \vdots & \vdots \\ 1 \ldots 1 & 1 \ldots 1 \end{array} \\
 & & & & \vdots & \\
t \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \begin{array}{c} 1 \ldots 1 \\ \vdots \\ 1 \ldots 1 \end{array} & \cdots & \begin{array}{cc} 1 \ldots 1 & 0 \ldots 0 \\ \vdots & \vdots \\ 1 \ldots 1 & 0 \ldots 0 \end{array}
\end{array}
$$

$k \cdot t \times k \cdot p$ binary matrix

- $k \times k$ blocks
- each block of size $t \times p$

1. $|\mathcal{M}| = k$
2. $|\mathcal{C}| = 2^k - 1$
3. $|\mathcal{F}| > 2^{(k-1)p} > \left(\frac{|\mathcal{C}|}{2}\right)^p$

q.e.d.

Condensed Representations of Frequent Itemsets II

closed frequent itemsets

- notions and basic properties ✓
- relative cardinalities of maximal frequent, closed frequent, and frequent itemsets ✓
- **a divide-and-conquer closed frequent itemset mining algorithm**
  - (folklore; see, e.g., Gély, 2005)

Computing Closed Frequent Itemsets with DF-Search

Problem: Given I, $\mathcal{D}$, and frequency threshold t, compute $\mathcal{C}$

Algorithm: (Gély, 2005; also other authors)

- compute first all closed frequent itemsets containing an item a,
- then all closed frequent itemsets which do not contain a
- apply recursively...

divide and conquer algorithm

Algorithm

Input : I with some total order $\leq$, $\mathcal{D}$, and frequency threshold t
Output : all closed frequent itemsets

Initial Call : $\textsc{ListClosed}(\emptyset, \emptyset, \min I)$

function $\textsc{ListClosed}(C, N, i)$ // $C, N \subseteq I, i \in I$

```
X := {k ∈ I \ C : k ≥ i}
if X ≠ ∅ then
    i' = min X
    C' = c(C ∪ {i'})
    if C' is frequent and C' ∩ N = ∅ then
        print C'
        ListClosed(C', N, i' + 1)
    Y := {k ∈ I \ C : k > i}
    if Y ≠ ∅ then
        i'' = min Y
        ListClosed(C, N ∪ {i'}, i'')
```

Algorithm

Thm.: The previous algorithm lists the set of closed frequent itemsets

(1) correctly,

(2) irredundantly,

(3) with polynomial delay, and

(4) in polynomial space.

Proof: *(exercise)*

Example

1. abde
2. bce
3. abde
4. abce
5. abcde
6. bcd

t = 3

$a<b<c<d<e$

- *ListClosed*(∅, ∅, a)
  - **print** c(a) = **abe** (frequent)
  - *ListClosed*(abe, ∅, c)
    - c(abce) = **abce** (infrequent)
    - *ListClosed*(abe, {c}, d)
      - **print** c(abde) = **abde** (frequent)
  - *ListClosed*(∅, {a}, b)
    - **print** c(b) = **b** (frequent)
    - *ListClosed*(b, {a}, c)
      - **print** c(bc) = **bc** (frequent)
      - *ListClosed*(bc, {a}, d)
        - c(bcd) = **bcd** (infrequent)
        - *ListClosed*(bc, {a,d}, e)
          - **print** c(bce) = **bce** (frequent)
      - *ListClosed*(b, {a,c}, d)
        - **print** c(bd) = **bd** (frequent)
        - *ListClosed*(bd, {a,c}, e)
          - c(bde) = **abde** (contains a)
        - *ListClosed*(b, {a,c,d}, e)
          - **print** c(be) = **be** (frequent)

Closed Frequent Itemsets: Summary

- another compact representation
- usually exponentially smaller than the set of frequent itemsets but exponentially larger then the set of maximal frequent itemsets
- divide and conqure: polynomial delay and polynomial space
- closure operators: also in other theory extraction problems
  - formal concept analysis
  - enumeration of maximal bipartite cliques of a bipartite graph

Literature to the lectures about Association Rules (I-V)

- J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed., Morgan Kaufmann, 2011.
- I. Witten and E. Frank, *Data Mining*, Morgan Kaufmann, 2000.
- R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo: *Fast Discovery of Association Rules*. In U.M. Fayyad et al. (Eds.), Advances in Knowledge Discovery and Data Mining, 307-328, AAAI/MIT Press, 1996.
- J. Han, J. Pei, Y. Yin, R. Mao: *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*. Data Mining and Knowledge Discovery 8(1): 53-87, 2004.
- D.-I. Lin, Z.M. Kedem: Pincer-Search: *An Efficient Algorithm for Discovering the Maximum Frequent Set*. IEEE Trans. Knowl. Data Eng. 14(3): 553-566, 2002.
- D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, R.S. Sharm: *Discovering all most specific sentences*. ACM Trans. Database Syst. 28(2):140-174, 2003.
- E. Boros, V. Gurvich, L. Khachiyan, K. Makino: *On Maximal Frequent and Minimal Infrequent Sets in Binary Matrices*. Ann. Math. Artif. Intell. 39(3): 211-221, 2003.
- N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: *Efficient Mining of Association Rules Using Closed Itemset Lattices*. Inf. Syst. 24(1): 25-46, 1999.
- A. Gély: *A Generic Algorithm for Generating Closed Sets of a Binary Relation*. In Proc. of the 3rd Int. Conference on Formal Concept Analysis (ICFCA 2005), LNCS 3403, pp. 223-234, Springer-Verlag, 2005.