Chapter 4: Frequent Itemsets and Association Rules

Jilles Vreeken

Revision 1, November 9th
small typo fixed

IRDM '15/16

3 Nov 2015

UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

Question of the week

How can we mine **interesting patterns** and **useful rules** from data?

Motivational Example

You run an on-line store, and want to increase sales.
You decide on **associative advertising**: show ads of relevant products **before** your users search for these

→

Easy, knowing the left-hand side. What if we don't?

IRDM Chapter 4, overview

1. Definitions
2. Algorithms for Frequent Itemset Mining
3. Association Rules and Interestingness
4. Summarising Itemset Collections

You'll find this covered in
Aggarwal Chapter 4, 5.2
Zaki & Meira, Ch. 10, 11

Chapter IV.1: Definitions

Transaction data model

The data type considered in **itemset mining** is called **transaction data**.

Let $\mathcal{I}$ be a set of items,

e.g. the products for sale in a shop.

A **transaction** $t \in \mathcal{P}(\mathcal{I})$, or, $t \subseteq \mathcal{I}$, is a set of items

e.g. representing the items a customer bought.

A **dataset** $\boldsymbol{D}$ is a bag of transactions,

e.g. the different sale transactions on a given day.

Market Basket Data

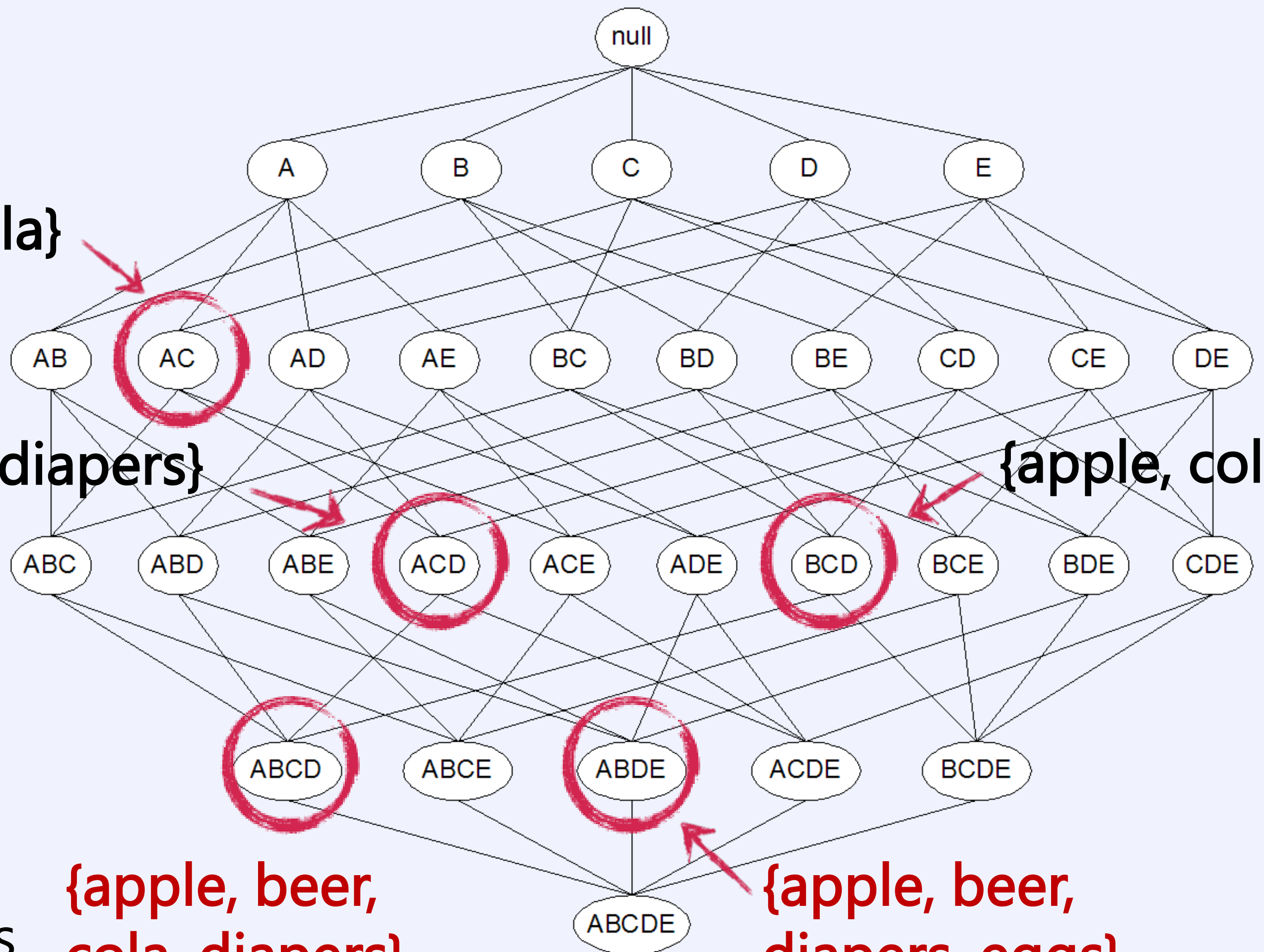
Items for sale: $\mathcal{I} = \{\text{apple}, \text{beer}, \text{cola}, \text{diapers}, \text{eggs}\}$

Transactions: $1{:}\,\{\text{apple}, \text{cola}\}$, $2{:}\,\{\text{apple}, \text{beer}, \text{diapers}, \text{eggs}\}$,
$3{:}\,\{\text{cola}, \text{beer}, \text{diapers}\}$, $4{:}\,\{\text{apple}, \text{beer}, \text{cola}, \text{diapers}\}$,
$5{:}\,\{\text{apple}, \text{cola}, \text{diapers}\}$

Transaction IDs

TID	Apple	Beer	Cola	Diapers	Eggs
1	✓		✓		
2	✓	✓		✓	✓
3		✓	✓	✓	
4	✓	✓	✓	✓	
5	✓		✓	✓	

Transaction data as subsets



a: apples
b: beer
c: cola
d: diapers
e: eggs

2^m subsets of m items. Layer k has $\binom{m}{k}$ subsets.

Transaction data as a binary matrix

TID	Apple	Beer	Cola	Diapers	Eggs
1	1	0	1	0	0
2	1	1	0	1	1
3	0	1	1	1	0
4	1	1	1	1	0
5	1	0	1	1	0

Any data that can be represented as a binary matrix can be used

Itemsets, support, and frequency

An **itemset** is a set of items, e.g. $X \subseteq \mathcal{I}$

- a transaction $t = (\mathrm{tid}, X)$ **contains** itemset Y if $Y \subseteq X$
- the **support** of itemset X in database $\boldsymbol{D}$ is the number of transactions in $\boldsymbol{D}$ that contain it,
$$supp(X, \boldsymbol{D}) = |\{t \in \boldsymbol{D} : t \text{ contains } X\}|$$
- the **frequency** of itemset X in database $\boldsymbol{D}$ is its relative support,
$$freq(X, \boldsymbol{D}) = \frac{supp(X, \boldsymbol{D})}{|\boldsymbol{D}|}$$

An itemset X is said to be **frequent** if its frequency is above a user-defined threshold σ.

  - people often exchange the meaning of frequency and support

Frequent itemset example

TID	Apple	Beer	Cola	Diapers	Eggs
1	1	0	1	0	0
2	1	1	0	1	1
3	0	1	1	1	0
4	1	1	1	1	0
5	1	0	1	1	0

Itemset $\{\text{apple}, \text{cola}\}$ has support 3 and frequency 3/5

Itemset $\{\text{apple}, \text{cola}, \text{eggs}\}$ has support and frequency 0

For $minfreq = \frac{1}{2}$, the frequent itemsets are:

$\{\text{apple}\}, \{\text{cola}\}, \{\text{diapers}\}, \{\text{beer}\}, \{\text{apple}, \text{cola}\},$
$\{\text{apple}, \text{diapers}\}, \{\text{cola}, \text{diapers}\}$, and $\{\text{diapers}, \text{beer}\}$

Association Rules and Confidence

An **association rule** is a rule of type $X \to Y$ where X and Y are disjoint itemsets ($X \cap Y = \emptyset$)

- "if a transaction supports X it likely also supports Y"

The support of rule $X \to Y$ in data $\boldsymbol{D}$ is

$$supp(X \to Y, \boldsymbol{D}) = supp(X \cup Y, \boldsymbol{D})$$

The **confidence** of a rule $X \to Y$ in data $\boldsymbol{D}$ is

$$conf(X \to Y, \boldsymbol{D}) = supp(X \cup Y, \boldsymbol{D}) / supp(X, \boldsymbol{D})$$

- confidence is the **empirical conditional probability** that a transaction t supporting itemset X also contains itemset Y

Association rule example

TID	Apple	Beer	Cola	Diapers	Eggs
1	1	0	1	0	0
2	1	1	0	1	1
3	0	1	1	1	0
4	1	1	1	1	0
5	1	0	1	1	0

$\{\text{apple}, \text{cola}\} \to \{\text{diapers}\}$ has support 2 and frequency 2/3

$\{\text{diapers}\} \to \{\text{apple}, \text{cola}\}$ has support 2 and frequency 1/2

$\{\text{eggs}\} \to \{\text{apple}, \text{beer}, \text{diapers}\}$ has support 1 and frequency 1

Applications

Frequent itemset mining

- which items often appear together?
  - what products do people buy together?
  - which pages of a website people often see in one visit?
  - which genes are often co-activated?
- later we'll learn better concepts for this

Association rule mining

- implication analysis: if X is bought/observed what else will probably be bought/observed?
  - if people who buy milk and cereal also buy bananas, we can locate bananas close to milk or cereal to improve sales
  - if people who search for swimsuits and cameras also search for holidays, we should show holiday advertisements to those who search for swimsuits and cameras

Chapter IV.2: **Algorithms**

The Naïve Algorithm

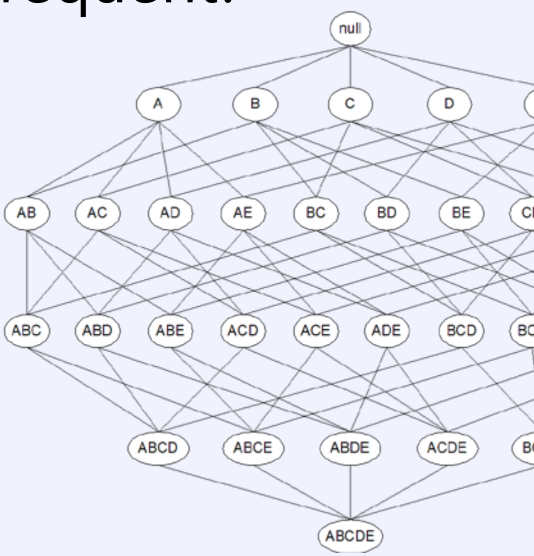
The Apriori Algorithm

Improving Apriori: Eclat

The FP-Growth Algorithm

The Naïve Algorithm

Try every possible itemset and check if it is frequent!



How to try the itemsets?

- breadth-first or depth-first in subset lattice

How to compute the support?

- check for every transaction is the itemset included

Time complexity

- computing the support of an itemset takes $O(|I| \times |\boldsymbol{D}|)$, and there are $2^{|I|}$ possible itemsets, so worst-case complexity is $O(|I| \times |D| \times 2^{|I|})$
- I/O complexity is $O(2^{|I|})$ database accesses

The Apriori Algorithm

The **downward closure** of support:

- if X and Y are itemsets s.t. $X \subseteq Y$ then $supp(X) \geq supp(Y)$
- in other words, if X is infrequent, so are **all its supersets**

The Apriori algorithm uses this to prune the search space

- Apriori never generates a candidate that has an infrequent subset

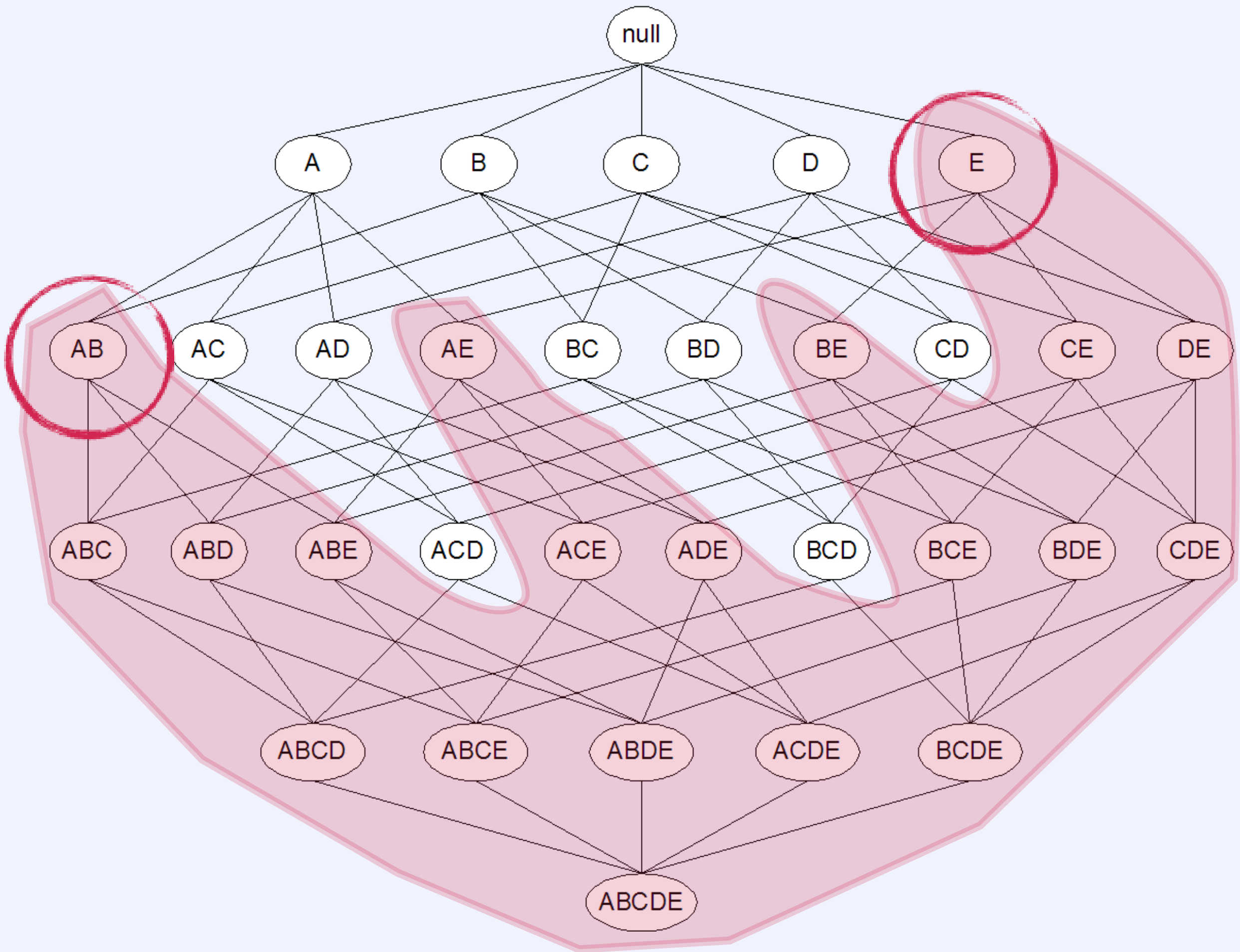
Worst-case time complexity is still $O\left(|\mathcal{I}| \times |\boldsymbol{D}| \times 2^{|\mathcal{I}|}\right)$

- in practice, it can be much much less

(Agrawal & Srikant, 1994, **18k cites**; Mannila, Toivonen & Verkamo, 1994, **1k cites**; Agrawal, Mannila, Srikant, Toivonen & Verkamo, 1996, **3k cites**)

Apriori pruning

What happens when {e} and {ab} are infrequent?



Improving I/O

The Naïve algorithm computes the frequency of every candidate itemset **indendepently**

- exponential number of database scans

It's much smarter to **loop over the transactions**:

- collect all candidate k-itemsets
- iterate over every transaction
  - for every k-subitemset of the transaction, if it is a candidate, increase the candidate's support by 1

Now we need to sweep over the data **only once per level** (!)

- at most $O(|\mathcal{I}|)$ database scans

Example of Apriori – blackboard

	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0
$\sum$	**4**	**6**	**4**	**4**	**5**

Improving over Apriori: Eclat

In Apriori, the support computation requires creating all k-subitemsets of all transactions

- many of them might not be in the candidate set

Way to speed up things: index the database so that we compute the support directly

- a **tidset** of itemset X, $t(X)$, is the set of transaction IDs of $\boldsymbol{D}$ that contain X, i.e. $t(X) = \{tid : (tid, Y) \in \boldsymbol{D} \text{ with } X \subseteq \mathrm{Y}\}$
  - $supp(X) = |t(X)|$
  - $t(XY) = t(X) \cap t(Y)$
    - XY is shorthand notation for $X \cup Y$

We can compute support by **intersecting** tidsets, and counting the cardinality of such an intersection.

(revised on November 9th , there was a wild \ that needed to be caught)

The Eclat algorithm

The **Eclat** algorithm uses tidsets to compute support

A **prefix equivalence class** (PEC) is
a set of all itemsets that share the same prefix

- we assume some (arbitrary) order on the items
- e.g. all itemsets that contain items A and B

Eclat merges two itemsets from the same PEC and intersects their tidsets to compute support

- if the result is frequent, it is moved down to a PEC with prefix matching the first itemset

Eclat traverses the prefix tree in a DFS-like manner

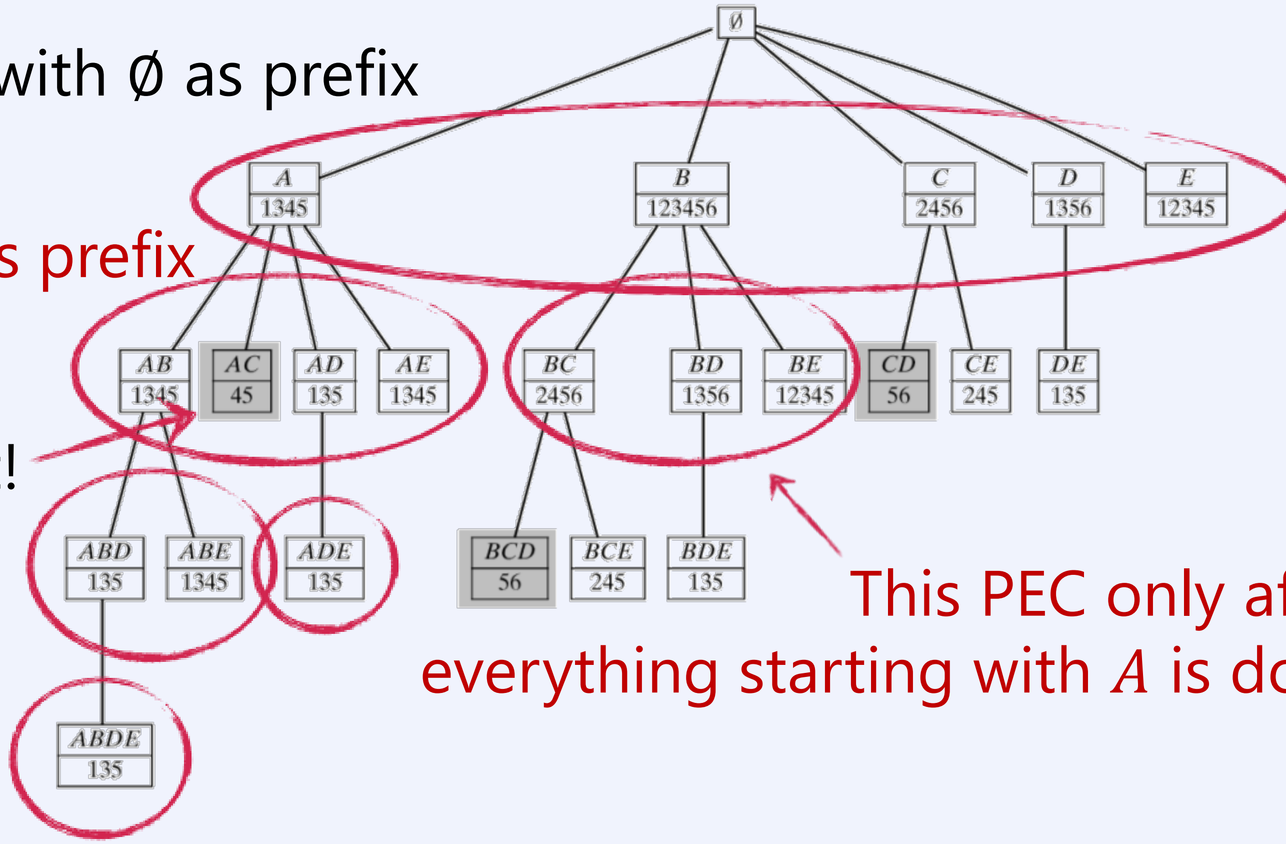
(Zaki et al. 1997, >1k citations)

Eclat in Action

First PEC with Ø as prefix

2nd PEC with A as prefix

Infrequent!



This PEC only after everything starting with A is done

(Figure 8.5 in Zaki & Meira)

dEclat: differences of tidsets

Long tidsets slow down Eclat

A **diffset** stores the differences of the tidsets

- the diffset of ABC, $d(ABC)$ is $t(AB) \setminus t(ABC)$
  - i.e. all tids that contain the prefix AB but **not** ABC

Updates: $d(ABC) = d(C) \setminus d(AB)$

Support: $supp(ABC) = supp(AB) - |d(ABC)|$

We can replace tidsets with diffsets if they are shorter

- this replacement can happen at any move to a new PEC

(Gouda & Zaki, 2003, 500+ cites)

The FP-Growth algorithm

The **FP-Growth** algorithm is the most widely-used algorithm for mining frequent itemsets

- it preprocesses the data to build an **FP-tree** data structure
- itemsets are then mined using this data structure

An **FP-tree** is a condensed representation of the data

- the smaller, the more efficient the mining

It looks very different but is intrinsically similar to Eclat.

(Han et al, 2000, 6000+ citations)

Building an FP-tree

Initially the tree contains the empty set as a root

For each transaction, we will add a branch that contains one node for each item in the transaction

- if a prefix of the transaction is already in the tree, we increase the counts of these nodes, and add only the suffix (with count 1)
- every transaction is now in a path from the root to a leaf
  - transactions that are proper subsets of others do not reach the leaf

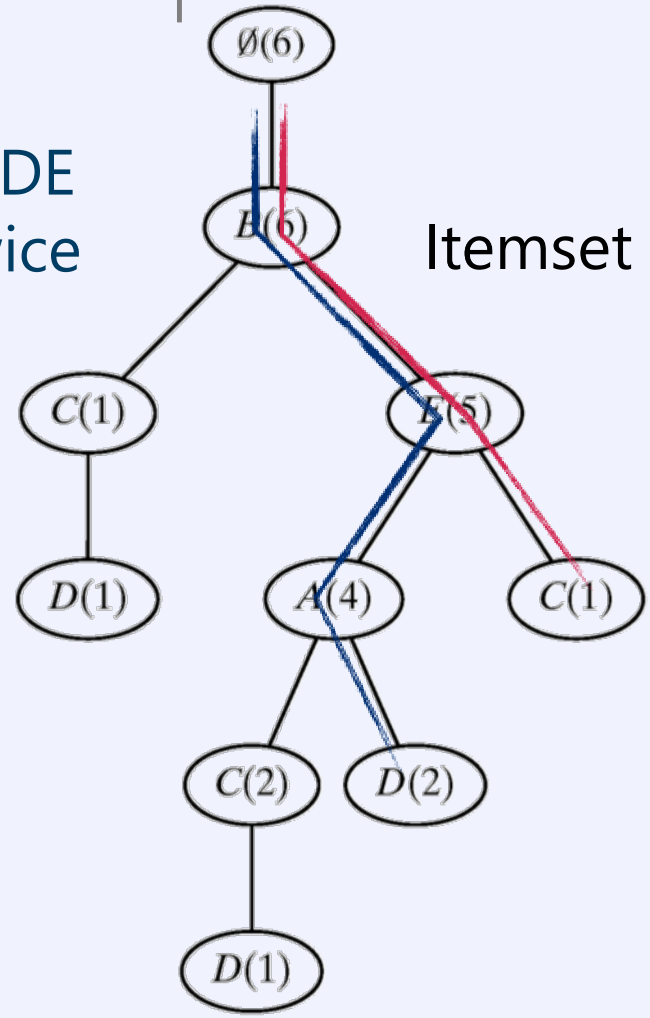
Items in transactions are added in decreasing order of support

- goal: as small as tree as possible

FP-tree example

Itemset ABDE appears twice

Itemset BCE



(Figure 8.9 of Zaki & Meira)

Mining frequent itemsets

To mine itemsets, we **project** the FP-tree onto a prefix

- initially these contain single items in increasing order of support
- the result is another FP-tree

If the projected tree is a path, we add all subsets of nodes together with the prefix as frequent itemsets

- the support is the smallest count
- if the projected tree is not a path, we call FP-growth recursively

How to project?

To project tree T to item i we first find all occurrences of i from T

- for each occurrence, find the path from root to node
- copy this path to the projected tree without the node corresponding to i
- increase the count of **every node** in the copied path by the count of the node corresponding to i

Item i is added to the prefix

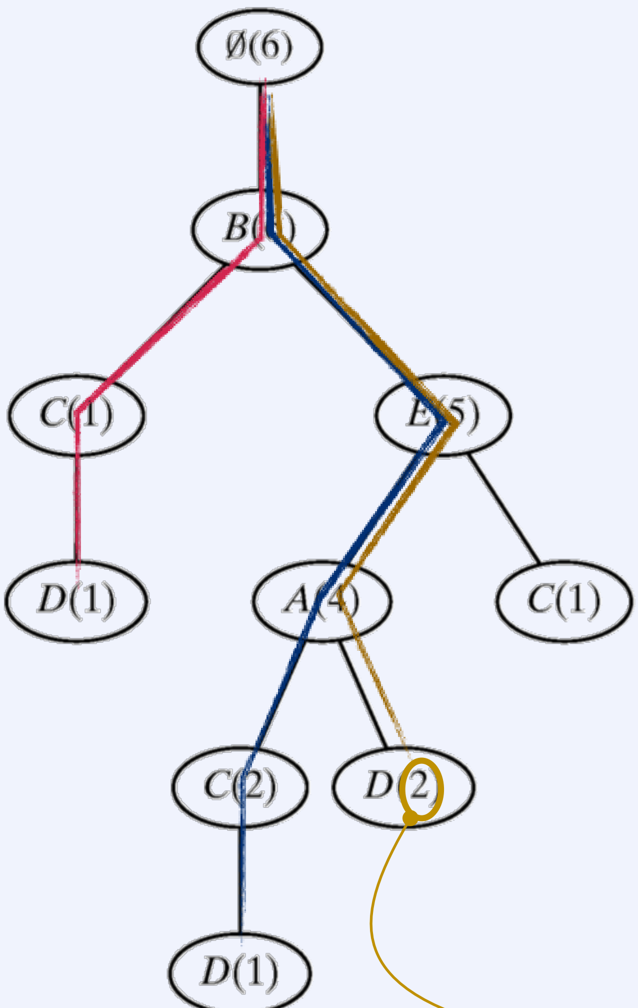
Remove nodes of elements with support $\leq$ **minsup**

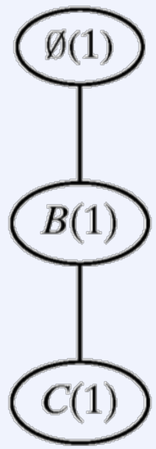
- the support of an element is the sum of counts in its corresponding nodes

If the resulting tree is a path, list the frequent itemsets

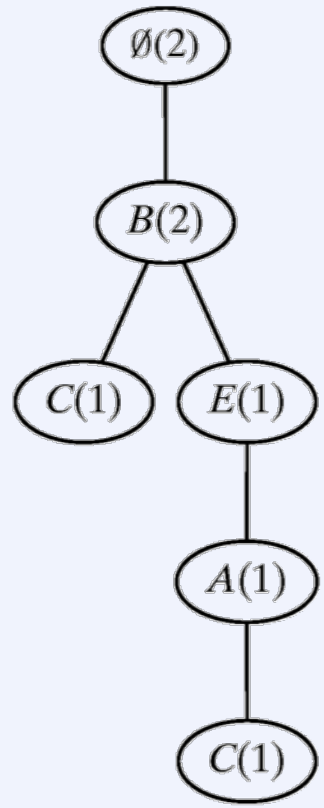
- else, add all itemsets with current prefix and any single item from the tree, and call FP-Growth recursively

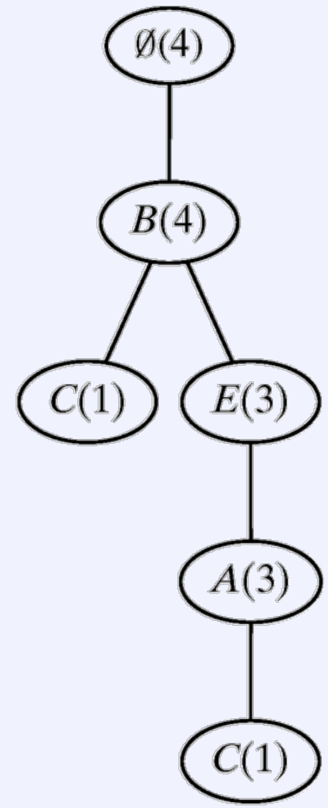
Example of projection





Add **BCD**
count = 1



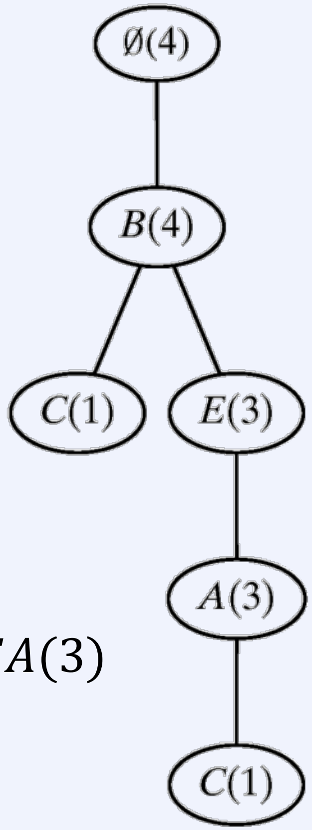


Add **BEACD**
count = 1

Add **BEAD**
count = 2

(from Fig 8.8 & 8.9 of Zaki & Meira)

Example of mining frequent itemsets



The tree projected onto prefix D

Nodes with C are infrequent

- can be removed

The result is a path, so the frequent itemsets are all subsets of nodes with prefix D

- their support is the smallest count
- $DB(4), DE(3), DA(3), DBE(3), DBA(3), DEA(3)$ and $DBEA(3)$

Similar process is done to other prefixes, possibly with recursive calls

(from Fig. 8.8 of Zaki & Meira)

An oldie but a goodie

Apriori is much faster than the naïve algorithm.

- it is not, however, the most efficient algorithm.

Eclat and FP-growth use tricks to speed-up counting.

- i.e. projection and smart data structures
- these tricks work only if **all data fits in memory**.

As Apriori limits the I/O operations to $O(|\mathcal{I}|)$ it **is the fastest** of the three when data does not fit in memory.

Conclusions

Transaction data

- co-occurrence data, any binary table or matrix can be considered.

Frequent itemsets

- those itemsets that occur more often in $\boldsymbol{D}$ than $minsup$ times

Mining frequent itemsets

- exponential output space
- Apriori prunes infrequent candidates by monotonicity
- Eclat considers tidlists to reduce number of database passes
- FP-growth considers prefix trees

Thank you!

Transaction data

- co-occurrence data, any binary table or matrix can be considered.

Frequent itemsets

- those itemsets that occur more often in $\boldsymbol{D}$ than $minsup$ times

Mining frequent itemsets

- exponential output space
- Apriori prunes infrequent candidates by monotonicity
- Eclat considers tidlists to reduce number of database passes
- FP-growth considers prefix trees