# Data Mining and Machine Learning: Fundamental Concepts and Algorithms

datamininingbook.info

Mohammed J. Zaki[1] Wagner Meira Jr.[2]

[1]Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY, USA

[2]Department of Computer Science
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

# Chapter 8: Itemset Mining

## Frequent Itemset Mining

In many applications one is interested in how often two or more objects of interest co-occur, the so-called *itemsets*.

The prototypical application was *market basket analysis*, that is, to mine the sets of items that are frequently bought together at a supermarket by analyzing the customer shopping carts (the so-called "market baskets").

Frequent itemset mining is a basic exploratory mining task, since the since the basic operation is to find the co-occurrence, which gives an estimate for the joint probability mass function.

Once we mine the frequent sets, they allow us to extract *association rules* among the itemsets, where we make some statement about how likely are two sets of items to co-occur or to conditionally occur.

## Frequent Itemsets: Terminology

**Itemsets:** Let $\mathcal{I} = \{x_1, x_2, \ldots, x_m\}$ be a set of elements called *items*. A set $X \subseteq \mathcal{I}$ is called an *itemset*. An itemset of cardinality (or size) $k$ is called a $k$-itemset. Further, we denote by $\mathcal{I}^{(k)}$ the set of all $k$-itemsets, that is, subsets of $\mathcal{I}$ with size $k$.

**Tidsets:** Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be another set of elements called transaction identifiers or *tids*. A set $T \subseteq \mathcal{T}$ is called a *tidset*. Itemsets and tidsets are kept sorted in lexicographic order.

**Transactions:** A *transaction* is a tuple of the form $\langle t, X \rangle$, where $t \in \mathcal{T}$ is a unique transaction identifier, and $X$ is an itemset.

**Database:** A binary database $\boldsymbol{D}$ is a binary relation on the set of tids and items, that is, $\boldsymbol{D} \subseteq \mathcal{T} \times \mathcal{I}$. We say that tid $t \in \mathcal{T}$ *contains* item $x \in \mathcal{I}$ iff $(t, x) \in \boldsymbol{D}$. In other words, $(t, x) \in \boldsymbol{D}$ iff $x \in X$ in the tuple $\langle t, X \rangle$. We say that tid $t$ *contains* itemset $X = \{x_1, x_2, \ldots, x_k\}$ iff $(t, x_i) \in \boldsymbol{D}$ for all $i = 1, 2, \ldots, k$.

## Database Representation

Let $2^X$ denote the powerset of $X$, that is, the set of all subsets of $X$. Let $\boldsymbol{i} : 2^{\mathcal{T}} \rightarrow 2^{\mathcal{I}}$ be a function, defined as follows:

$$\boldsymbol{i}(T) = \{x \mid \forall t \in T,\ t \text{ contains } x\}$$

where $T \subseteq \mathcal{T}$, and $\boldsymbol{i}(T)$ is the set of items that are common to *all* the transactions in the tidset $T$. In particular, $\boldsymbol{i}(t)$ is the set of items contained in tid $t \in \mathcal{T}$.

Let $\boldsymbol{t} : 2^{\mathcal{I}} \rightarrow 2^{\mathcal{T}}$ be a function, defined as follows:

$$\boldsymbol{t}(X) = \{t \mid \ t \in \mathcal{T} \text{ and } t \text{ contains } X\} \tag{1}$$

where $X \subseteq \mathcal{I}$, and $\boldsymbol{t}(X)$ is the set of tids that contain *all* the items in the itemset $X$. In particular, $\boldsymbol{t}(x)$ is the set of tids that contain the single item $x \in \mathcal{I}$.

The binary database $\boldsymbol{D}$ can be represented as a *horizontal* or *transaction database* consisting of tuples of the form $\langle t, \boldsymbol{i}(t) \rangle$, with $t \in \mathcal{T}$.

The binary database $\boldsymbol{D}$ can also be represented as a *vertical* or *tidset database* containing a collection of tuples of the form $\langle x, \boldsymbol{t}(x) \rangle$, with $x \in \mathcal{I}$.

## Binary Database: Transaction and Vertical Format

| $\boldsymbol{D}$ | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| 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 |

Binary Database

| $t$ | $\boldsymbol{i}(t)$ |
|---|---|
| 1 | $ABDE$ |
| 2 | $BCE$ |
| 3 | $ABDE$ |
| 4 | $ABCE$ |
| 5 | $ABCDE$ |
| 6 | $BCD$ |

Transaction Database

| $\boldsymbol{t}(x)$ | | | | |
|---|---|---|---|---|
| $A$ | $B$ | $C$ | $D$ | $E$ |
| 1 | 1 | 2 | 1 | 1 |
| 3 | 2 | 4 | 3 | 2 |
| 4 | 3 | 5 | 5 | 3 |
| 5 | 4 | 6 | 6 | 4 |
| | 5 | | | 5 |
| | 6 | | | |

Vertical Database

This dataset $\boldsymbol{D}$ has 5 items, $\mathcal{I} = \{A, B, C, D, E\}$ and 6 tids $\mathcal{T} = \{1, 2, 3, 4, 5, 6\}$.

The the first transaction is $\langle 1, \{A, B, D, E\}\rangle$, where we omit item $C$ since $(1, C) \notin \boldsymbol{D}$. Henceforth, for convenience, we drop the set notation for itemsets and tidsets. Thus, we write $\langle 1, \{A, B, D, E\}\rangle$ as $\langle 1, ABDE\rangle$.

## Support and Frequent Itemsets

The *support* of an itemset $X$ in a dataset $\boldsymbol{D}$, denoted $sup(X)$, is the number of transactions in $\boldsymbol{D}$ that contain $X$:

$$\boxed{sup(X) = \left|\{t \mid \langle t, \boldsymbol{i}(t)\rangle \in \boldsymbol{D} \text{ and } X \subseteq \boldsymbol{i}(t)\}\right| = |\boldsymbol{t}(X)|}$$

The *relative support* of $X$ is the fraction of transactions that contain $X$:

$$\boxed{rsup(X) = \frac{sup(X)}{|\boldsymbol{D}|}}$$

It is an estimate of the *joint probability* of the items comprising $X$.

An itemset $X$ is said to be *frequent* in $\boldsymbol{D}$ if $sup(X) \geq minsup$, where *minsup* is a user defined *minimum support threshold*.

The set $\mathcal{F}$ to denotes the set of all frequent itemsets, and $\mathcal{F}^{(k)}$ denotes the set of frequent $k$-itemsets.

## Frequent Itemsets

Minimum support: $minsup = 3$

| $t$ | $\boldsymbol{i}(t)$ |
|---|---|
| 1 | $ABDE$ |
| 2 | $BCE$ |
| 3 | $ABDE$ |
| 4 | $ABCE$ |
| 5 | $ABCDE$ |
| 6 | $BCD$ |

Transaction Database

| $sup$ | itemsets |
|---|---|
| 6 | $B$ |
| 5 | $E, BE$ |
| 4 | $A, C, D, AB, AE, BC, BD, ABE$ |
| 3 | $AD, CE, DE, ABD, ADE, BCE, BDE, ABDE$ |

Frequent Itemsets

The 19 frequent itemsets shown in the table comprise the set $\mathcal{F}$. The sets of all frequent $k$-itemsets are

$$\mathcal{F}^{(1)} = \{A, B, C, D, E\}$$
$$\mathcal{F}^{(2)} = \{AB, AD, AE, BC, BD, BE, CE, DE\}$$
$$\mathcal{F}^{(3)} = \{ABD, ABE, ADE, BCE, BDE\}$$
$$\mathcal{F}^{(4)} = \{ABDE\}$$

## Itemset Mining Algorithms: Brute Force

The brute-force algorithm enumerates all the possible itemsets $X \subseteq \mathcal{I}$, and for each such subset determines its support in the input dataset $\boldsymbol{D}$. The method comprises two main steps: (1) candidate generation and (2) support computation.

**Candidate Generation:** This step generates all the subsets of $\mathcal{I}$, which are called *candidates*, as each itemset is potentially a candidate frequent pattern. The candidate itemset search space is clearly exponential because there are $2^{|\mathcal{I}|}$ potentially frequent itemsets.

**Support Computation:** This step computes the support of each candidate pattern $X$ and determines if it is frequent. For each transaction $\langle t, \boldsymbol{i}(t)\rangle$ in the database, we determine if $X$ is a subset of $\boldsymbol{i}(t)$. If so, we increment the support of $X$.

**Computational Complexity:** Support computation takes time $O(|\mathcal{I}| \cdot |\boldsymbol{D}|)$ in the worst case, and because there are $O(2^{|\mathcal{I}|})$ possible candidates, the computational complexity of the brute-force method is $O(|\mathcal{I}| \cdot |\boldsymbol{D}| \cdot 2^{|\mathcal{I}|})$.

## Brute Force Algorithm

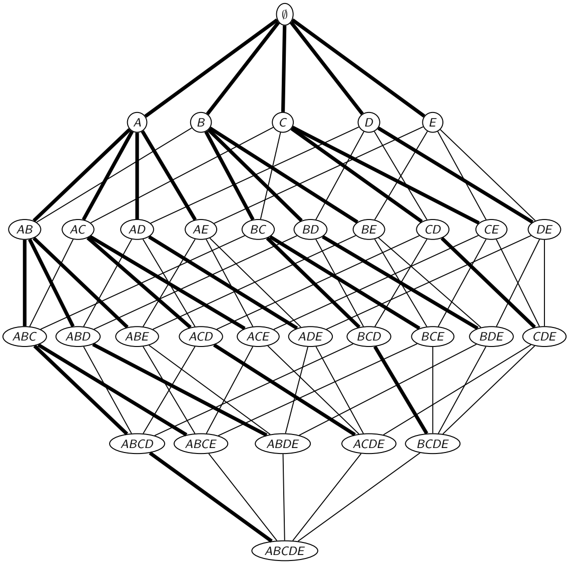
```
BruteForce ($\boldsymbol{D}$, $\mathcal{I}$, $minsup$):
1 $\mathcal{F} \leftarrow \emptyset$ // set of frequent itemsets
2
3 foreach $X \subseteq \mathcal{I}$ do
4     $sup(X) \leftarrow$ ComputeSupport $(X, \boldsymbol{D})$
5     if $sup(X) \geq minsup$ then
6         $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
7 return $\mathcal{F}$
```

```
ComputeSupport $(X, \boldsymbol{D})$:
1 $sup(X) \leftarrow 0$
2 foreach $\langle t, \boldsymbol{i}(t) \rangle \in \boldsymbol{D}$ do
3     if $X \subseteq \boldsymbol{i}(t)$ then
4         $sup(X) \leftarrow sup(X) + 1$
5 return $sup(X)$
```

## Itemset lattice and prefix-based search tree

Itemset search space is a lattice where any two itemsets $X$ and $Y$ are connected by a link iff $X$ is an *immediate subset* of $Y$, that is, $X \subseteq Y$ and $|X| = |Y| - 1$.

Frequent itemsets can enumerated using either a BFS or DFS search on the *prefix tree*, where two itemsets $X, Y$ are connected by a link iff $X$ is an immediate subset and prefix of $Y$. This allows one to enumerate itemsets starting with an empty set, and adding one more item at a time.



## Level-wise Approach: Apriori Algorithm

If $X \subseteq Y$, then $sup(X) \geq sup(Y)$, which leads to the following two observations: (1) if $X$ is frequent, then any subset $Y \subseteq X$ is also frequent, and (2) if $X$ is not frequent, then any superset $Y \supseteq X$ cannot be frequent.

The *Apriori algorithm* utilizes these two properties to significantly improve the brute-force approach. It employs a level-wise or breadth-first exploration of the itemset search space, and prunes all supersets of any infrequent candidate, as no superset of an infrequent itemset can be frequent. It also avoids generating any candidate that has an infrequent subset.

In addition to improving the candidate generation step via itemset pruning, the Apriori method also significantly improves the I/O complexity. Instead of counting the support for a single itemset, it explores the prefix tree in a breadth-first manner, and computes the support of all the valid candidates of size $k$ that comprise level $k$ in the prefix tree.

## The Apriori Algorithm

```
**Apriori** ($\boldsymbol{D}$, $\mathcal{I}$, *minsup*):
1  $\mathcal{F} \leftarrow \emptyset$
2  $\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$ // Initial prefix tree with single items
3  foreach $i \in \mathcal{I}$ do Add $i$ as child of $\emptyset$ in $\mathcal{C}^{(1)}$ with $sup(i) \leftarrow 0$
4  $k \leftarrow 1$ // $k$ denotes the level
5  while $\mathcal{C}^{(k)} \neq \emptyset$ do
6      ComputeSupport ($\mathcal{C}^{(k)}, \boldsymbol{D}$)
7      foreach leaf $X \in \mathcal{C}^{(k)}$ do
8          if $sup(X) \geq minsup$ then $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
9          else remove $X$ from $\mathcal{C}^{(k)}$
10     $\mathcal{C}^{(k+1)} \leftarrow$ ExtendPrefixTree ($\mathcal{C}^{(k)}$)
11     $k \leftarrow k+1$
12 return $\mathcal{F}^{(k)}$
```

## Apriori Algorithm

```
ComputeSupport ($\mathcal{C}^{(k)}, \boldsymbol{D}$):
1 foreach $\langle t, \boldsymbol{i}(t) \rangle \in \boldsymbol{D}$ do
2     foreach $k$-subset $X \subseteq \boldsymbol{i}(t)$ do
3         if $X \in \mathcal{C}^{(k)}$ then $sup(X) \leftarrow sup(X) + 1$

ExtendPrefixTree ($\mathcal{C}^{(k)}$):
1 foreach leaf $X_a \in \mathcal{C}^{(k)}$ do
2     foreach leaf $X_b \in \text{sibling}(X_a)$, such that $b > a$ do
3         $X_{ab} \leftarrow X_a \cup X_b$
          // prune if there are any infrequent subsets
4         if $X_j \in \mathcal{C}^{(k)}$, for all $X_j \subset X_{ab}$, such that $|X_j| = |X_{ab}| - 1$ then
5             Add $X_{ab}$ as child of $X_a$ with $sup(X_{ab}) \leftarrow 0$
6     if no extensions from $X_a$ then
7         remove $X_a$ and its ancestors with no extensions from $\mathcal{C}^{(k)}$
8 return $\mathcal{C}^{(k)}$
```

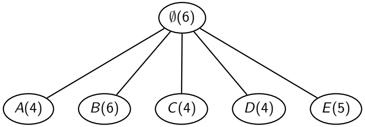
## Itemset Mining: Apriori Algorithm

Infrequent itemsets in gray

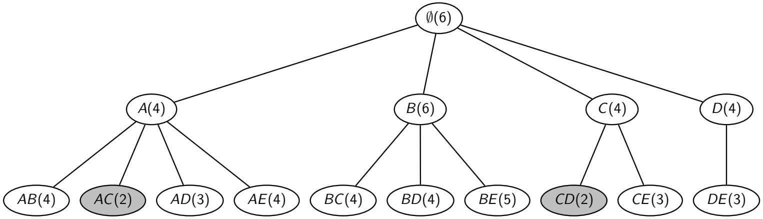
***D***

| $t$ | $\boldsymbol{i}(t)$ |
|---|---|
| 1 | *ABDE* |
| 2 | *BCE* |
| 3 | *ABDE* |
| 4 | *ABCE* |
| 5 | *ABCDE* |
| 6 | *BCD* |

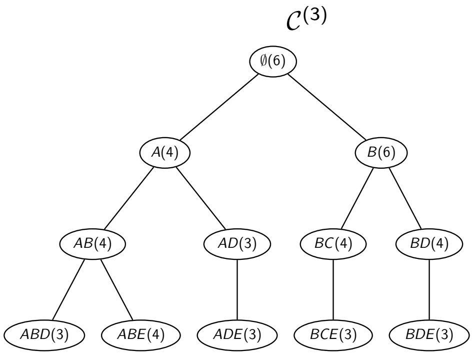
$\mathcal{C}^{(1)}$

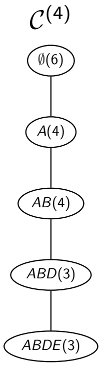


$\mathcal{C}^{(2)}$

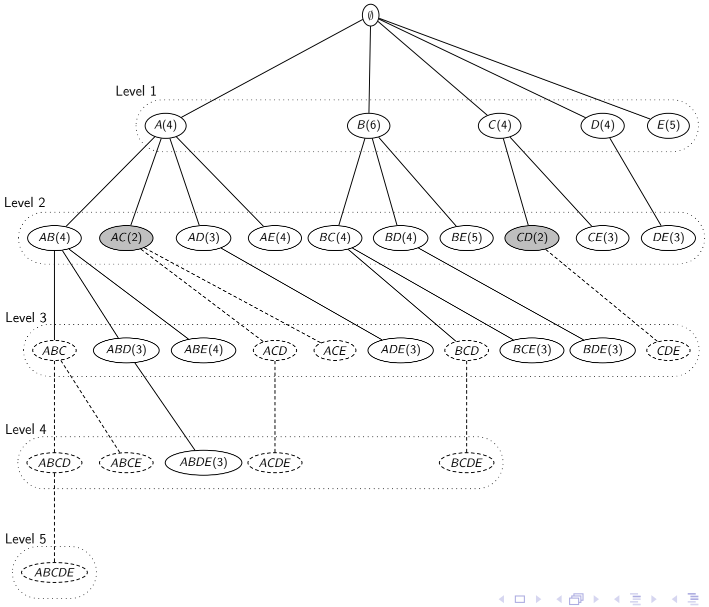


## Itemset Mining: Apriori Algorithm





# Apriori Algorithm: Prefix Search Tree and Pruning



## Apriori Algorithm: Details

Let $\mathcal{C}^{(k)}$ denote the prefix tree comprising all the candidate $k$-itemsets.

Apriori begins by inserting the single items into an initially empty prefix tree to populate $\mathcal{C}^{(1)}$.

The support for the current candidates is obtained via ComputeSupport procedure that generates $k$-subsets of each transaction in the database $\boldsymbol{D}$, and for each such subset it increments the support of the corresponding candidate in $\mathcal{C}^{(k)}$ if it exists. Next, we remove any infrequent candidate.

The leaves of the prefix tree that survive comprise the set of frequent $k$-itemsets $\mathcal{F}^{(k)}$, which are used to generate the candidate $(k+1)$-itemsets for the next level. The ExtendPrefixTree procedure employs prefix-based extension for candidate generation. Given two frequent $k$-itemsets $X_a$ and $X_b$ with a common $k-1$ length prefix, that is, given two sibling leaf nodes with a common parent, we generate the $(k+1)$-length candidate $X_{ab} = X_a \cup X_b$. This candidate is retained only if it has no infrequent subset. Finally, if a $k$-itemset $X_a$ has no extension, it is pruned from the prefix tree, and we recursively prune any of its ancestors with no $k$-itemset extension, so that in $\mathcal{C}^{(k)}$ all leaves are at level $k$.

If new candidates were added, the whole process is repeated for the next level. This process continues until no new candidates are added.

## Tidset Intersection Approach: Eclat Algorithm

The support counting step can be improved significantly if we can index the database in such a way that it allows fast frequency computations.

The Eclat algorithm leverages the tidsets directly for support computation. The basic idea is that the support of a candidate itemset can be computed by intersecting the tidsets of suitably chosen subsets. In general, given $\boldsymbol{t}(X)$ and $\boldsymbol{t}(Y)$ for any two frequent itemsets $X$ and $Y$, we have

$$\boldsymbol{t}(XY) = \boldsymbol{t}(X) \cap \boldsymbol{t}(Y)$$

The support of candidate $XY$ is simply the cardinality of $\boldsymbol{t}(XY)$, that is, $sup(XY) = |\boldsymbol{t}(XY)|$.

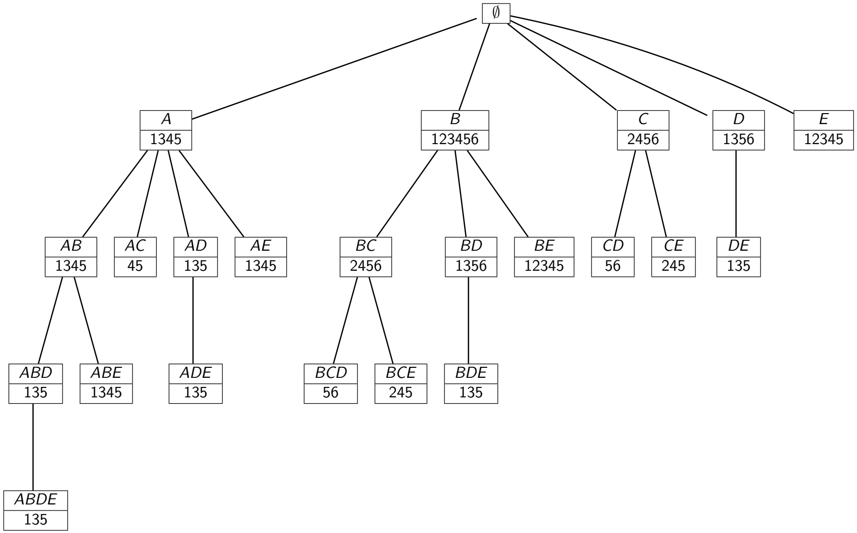
Eclat intersects the tidsets only if the frequent itemsets share a common prefix, and it traverses the prefix search tree in a DFS-like manner, processing a group of itemsets that have the same prefix, also called a *prefix equivalence class*.

## Eclat Algorithm

```
// Initial Call: $\mathcal{F} \leftarrow \emptyset, P \leftarrow \{\langle i, \boldsymbol{t}(i)\rangle \mid i \in \mathcal{I}, |\boldsymbol{t}(i)| \geq minsup\}$
Eclat ($P$, $minsup$, $\mathcal{F}$):
1 foreach $\langle X_a, \boldsymbol{t}(X_a)\rangle \in P$ do
2     $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, sup(X_a))\}$
3     $P_a \leftarrow \emptyset$
4     foreach $\langle X_b, \boldsymbol{t}(X_b)\rangle \in P$, with $X_b > X_a$ do
5         $X_{ab} = X_a \cup X_b$
6         $\boldsymbol{t}(X_{ab}) = \boldsymbol{t}(X_a) \cap \boldsymbol{t}(X_b)$
7         if $sup(X_{ab}) \geq minsup$ then
8             $P_a \leftarrow P_a \cup \{\langle X_{ab}, \boldsymbol{t}(X_{ab})\rangle\}$
9     if $P_a \neq \emptyset$ then Eclat ($P_a$, $minsup$, $\mathcal{F}$)
```

## Eclat Algorithm: Tidlist Intersections

Infrequent itemsets in gray



## Diffsets: Difference of Tidsets

The Eclat algorithm can be significantly improved if we can shrink the size of the intermediate tidsets. This can be achieved by keeping track of the differences in the tidsets as opposed to the full tidsets.

Let $X_a = \{x_1, \dots, x_{k-1}, x_a\}$ and $X_b = \{x_1, \dots, x_{k-1}, x_b\}$, so that $X_{ab} = X_a \cup X_b = \{x_1, \dots, x_{k-1}, x_a, x_b\}$.

The *diffset* of $X_{ab}$ is the set of tids that contain the prefix $X_a$, but not the item $X_b$

$$\boldsymbol{d}(X_{ab}) = \boldsymbol{t}(X_a) \setminus \boldsymbol{t}(X_{ab}) = \boldsymbol{t}(X_a) \setminus \boldsymbol{t}(X_b)$$

We can obtain an expression for $\boldsymbol{d}(X_{ab})$ in terms of $\boldsymbol{d}(X_a)$ and $\boldsymbol{d}(X_b)$ as follows:

$$\boldsymbol{d}(X_{ab}) = \boldsymbol{d}(X_b) \setminus \boldsymbol{d}(X_a)$$

which means that we can replace all intersection operations in Eclat with diffset operations.

## Diffsets: Difference of Tidsets

Given $\mathcal{T} = 123456$:

$$\boldsymbol{d}(A) = \mathcal{T} \setminus 1345 = 26 \qquad \boldsymbol{d}(B) = \mathcal{T} \setminus 123456 = \emptyset$$
$$\boldsymbol{d}(C) = \mathcal{T} \setminus 2456 = 13 \qquad \boldsymbol{d}(D) = \mathcal{T} \setminus 1356 = 24$$
$$\boldsymbol{d}(E) = \mathcal{T} \setminus 12345 = 6$$

For instance, the diffsets of $AB$ and $AC$ are given as

$$\boldsymbol{d}(AB) = \boldsymbol{d}(B) \setminus \boldsymbol{d}(A) = \emptyset \setminus \{2,6\} = \emptyset$$
$$\boldsymbol{d}(AC) = \boldsymbol{d}(C) \setminus \boldsymbol{d}(A) = \{1,3\} \setminus \{2,6\} = 13$$

and their support values are

$$sup(AB) = sup(A) - |\boldsymbol{d}(AB)| = 4 - 0 = 4$$
$$sup(AC) = sup(A) - |\boldsymbol{d}(AC)| = 4 - 2 = 2$$

The new prefix equivalence class $P_A$ is:

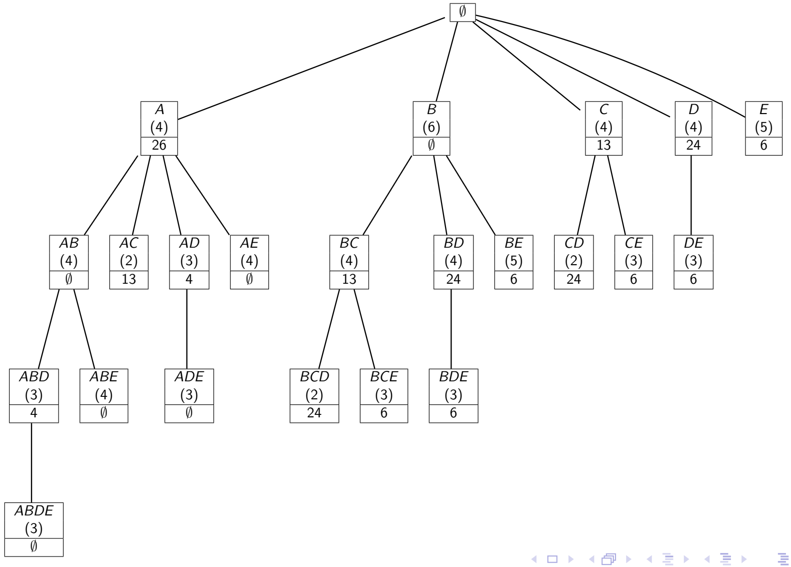
$$P_A = \big\{\langle AB, \emptyset, 4\rangle, \langle AD, 4, 3\rangle, \langle AE, \emptyset, 4\rangle\big\}$$

## Algorithm dEclat

```
// Initial Call: $\mathcal{F} \leftarrow \emptyset$,
    $P \leftarrow \{\langle i, \mathbf{d}(i), sup(i)\rangle \mid i \in \mathcal{I}, \mathbf{d}(i) = \mathcal{T} \setminus \mathbf{t}(i), sup(i) \geq minsup\}$
dEclat ($P$, $minsup$, $\mathcal{F}$):
1  foreach $\langle X_a, \mathbf{d}(X_a), sup(X_a)\rangle \in P$ do
2      $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, sup(X_a))\}$
3      $P_a \leftarrow \emptyset$
4      foreach $\langle X_b, \mathbf{d}(X_b), sup(X_b)\rangle \in P$, with $X_b > X_a$ do
5          $X_{ab} = X_a \cup X_b$
6          $\mathbf{d}(X_{ab}) = \mathbf{d}(X_b) \setminus \mathbf{d}(X_a)$
7          $sup(X_{ab}) = sup(X_a) - |\mathbf{d}(X_{ab})|$
8          if $sup(X_{ab}) \geq minsup$ then
9              $P_a \leftarrow P_a \cup \{\langle X_{ab}, \mathbf{d}(X_{ab}), sup(X_{ab})\rangle\}$
10     if $P_a \neq \emptyset$ then dEclat ($P_a$, $minsup$, $\mathcal{F}$)
```

## dEclat Algorithm: Diffsets

support shown within brackets; infrequent itemsets in gray



## Frequent Pattern Tree Approach: FPGrowth Algorithm

The FPGrowth method indexes the database for fast support computation via the use of an augmented prefix tree called the *frequent pattern tree* (FP-tree).

Each node in the tree is labeled with a single item, and each child node represents a different item. Each node also stores the support information for the itemset comprising the items on the path from the root to that node.

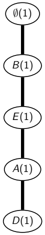
The FP-tree is constructed as follows. Initially the tree contains as root the null item $\emptyset$. Next, for each tuple $\langle t, X \rangle \in \boldsymbol{D}$, where $X = \boldsymbol{i}(t)$, we insert the itemset $X$ into the FP-tree, incrementing the count of all nodes along the path that represents $X$.

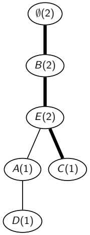
If $X$ shares a prefix with some previously inserted transaction, then $X$ will follow the same path until the common prefix. For the remaining items in $X$, new nodes are created under the common prefix, with counts initialized to 1. The FP-tree is complete when all transactions have been inserted.

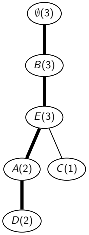
## Frequent Pattern Tree

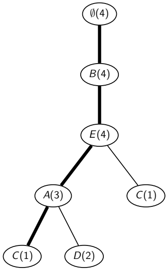
The FP-tree is a prefix compressed representation of $\boldsymbol{D}$. For most compression items are sorted in descending order of support.

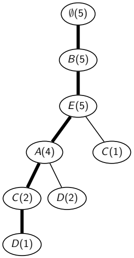
| Transactions |
|---|
| BEAD |
| BEC |
| BEAD |
| BEAC |
| BEACD |
| BCD |

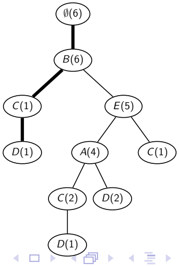












## FPGrowth Algorithm: Rationale

Given an FP-tree $R$, projected FP-trees are built for each frequent item $i$ in $R$ in increasing order of support in a recursive manner.

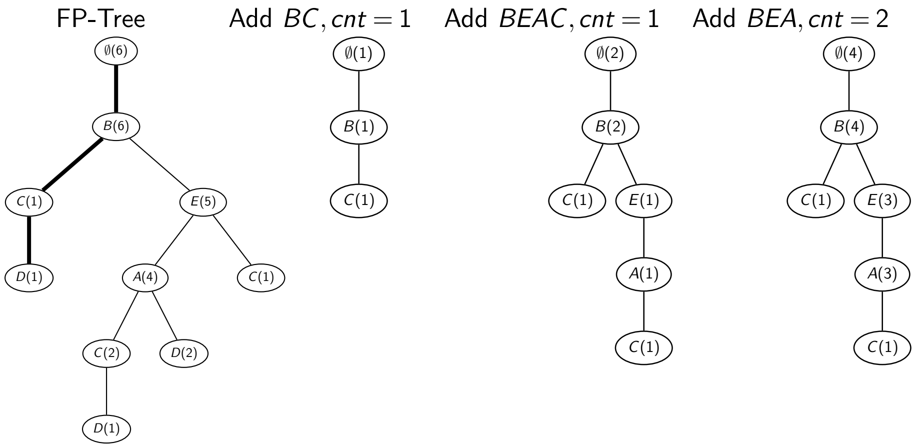
To project $R$ on item $i$, we find all the occurrences of $i$ in the tree, and for each occurrence, we determine the corresponding path from the root to $i$. The count of item $i$ on a given path is recorded in $cnt(i)$ and the path is inserted into the new projected tree $R_X$, where $X$ is the itemset obtained by extending the prefix $P$ with the item $i$. While inserting the path, the count of each node in $R_X$ along the given path is incremented by the path count $cnt(i)$.

The base case for the recursion happens when the input FP-tree $R$ is a single path. FP-trees that are paths are handled by enumerating all itemsets that are subsets of the path, with the support of each such itemset being given by the least frequent item in it.

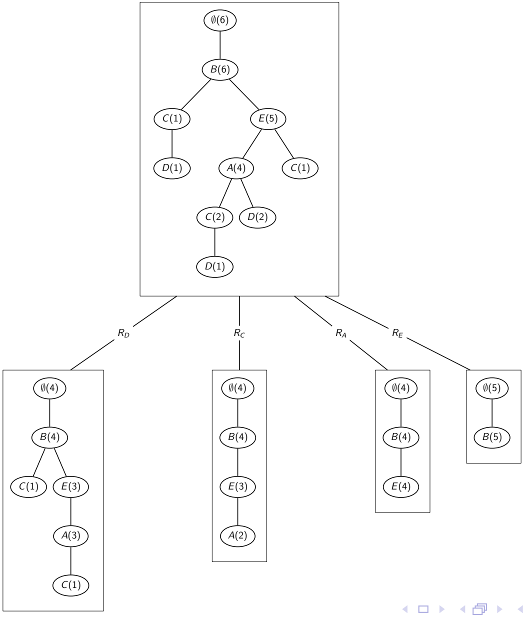
## FPGrowth Algorithm

```
// Initial Call: $R \leftarrow$ FP-tree($\mathbf{D}$), $P \leftarrow \emptyset$, $\mathcal{F} \leftarrow \emptyset$
**FPGrowth** ($R$, $P$, $\mathcal{F}$, $minsup$):
Remove infrequent items from $R$
if IsPath($R$) then // insert subsets of $R$ into $\mathcal{F}$
    foreach $Y \subseteq R$ do
        $X \leftarrow P \cup Y$
        $sup(X) \leftarrow \min_{x \in Y}\{cnt(x)\}$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
else    // process projected FP-trees for each frequent item $i$
    foreach $i \in R$ in increasing order of $sup(i)$ do
        $X \leftarrow P \cup \{i\}$
        $sup(X) \leftarrow sup(i)$ // sum of $cnt(i)$ for all nodes labeled $i$

        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
        $R_X \leftarrow \emptyset$ // projected FP-tree for $X$

        foreach $path \in$ PathFromRoot($i$) do
            $cnt(i) \leftarrow$ count of $i$ in $path$
            Insert $path$, excluding $i$, into FP-tree $R_X$ with count $cnt(i)$
        if $R_X \neq \emptyset$ then FPGrowth ($R_X$, $X$, $\mathcal{F}$, $minsup$)
```

## Projected Frequent Pattern Tree for $D$



## FPGrowth Algorithm: Frequent Pattern Tree Projection



## Association Rules

An *association rule* is an expression

$$X \xrightarrow{s,c} Y$$

where $X$ and $Y$ are itemsets and they are disjoint, that is, $X, Y \subseteq \mathcal{I}$, and $X \cap Y = \emptyset$. Let the itemset $X \cup Y$ be denoted as $XY$.

The *support* of the rule is the number of transactions in which both $X$ and $Y$ co-occur as subsets:

$$s = sup(X \longrightarrow Y) = |\boldsymbol{t}(XY)| = sup(XY)$$

The *relative support* of the rule is defined as the fraction of transactions where $X$ and $Y$ co-occur, and it provides an estimate of the joint probability of $X$ and $Y$:

$$rsup(X \longrightarrow Y) = \frac{sup(XY)}{|\boldsymbol{D}|} = P(X \wedge Y)$$

The *conf idence* of a rule is the conditional probability that a transaction contains $Y$ given that it contains $X$:

$$c = conf(X \longrightarrow Y) = P(Y|X) = \frac{P(X \wedge Y)}{P(X)} = \frac{sup(XY)}{sup(X)}$$

## Generating Association Rules

Given a frequent itemset $Z \in \mathcal{F}$, we look at all proper subsets $X \subset Z$ to compute rules of the form

$$X \xrightarrow{s,c} Y, \text{ where } Y = Z \setminus X$$

where $Z \setminus X = Z - X$.

The rule must be frequent because

$$s = sup(XY) = sup(Z) \geq minsup$$

We compute the confidence as follows:

$$\boxed{c = \frac{sup(X \cup Y)}{sup(X)} = \frac{sup(Z)}{sup(X)}}$$

If $c \geq minconf$, then the rule is a strong rule. On the other hand, if $conf(X \longrightarrow Y) < c$, then $conf(W \longrightarrow Z \setminus W) < c$ for all subsets $W \subset X$, as $sup(W) \geq sup(X)$. We can thus avoid checking subsets of $X$.

## Association Rule Mining Algorithm

```
    AssociationRules ($\mathcal{F}$, $minconf$):
1   foreach $Z \in \mathcal{F}$, such that $|Z| \geq 2$ do
2       $\mathcal{A} \leftarrow \{X \mid X \subset Z, X \neq \emptyset\}$
3       while $\mathcal{A} \neq \emptyset$ do
4           $X \leftarrow$ maximal element in $\mathcal{A}$
5           $\mathcal{A} \leftarrow \mathcal{A} \setminus X$// remove $X$ from $\mathcal{A}$
6           $c \leftarrow sup(Z)/sup(X)$
7           if $c \geq minconf$ then
8               print $X \longrightarrow Y$, $sup(Z)$, $c$
9           else
10              $\mathcal{A} \leftarrow \mathcal{A} \setminus \{W \mid W \subset X\}$
                // remove all subsets of $X$ from $\mathcal{A}$
```

# Data Mining and Machine Learning: Fundamental Concepts and Algorithms

datamininingbook.info

Mohammed J. Zaki[1] Wagner Meira Jr.[2]

[1]Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY, USA

[2]Department of Computer Science
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

# Chapter 8: Itemset Mining