

Association Analysis

General Idea of Association Analysis

- Infer knowledge from databases
- Model (experts') knowledge

Association Rules

- Market basket analysis
- Generate rules that represent that knowledge

Bayesian Networks

- Probabilistic networks (graphs)
- Model (high-dimensional) distribution of attributes as combination of several lower dimensional distributions that are easier to handle

Association Rules and Frequent Item Sets

Frequent Item Set Mining: Motivation

Frequent Item Set Mining is a method for **market basket analysis**.

It aims at finding regularities in the shopping behavior of customers of supermarkets, mail-order companies, on-line shops etc.

More specifically:

Find sets of products that are frequently bought together.

Possible applications of found frequent item sets:

- Improve arrangement of products in shelves, on a catalog's pages.
- Support cross-selling (suggestion of other products), product bundling.
- Fraud detection, technical dependence analysis.

Often found patterns are expressed as **association rules**, for example:

If a customer buys **bread** and **wine**,
then she/he will probably also buy **cheese**.

Frequent Item Set Mining: Basic Notions

Let $A = \{a_1, \dots, a_m\}$ be a set of **items**.

Items may be products, special equipment items, service options etc.

Any subset $I \subseteq A$ is called an **item set**.

An item set may be any set of products that can be bought (together).

Let $T = (t_1, \dots, t_n)$ with $\forall i, 1 \leq i \leq n : t_i \subseteq A$
be a vector of **transactions** over A .

Each transaction is an item set, but some item sets may not appear in T .

Transactions need not be pairwise different: it may be $t_i = t_k$ for $i \neq k$.

T may also be defined as a *bag* or *multiset* of transactions.

The set A may not be explicitly given, but only implicitly as $A = \bigcup_{i=1}^n t_i$.

A vector of transactions can list, for example, the sets of products bought by the customers of a supermarket in a given period of time.

Frequent Item Set Mining: Basic Notions

Let $I \subseteq A$ be an item set and T a vector of transactions over A .

A transaction $t \in T$ **covers** the item set I or
the item set I is **contained in** a transaction $t \in T$ iff $I \subseteq t$.

The set $K_T(I) = \{k \in \{1, \dots, n\} \mid I \subseteq t_k\}$ is called the **cover** of I w.r.t. T .

The cover of an item set is the index set of the transactions that cover it.

It may also be defined as a vector of all transactions that cover it
(which, however, is complicated to write in formally correct way).

The value $s_T(I) = |K_T(I)|$ is called the **(absolute) support** of I w.r.t. T .

The value $\sigma_T(I) = \frac{1}{n} |K_T(I)|$ is called the **relative support** of I w.r.t. T .

The support of I is the number or fraction of transactions that contain it.

Sometimes $\sigma_T(I)$ is also called the *(relative) frequency* of I w.r.t. T .

Frequent Item Set Mining: Formal Definition

Given:

a set $A = \{a_1, \dots, a_m\}$ of items,

a vector $T = (t_1, \dots, t_n)$ of transactions over A ,

a number $s_{\min} \in \mathbb{N}$, $0 < s_{\min} \leq n$ or (equivalently)

a number $\sigma_{\min} \in \mathbb{R}$, $0 < \sigma_{\min} \leq 1$, the **minimum support**.

Desired:

the set of **frequent item sets**, that is,

the set $F_T(s_{\min}) = \{I \subseteq A \mid s_T(I) \geq s_{\min}\}$ or (equivalently)

the set $\Phi_T(\sigma_{\min}) = \{I \subseteq A \mid \sigma_T(I) \geq \sigma_{\min}\}$.

Note that with the relations $s_{\min} = \lceil n\sigma_{\min} \rceil$ and $\sigma_{\min} = \frac{1}{n}s_{\min}$ the two versions can easily be transformed into each other.

Frequent Item Sets: Example

transaction vector

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

frequent item sets

0 items	1 item	2 items	3 items
\emptyset : 100%	{a}: 70%	{a, c}: 40%	{a, c, d}: 30%
	{b}: 30%	{a, d}: 50%	{a, c, e}: 30%
	{c}: 70%	{a, e}: 60%	{a, d, e}: 40%
	{d}: 60%	{b, c}: 30%	
	{e}: 70%	{c, d}: 40%	
		{c, e}: 40%	
		{d, e}: 40%	

The minimum support is $s_{\min} = 3$ or $\sigma_{\min} = 0.3 = 30\%$ in this example.

There are $2^5 = 32$ possible item sets over $A = \{a, b, c, d, e\}$.

There are 16 frequent item sets (but only 10 transactions).

Properties of the Support of an Item Set

A **brute force approach** that enumerates all possible item sets, determines their support, and discards infrequent item sets is usually **infeasible**:

The number of possible item sets grows exponentially with the number of items.

A typical supermarket has thousands of different products.

Idea: Consider the properties of the support, in particular:

$$\forall I : \forall J \supseteq I : K_T(J) \subseteq K_T(I).$$

This property holds, since $\forall t : \forall I : \forall J \supseteq I : J \subseteq t \rightarrow I \subseteq t$.

Each additional item is another condition a transaction has to satisfy.

Transactions that do not satisfy this condition are removed from the cover.

It follows:

$$\forall I : \forall J \supseteq I : s_T(I) \geq s_T(J).$$

That is: **If an item set is extended, its support cannot increase.**

One also says that support is **anti-monotone** or **downward closed**.

Properties of the Support of an Item Set

From $\forall I : \forall J \supseteq I : s_T(I) \geq s_T(J)$ it follows

$$\forall s_{\min} : \forall I : \forall J \supseteq I : s_T(I) < s_{\min} \rightarrow s_T(J) < s_{\min}.$$

That is: **No superset of an infrequent item set can be frequent.**

This property is often referred to as the **Apriori Property**.

Rationale: Sometimes we can know *a priori*, that is, before checking its support by accessing the given transaction vector, that an item set cannot be frequent.

Of course, the contraposition of this implication also holds:

$$\forall s_{\min} : \forall I : \forall J \subseteq I : s_T(I) \geq s_{\min} \rightarrow s_T(J) \geq s_{\min}.$$

That is: **All subsets of a frequent item set are frequent.**

This suggests a compressed representation of the set of frequent item sets.

Maximal Item Sets

Consider the set of **maximal (frequent) item sets**:

$$M_T(s_{\min}) = \{I \subseteq A \mid s_T(I) \geq s_{\min} \wedge \forall J \supset I : s_T(J) < s_{\min}\}.$$

That is: An item set is maximal if it is frequent,
but none of its proper supersets is frequent.

Since with this definition we know that

$$\forall s_{\min} : \forall I : I \in M_T(s_{\min}) \vee \exists J \supset I : s_T(J) \geq s_{\min}$$

it follows (can easily be proven by successively extending the item set I)

$$\forall s_{\min} : \forall I : I \in F_T(s_{\min}) \rightarrow \exists J \in M_T(s_{\min}) : I \subseteq J.$$

That is: **Every frequent item set has a maximal superset.**

Therefore:

$$\forall s_{\min} : F_T(s_{\min}) = \bigcup_{I \in M_T(s_{\min})} 2^I$$

Maximal Frequent Item Sets: Example

transaction vector

- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{c, b, e\}$
- 10: $\{a, d, e\}$

frequent item sets

0 items	1 item	2 items	3 items
\emptyset : 100%	$\{a\}$: 70%	$\{a, c\}$: 40%	$\{a, c, d\}$: 30%
	$\{b\}$: 30%	$\{a, d\}$: 50%	$\{a, c, e\}$: 30%
	$\{c\}$: 70%	$\{a, e\}$: 60%	$\{a, d, e\}$: 40%
	$\{d\}$: 60%	$\{b, c\}$: 30%	
	$\{e\}$: 70%	$\{c, d\}$: 40%	
		$\{c, e\}$: 40%	
		$\{d, e\}$: 40%	

The maximal item sets are:

$$\{b, c\}, \quad \{a, c, d\}, \quad \{a, c, e\}, \quad \{a, d, e\}.$$

Every frequent item set is a subset of at least one of these sets.

Limits of Maximal Item Sets

The set of maximal item sets captures the set of all frequent item sets, but then we know only the support of the maximal item sets.

About the support of a non-maximal frequent item set we only know:

$$\forall s_{\min} : \forall I \in F_T(s_{\min}) - M_T(s_{\min}) : s_T(I) \geq \max_{J \in M_T(s_{\min}), J \supset I} s_T(J).$$

This relation follows immediately from $\forall I : \forall J \supseteq I : s_T(I) \geq s_T(J)$, that is, an item set cannot have a lower support than any of its supersets.

Note that we have generally

$$\forall s_{\min} : \forall I \in F_T(s_{\min}) : s_T(I) \geq \max_{J \in M_T(s_{\min}), J \supseteq I} s_T(J).$$

Question: Can we find a subset of the set of all frequent item sets, which also preserves knowledge of all support values?

Closed Item Sets

Consider the set of **closed (frequent) item sets**:

$$C_T(s_{\min}) = \{I \subseteq A \mid s_T(I) \geq s_{\min} \wedge \forall J \supset I : s_T(J) < s_T(I)\}.$$

That is: An item set is closed if it is frequent,
but none of its proper supersets has the same support.

Since with this definition we know that

$$\forall s_{\min} : \forall I : I \in C_T(s_{\min}) \vee \exists J \supset I : s_T(J) = s_T(I)$$

it follows (can easily be proven by successively extending the item set I)

$$\forall s_{\min} : \forall I : I \in F_T(s_{\min}) \rightarrow \exists J \in C_T(s_{\min}) : I \subseteq J.$$

That is: **Every frequent item set has a closed superset.**

Therefore:

$$\forall s_{\min} : F_T(s_{\min}) = \bigcup_{I \in C_T(s_{\min})} 2^I$$

Closed Item Sets

However, not only has every frequent item set a closed superset, but it has a **closed superset with the same support**:

$$\forall s_{\min} : \forall I : I \in F_T(s_{\min}) \rightarrow \exists J \supseteq I : J \in C_T(s_{\min}) \wedge s_T(J) = s_T(I).$$

(Proof: see the considerations on the next slide)

The set of all closed item sets preserves knowledge of all support values:

$$\forall s_{\min} : \forall I \in F_T(s_{\min}) : s_T(I) = \max_{J \in C_T(s_{\min}), J \supseteq I} s_T(J).$$

Note that the weaker statement

$$\forall s_{\min} : \forall I \in F_T(s_{\min}) : s_T(I) \geq \max_{J \in C_T(s_{\min}), J \supseteq I} s_T(J)$$

follows immediately from $\forall I : \forall J \supseteq I : s_T(I) \geq s_T(J)$, that is, an item set cannot have a lower support than any of its supersets.

Closed Item Sets

Alternative characterization of closed item sets:

$$I \text{ closed} \iff s_T(I) \geq s_{\min} \quad \wedge \quad I = \bigcap_{k \in K_T(I)} t_k.$$

Reminder: $K_T(I) = \{k \in \{1, \dots, n\} \mid I \subseteq t_k\}$ is the *cover* of I w.r.t. T .

This is derived as follows: since $\forall k \in K_T(I) : I \subseteq t_k$, it is obvious that

$$\forall s_{\min} : \forall I \in F_T(s_{\min}) : I \subseteq \bigcap_{k \in K_T(I)} t_k,$$

If $I \subset \bigcap_{k \in K_T(I)} t_k$, it is not closed, since $\bigcap_{k \in K_T(I)} t_k$ has the same support.

On the other hand, no superset of $\bigcap_{k \in K_T(I)} t_k$ has the cover $K_T(I)$.

Note that the above characterization allows us to construct the (uniquely determined) closed superset of a frequent item set that has the same support.

Closed Frequent Item Sets: Example

transaction vector

- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{c, b, e\}$
- 10: $\{a, d, e\}$

frequent item sets

0 items	1 item	2 items	3 items
\emptyset : 100%	$\{a\}$: 70%	$\{a, c\}$: 40%	$\{a, c, d\}$: 30%
	$\{b\}$: 30%	$\{a, d\}$: 50%	$\{a, c, e\}$: 30%
	$\{c\}$: 70%	$\{a, e\}$: 60%	$\{a, d, e\}$: 40%
	$\{d\}$: 60%	$\{b, c\}$: 30%	
	$\{e\}$: 70%	$\{c, d\}$: 40%	
		$\{c, e\}$: 40%	
		$\{d, e\}$: 40%	

All frequent item sets are closed with the exception of $\{b\}$ and $\{d, e\}$.

$\{b\}$ is a subset of $\{b, c\}$, both have support 30%.

$\{d, e\}$ is a subset of $\{a, d, e\}$, both have a support of 40%.

Types of Frequent Item Sets

Frequent Item Set

Any frequent item set (support is higher than the minimal support):

$$I \text{ frequent} \iff s_T(I) \geq s_{\min}$$

Closed Item Set

A frequent item set is called *closed* if no superset has the same support:

$$I \text{ closed} \iff s_T(I) \geq s_{\min} \quad \wedge \quad \forall J \supset I : s_T(J) < s_T(I)$$

Maximal Item Set

A frequent item set is called *maximal* if no superset is frequent:

$$I \text{ maximal} \iff s_T(I) \geq s_{\min} \quad \wedge \quad \forall J \supset I : s_T(J) < s_{\min}$$

Obvious relations between these types of item sets:

- All maximal and all closed item sets are frequent.
- All maximal item sets are closed.

Types of Frequent Item Sets: Example

0 items	1 item	2 items	3 items
\emptyset^+ : 100%	$\{a\}^+$: 70% $\{b\}$: 30% $\{c\}^+$: 70% $\{d\}^+$: 60% $\{e\}^+$: 70%	$\{a, c\}^+$: 40% $\{a, d\}^+$: 50% $\{a, e\}^+$: 60% $\{b, c\}^{+*}$: 30% $\{c, d\}^+$: 40% $\{c, e\}^+$: 40% $\{d, e\}$: 40%	$\{a, c, d\}^{+*}$: 30% $\{a, c, e\}^{+*}$: 30% $\{a, d, e\}^{+*}$: 40%

Frequent Item Set

Any frequent item set (support is higher than the minimal support).

Closed Item Set (marked with $^+$)

A frequent item set is called *closed* if no superset has the same support.

Maximal Item Set (marked with *)

A frequent item set is called *maximal* if no superset is frequent.

Searching for Frequent Item Sets

We know that it suffices to find the closed item sets together with their support.

The characterization of closed item sets by

$$I \text{ closed} \iff s_T(I) \geq s_{\min} \quad \wedge \quad I = \bigcap_{k \in K_T(I)} t_k$$

suggests to find them by forming all possible intersections of the transactions and checking their support.

However, approaches using this idea are not competitive with other methods.

If the support of all frequent item sets is needed, it can be clumsy and tedious to compute the support of a non-closed frequent item set with

$$\forall s_{\min} : \forall I \in F_T(s_{\min}) - C_T(s_{\min}) : s_T(I) = \max_{J \in C_T(s_{\min}), J \supset I} s_T(J).$$

In order to find the closed sets one may have to visit many frequent sets anyway.

Finding the Frequent Item Sets

Idea: Use the properties of the support to organize the search for all frequent item sets, especially

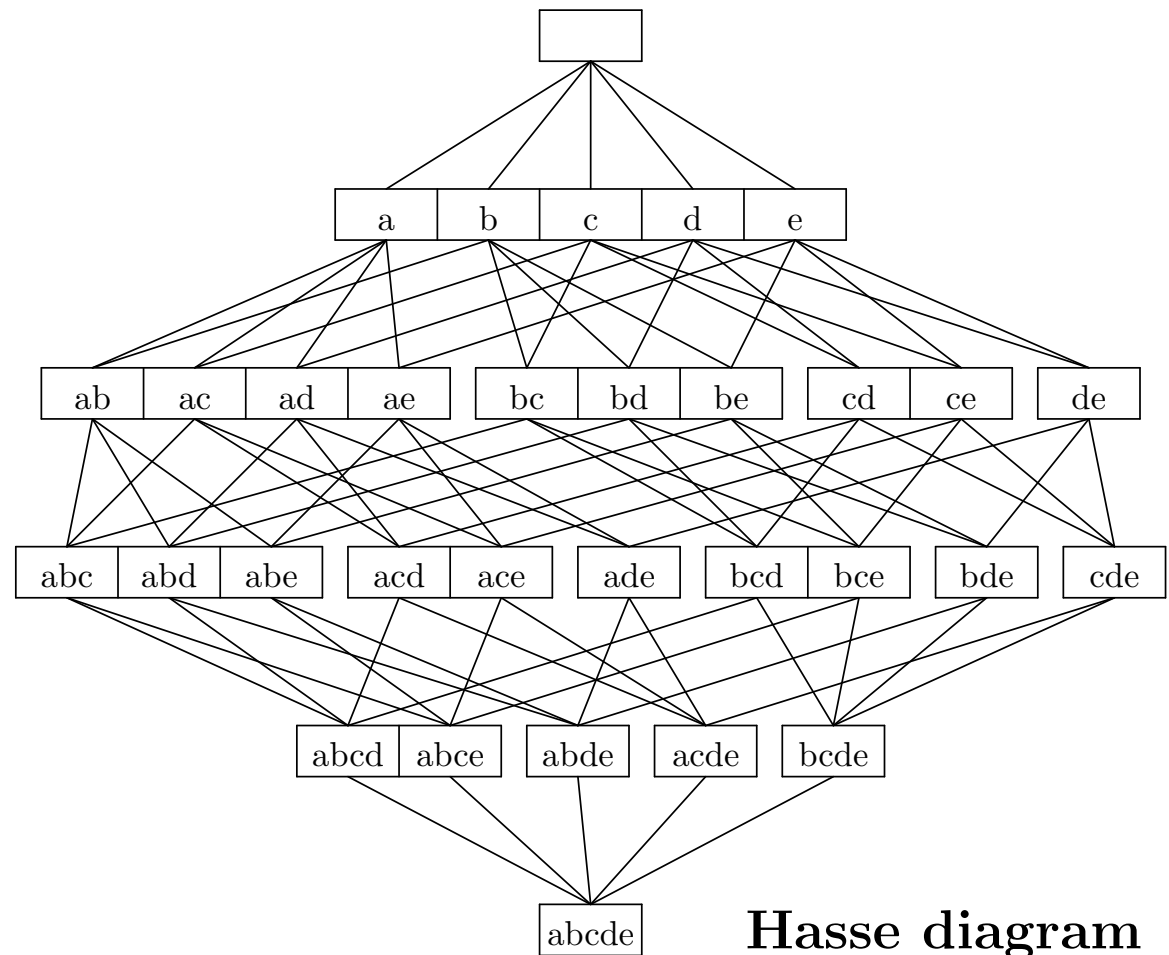
$$\forall I : \forall J \supset I :$$

$$s_T(I) < s_{\min}$$

$$\rightarrow s_T(J) < s_{\min}.$$

Since these properties relate the support of an item set to the support of its **subsets** and **supersets**, it is reasonable to organize the search based on the **subset lattice** of the set A , the set of all items.

A subset lattice for five items $\{a, b, c, d, e\}$:



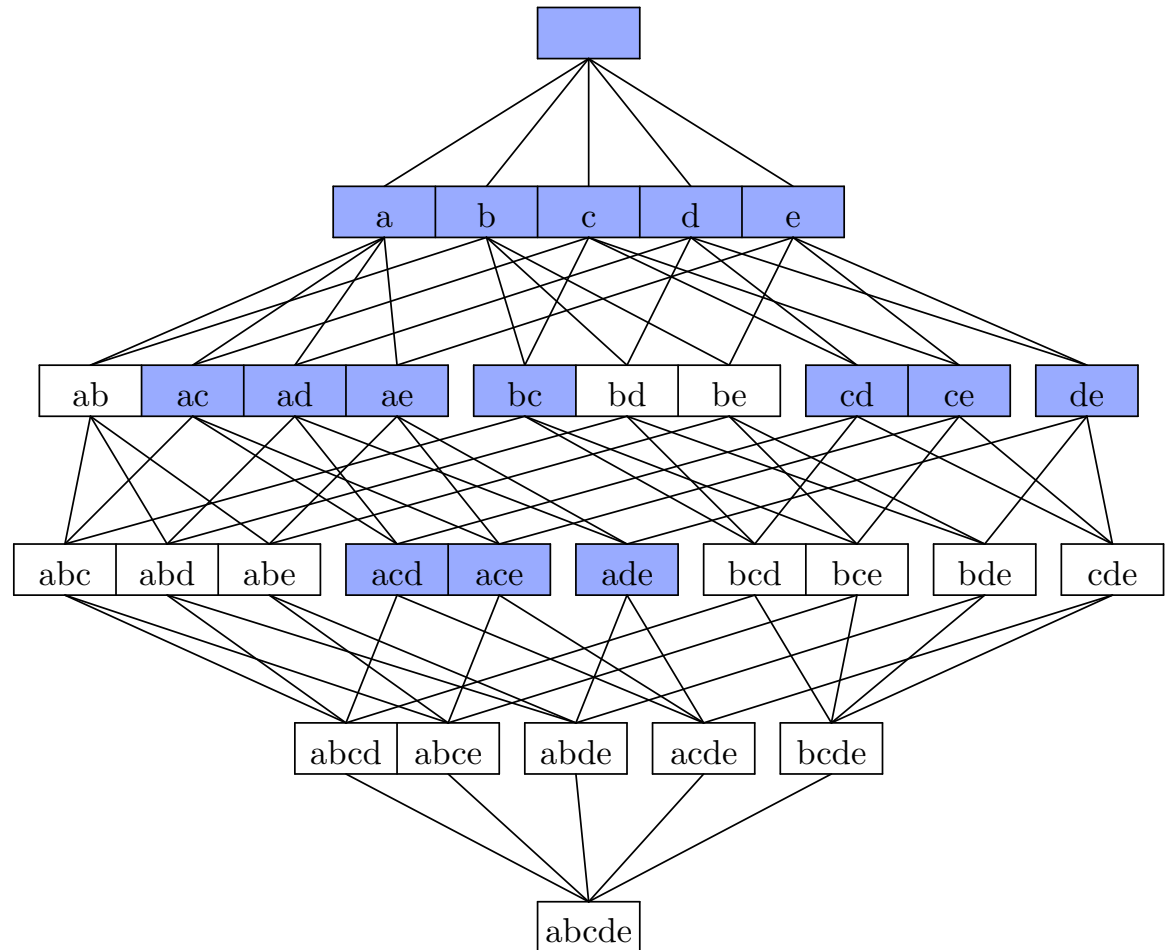
Subset Lattice and Frequent Item Sets

transaction vector

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

Blue boxes are frequent item sets, white boxes infrequent item sets.

subset lattice with frequent item sets ($s_{\min} = 3$):



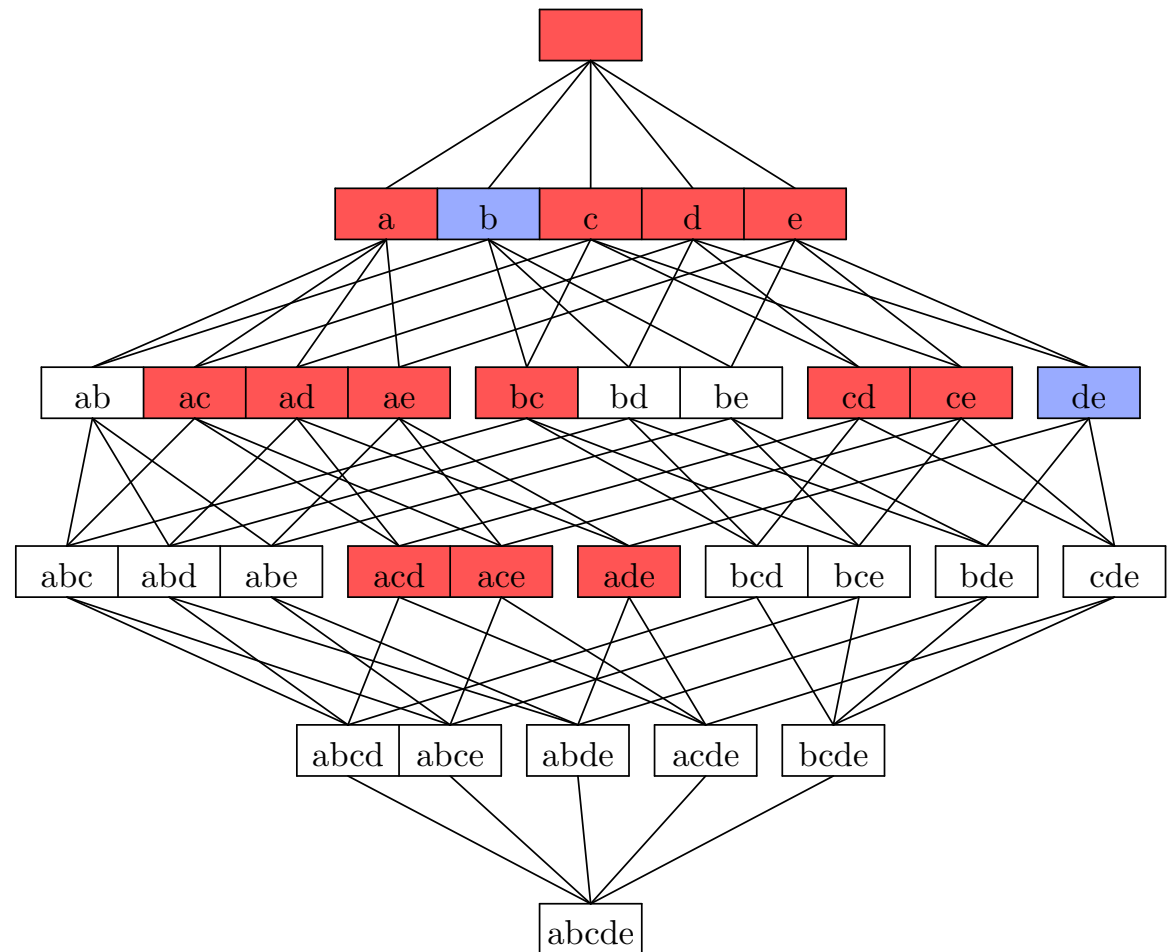
Subset Lattice and Closed Item Sets

transaction vector

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

Red boxes are closed item sets, white boxes infrequent item sets.

subset lattice with closed item sets ($s_{\min} = 3$):



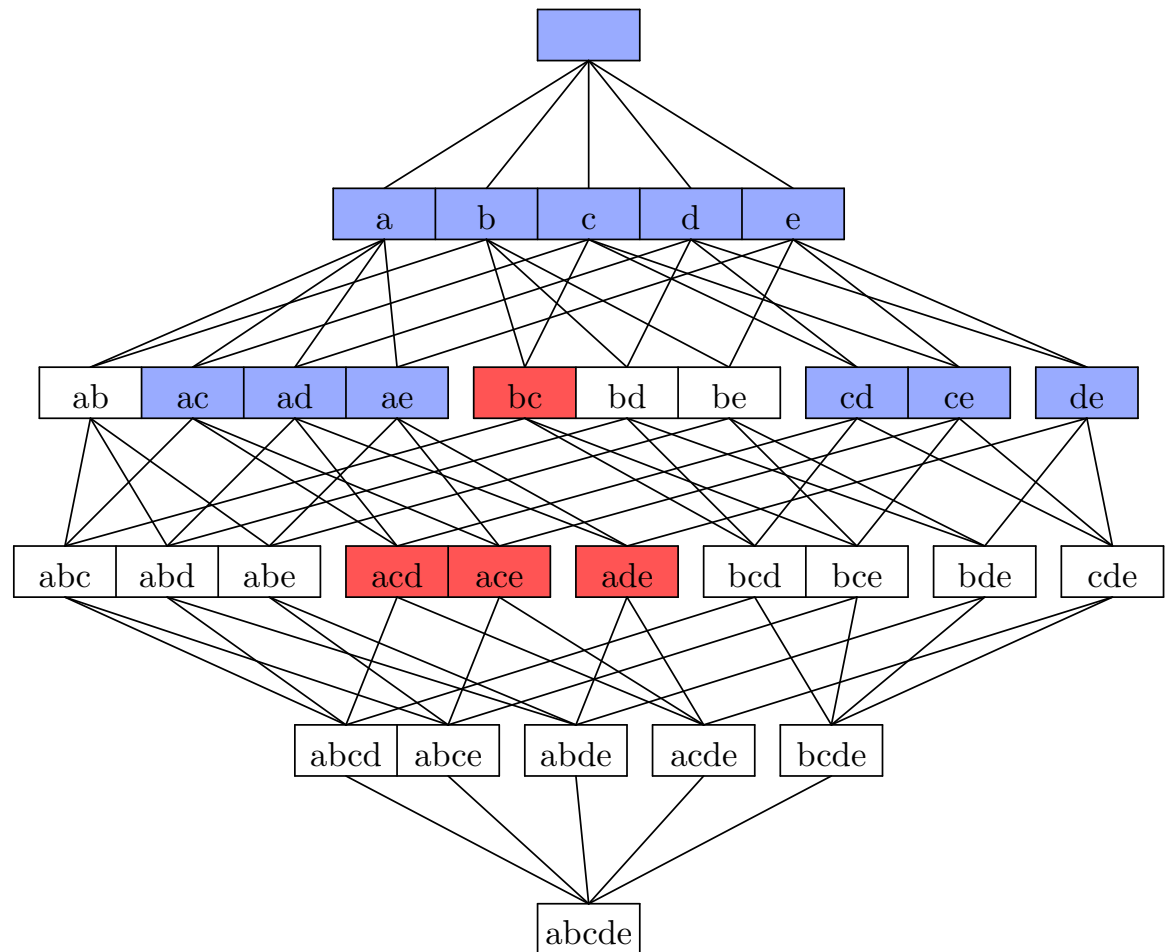
Subset Lattice and Maximal Item Sets

transaction vector

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

Red boxes are maximal item sets, white boxes infrequent item sets.

subset lattice with maximal item sets ($s_{\min} = 3$):



The Apriori Algorithm

[Agrawal and Srikant 1994]

Searching for Frequent Item Sets

One possible scheme for the search:

Determine the support of the one element item sets and discard the infrequent items.

Form candidate item sets with two items (both items must be frequent), determine their support, and discard the infrequent item sets.

Form candidate item sets with three items (all pairs must be frequent), determine their support, and discard the infrequent item sets.

Continue by forming candidate item sets with four, five etc. items until no candidate item set is frequent.

This is the general scheme of the **Apriori Algorithm**.

It is based on two main steps: **candidate generation** and **pruning**.

All frequent item set mining algorithms are based on these steps in some form.

The Apriori Algorithm 1

```
function apriori ( $A, T, s_{\min}$ )           (* Apriori algorithm *)
begin
   $k := 1$ ;                                (* initialize the item set size *)
   $E_k := \bigcup_{a \in A} \{\{a\}\}$ ;         (* start with single element sets *)
   $F_k := \text{prune}(E_k, T, s_{\min})$ ;      (* and determine the frequent ones *)
  while  $F_k \neq \emptyset$  do begin       (* while there are frequent item sets *)
     $E_{k+1} := \text{candidates}(F_k)$ ;      (* create item sets with one item more *)
     $F_{k+1} := \text{prune}(E_{k+1}, T, s_{\min})$ ; (* and determine the frequent ones *)
     $k := k + 1$ ;                          (* increment the item counter *)
  end;
  return  $\bigcup_{j=1}^k F_j$ ;                (* return the frequent item sets *)
end (* apriori *)
```

The Apriori Algorithm 2

```
function candidates ( $F_k$ )      (* generate candidates with  $k + 1$  items *)
begin
   $E := \emptyset$ ;                (* initialize the set of candidates *)
  forall  $f_1, f_2 \in F_k$         (* traverse all pairs of frequent item sets *)
  with  $f_1 = \{a_1, \dots, a_{k-1}, a_k\}$  (* that differ only in one item and *)
  and  $f_2 = \{a_1, \dots, a_{k-1}, a'_k\}$  (* are in a lexicographic order *)
  and  $a_k < a'_k$  do begin      (* (the order is arbitrary, but fixed) *)
     $f := f_1 \cup f_2 = \{a_1, \dots, a_{k-1}, a_k, a'_k\}$ ; (* union has  $k + 1$  items *)
    if  $\forall a \in f : f - \{a\} \in F_k$  (* only if all subsets are frequent, *)
    then  $E := E \cup \{f\}$ ;      (* add the new item set to the candidates *)
  end;                          (* (otherwise it cannot be frequent) *)
  return  $E$ ;                    (* return the generated candidates *)
end (* candidates *)
```

The Apriori Algorithm 3

```
function prune ( $E, T, s_{\min}$ )      (* prune infrequent candidates *)
begin
  forall  $e \in E$  do                (* initialize the support counters *)
     $s_T(e) := 0;$                   (* of all candidates to be checked *)
  forall  $t \in T$  do              (* traverse the transactions *)
    forall  $e \in E$  do            (* traverse the candidates *)
      if  $e \subseteq t$             (* if transaction contains the candidate, *)
      then  $s_T(e) := s_T(e) + 1;$  (* increment the support counter *)
   $F := \emptyset;$                 (* initialize the set of frequent candidates *)
  forall  $e \in E$  do              (* traverse the candidates *)
    if  $s_T(e) \geq s_{\min}$         (* if a candidate is frequent, *)
    then  $F := F \cup \{e\};$       (* add it to the set of frequent candidates *)
  return  $F;$                     (* return the pruned set of candidates *)
end (* prune *)
```

Searching for Frequent Item Sets

The Apriori algorithm searches the subset lattice top-down level by level.

Collecting the frequent item sets of size k in a set F_k has drawbacks:
A frequent item set of size $k + 1$ can be formed in

$$j = \frac{k(k + 1)}{2}$$

possible ways. (For infrequent item sets the number may be smaller.)

As a consequence, the candidate generation step may carry out a lot of redundant work, since it suffices to generate each candidate item set once.

Question: Can we reduce or even eliminate this redundant work?

More generally:

How can we make sure that any candidate item set is generated at most once?

Idea: Assign to each item set a unique parent item set,
from which this item set is to be generated.

Searching for Frequent Item Sets

A core problem is that an item set of size k (that is, with k items) can be generated in $k!$ different ways (on $k!$ paths in the Hasse diagram), because in principle the items may be added in any order.

If we consider an item by item process of building an item set (which can be imagined as a levelwise traversal of the lattice), there are k possible ways of forming an item set of size k from item sets of size $k - 1$ by adding the remaining item.

It is obvious that it suffices to consider each item set at most once in order to find the frequent ones (infrequent item sets need not be generated at all).

Question: Can we reduce or even eliminate this variety?

More generally:

How can we make sure that any candidate item set is generated at most once?

Idea: Assign to each item set a unique parent item set, from which this item set is to be generated.

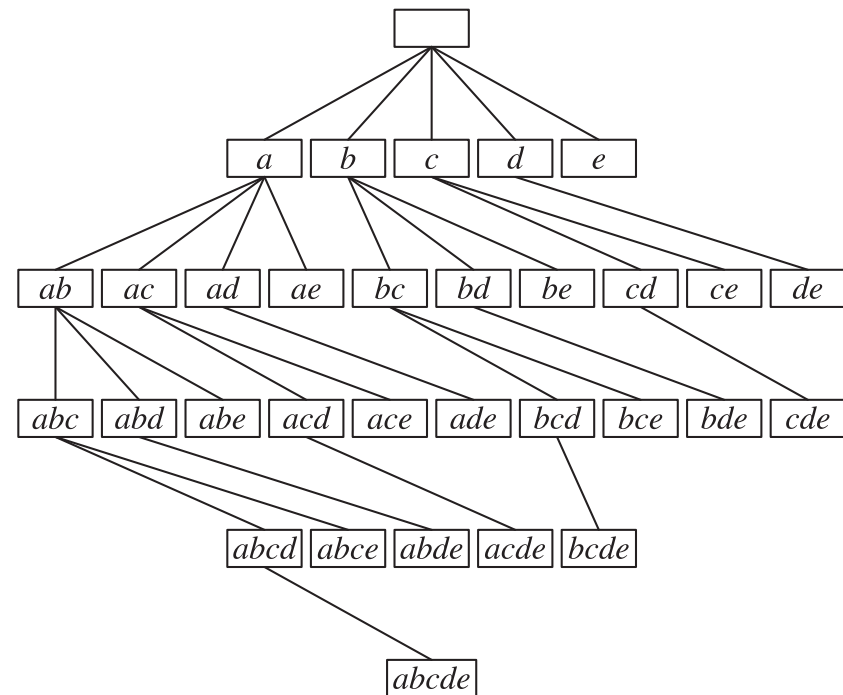
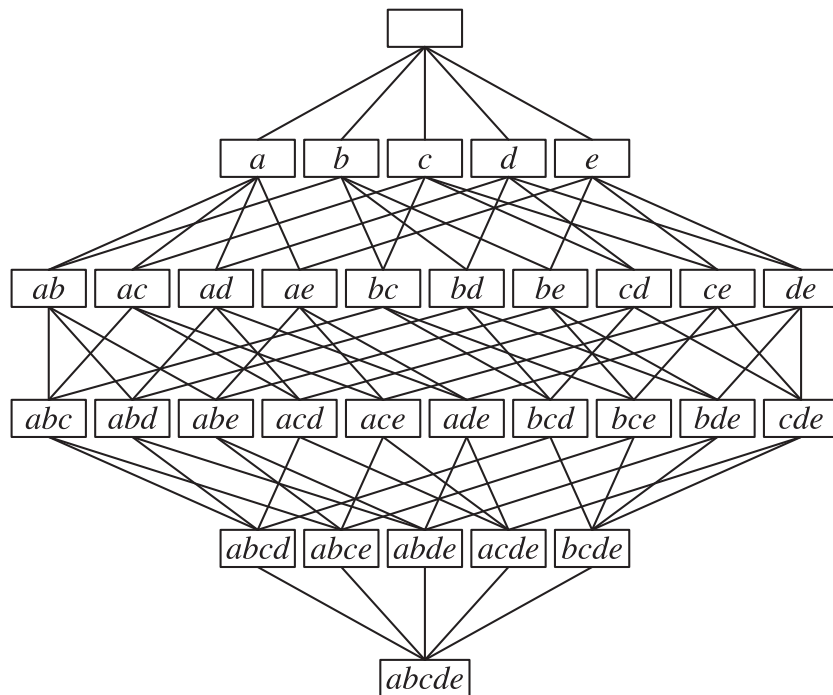
Searching for Frequent Item Sets

We have to search the item subset lattice / its Hasse diagram.

Assigning unique parents turns the Hasse diagram into a tree.

Traversing the resulting tree explores each item set exactly once.

Subset lattice (Hasse diagram) and a possible tree for five items:



Searching with Unique Parents

Principle of a Search Algorithm based on Unique Parents:

Base Loop:

- Traverse all one-element item sets (their unique parent is the empty set).
- Recursively process all one-element item sets that are frequent.

Recursive Processing:

For a given frequent item set I :

- Generate all extensions J of I by one item (that is, $J \supset I$, $|J| = |I| + 1$) for which the item set I is the chosen unique parent.
- For all J : if J is frequent, process J recursively, otherwise discard J .

Questions:

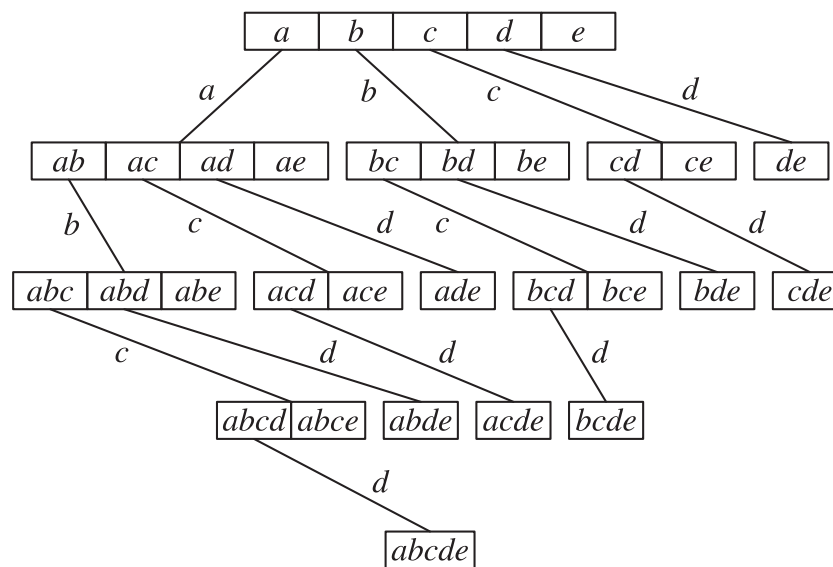
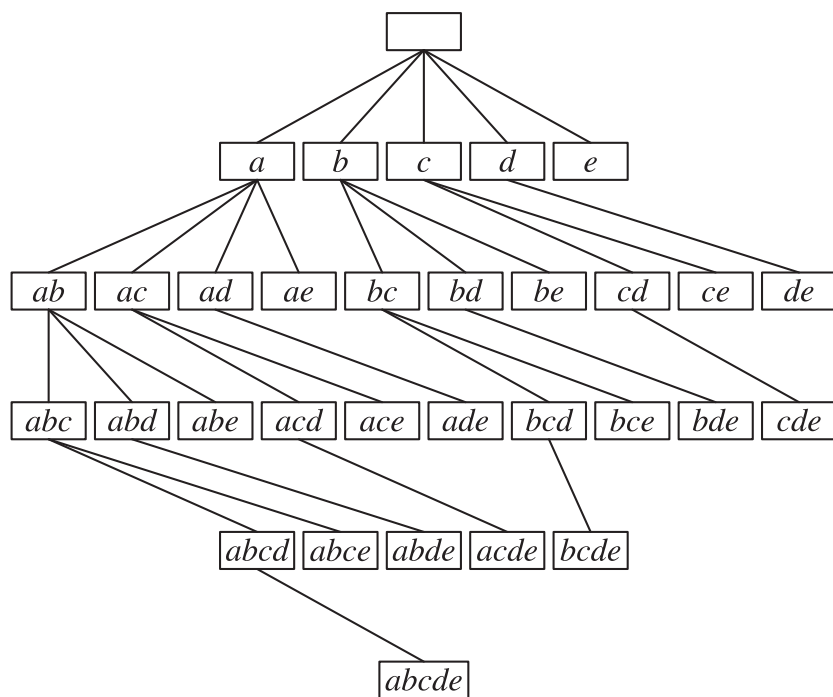
- How can we formally assign unique parents?
- How can we make sure that we generate only those extensions for which the item set that is extended is the chosen unique parent?

Unique Parents and Prefix Trees

Item sets sharing the same longest proper prefix are siblings, because they have the same unique parent.

This allows us to represent the unique parent tree as a **prefix tree** or **trie**.

Canonical parent tree and corresponding prefix tree for five items:



Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

a: 7	b: 3	c: 7	d: 6	e: 7
------	------	------	------	------

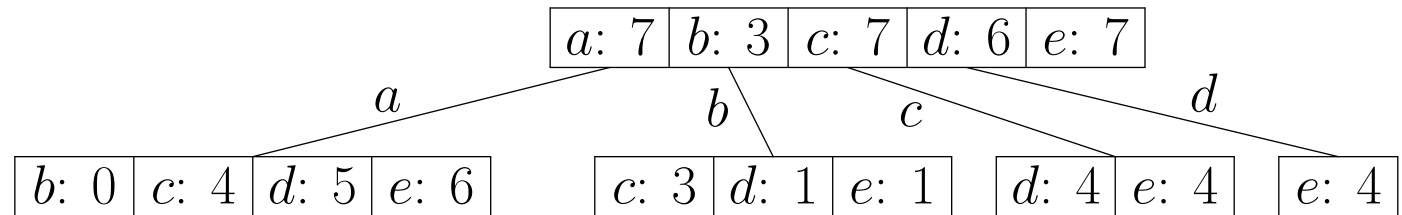
Example transaction database with 5 items and 10 transactions.

Minimum support: 30%, i.e., at least 3 transactions must contain the item set.

All one item sets are frequent → full second level is needed.

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

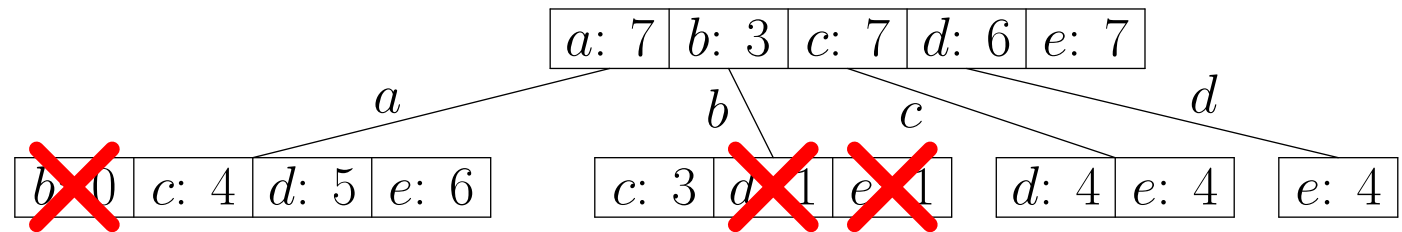


Determining the support of item sets: For each item set traverse the database and count the transactions that contain it (highly inefficient).

Better: Traverse the tree for each transaction and find the item sets it contains (efficient: can be implemented as a simple doubly recursive procedure).

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



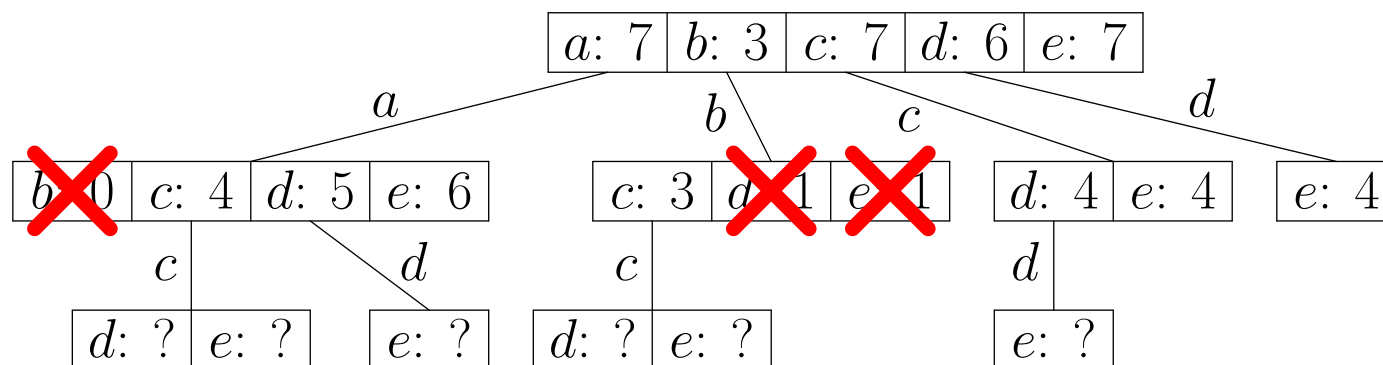
Minimum support: 30%, i.e., at least 3 transactions must contain the item set.

Infrequent item sets: {a, b}, {b, d}, {b, e}.

The subtrees starting at these item sets can be pruned.

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



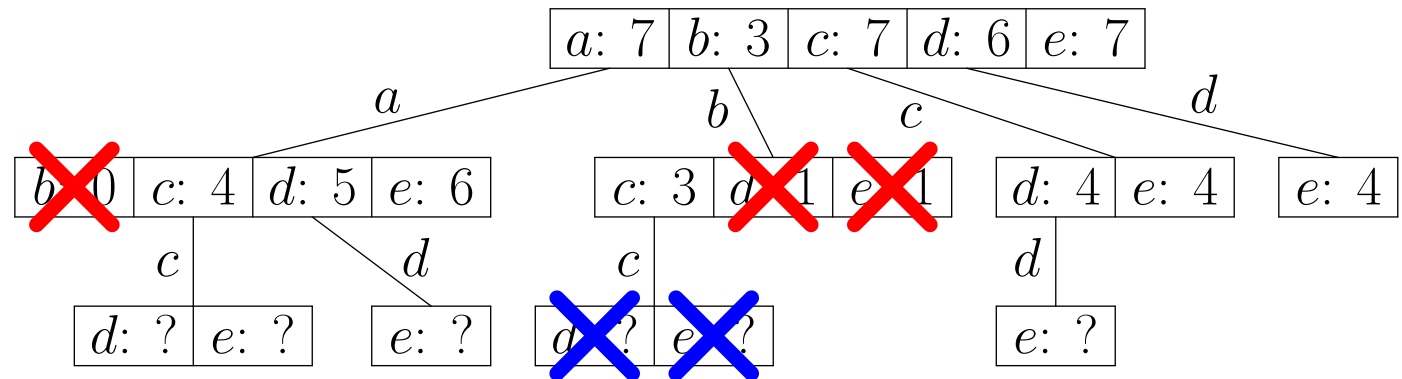
Generate candidate item sets with 3 items (parents must be frequent).

Before counting, check whether the candidates contain an infrequent item set.

- An item set with k items has k subsets of size $k - 1$.
- The parent is only one of these subsets.

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

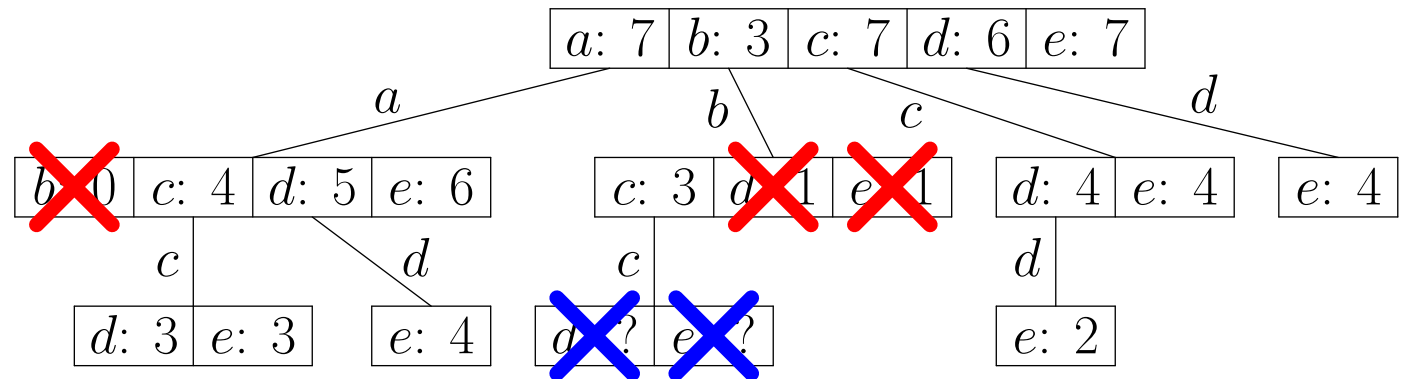


The item sets $\{b, c, d\}$ and $\{b, c, e\}$ can be pruned, because

- $\{b, c, d\}$ contains the infrequent item set $\{b, d\}$ and
- $\{b, c, e\}$ contains the infrequent item set $\{b, e\}$.

Apriori: Levelwise Search

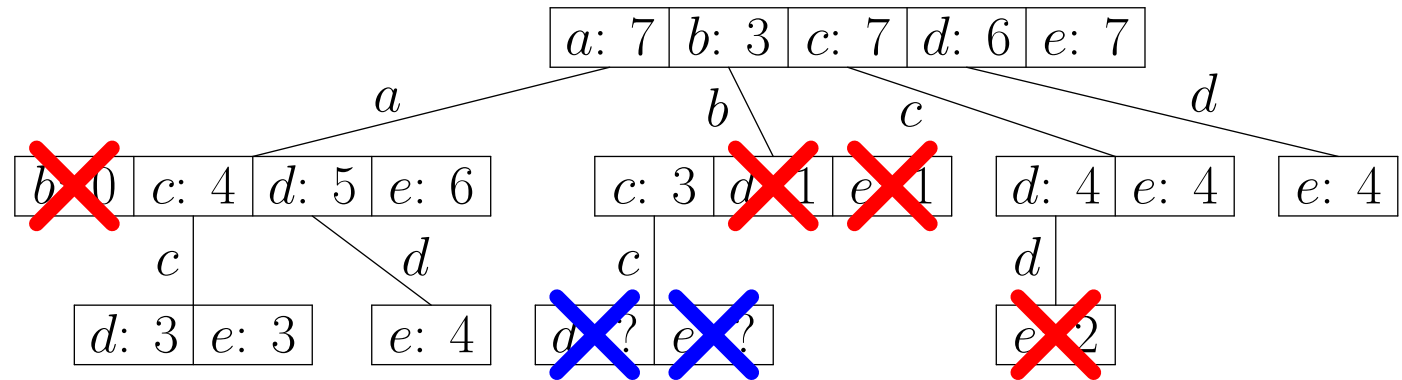
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



Only the remaining four item sets of size 3 are evaluated.

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

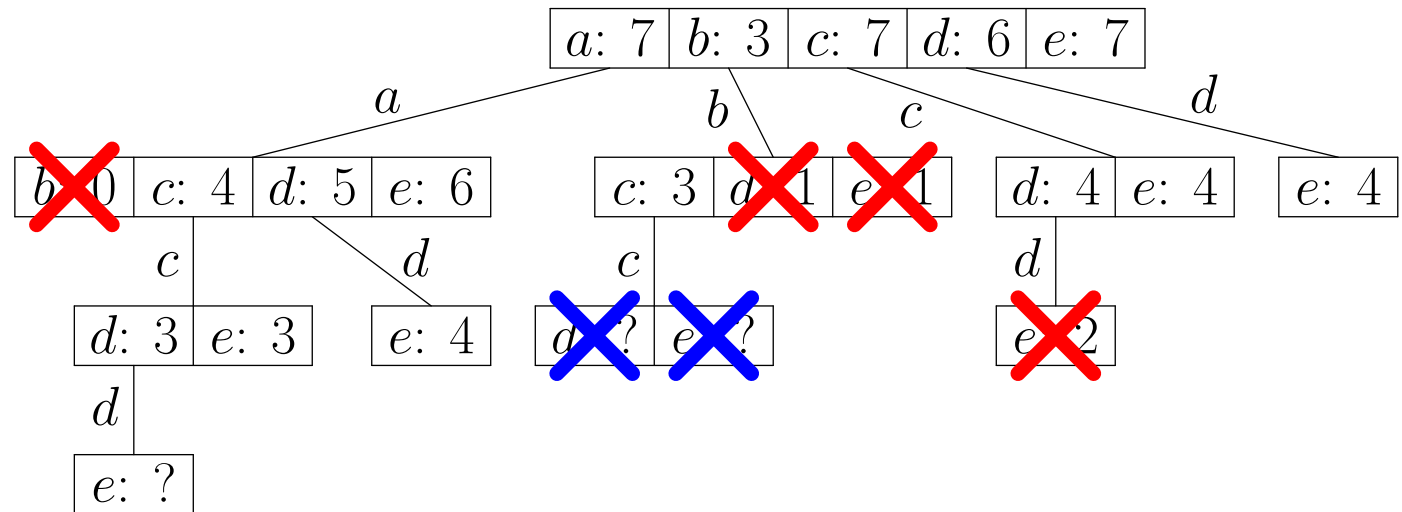


Minimum support: 30%, i.e., at least 3 transactions must contain the item set.

Infrequent item set: {c, d, e}.

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

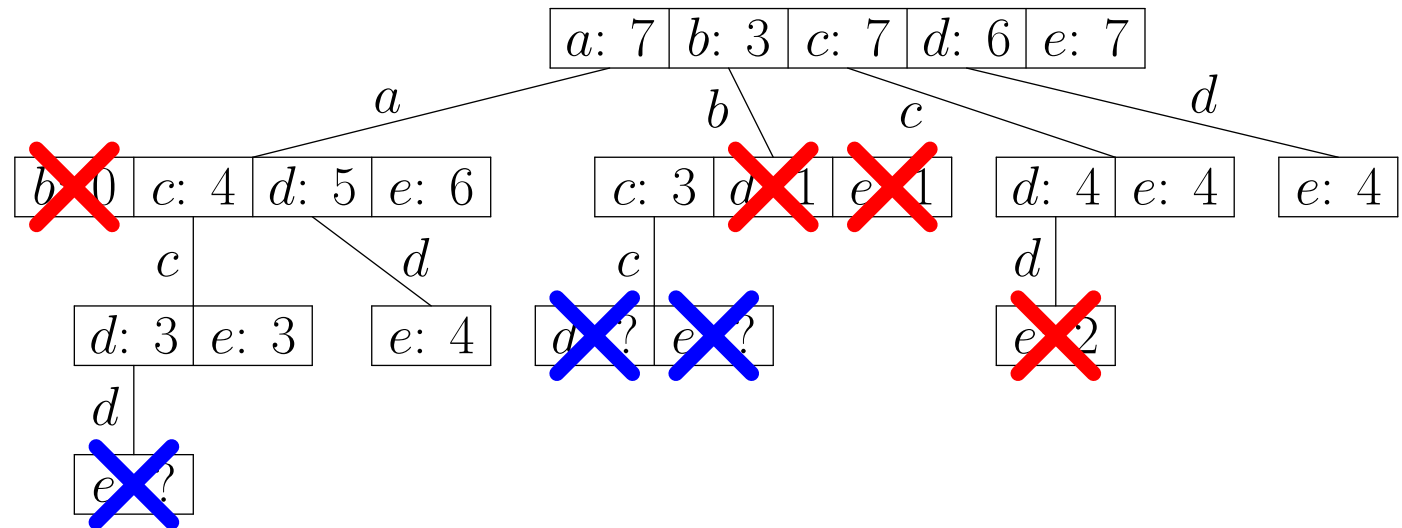


Generate candidate item sets with 4 items (parents must be frequent).

Before counting, check whether the candidates contain an infrequent item set.

Apriori: Levelwise Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



The item set {a, c, d, e} can be pruned,
because it contains the infrequent item set {c, d, e}.

Consequence: No candidate item sets with four items.

Fourth access to the transaction database is not necessary.

Apriori: Node Organization 1

Idea: Optimize the organization of the counters and the child pointers.

Direct Indexing:

Each node is a simple vector (array) of counters.

An item is used as a direct index to find the counter.

Advantage: Counter access is extremely fast.

Disadvantage: Memory usage can be high due to “gaps” in the index space.

Sorted Vectors:

Each node is a vector (array) of item/counter pairs.

A binary search is necessary to find the counter for an item.

Advantage: Memory usage may be smaller, no unnecessary counters.

Disadvantage: Counter access is slower due to the binary search.

Apriori: Node Organization 2

Hash Tables:

Each node is a vector (array) of item/counter pairs (closed hashing).

The index of a counter is computed from the item code.

Advantage: Faster counter access than with binary search.

Disadvantage: Higher memory usage than sorted vectors (pairs, fill rate).
The order of the items cannot be exploited.

Child Pointers:

The deepest level of the item set tree does not need child pointers.

Fewer child pointers than counters are needed.

→ It pays to represent the child pointers in a separate array.

The sorted array of item/counter pairs can be reused for a binary search.

Apriori: Item Coding

Items are coded as consecutive integers starting with 0 (needed for the direct indexing approach).

The size and the number of the “gaps” in the index space depends on how the items are coded.

Idea: It is plausible that frequent item sets consist of frequent items.

- Sort the items w.r.t. their frequency (group frequent items).
- Sort descendingly: Prefix tree has fewer nodes.
- Sort ascendingly: There are fewer and smaller index “gaps”.
- Empirical evidence: sorting ascendingly is better.

Extension: Sort items w.r.t. the sum of the sizes of the transactions that cover them.

- Empirical evidence: Better than simple item frequencies.

Apriori: Recursive Counting

The items in a transaction are sorted (ascending item codes).

Processing a transaction is then a *doubly recursive procedure*.

To process a transaction for a node of the item set tree:

- Go to the child corresponding to the first item in the transaction and count the remainder of the transaction recursively for that child.
(In the currently deepest level of the tree we increment the counter corresponding to the item instead of going to the child node.)
- Discard the first item of the transaction and process it recursively for the node itself.

Optimizations:

- Directly skip all items preceding the first item in the node.
- Abort the recursion if the first item is beyond the last one in the node.
- Abort the recursion if a transaction is too short to reach the deepest level.

Apriori: Transaction Representation

Direct Representation:

Each transaction is represented as an array of items.

The transactions are stored in a simple list.

Organization as a Prefix Tree:

The items in each transaction are sorted.

Transactions with the same prefix are grouped together.

Advantage: a common prefix is processed only once.

Gains from this organization depend on how the items are coded:

- Common transaction prefixes are more likely if the items are sorted with descending frequency.

Summary Apriori

Basic Processing Scheme

Breadth-first/levelwise traversal of the subset lattice.

Candidates are formed by merging item sets that differ in only one item.

Support counting is done with a doubly recursive procedure.

Advantages

“Perfect” pruning of infrequent candidate item sets (with infrequent subsets).

Disadvantages

Can require a lot of memory (since all frequent item sets are represented).

Support counting takes very long for large transactions.

Software

<http://www.borgelt.net/apriori.html>

Depth-First Search and Conditional Databases

In contrast to the levelwise search of the Apriori algorithm, the **Eclat Algorithm** executes a depth-first search in the prefix tree.

This depth-first search can also be seen as a **divide-and-conquer scheme**:

- Let the item order be $a < b < c \dots$
- Restrict the transaction vector to those transactions that contain a .
This is the **conditional database** for the prefix a .

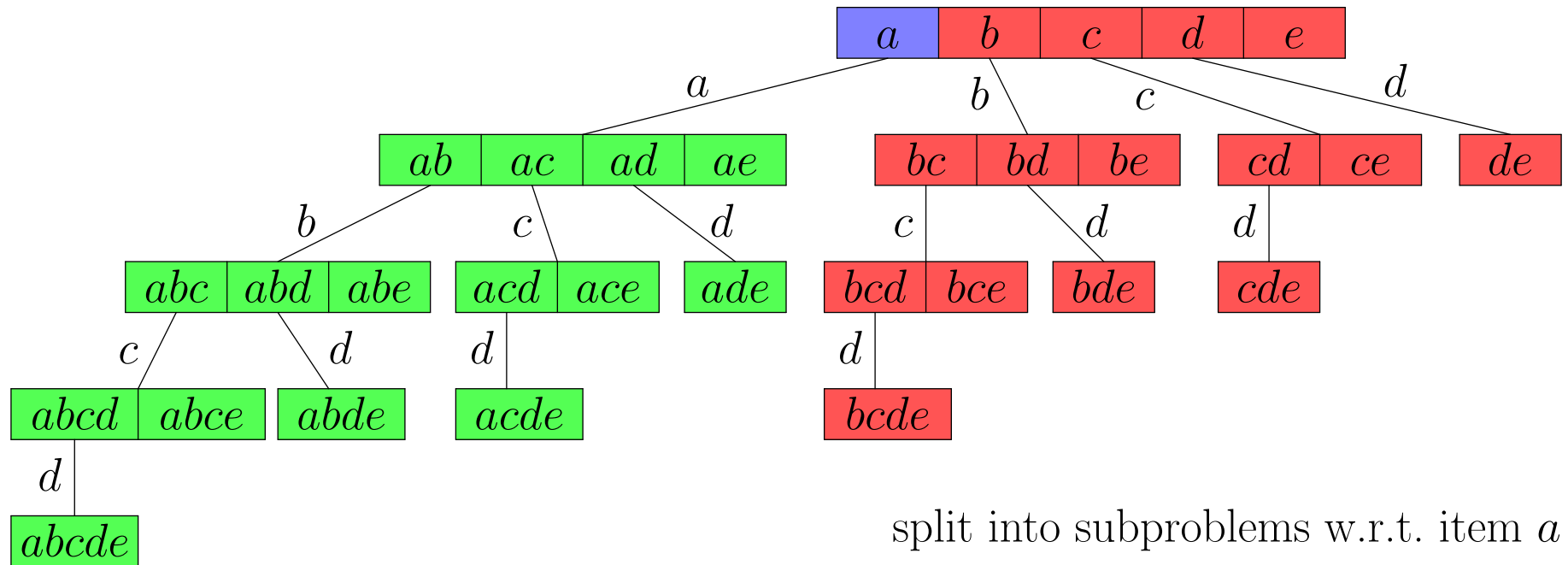
Recursively search this conditional database for frequent item sets and add the prefix a to all frequent item sets found in the recursion.

- Remove the item a from the transactions in the full transaction vector.
This is the **conditional database** for item sets without a .

Recursively search this conditional database for frequent item sets.

With this scheme only frequent one-element item sets have to be determined. Larger item sets result from adding possible prefixes.

Depth-First Search and Conditional Databases



split into subproblems w.r.t. item a

blue : item set consisting of only item a .

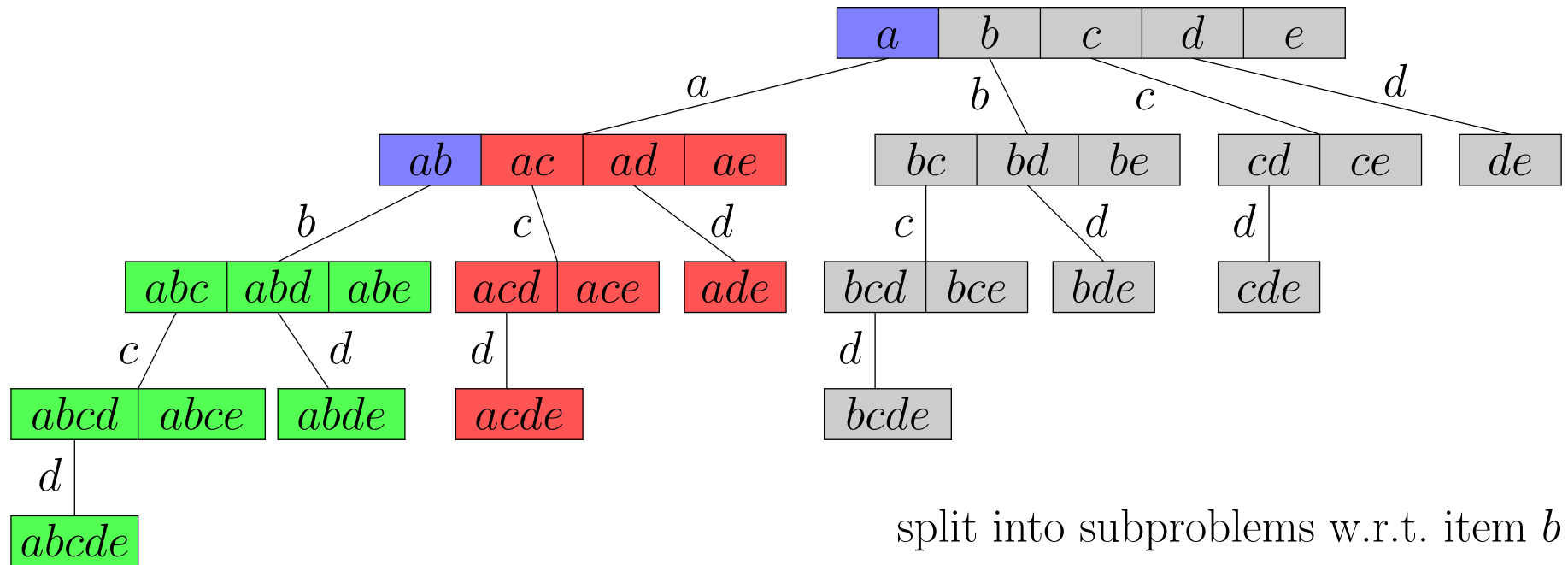
green: item sets containing item a (and at least one other item).

red : item sets not containing item a (but at least one other item).

green: database with transactions containing a .

red : database with all transactions, but with item a removed.

Depth-First Search and Conditional Databases



blue : item sets $\{a\}$ and $\{a, b\}$.

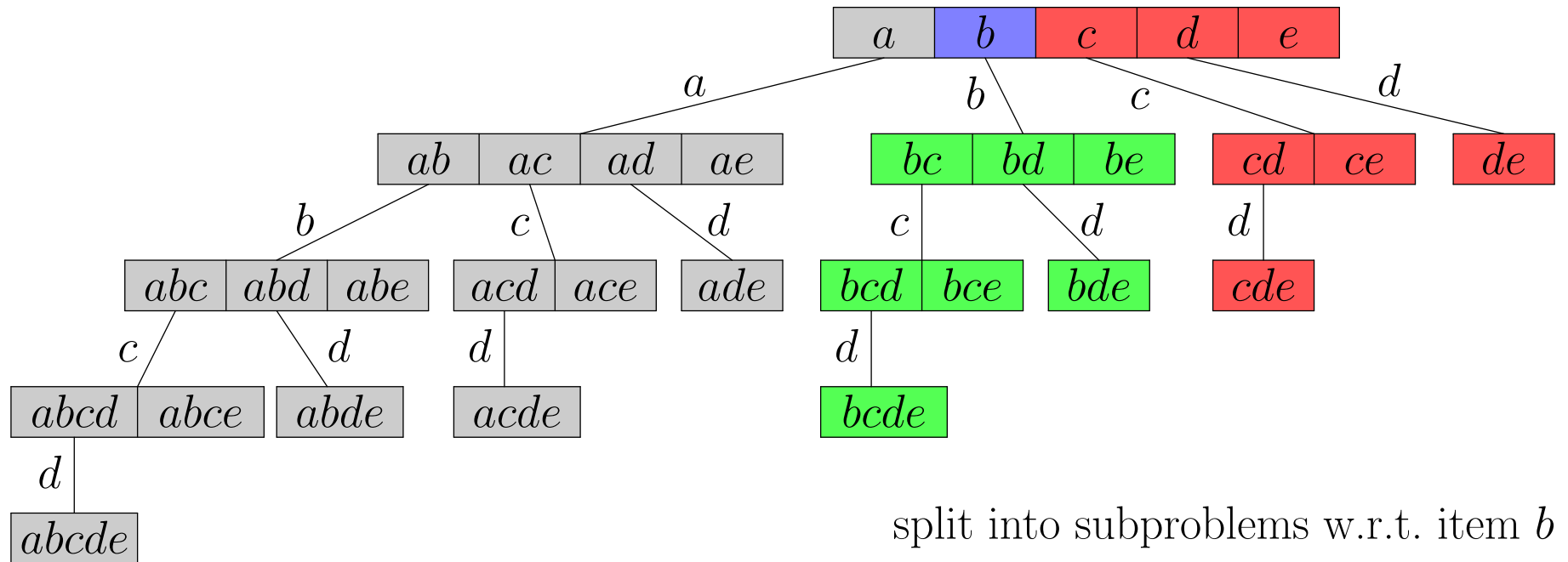
green: item sets containing items a and b (and at least one other item).

red : item sets containing item a , but not item b .

green: database with transactions containing a and b .

red : database with transactions containing a , but with b removed.

Depth-First Search and Conditional Databases



blue : item set consisting of only item b .

green: item sets containing item b , but not item a .

red : item sets containing neither item a nor item b .

green: database with transactions containing b , but not a .

red : database with all transactions, but with a and b removed.

The Eclat Algorithm

[Zaki, Parthasarathy, Ogihara, and Li 1997]

Eclat: Basic Ideas

The item sets are checked in lexicographic order (depth-first traversal of the prefix tree).

Eclat generates more candidate item sets than Apriori, because it does not store the support of all visited item sets.

Eclat uses a *vertical transaction representation* (see next slide for details).

No subset tests and no subset generation is needed for the support computation.

Eclat: Transaction Representation

The Apriori algorithm uses a **horizontal transaction representation**: each transaction is an array of the contained items.

- Note that the alternative prefix tree organization is still an essentially *horizontal* representation.


The Eclat algorithm uses a **vertical transaction representation**:

- For each item a **transaction list** is created.
- The transaction list of item a indicates the transactions that contain it, that is, it represents its *cover* $K_T(\{a\})$.
- Advantage: the transaction list for a pair of items can be computed by intersecting the transaction lists of the individual items.
- Generally, a vertical transaction representation can exploit

$$\forall I, J \subseteq A : K_T(I \cup J) = K_T(I) \cap K_T(J).$$

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

a: 7	b: 3	c: 7	d: 6	e: 7
				

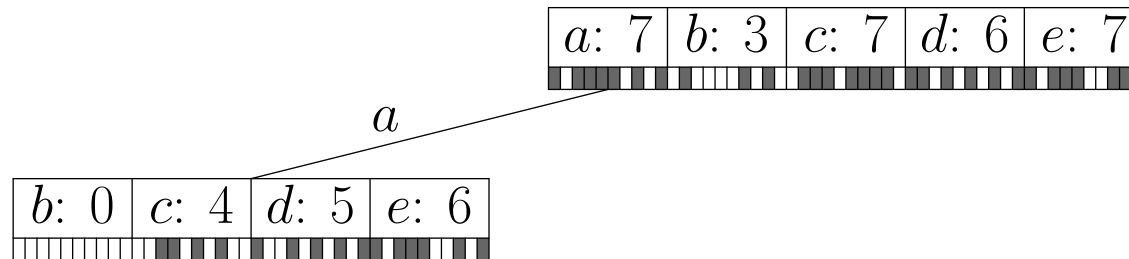
Form a transaction list for each item. Here: bit vector representation.

- grey: item is contained in transaction
- white: item is not contained in transaction

Transaction database is needed only once (for the single item transaction lists).

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



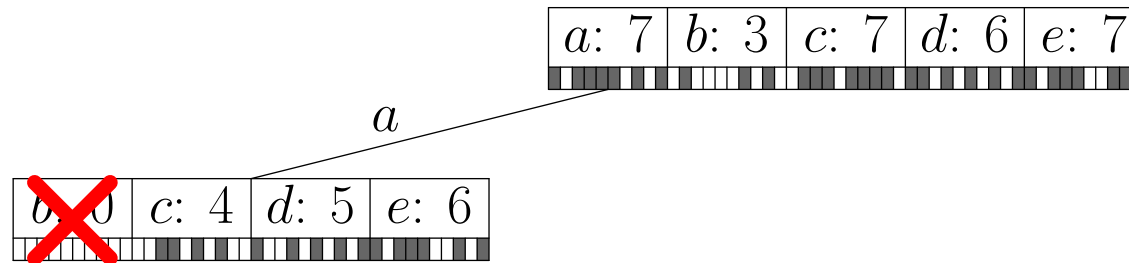
Intersect the transaction list for item a with the transaction lists of all other items (*conditional database*).

Count the number of set bits (number of containing transactions).

The item set $\{a, b\}$ is infrequent and can be pruned.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



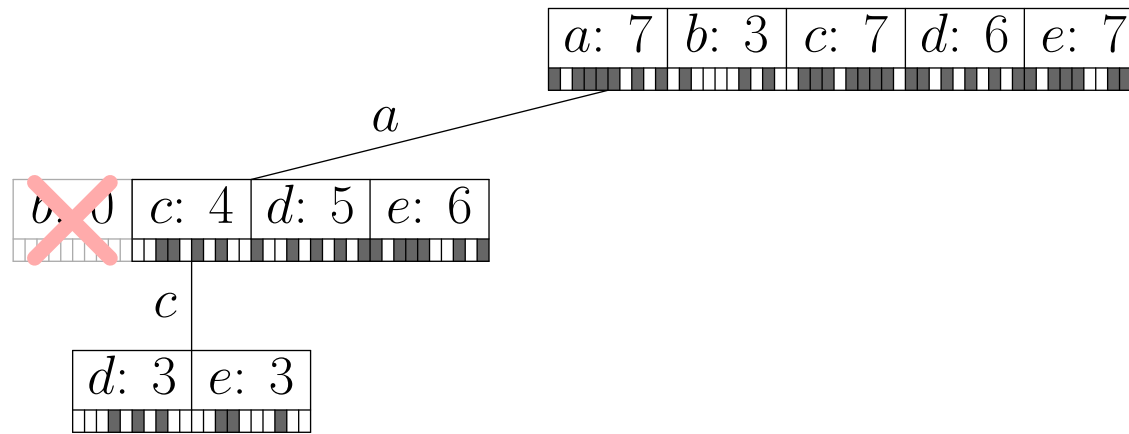
Intersect the transaction list for item a
with the transaction lists of all other items (*conditional database*).

Count the number of set bits (number of containing transactions).

The item set $\{a, b\}$ is infrequent and can be pruned.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

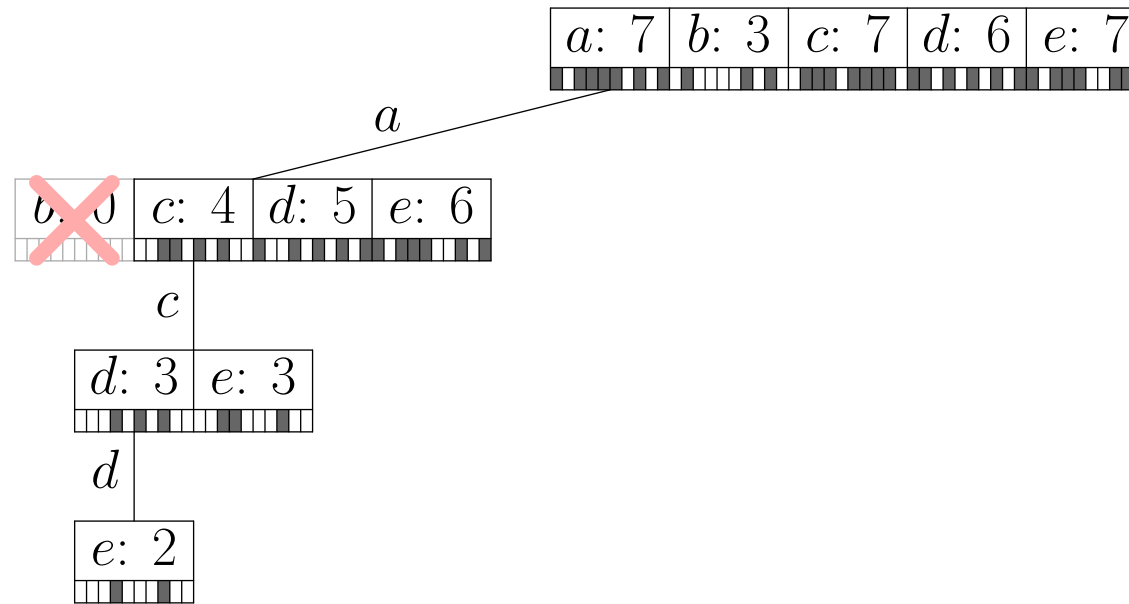


Intersect the transaction list for {a, c}
with the transaction lists of {a, x}, $x \in \{d, e\}$.

Result: Transaction lists for the item sets {a, c, d} and {a, c, e}.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



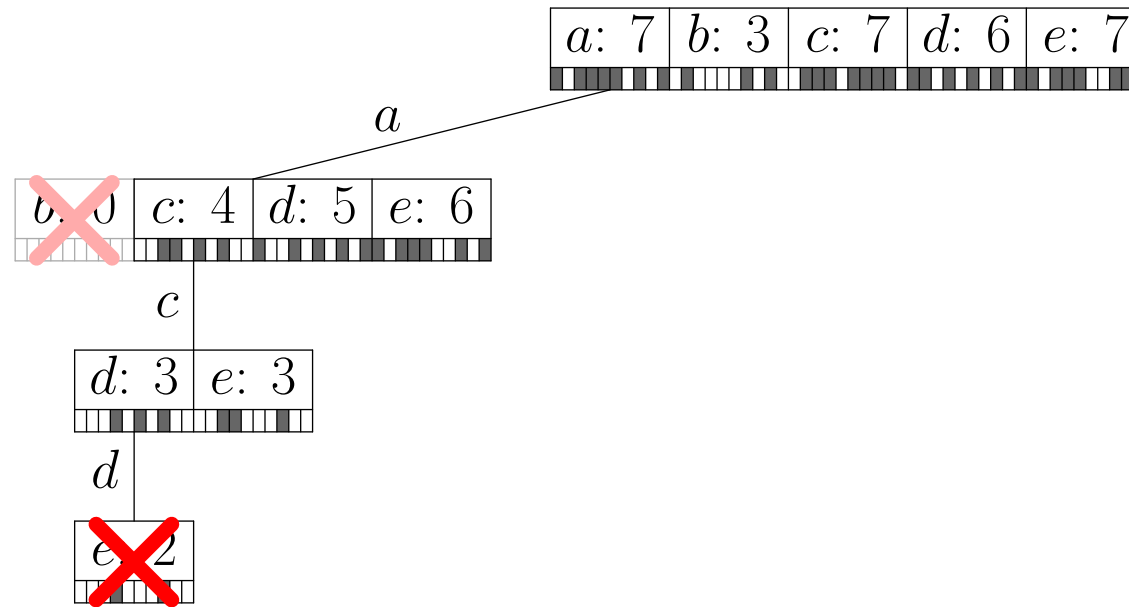
Intersect the transaction list for {a, c, d} and {a, c, e}.

Result: Transaction list for the item set {a, c, d, e}.

With Apriori this item set could be pruned before counting, because it was known that {c, d, e} is infrequent.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



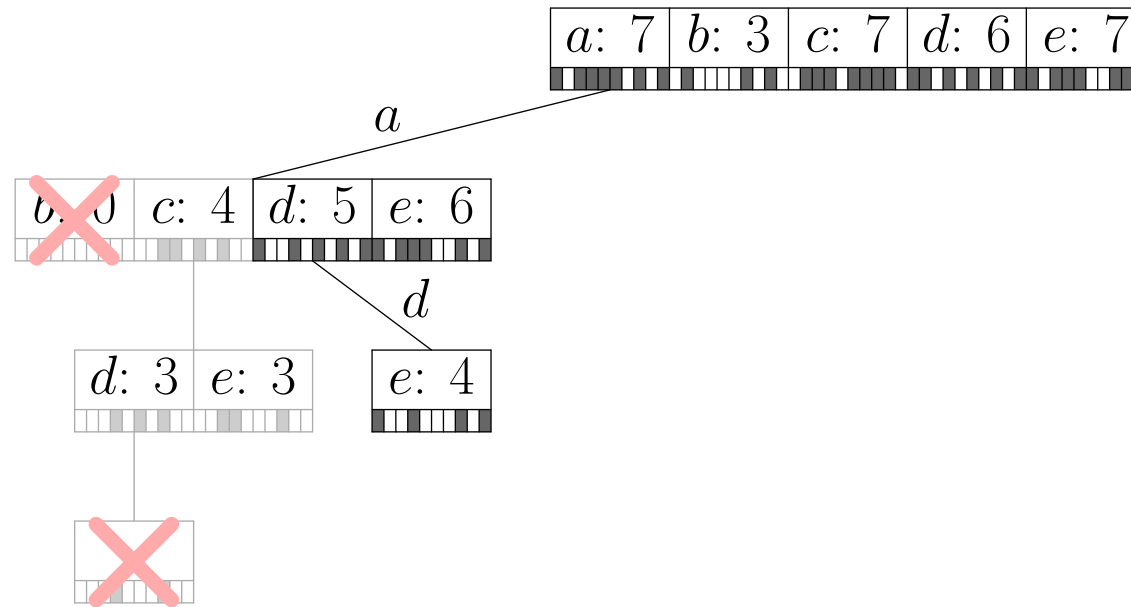
Intersect the transaction list for {a, c, d} and {a, c, e}.

Result: Transaction list for the item set {a, c, d, e}.

With Apriori this item set could be pruned before counting, because it was known that {c, d, e} is infrequent.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

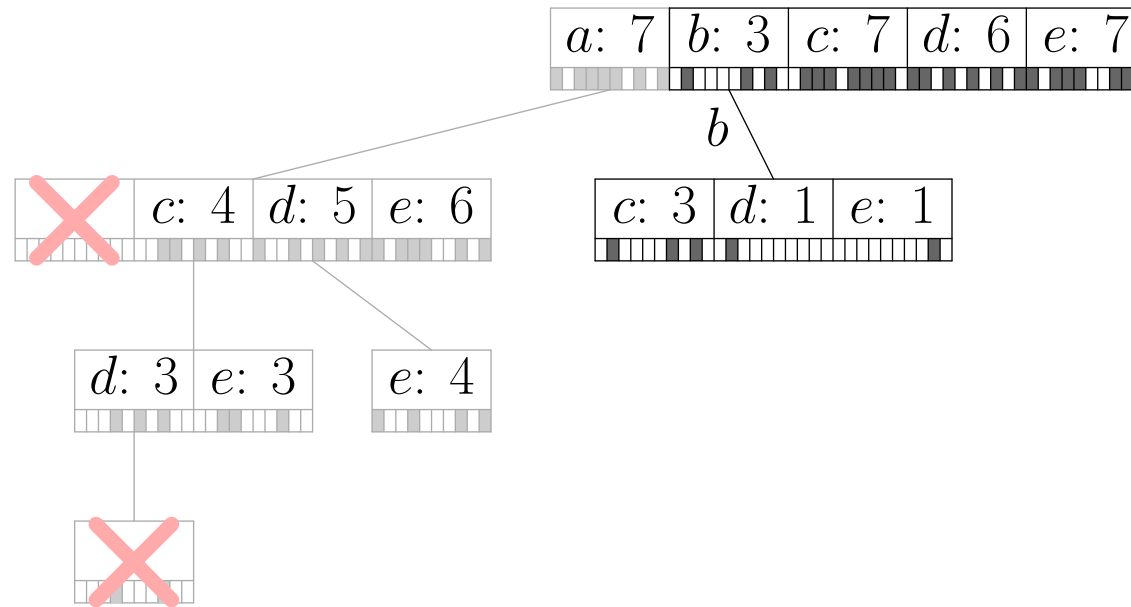


Backtrack to the second level of the search tree and intersect the transaction list for $\{a, d\}$ and $\{a, e\}$.

Result: Transaction list for $\{a, d, e\}$.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



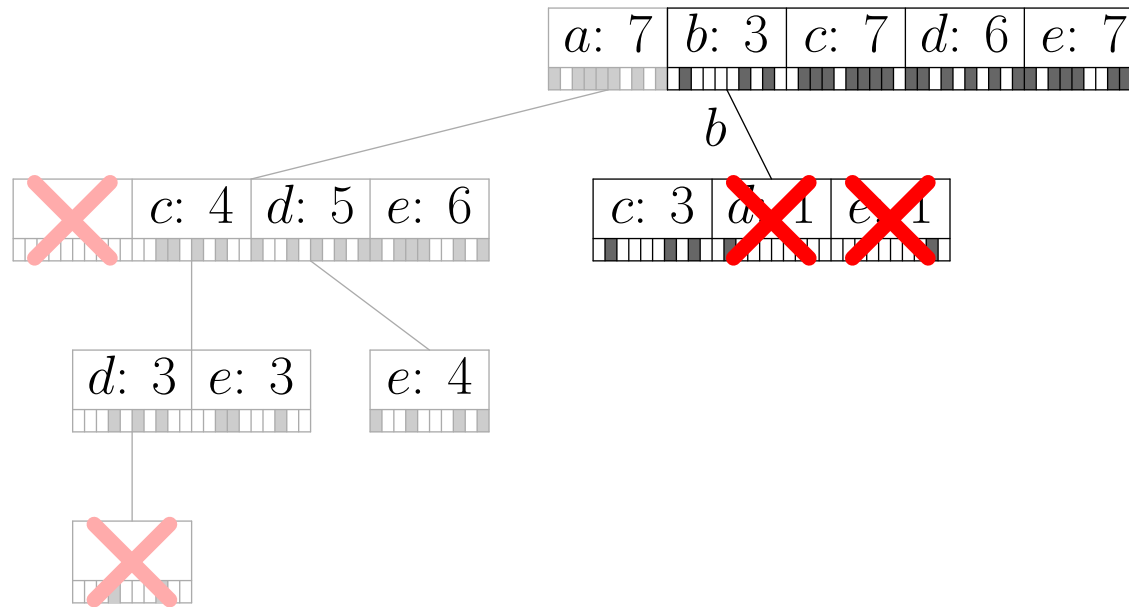
Backtrack to the first level of the search tree and intersect the transaction list for b with the transaction lists for c , d , and e .

Result: Transaction lists for the item sets $\{b, c\}$, $\{b, d\}$, and $\{b, e\}$.

Only one item set with sufficient support \rightarrow prune all subtrees.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



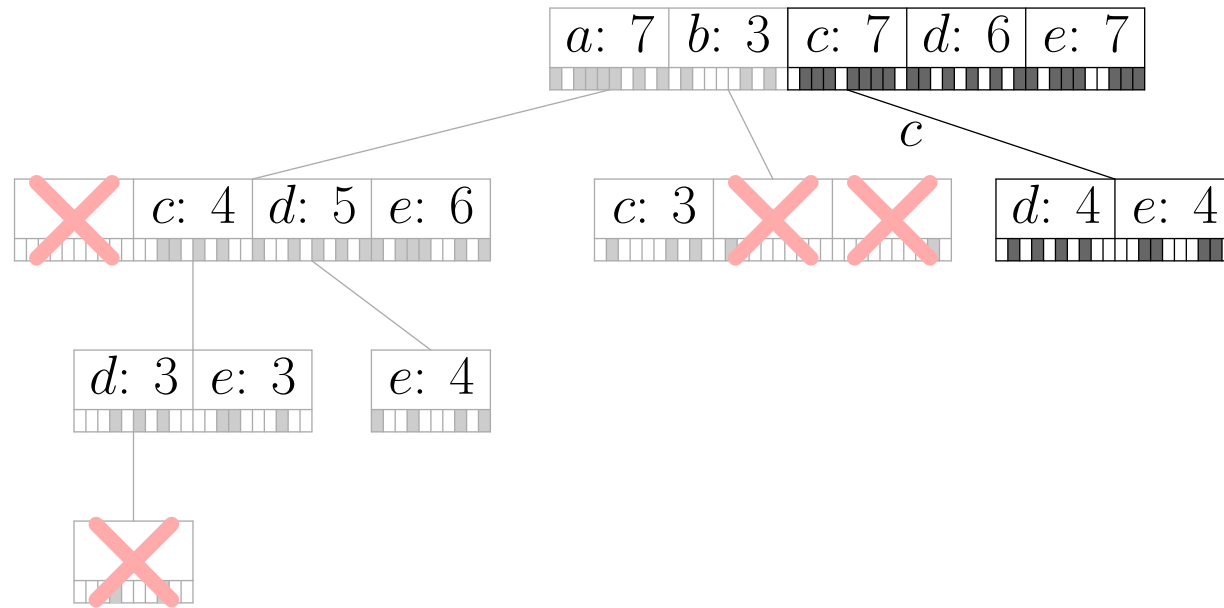
Backtrack to the first level of the search tree and intersect the transaction list for b with the transaction lists for c , d , and e .

Result: Transaction lists for the item sets $\{b, c\}$, $\{b, d\}$, and $\{b, e\}$.

Only one item set with sufficient support \rightarrow prune all subtrees.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

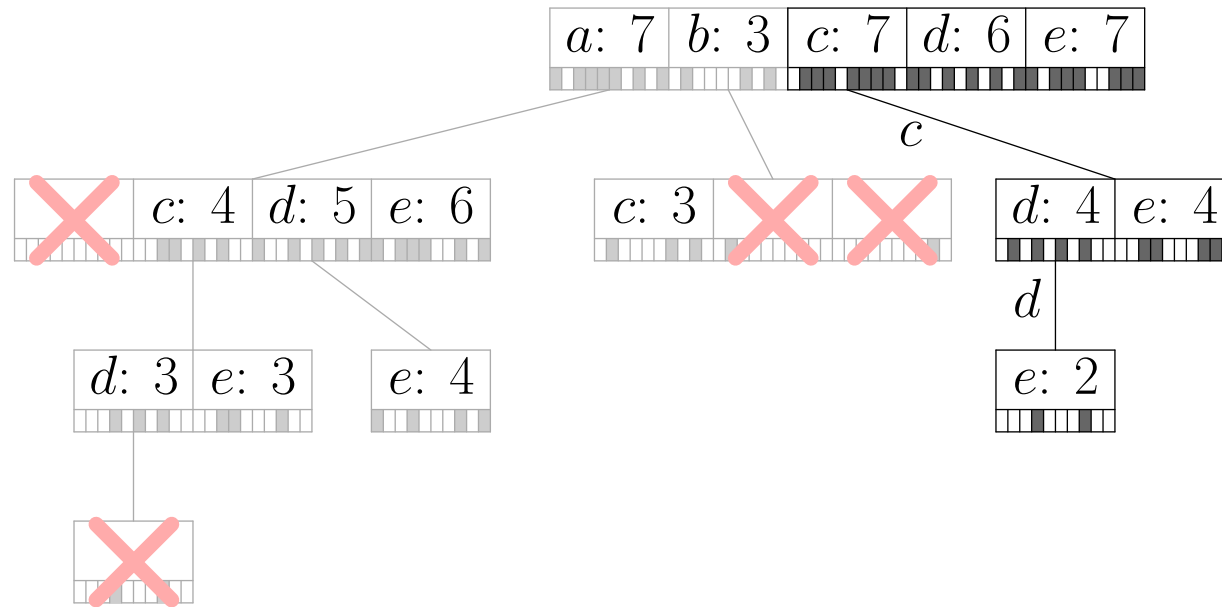


Backtrack to the first level of the search tree and intersect the transaction list for c with the transaction lists for d and e .

Result: Transaction lists for the item sets $\{c, d\}$ and $\{c, e\}$.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



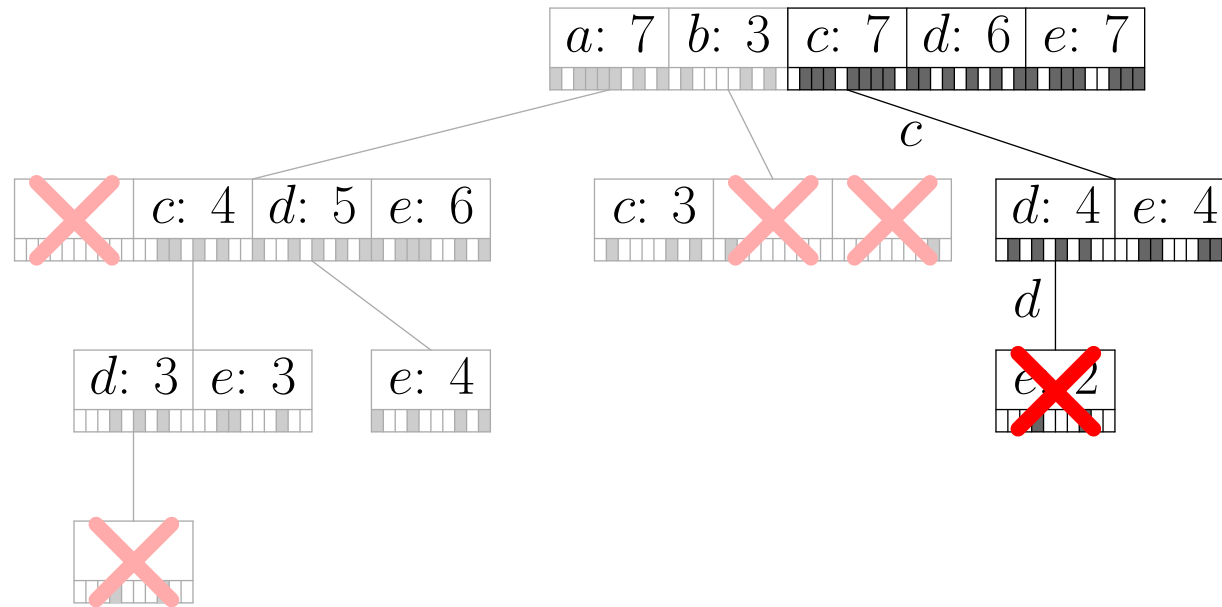
Intersect the transaction list for {c, d} and {c, e}.

Result: Transaction list for {c, d, e}.

Infrequent item set: {c, d, e}.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



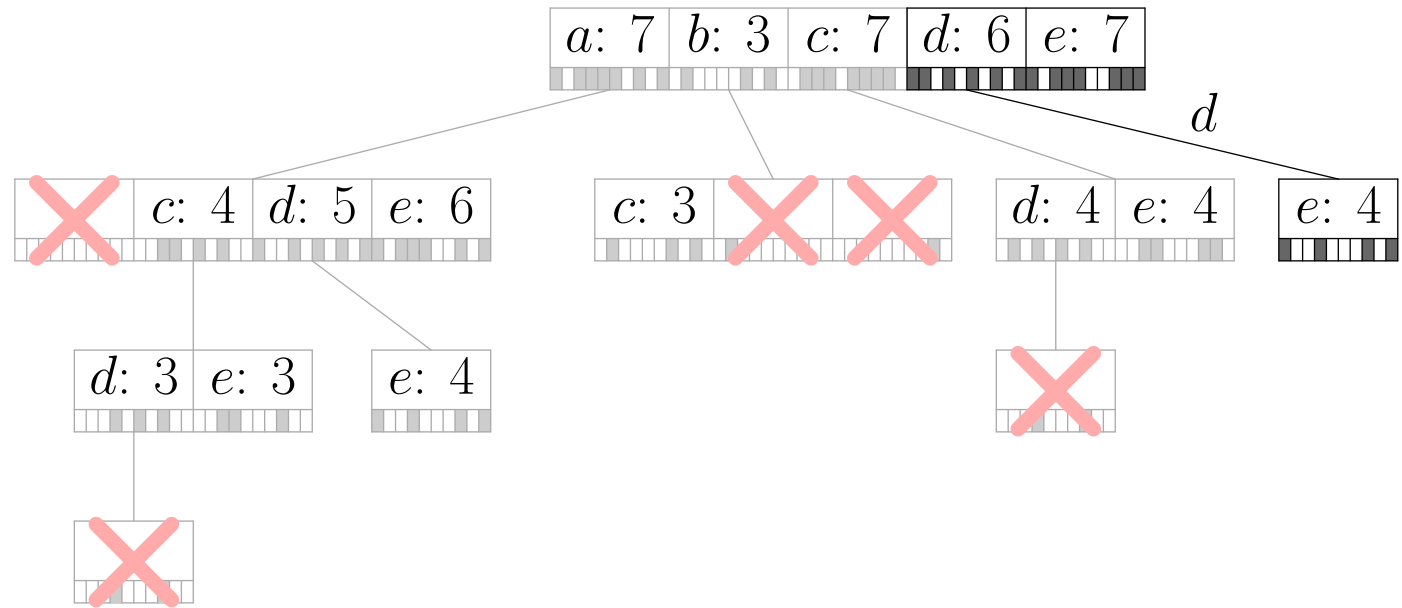
Intersect the transaction list for {c, d} and {c, e}.

Result: Transaction list for {c, d, e}.

Infrequent item set: {c, d, e}.

Eclat: Depth-First Search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



Backtrack to the first level of the search tree and intersect the transaction list for d with the transaction list for e .

Result: Transaction list for the item set $\{d, e\}$.

With this step the search is finished.

Eclat: Bit Matrices and Item Coding

Bit Matrices

Represent transactions as a bit matrix:

- Each column corresponds to an item.
- Each row corresponds to a transaction.

Normal and sparse representation of bit matrices:

- Normal: one memory bit per matrix bit, zeros represented.
- Sparse: lists of column indices of set bits (transaction lists).

Which representation is preferable depends on the ratio of set bits to cleared bits.

Item Coding

Sorting the item descendingly w.r.t. their frequency (individual or transaction size sum) leads to a better structure of the search tree.

Basic Processing Scheme

Depth-first traversal of the prefix tree.

Data is represented as lists of transaction ids (one per item).

Support counting is done by intersecting lists of transaction ids.

Advantages

Depth-first search reduces memory requirements.

Usually (considerably) faster than Apriori.

Disadvantages

Difficult to execute for modern processors (branch prediction).

Software

<http://www.borgelt.net/eclat.html>

Additional Frequent Item Set Filtering

General problem of frequent item set mining:

The number of frequent item sets, even the number of closed or maximal item sets, can exceed the number of transactions in the database by far.

Therefore: Additional filtering is necessary to find the "relevant" or "interesting" frequent item sets.

General idea: **Compare support to expectation.**

- Item sets consisting of items that appear frequently are likely to have a high support.
- However, this is not surprising:
we expect this even if the occurrence of the items is independent.
- Additional filtering should remove item sets with a support close to the support expected from an independent occurrence.

Additional Frequent Item Set Filtering

Full Independence

Evaluate item sets with

$$e_{\text{fi}}(I) = \frac{s_T(I) \cdot n^{|I|-1}}{\prod_{a \in I} s_T(\{a\})} = \frac{\hat{p}_T(I)}{\prod_{a \in I} \hat{p}_T(\{a\})}.$$

an require a minimum value for this measure.

(\hat{p}_T is the probability estimate based on T .)

Assumes full independence of the items in order to form an expectation about the support of an item set.

Advantage: Can be computed from only the support of the item set and the support values of the individual items.

Disadvantage: If some item set I scores high on this measure, then all $J \supset I$ are also likely to score high, even if the items in $J - I$ are independent of I .

Additional Frequent Item Set Filtering

Incremental Independence

Evaluate item sets with

$$q_{ii}(I) = \min_{a \in I} \frac{n s_T(I)}{s_T(I - \{a\}) \cdot s_T(\{a\})} = \min_{a \in I} \frac{\hat{p}_T(I)}{\hat{p}_T(I - \{a\}) \cdot \hat{p}_T(\{a\})}.$$

an require a minimum value for this measure.

(\hat{p}_T is the probability estimate based on T .)

Advantage: If I contains independent items,
the minimum ensures a low value.

Disadvantages: We need to know the support values of all subsets $I - \{a\}$.

If there exist high scoring independent subsets I_1 and I_2
with $|I_1| > 1$, $|I_2| > 1$, $I_1 \cap I_2 = \emptyset$ and $I_1 \cup I_2 = I$,
the item set I still receives a high evaluation.

Additional Frequent Item Set Filtering

Subset Independence

Evaluate item sets with

$$q_{\text{si}}(I) = \min_{J \subset I, J \neq \emptyset} \frac{n s_T(I)}{s_T(I - J) \cdot s_T(J)} = \min_{J \subset I, J \neq \emptyset} \frac{\hat{p}_T(I)}{\hat{p}_T(I - J) \cdot \hat{p}_T(J)}.$$

an require a minimum value for this measure.

(\hat{p}_T is the probability estimate based on T .)

Advantage: Detects all cases where a decomposition is possible and evaluates them with a low value.

Disadvantages: We need to know the support values of all proper subsets J .

Improvement: Use incremental independence and in the minimum consider only items $\{a\}$ for which $I - \{a\}$ has been evaluated high.

This captures subset independence “incrementally”.

Summary Frequent Item Set Mining

Algorithms for frequent item set mining differ in:

- the **traversal order** of the prefix tree:
(breadth-first/levelwise versus depth-first traversal)
- the **transaction representation**:
horizontal (item arrays) versus *vertical* (transaction lists)
versus *specialized data structures* like FP-trees
- the **types of frequent item sets** found:
frequent versus *closed* versus *maximal item sets*
(additional pruning methods for closed and maximal item sets)

Additional filtering is necessary to reduce the size of the output.

Association Rules: Basic Notions

Often found patterns are expressed as **association rules**, for example:

If a customer buys **bread** and **wine**,
then she/he will probably also buy **cheese**.

Formally, we consider rules of the form $X \rightarrow Y$,
with $X, Y \subseteq A$ and $X \cap Y = \emptyset$.

Support of a Rule $X \rightarrow Y$:

Either: $\varsigma_T(X \rightarrow Y) = \sigma_T(X \cup Y)$ (more common: rule is correct)

Or: $\varsigma_T(X \rightarrow Y) = \sigma_T(X)$ (more plausible: rule is applicable)

Confidence of a Rule $X \rightarrow Y$:

$$c_T(X \rightarrow Y) = \frac{\sigma_T(X \cup Y)}{\sigma_T(X)} = \frac{s_T(X \cup Y)}{s_T(X)} = \frac{s_T(I)}{s_T(X)}$$

The confidence can be seen as an estimate of $P(Y \mid X)$.

Association Rules: Formal Definition

Given:

a set $A = \{a_1, \dots, a_m\}$ of items,

a vector $T = (t_1, \dots, t_n)$ of transactions over A ,

a real number ς_{\min} , $0 < \varsigma_{\min} \leq 1$, the **minimum support**,

a real number c_{\min} , $0 < c_{\min} \leq 1$, the **minimum confidence**.

Desired:

the set of all **association rules**, that is, the set

$$\mathcal{R} = \{R : X \rightarrow Y \mid \varsigma_T(R) \geq \varsigma_{\min} \wedge c_T(R) \geq c_{\min}\}.$$

General Procedure:

Find the frequent item sets.

Construct rules and filter them w.r.t. ς_{\min} and c_{\min} .

Generating Association Rules

Which minimum support has to be used for finding the frequent item sets depends on the definition of the support of a rule:

- If $\varsigma_T(X \rightarrow Y) = \sigma_T(X \cup Y)$,
then $\sigma_{\min} = \varsigma_{\min}$ or equivalently $s_{\min} = \lceil n\varsigma_{\min} \rceil$.
- If $\varsigma_T(X \rightarrow Y) = \sigma_T(X)$,
then $\sigma_{\min} = \varsigma_{\min}c_{\min}$ or equivalently $s_{\min} = \lceil n\varsigma_{\min}c_{\min} \rceil$.

After the frequent item sets have been found, the rule construction then traverses all frequent item sets I and splits them into disjoint subsets X and Y ($X \cap Y = \emptyset$ and $X \cup Y = I$), thus forming rules $X \rightarrow Y$.

- Filtering rules w.r.t. confidence is always necessary.
- Filtering rules w.r.t. support is only necessary if $\varsigma_T(X \rightarrow Y) = \sigma_T(X)$.

Properties of the Confidence

From $\forall I : \forall J \subseteq I : s_T(I) \leq s_T(J)$ it obviously follows

$$\forall X, Y : \forall a \in X : \frac{s_T(X \cup Y)}{s_T(X)} \geq \frac{s_T(X \cup Y)}{s_T(X - \{a\})}$$

and therefore

$$\forall X, Y : \forall a \in X : c_T(X \rightarrow Y) \geq c_T(X - \{a\} \rightarrow Y \cup \{a\}).$$

That is: **Moving an item from the antecedent to the consequent cannot increase the confidence of a rule.**

As an immediate consequence we have

$$\forall X, Y : \forall a \in X : c_T(X \rightarrow Y) < c_{\min} \rightarrow c_T(X - \{a\} \rightarrow Y \cup \{a\}) < c_{\min}.$$

That is: **If a rule fails to meet the minimum confidence, no rules over the same item set and with a larger consequent need to be considered.**

Generating Association Rules

```
function rules (F); (* — generate association rules *)
  R := ∅; (* initialize the set of rules *)
  forall f ∈ F do begin (* traverse the frequent item sets *)
    m := 1; (* start with rule heads (consequents) *)
    Hm := ∪i∈f {{i}}; (* that contain only one item *)
    repeat (* traverse rule heads of increasing size *)
      forall h ∈ Hm do (* traverse the possible rule heads *)
        if  $\frac{s_T(f)}{s_T(f-h)} \geq c_{\min}$  (* if the confidence is high enough, *)
          then R := R ∪ {(f - h) → h}; (* add rule to the result *)
          else Hm := Hm - {h}; (* otherwise discard the head *)
        Hm+1 := candidates(Hm); (* create heads with one item more *)
        m := m + 1; (* increment the head item counter *)
      until Hm = ∅ or m ≥ |f|; (* until there are no more rule heads *)
    end; (* or antecedent would become empty *)
  return R; (* return the rules found *)
end; (* rules *)
```

Generating Association Rules

```
function candidates ( $F_k$ )      (* generate candidates with  $k + 1$  items *)
begin
   $E := \emptyset$ ;                (* initialize the set of candidates *)
  forall  $f_1, f_2 \in F_k$         (* traverse all pairs of frequent item sets *)
  with  $f_1 = \{a_1, \dots, a_{k-1}, a_k\}$  (* that differ only in one item and *)
  and  $f_2 = \{a_1, \dots, a_{k-1}, a'_k\}$  (* are in a lexicographic order *)
  and  $a_k < a'_k$  do begin      (* (the order is arbitrary, but fixed) *)
     $f := f_1 \cup f_2 = \{a_1, \dots, a_{k-1}, a_k, a'_k\}$ ; (* union has  $k + 1$  items *)
    if  $\forall a \in f : f - \{a\} \in F_k$  (* only if all subsets are frequent, *)
    then  $E := E \cup \{f\}$ ;      (* add the new item set to the candidates *)
  end;                          (* (otherwise it cannot be frequent) *)
  return  $E$ ;                    (* return the generated candidates *)
end (* candidates *)
```

Frequent Item Sets: Example

transaction vector

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}

frequent item sets

0 items	1 item	2 items	3 items
\emptyset : 100%	{a}: 70%	{a, c}: 40%	{a, c, d}: 30%
	{b}: 30%	{a, d}: 50%	{a, c, e}: 30%
	{c}: 70%	{a, e}: 60%	{a, d, e}: 40%
	{d}: 60%	{b, c}: 30%	
	{e}: 70%	{c, d}: 40%	
		{c, e}: 40%	
		{d, e}: 40%	

The minimum support is $s_{\min} = 3$ or $\sigma_{\min} = 0.3 = 30\%$ in this example.

There are $2^5 = 32$ possible item sets over $A = \{a, b, c, d, e\}$.

There are 16 frequent item sets (but only 10 transactions).

Generating Association Rules

Example: $I = \{a, c, e\}$, $X = \{c, e\}$, $Y = \{a\}$.

$$c_T(c, e \rightarrow a) = \frac{s_T(\{a, c, e\})}{s_T(\{c, e\})} = \frac{30\%}{40\%} = 75\%$$

Minimum confidence: 80%

association rule	support of all items	support of antecedent	confidence
$b \rightarrow c$:	30%	30%	100%
$d \rightarrow a$:	50%	60%	83.3%
$e \rightarrow a$:	60%	70%	85.7%
$a \rightarrow e$:	60%	70%	85.7%
$d, e \rightarrow a$:	40%	40%	100%
$a, d \rightarrow e$:	40%	50%	80%

Support of an Association Rule

The two rule support definitions are not equivalent:

transaction vector

- 1: $\{a, c, e\}$
- 2: $\{b, d\}$
- 3: $\{b, c, d\}$
- 4: $\{a, e\}$
- 5: $\{a, b, c, d\}$
- 6: $\{c, e\}$
- 7: $\{a, b, d\}$
- 8: $\{a, c, d\}$

two association rules

association rule	support of all items	support of antecedent	confidence
$a \rightarrow c$	3 (37.5%)	5 (62.5%)	67.7%
$b \rightarrow d$	4 (50.0%)	4 (50.0%)	100.0%

Let the minimum confidence be $c_{\min} = 65\%$.

For $\varsigma_T(R) = \sigma(X \cup Y)$ and $3 < \varsigma_{\min} \leq 4$ only the rule $b \rightarrow d$ is generated, but not the rule $a \rightarrow c$.

For $\varsigma_T(R) = \sigma(X)$ there is no value ς_{\min} that generates only the rule $b \rightarrow d$, but not at the same time also the rule $a \rightarrow c$.

Additional Rule Filtering

Simple Measures

General idea: Compare $\hat{p}_T(Y | X) = c_T(X \rightarrow Y)$
and $\hat{p}_T(Y) = c_T(\emptyset \rightarrow Y) = \sigma_T(Y)$.

(Absolute) confidence difference to prior:

$$d_T(R) = |c_T(X \rightarrow Y) - \sigma_T(Y)|$$

(Absolute) difference of confidence quotient to 1:

$$q_T(R) = \left| 1 - \min \left\{ \frac{c_T(X \rightarrow Y)}{\sigma_T(Y)}, \frac{\sigma_T(Y)}{c_T(X \rightarrow Y)} \right\} \right|$$

Confidence to prior ratio (lift):

$$l_T(R) = \frac{c_T(X \rightarrow Y)}{\sigma_T(Y)}$$

Additional Rule Filtering

More Sophisticated Measures

Consider the 2×2 contingency table or the estimated probability table:

	$X \not\subseteq t$	$X \subseteq t$	
$Y \not\subseteq t$	n_{00}	n_{01}	$n_{0.}$
$Y \subseteq t$	n_{10}	n_{11}	$n_{1.}$
	$n_{.0}$	$n_{.1}$	$n_{..}$

	$X \not\subseteq t$	$X \subseteq t$	
$Y \not\subseteq t$	p_{00}	p_{01}	$p_{0.}$
$Y \subseteq t$	p_{10}	p_{11}	$p_{1.}$
	$p_{.0}$	$p_{.1}$	1

$n_{..}$ is the total number of transactions.

$n_{1.}$ is the number of transactions to which the rule is applicable.

n_{11} is the number of transactions for which the rule is correct.

It is $p_{ij} = \frac{n_{ij}}{n_{..}}$, $p_{i.} = \frac{n_{i.}}{n_{..}}$, $p_{.j} = \frac{n_{.j}}{n_{..}}$ for $i, j = 1, 2$.

General idea: Use measures for the strength of dependence of X and Y .

An Information-theoretic Evaluation Measure

Information Gain (Kullback and Leibler 1951, Quinlan 1986)

Based on Shannon Entropy $H = - \sum_{i=1}^n p_i \log_2 p_i$ (Shannon 1948)

$$\begin{aligned} I_{\text{gain}}(X, Y) &= \overbrace{H(Y)} - \overbrace{H(Y|X)} \\ &= - \sum_{i=1}^{k_Y} p_{i.} \log_2 p_{i.} - \sum_{j=1}^{k_X} p_{.j} \left(- \sum_{i=1}^{k_Y} p_{i|j} \log_2 p_{i|j} \right) \end{aligned}$$

$H(Y)$ Entropy of the distribution of Y

$H(Y|X)$ *Expected entropy* of the distribution of Y
if the value of the X becomes known

$H(Y) - H(Y|X)$ Expected entropy reduction or *information gain*

A Statistical Evaluation Measure

χ^2 Measure

Compares the actual joint distribution with a **hypothetical independent distribution**.

Uses absolute comparison.

Can be interpreted as a difference measure.

$$\chi^2(X, Y) = \sum_{i=1}^{k_X} \sum_{j=1}^{k_Y} n_{..} \frac{(p_{i.p.j} - p_{ij})^2}{p_{i.p.j}}$$

Side remark: Information gain can also be interpreted as a difference measure.

$$I_{\text{gain}}(X, Y) = \sum_{j=1}^{k_Y} \sum_{i=1}^{k_X} p_{ij} \log_2 \frac{p_{ij}}{p_{i.p.j}}$$

A Statistical Evaluation Measure

χ^2 Measure

Compares the actual joint distribution with a **hypothetical independent distribution**.

Uses absolute comparison.

Can be interpreted as a difference measure.

$$\chi^2(X, Y) = \sum_{i=1}^{k_X} \sum_{j=1}^{k_Y} n_{..} \frac{(p_{i.p.j} - p_{ij})^2}{p_{i.p.j}}$$

For $k_X = k_Y = 2$ (as for rule evaluation) the χ^2 measure simplifies to

$$\chi^2(X, Y) = n_{..} \frac{(p_{1.} p_{.1} - p_{11})^2}{p_{1.}(1 - p_{1.})p_{.1}(1 - p_{.1})} = n_{..} \frac{(n_{1.}n_{.1} - n_{..}n_{11})^2}{n_{1.}(n_{..} - n_{1.})n_{.1}(n_{..} - n_{.1})}$$

Association Rule Induction is a Two Step Process

- Find the frequent item sets (minimum support).
- Form the relevant association rules (minimum confidence).

Generating the Association Rules

- Form all possible association rules from the frequent item sets.
- Filter “interesting” association rules based on minimum support and minimum confidence.

Filtering the Association Rules

- Compare rule confidence and consequent support.
- Information gain
- χ^2 measure

Industrial Applications

Car manufacturer collects servicing tasks on all their vehicles.

- What are interesting subgroups of cars?
- How do these subgroups behave over time?
- Which cars' suspension failure rate is strongly increasing in winter?

Bank assesses credit contracts w. r. t. terminability.

- What changes were there in the past?
- Any common factors?
- How do I communicate this to a non-statistician?

Tracking user activity in a virtual environment

- Are there any oddities in user behavior?
- How do I parameterize “odd” things?

Or: What they have and what they want

Given:

High-dimensional data

Many-valued data

Time-stamped data

Asked for:

Easy-to-understand patterns (rules)

Exploratory tools (visualization and inspection)

Natural way of interaction

Exploit temporal information (if desired)

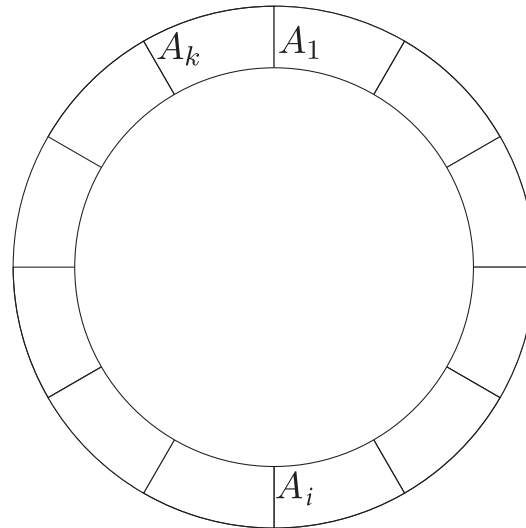
Rule Icons

Every rule

$$\langle A_1 = a_1 \wedge \cdots \wedge A_k = a_k \rangle \rightarrow C = c$$

of a given rule set is represented as an icon:

- For every possible item there is a reserved segment on the outer border.



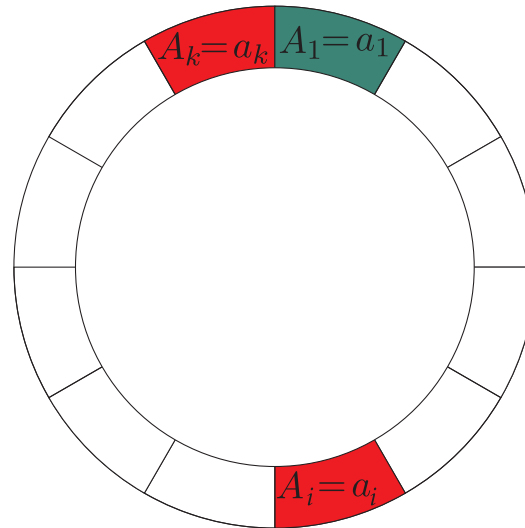
Rule Icons

Every rule

$$\langle A_1 = a_1 \wedge \cdots \wedge A_k = a_k \rangle \rightarrow C = c$$

of a given rule set is represented as an icon:

- For every possible item there is a reserved segment on the outer border.
- If the item is present in the antecedent, the segment is colored.



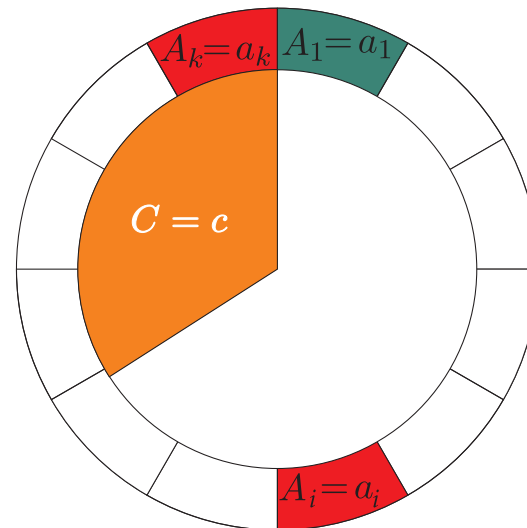
Rule Icons

Every rule

$$\langle A_1 = a_1 \wedge \cdots \wedge A_k = a_k \rangle \rightarrow C = c$$

of a given rule set is represented as an icon:

- For every possible item there is a reserved segment on the outer border.
- If the item is present in the antecedent, the segment is colored.
- Interior encodes a rule-measure: here confidence.



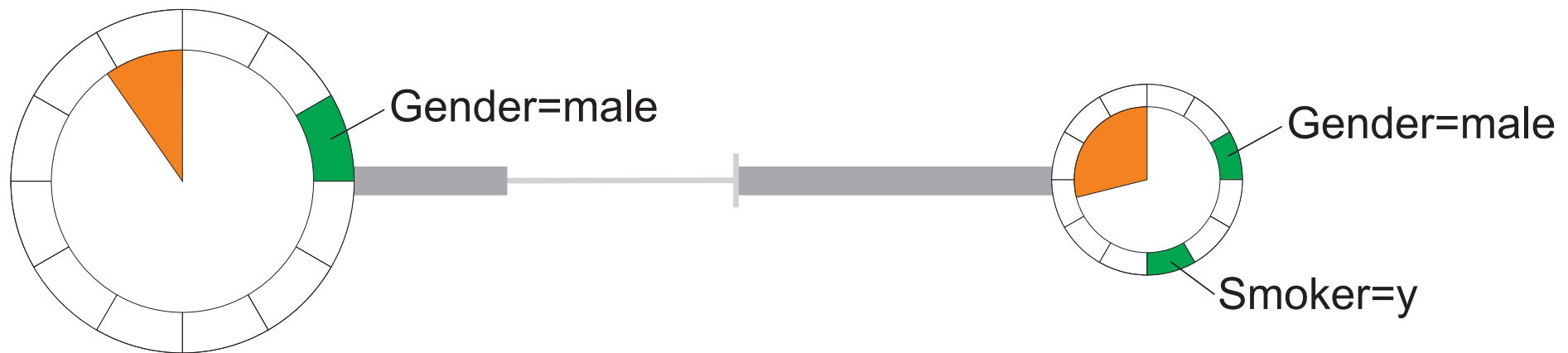
Rule Icons: Overlapping

The cover of two rules may be non-empty. Use a percentage bar to display the mutual overlap.

Special case: inclusion

$\text{Gender} = \text{male} \rightarrow \text{Cancer} = \text{yes}$

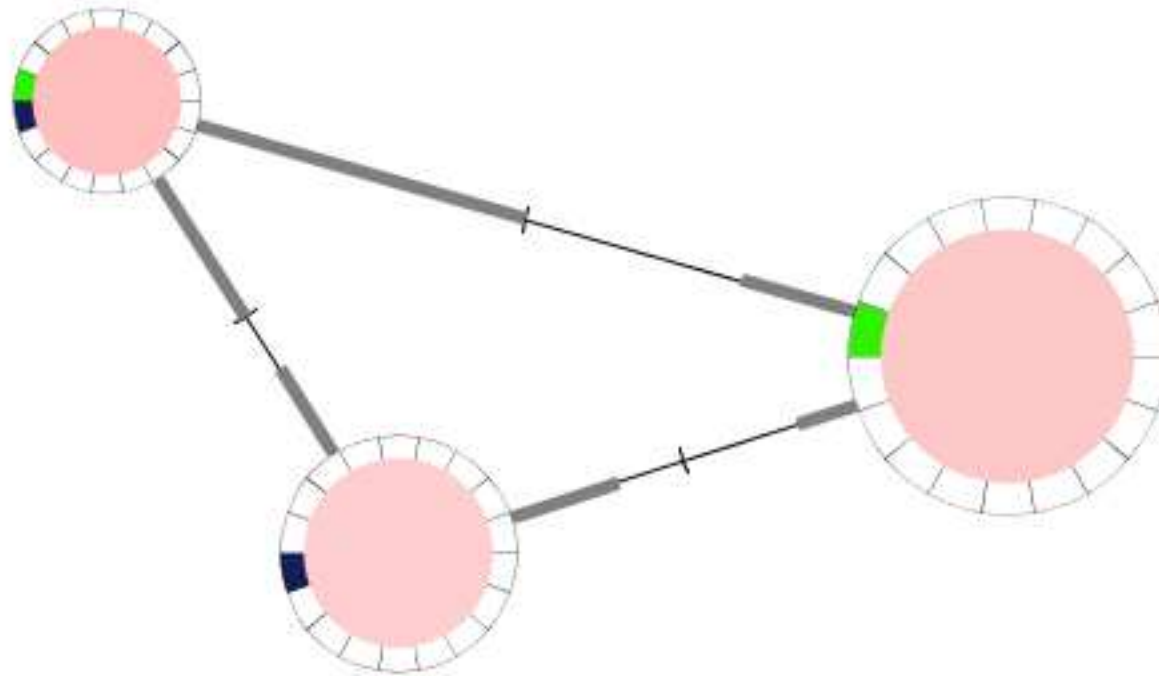
$\text{Gender} = \text{male} \wedge \text{Smoker} = \text{yes} \rightarrow \text{Cancer} = \text{yes}$



Rule Icons: Overlapping

The cover of two rules may be non-empty. Use a percentage bar to display the mutual overlap.

General case:



Rule Icons: Location

Finally, arrange the icons in a two-dimensional chart.

Choose association rule measures for the two axes and the size of the icon

Our suggestion: For a rule $X \rightarrow Y$ choose the following measures:

- x-coordinate: recall, i. e. $c_T(Y \rightarrow X)$
- y-coordinate: lift, i. e. $c_T(X \rightarrow Y)/\sigma_T(Y)$
- size: support, i. e. $\sigma_T(X \cup Y)$

Real-world Example: Daimler AG

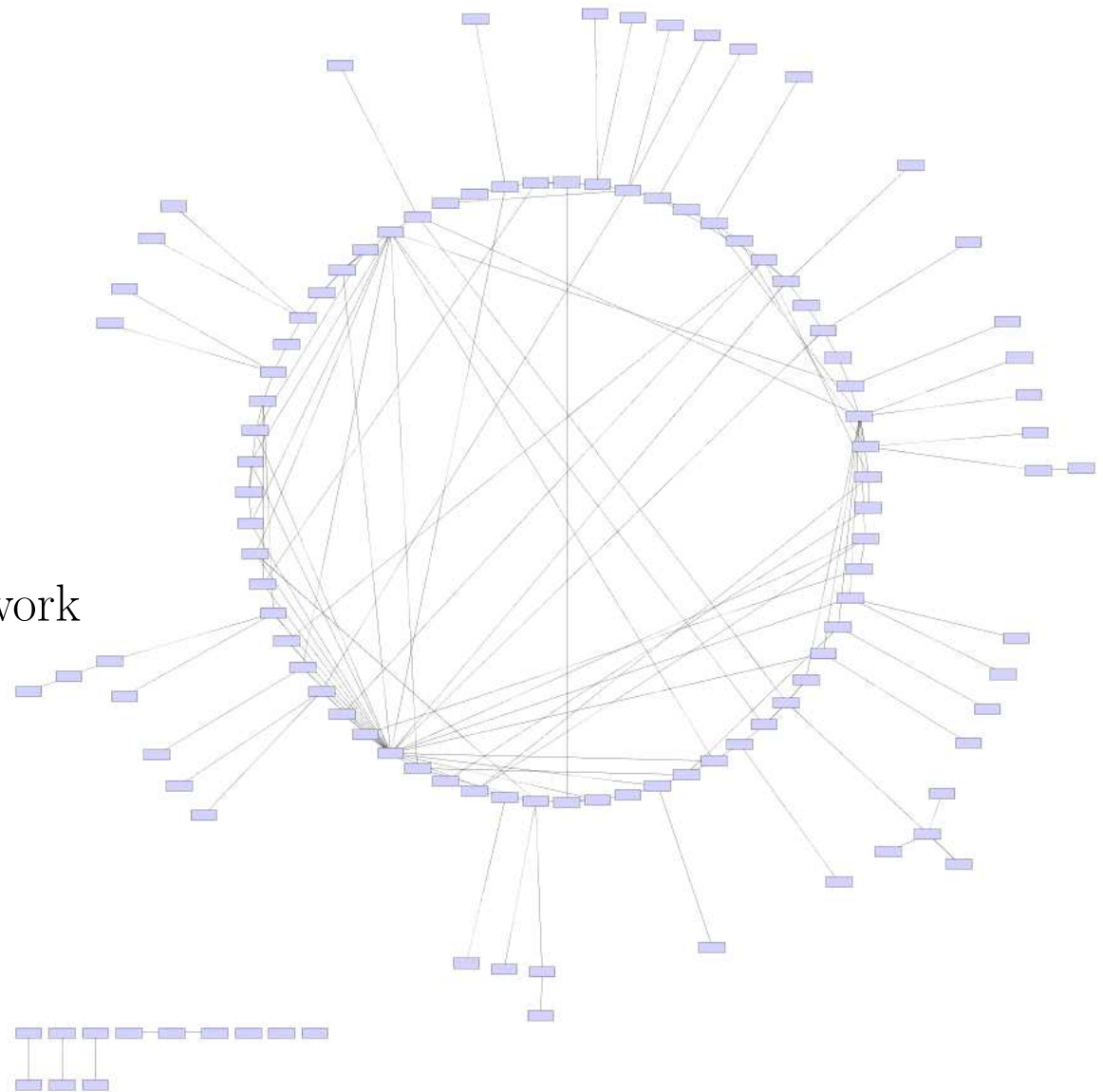
Car database

300 000 vehicles

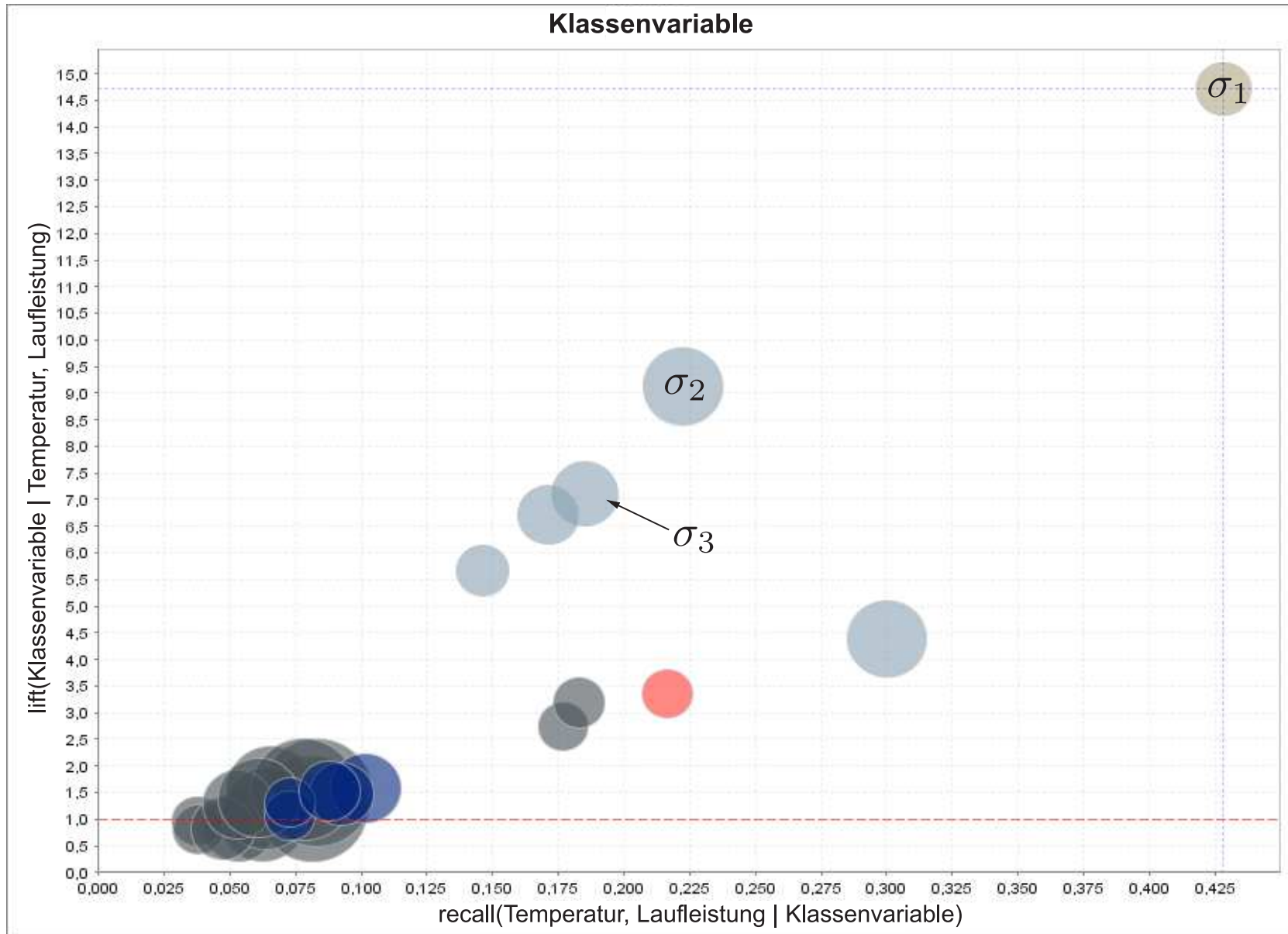
subset of 180 attributes

2 to 300 values per attribute

Probabilistic Dependency Network

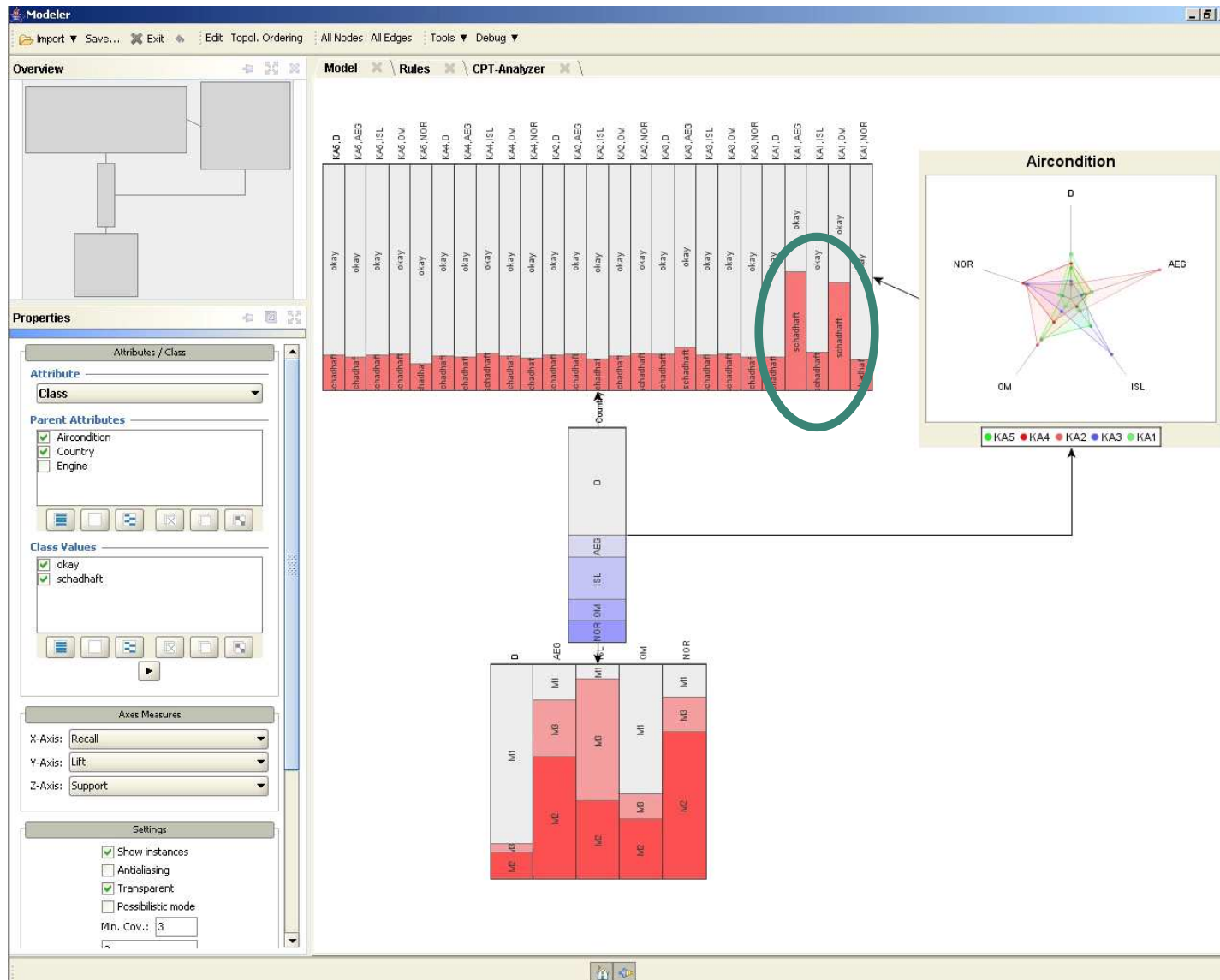


Real-world Example: Daimler AG



Real-world Example: Daimler AG

Explorative Analysis Tool



Real-world Example: ADAC

Customer database

Car and customer information

Assessment of vehicle quality

The screenshot displays the ADAC ADD Miner software interface. The main window is titled "ADAC ADD Miner" and features the ADAC Autodiagnose logo in the top right corner. The interface is divided into several sections:

- Left Panel (Attribute Selection):** Contains a list of attributes under "Parent Attributes" (PLZ, KundenAlter, Geschlecht, MotorLeistung, Kilometerstand) and "Class Values" (Ab 15, 5-9, ?, 10-14, Unter 5). It also includes "Axes Measures" (X-Axis: Recall, Y-Axis: Lift, Z-Axis: Support) and "Settings" (Show instances, Antialiasing, Transparent, Possibilistic mode, Min. Cov., and a numeric input field).
- Center Panel (Visualizations):** Displays several pie charts representing data distributions. A table below the charts shows the selected rule:

Attribute	Value
FahrzeugAlter	10-14
MotorLeistung	60-79
Geschlecht	m
- Right Panel (Rule Extraction):** Shows a tree diagram of rules. Below it, a smaller window displays a pie chart visualization similar to the center panel. A table of "Heuristics Values" is also present:

Heuristics Values
LHS Support: 0,145
Confidence: 0,241
Recall: 0,264
Lift: 1,822
RHS Support: 0,132
Leverage: 0,016
Support: 0,035
Interestingness: 0,481
Inv. Interestingness: 0,44

Why considering the temporal development of rules?

(i. e. the change of certain rule evaluation measures)

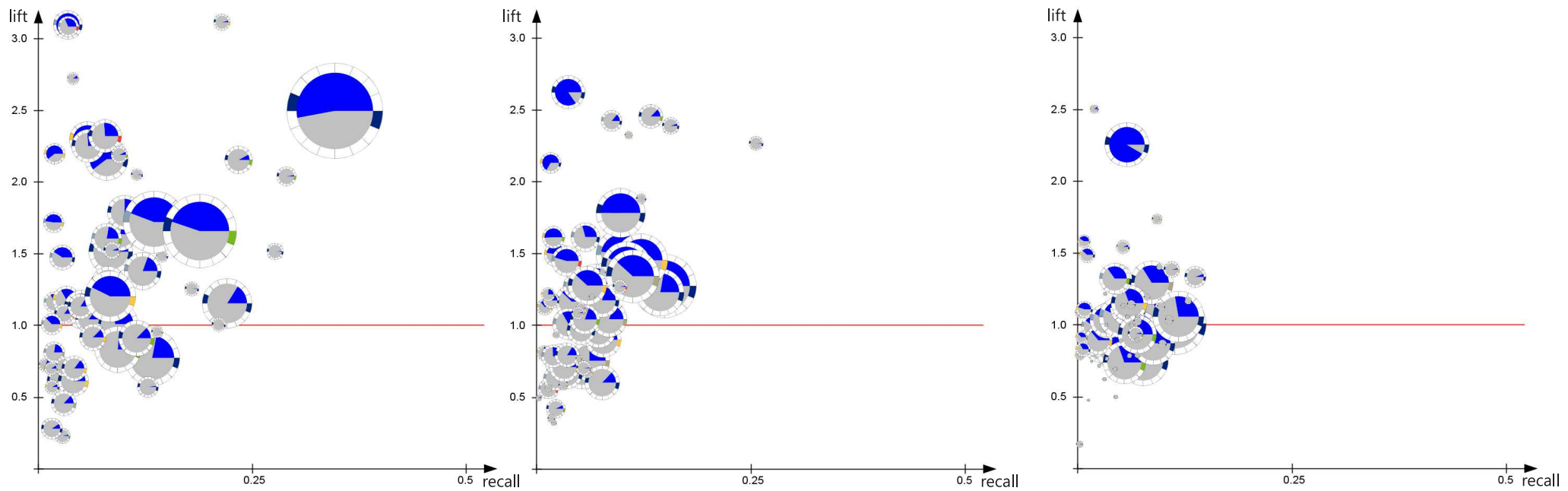
- Failure patterns usually do not arise out of a sudden but rather evolve slowly over time.
- A fixed problem takes some while to have a measurable effect.

How to present this evolution to the user?

- Create a time series for every measure used for locating and scaling the rule icon.
- Interpolate between the frames and present an animation.

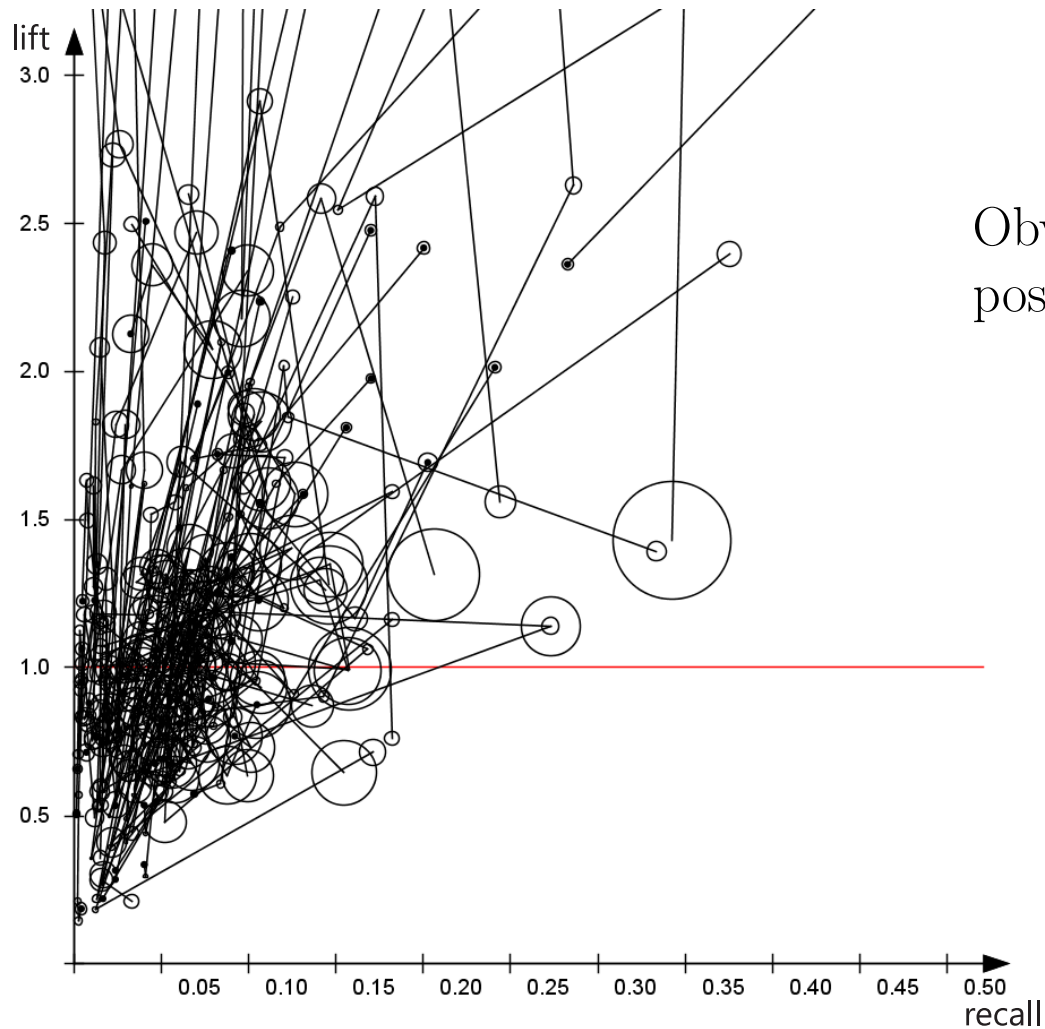
Problem: Need to reduce the number of rules.

How does that look like?



Real-world dataset

How does that look like?



Obviously, there is a demand for post-processing the rule set!

Temporal Change of Rules

Divide dataset into reasonable time frames

Run respective pattern induction algorithm

Quantify each pattern w. r. t. any desired measure(s)

Generate time series for each measure and each pattern

Match the time series against a user-specified concept

Rank them according to the membership of the concept

User-driven Post-processing

Users often have an idea in which direction to investigate but cannot explicitly phrase a query to a Data Mining system. However, we can use intentions like

“Show me only those rules that had a strong increasing support and a increasing confidence in the last quarter.”

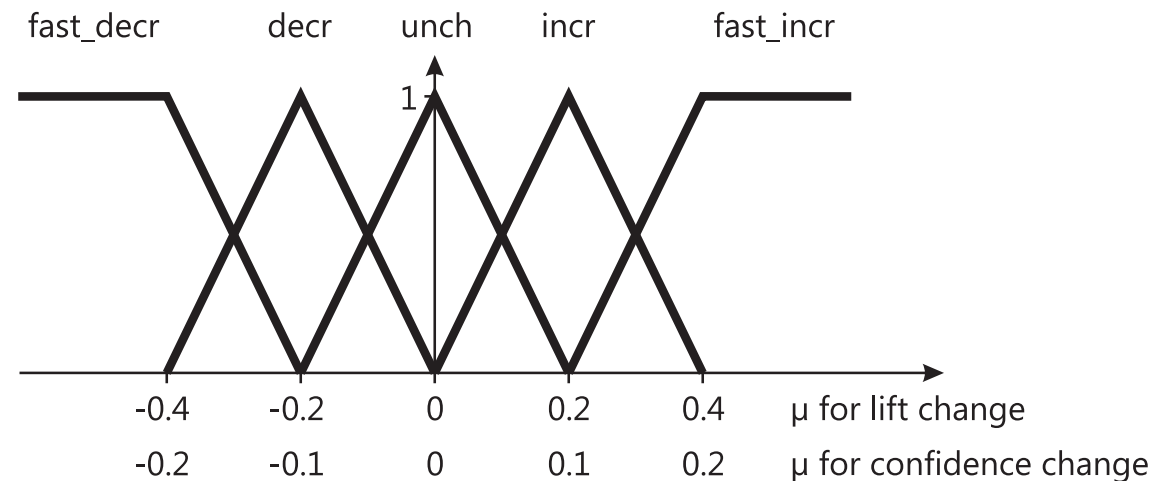
or

“Which patterns exhibit an increasing lift while the support was stable or at most slightly decreasing?”

to thin out the rule set.

User-driven Post-processing

Specify a fuzzy partition on the change rate domain of every pattern evaluation measure.



User-driven Post-processing

Specify a fuzzy partition on the change rate domain of every pattern evaluation measure.

Encode the user-concept as a fuzzy antecedent.

E. g. “lift is unchanged and confidence is increasing”:

$$\langle \Delta_{\text{lift}} \mathbf{is\ unch} \wedge \Delta_{\text{conf}} \mathbf{is\ incr} \rangle$$

will be evaluated as

$$\top \left(\mu_{\Delta_{\text{lift}}}^{(\text{unch})}(\vec{a} \rightarrow c), \mu_{\Delta_{\text{conf}}}^{(\text{incr})}(\vec{a} \rightarrow c) \right)$$

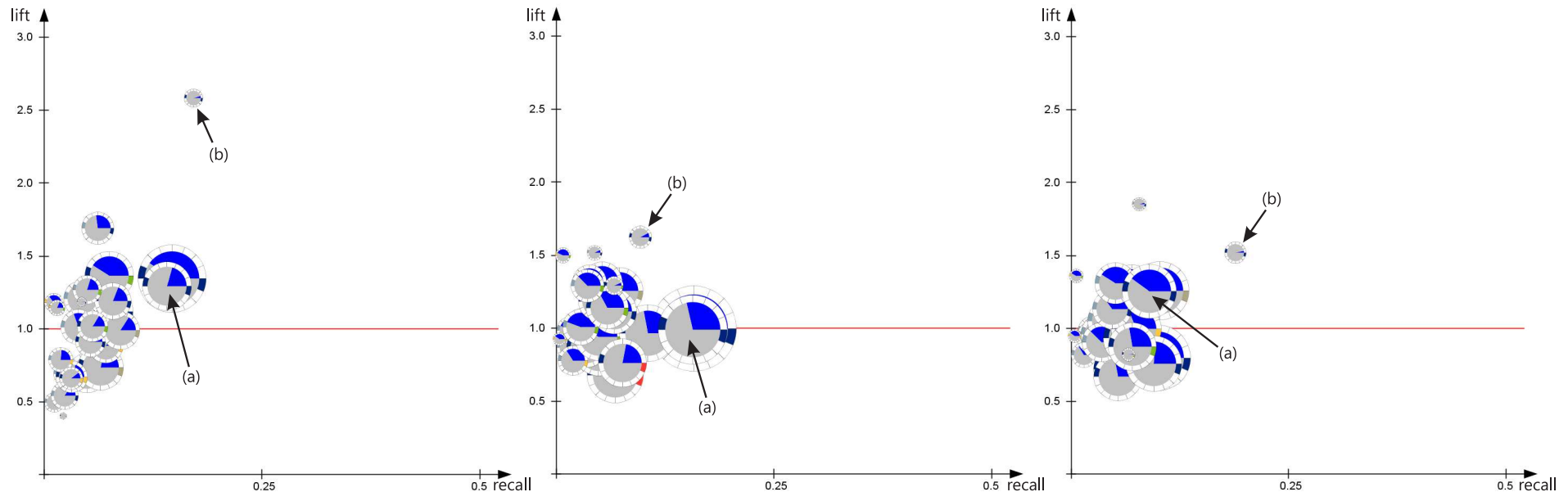
where \top is a t-norm that represents a fuzzy conjunction.

User-driven Post-processing

Specify a fuzzy partition on the change rate domain of every pattern evaluation measure.

Encode the user-concept as a fuzzy antecedent.

Order the patterns w. r. t. concept membership degrees.



Requirements

- Easy-to-understand patterns
- Exploratory visual tools
- Natural and intuitive interaction
- Exploitation of temporal information

Desired Properties of Rules

- Almost free of parameters (support and confidence have a clear notion and can even be increased after the induction)
- No black box approach
- Intuitive type of patterns (decision rules, business rules)
- Natural way of treating missing values.
- Light data preprocessing overhead

Probabilistic Causal Networks (Bayes Networks)

The Big Objective(s)

In a wide variety of application fields two main problems need to be addressed over and over:

How can (expert) knowledge of complex domains be efficiently represented?

How can inferences be carried out within these representations?

How can such representations be (automatically) extracted from collected data?

Example 1: Planning in car manufacturing

Available information

“Engine type e_1 can only be combined with transmission t_2 or t_5 .”

“Transmission t_5 requires crankshaft c_2 .”

“Convertibles have the same set of radio options as SUVs.”

Possible questions/inferences:

“Can a station wagon with engine e_4 be equipped with tire set y_6 ?”

“Supplier S_8 failed to deliver on time. What production line has to be modified and how?”

“Are there any peculiarities within the set of cars that suffered an aircondition failure?”

Example 2: Medical reasoning

Available information:

“Malaria is much less likely than flu.”

“Flu causes cough and fever.”

“Nausea can indicate malaria as well as flu.”

“Nausea never indicated pneumonia before.”

Possible questions/inferences

“The patient has fever. How likely is he to have malaria?”

“How much more likely does flu become if we can exclude malaria?”

Common Problems

Both scenarios share some severe problems:

Large Data Space

It is intractable to store all value combinations, i. e. all car part combinations or inter-disease dependencies.

(Example: VW Bora has 10^{200} theoretical value combinations*)

Sparse Data Space

Even if we could handle such a space, it would be extremely sparse, i. e. it would be impossible to find good estimates for all the combinations.

(Example: with 100 diseases and 200 symptoms, there would be about 10^{62} different scenarios for which we had to estimate the probability.*)

* The number of particles in the observable universe is estimated to be between 10^{78} and 10^{85} .

Idea to Solve the Problems

Given: A large (high-dimensional) distribution δ representing the domain knowledge.

Desired: A set of smaller (lower-dimensional) distributions $\{\delta_1, \dots, \delta_s\}$ (maybe overlapping) from which the original δ *could* be reconstructed with no (or as few as possible) errors.

With such a decomposition we can draw any conclusions from $\{\delta_1, \dots, \delta_s\}$ that could be inferred from δ — without, however, actually reconstructing it.

Example: Car Manufacturing

Let us consider a car configuration is described by three attributes:

- Engine E , $\text{dom}(E) = \{e_1, e_2, e_3\}$
- Breaks B , $\text{dom}(B) = \{b_1, b_2, b_3\}$
- Tires T , $\text{dom}(T) = \{t_1, t_2, t_3, t_4\}$

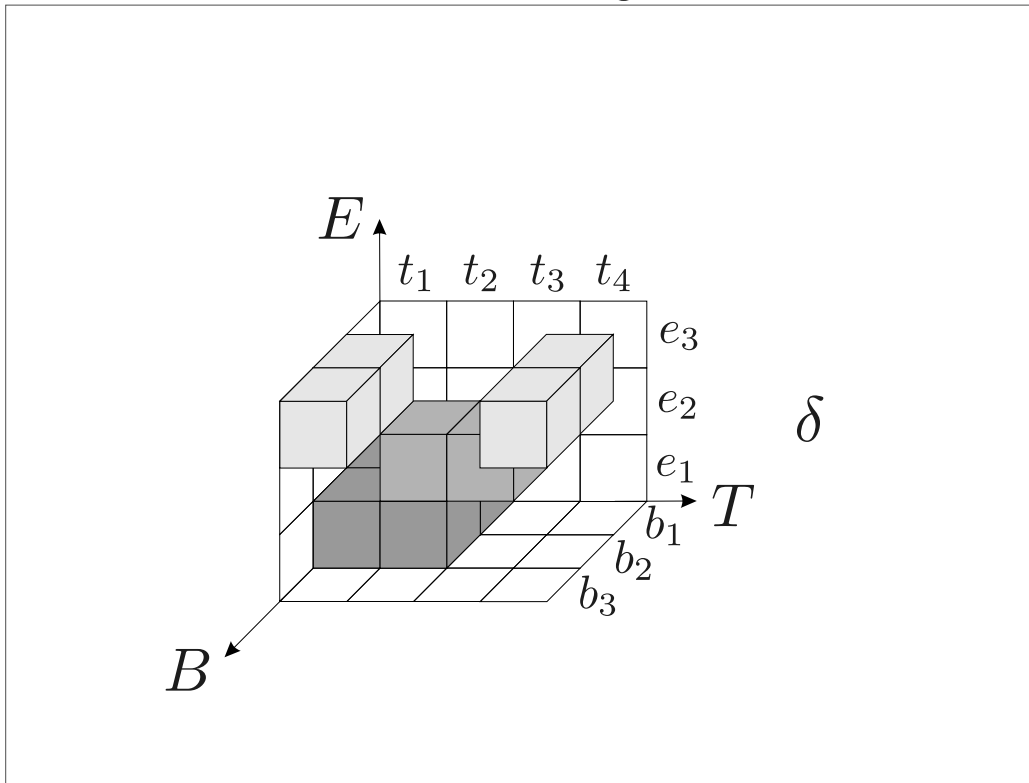
Therefore the set of all (theoretically) possible car configurations is:

$$\Omega = \text{dom}(E) \times \text{dom}(B) \times \text{dom}(T)$$

Since not all combinations are technically possible (or wanted by marketing) a set of rules is used to cancel out invalid combinations.

Example: Car Manufacturing

Possible car configurations



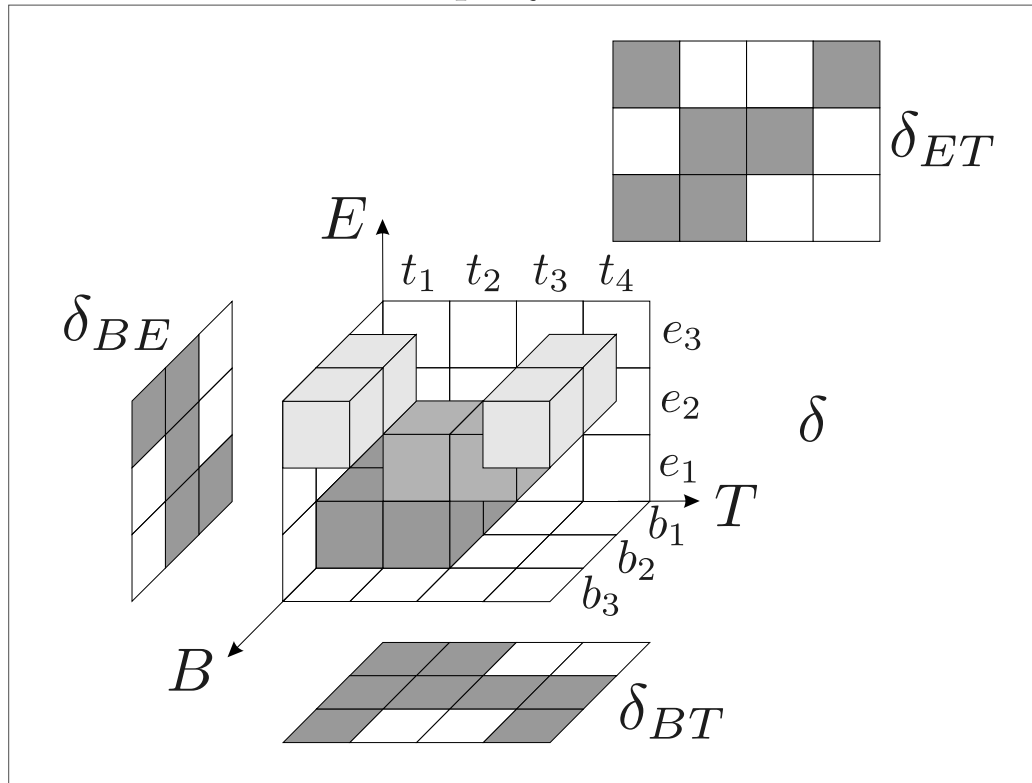
Every cube designates a valid value combination.

10 car configurations in our model.

Different colors are intended to distinguish the cubes only.

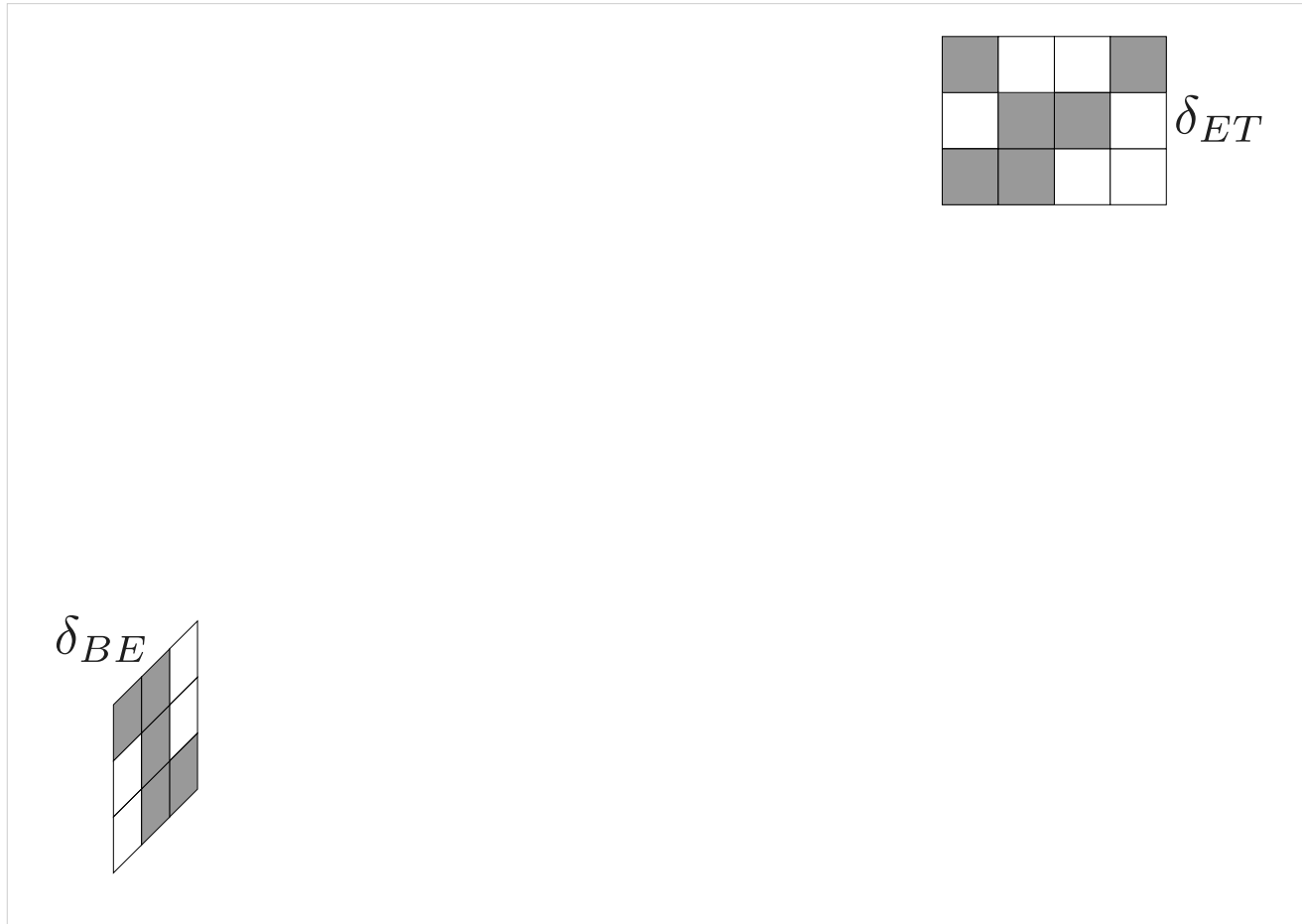
Example

2-D projections

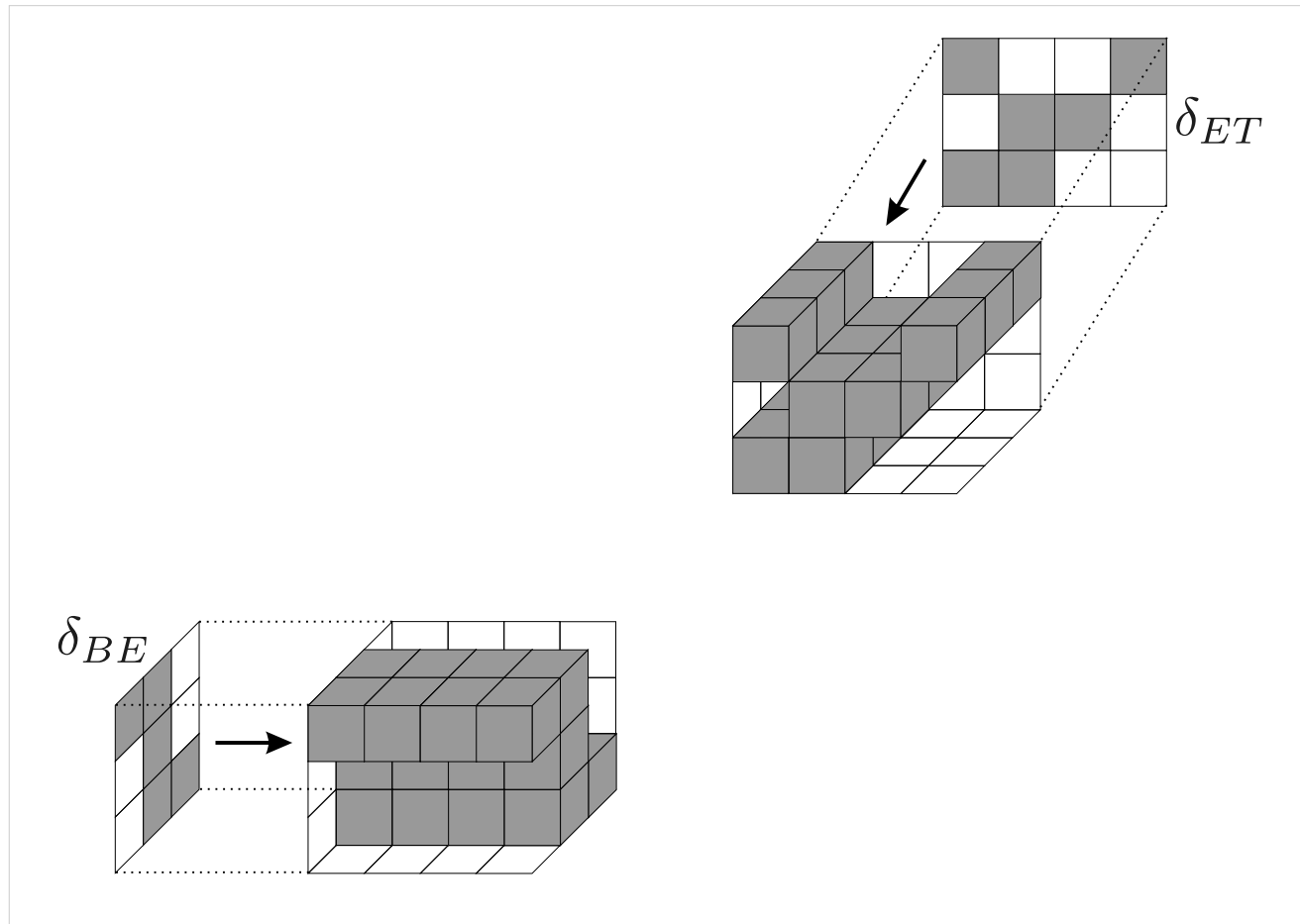


Is it possible to reconstruct δ from the δ_i ?

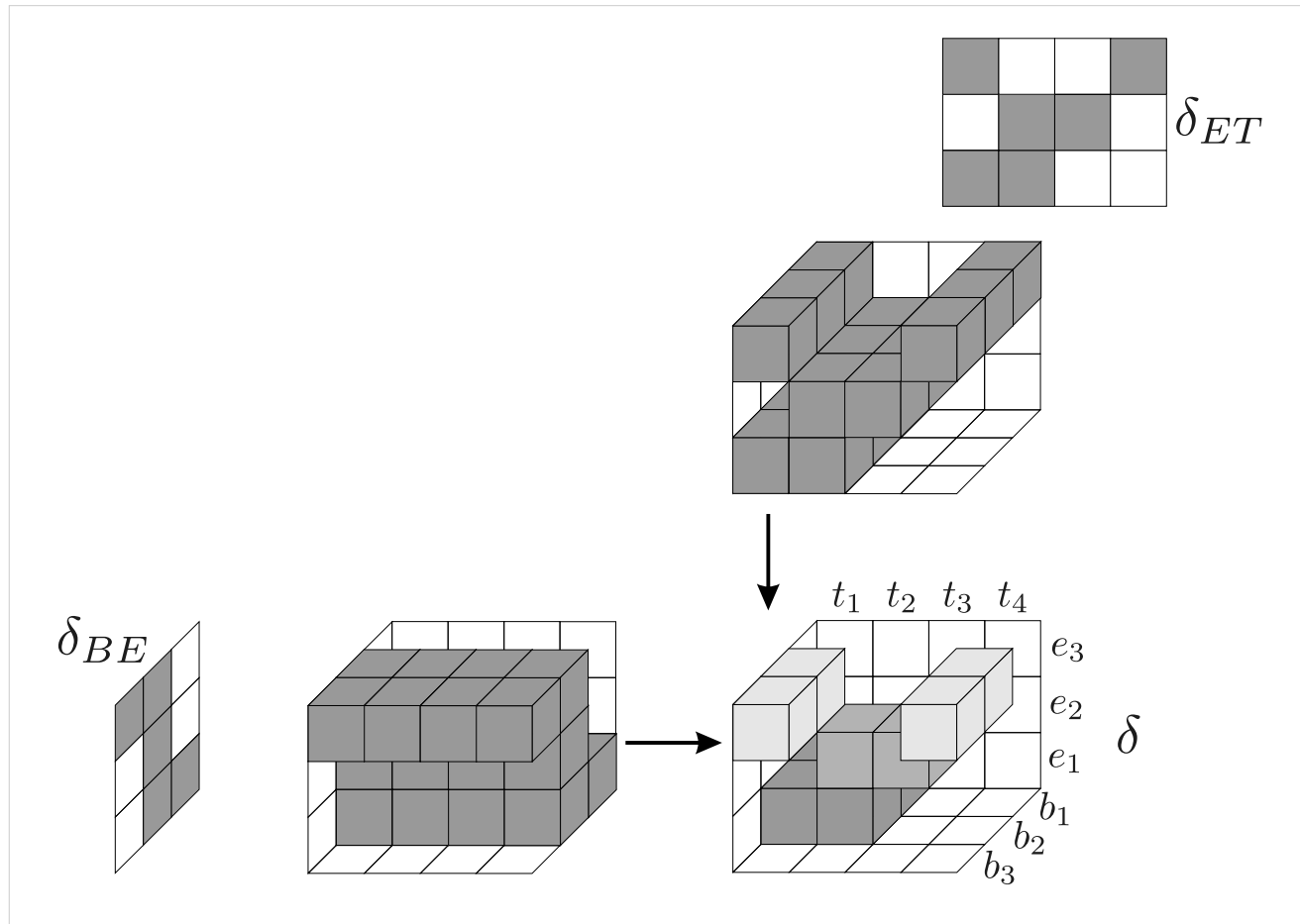
Example: Reconstruction of δ with δ_{BE} and δ_{ET}



Example: Reconstruction of δ with δ_{BE} and δ_{ET}



Example: Reconstruction of δ with δ_{BE} and δ_{ET}



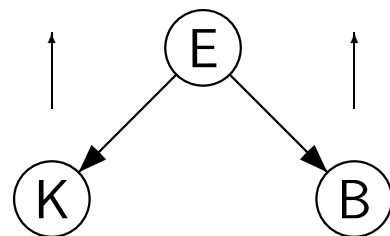
Example — Qualitative Aspects

Lecture theatre in winter: Waiting for Mr. **K** and Mr. **B**.
Not clear whether there is ice on the roads.

3 variables:

- **E** road condition: $\text{dom}(\mathbf{E}) = \{\text{ice}, \neg\text{ice}\}$
- **K** **K** had an accident: $\text{dom}(\mathbf{K}) = \{\text{yes}, \text{no}\}$
- **B** **B** had an accident: $\text{dom}(\mathbf{B}) = \{\text{yes}, \text{no}\}$

Ignorance about these states is modelled via the observer's belief.



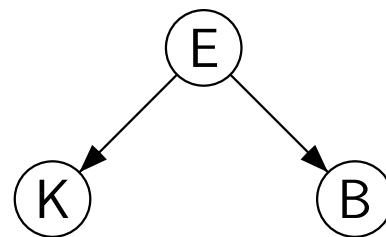
↓ **E** influences **K** and **B**

(the more ice the more accidents)

↑ Knowledge about accident increases belief in ice

Example

A priori knowledge	Evidence	Inferences
E unknown	B has accident	$\Rightarrow E = \text{ice}$ more likely $\Rightarrow K$ has accident more likely
$E = \neg \text{ice}$	B has accident	\Rightarrow no change in belief about E \Rightarrow no change in belief about accident of K
E unknown		K and B dependent
E known		K and B independent



Dependence vs. Reasoning

Rule: A entails B with certainty x :

$$\boxed{A \xrightarrow{x} B}$$

Deduction (\rightarrow):

A and $A \xrightarrow{x} B$, therefore B more likely as effect (causality)

Abduction (\leftarrow):

B and $A \xrightarrow{x} B$, therefore A more likely as cause (no causality)

For this reason, the notion “dependency model” is to be preferred to “causal network”.

Objective

Is it possible to exploit local constraints (wherever they may come from — both structural and expert knowledge-based) in a way that allows for a decomposition of the large (intractable) distribution $P(X_1, \dots, X_n)$ into several sub-structures $\{C_1, \dots, C_m\}$ such that:

The collective size of those sub-structures is much smaller than that of the original distribution P .

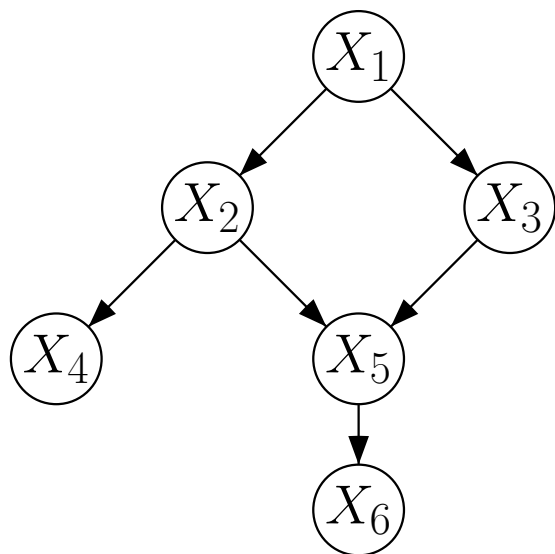
The original distribution P is recomposable (with no or at least as few as possible errors) from these sub-structures in the following way:

$$P(X_1, \dots, X_n) = \prod_{i=1}^m \Psi_i(c_i)$$

where c_i is an instantiation of C_i and $\Psi_i(c_i) \in \mathbb{R}^+$ a *factor potential*.

Bayes Networks

Probabilistic causal networks are directed acyclic graphs (DAGs) where the nodes represent propositions or variables and the directed edges model a direct causal dependence between the connected nodes. The strength of dependence is defined by conditional probabilities.

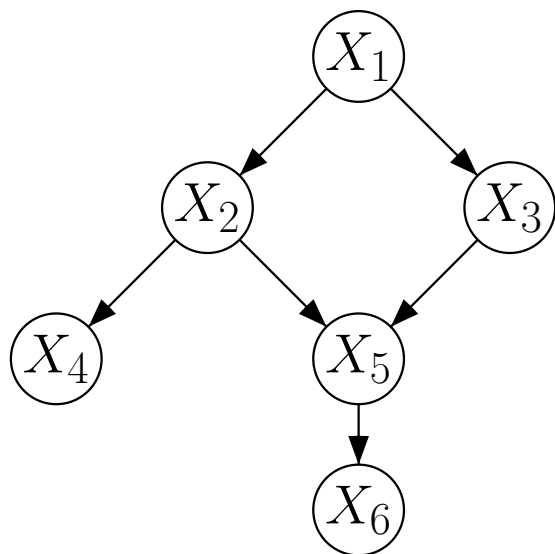


In general (according chain rule):

$$\begin{aligned} P(X_1, \dots, X_6) &= P(X_6 \mid X_5, \dots, X_1) \cdot \\ &P(X_5 \mid X_4, \dots, X_1) \cdot \\ &P(X_4 \mid X_3, X_2, X_1) \cdot \\ &P(X_3 \mid X_2, X_1) \cdot \\ &P(X_2 \mid X_1) \cdot \\ &P(X_1) \end{aligned}$$

Bayes Networks

Probabilistic causal networks are directed acyclic graphs (DAGs) where the nodes represent propositions or variables and the directed edges model a direct causal dependence between the connected nodes. The strength of dependence is defined by conditional probabilities.



According graph (independence structure):

$$\begin{aligned} P(X_1, \dots, X_6) = & P(X_6 \mid X_5) \cdot \\ & P(X_5 \mid X_2, X_3) \cdot \\ & P(X_4 \mid X_2) \cdot \\ & P(X_3 \mid X_1) \cdot \\ & P(X_2 \mid X_1) \cdot \\ & P(X_1) \end{aligned}$$

Formal Framework

Nomenclature for the next slides:

- X_1, \dots, X_n Variables
(properties, attributes, random variables, propositions)
- $\Omega_1, \dots, \Omega_n$ respective finite domains
(also designated with $\text{dom}(X_i)$)
- $\Omega = \prod_{i=1}^n \Omega_i$ Universe of Discourse (tuples that characterize objects described by X_1, \dots, X_n)
- $\Omega_i = \{x_i^{(1)}, \dots, x_i^{(n_i)}\}$ $n = 1, \dots, n, n_i \in \mathbb{N}$

Belief Network

A *Belief Network* (V, E, P) consists of a set $V = \{X_1, \dots, X_n\}$ of random variables and a set E of directed edges between the variables.

Each variable has a finite set of mutual exclusive and collectively exhaustive states.

The variables in combination with the edges form a directed, acyclic graph.

Each variable with parent nodes B_1, \dots, B_m is assigned a potential table $P(A | B_1, \dots, B_m)$.

Note, that the connections between the nodes not necessarily express a causal relationship.

For every belief network, the following equation holds:

$$P(V) = \prod_{v \in V: P(c(v)) > 0} P(v | c(v))$$

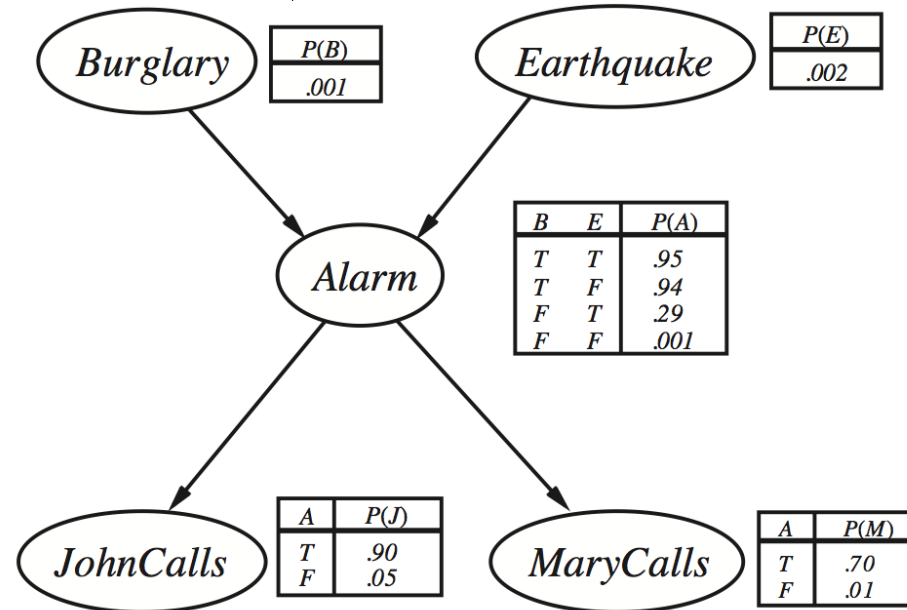
with $c(v)$ being the parent nodes of v .

Example

New burglar alarm has been installed and is fairly reliable, yet sometimes also reacts on earthquakes

Neighbours John and Mary agree to call each other, when they hear the alarm.

John sometimes mistakes the ringing of the phone for an alarm and Mary sometimes does not hear the alarm, because she listens to loud music.



Example

Choice of universe of discourse

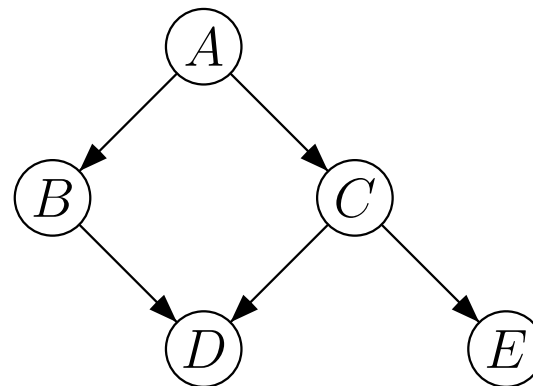
	Variable	Domain
<i>A</i>	metastatic cancer	$\{a_1, a_2\}$
<i>B</i>	increased serum calcium	$\{b_1, b_2\}$
<i>C</i>	brain tumor	$\{c_1, c_2\}$
<i>D</i>	coma	$\{d_1, d_2\}$
<i>E</i>	headache	$\{e_1, e_2\}$

(\cdot_1 — present, \cdot_2 — absent)

$$\Omega = \{a_1, a_2\} \times \cdots \times \{e_1, e_2\}$$

$$|\Omega| = 32$$

Analysis of dependencies



Example

Choice of probability parameters

$$P(a, b, c, d, e) \stackrel{\text{abbr.}}{=} P(A = a, B = b, C = c, D = d, E = e)$$
$$= P(e | c)P(d | b, c)P(c | a)P(b | a)P(a)$$

↑
Shorthand notation

11 values to store instead of 31

Consult experts, textbooks, case studies, surveys, etc.

Calculation of conditional probabilities

Calculation of marginal probabilities

Crux of the Matter

Knowledge acquisition (Where do the numbers come from?)

→ learning strategies

Computational complexities

→ exploit independencies

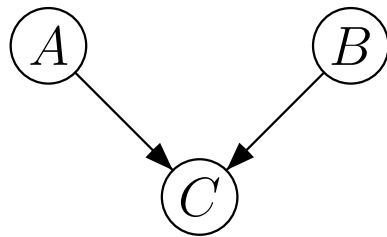
Problem:

When does the independency of X and Y given Z hold in (V, E, P) ?

How can we determine $P(X, Y | Z) = P(X | Z)P(Y | Z)$ solely using the graph structure?

Dependencies

Converging Connection



Meal quality

A quality of ingredients

B cook's skill

C meal quality

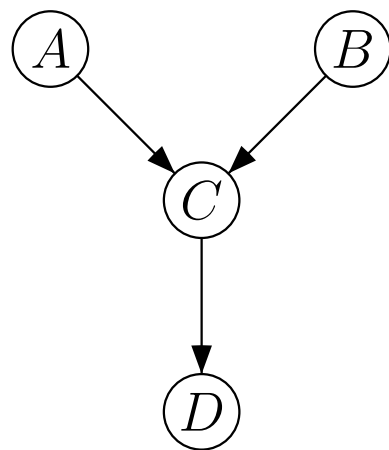
If C is not instantiated (i. e., no value specified/observed), A and B are marginally independent.

After instantiation (observation) of C the variables A and B become conditionally dependent given C .

Evidence can only be transferred over a converging connection if the variable in between (or one of its successors) is initialized.

Dependencies

Converging Connection (cont.)



Meal quality

A quality of ingredients

B cook's skill

C meal quality

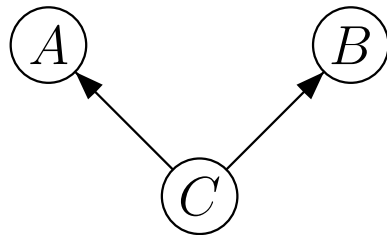
D restaurant success

If nothing is known about the restaurant success or meal quality or both, the cook's skills and quality of the ingredients are unrelated, that is, *independent*.

However, if we observe that the restaurant has no success, we can infer that the meal quality might be bad.

If we further learn that the ingredients quality is high, we will conclude that the cook's skills must be low, thus rendering both variables *dependent*.

Diverging Connection



Diagnosis

A body temperature

B cough

C disease

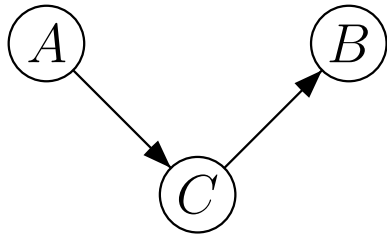
If *C* is unknown, knowledge about *A* is relevant for *B* and vice versa, i. e. *A* and *B* are marginally dependent.

However, if *C* is observed, *A* and *B* become conditionally independent given *C*.

A influences *B* via *C*. If *C* is known it in a way blocks the information from flowing from *A* to *B*, thus rendering *A* and *B* (conditionally) independent.

Dependencies

Serial Connection



Accidents

A rain

B accident risk

C road conditions

Analog scenario to case 2

A influences *C* and *C* influences *B*. Thus, *A* influences *B*.

If *C* is known, it blocks the path between *A* and *B*.

Summary

- Knowledge about conditional dependencies can be modelled as graphs.
- Problems arise, if the resulting graph is not acyclic (evidence can be propagated in several ways).
- Evidence propagation (not covered here) still easily possible with hyper-tree structure. (More on that in the lecture on Bayesian Networks next winter semester)