Formal Concept Analysis

II Closure Systems and Implications

Sebastian Rudolph

Computational Logic Group
Technische Universität Dresden

slides based on a lecture by Prof. Gerd Stumme

Agenda

3. Closure Systems
   - Concept Intents as Closed Sets
   - NEXT CLOSURE Algorithm
   - Iceberg Concept Lattices
   - TITANIC Algorithm

Closure Systems

Def.: A *closure system* on a set G is a set $\mathfrak{A} \subseteq 2^G$ of subsets of G if

- it contains G, i.e., $G \in \mathfrak{A}$ and
- it is closed under intersections, i.e., $\mathfrak{X} \subseteq \mathfrak{A}$ implies $\bigcap_{X \in \mathfrak{X}} X \in \mathfrak{A}$.

Def.: A *closure operator* on G is a map $\varphi : 2^G \to 2^G$ assigning to each subset $X \subseteq G$ its *closure* $\varphi(X) \subseteq G$ satisfying the following conditions:

- $X \subseteq Y$ implies $\varphi(X) \subseteq \varphi(Y)$, (monotonicity)
- $X \subseteq \varphi(X)$, and (extensivity)
- $\varphi(\varphi(X)) = \varphi(X)$. (idempotency)

Theorem (correspondence of closure systems and closure operators)

If $\mathfrak{A}$ is a closure system on G then $\varphi_{\mathfrak{A}}(X) := \bigcap_{A \in \mathfrak{A}, X \subseteq A} A$ defines a closure operator on G. Conversely, for a closure operator φ on G, the set $\mathfrak{A}_\varphi = \{\varphi(A) \mid A \subseteq G\}$ of all closures forms a closure system on G. Moreover, $\varphi_{\mathfrak{A}_\varphi} = \varphi$ and $\mathfrak{A}_{\varphi_{\mathfrak{A}}} = \mathfrak{A}$.

Closure Systems

Theorem (closure systems and complete lattices)

If $\mathfrak{A}$ is a closure system, then $(\mathfrak{A}, \subseteq)$ is a complete lattice with $\bigwedge \mathfrak{X} = \bigcap_{X \in \mathfrak{X}} X$ and $\bigvee \mathfrak{X} = \varphi_{\mathfrak{A}}(\bigcup_{X \in \mathfrak{X}} X)$.
Conversely, every complete lattice is isomorphic to the lattice of all closures of a closure system.

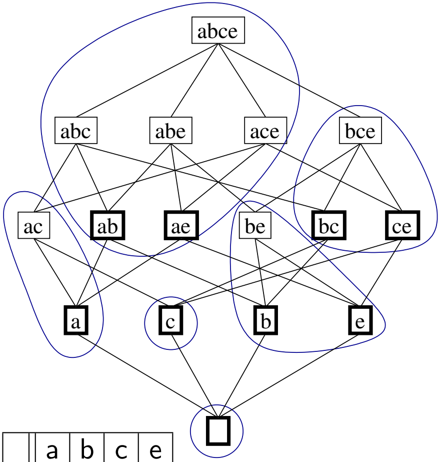
In mathematics and computer science, we find a plethora of examples for closure systems (e.g., subtrees, subintervals, convex sets, equivalence relations).

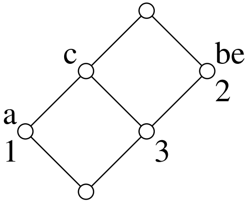
For every formal context (G, M, I) holds:

- The extents form a closure system on G.
- The intents form a closure system on M.
- $''$ is a closure operator.

Concept Intents as Closed Sets

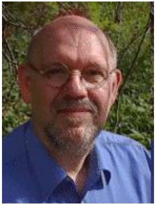
- the line diagram of the powerset of $\{a, b, c, e\}$
- classes of attributes that describe the same set of objects
- unique representatives: concept intents (=closed sets)
- minimal generator





	a	b	c	e
1	×		×	
2		×		×
3		×	×	×

Next Closure Algorithm



Developed 1984 by Bernhard Ganter.

Can be used

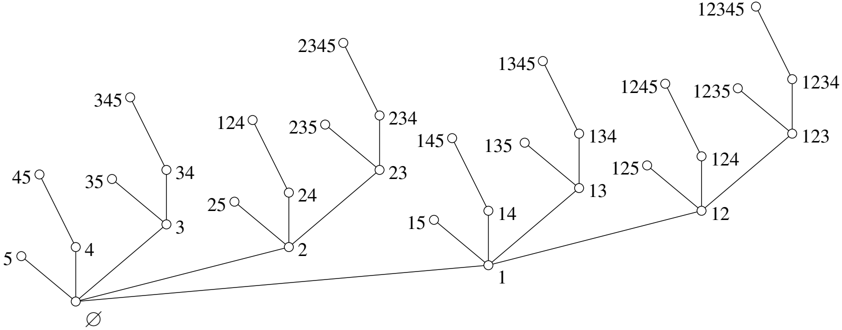
- to compute the concept lattice, or
- to compute the concept lattice together with the stem base, or
- for interactive knowledge exploration.

The algorithm computes the concept intents in the *lectic order*.

Next Closure Algorithm: Lectic Order

Let $M = \{1, \ldots, n\}$. We say that $A \subseteq M$ is *lectically smaller* than $B \subseteq M$, if $B \neq A$ and the smallest element in which A and B differ belongs to B:

$$A < B :\Leftrightarrow \exists i \in B \backslash A : A \cap \{1, 2, \ldots, i-1\} = B \cap \{1, 2, \ldots, i-1\}$$



Next Closure Algorithm: Theorem

Some definitions before we start:

$$A <_i B :\Leftrightarrow i \in B \backslash A \wedge A \cap \{1, 2, \ldots, i-1\} = B \cap \{1, 2, \ldots, i-1\}$$

$$A + i := (A \cap \{1, 2, \ldots, i-1\}) \cup \{i\}$$

Theorem

The smallest concept intent larger than a given set $A \subset M$ with respect to the lectic order is

$$A \oplus i := (A + i)'',$$

with i being the largest element of M with $A <_i A \oplus i$.

Next Closure Algorithm

The Next Closure algorithm to compute all concept intents:

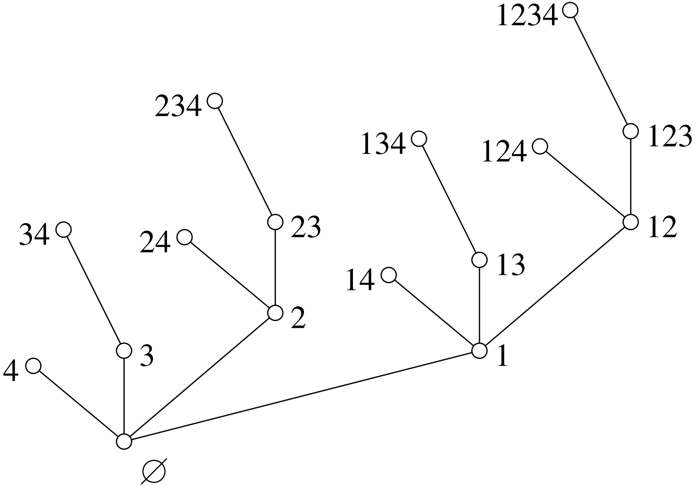
1. The lectically smallest concept intent is $\varnothing''$.
2. If A is a concept intent, we find the lectically next intent by checking all attributes $i \in M \setminus A$ (starting with the largest), continuing in descending order until for the first time $A <_i A \oplus i$. Then $A \oplus i$ is the lectically next intent.
3. If $A \oplus i = M$, we stop. Otherwise we set $A := A \oplus i$ and go to step 2.

Next Closure Algorithm: Example

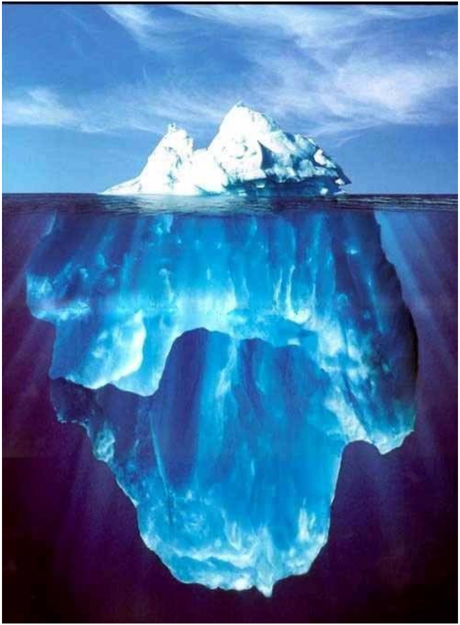
	mobile (1)	phone (2)	fax (3)	paper fax (4)
Sinus 44		×		
Nokia 6110	×	×		
T-Fax 301			×	×
T-Fax 360 PC				×

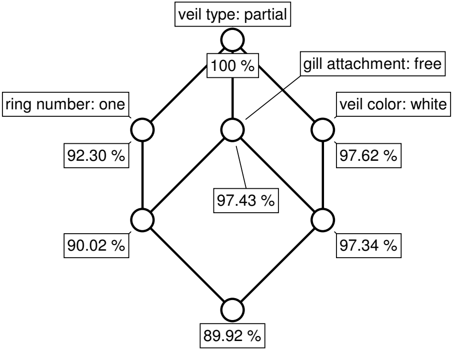
A	i	$A + i$	$A \oplus i := (A + i)''$	$A <_i A \oplus i$?	new intent

Next Closure Algorithm: Lectic Order



Iceberg Concept Lattices

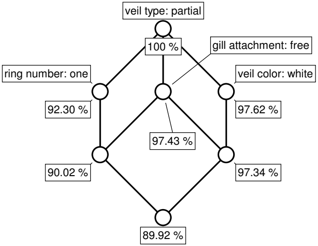


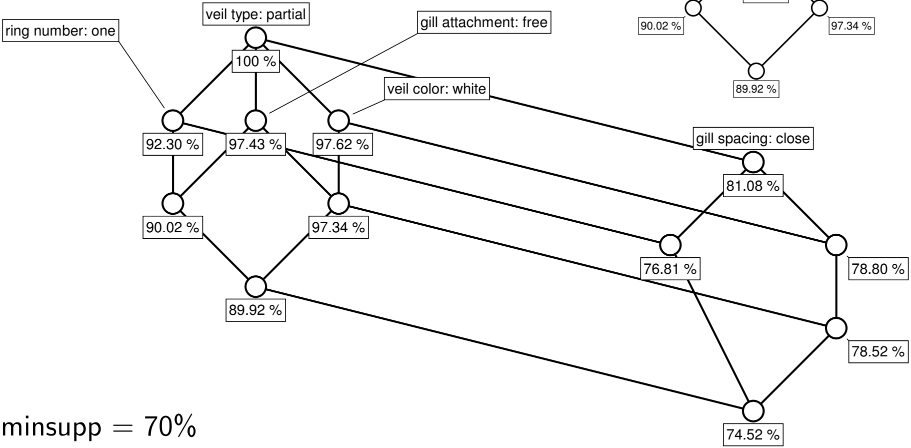


The seven most general concepts (for minsupp = 85%) of the 32086 concepts of the mushroom database (`http://kdd.ics.uci.edu/`).

Iceberg Concept Lattices

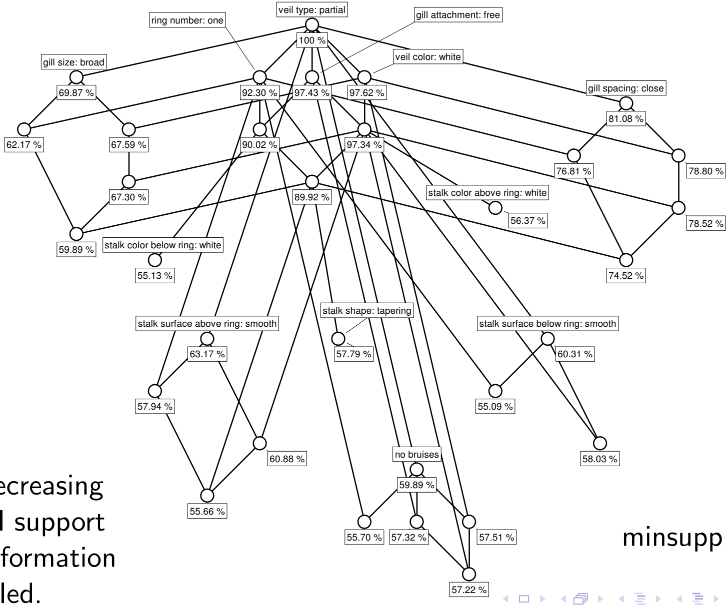
minsupp = 85%





minsupp = 70%

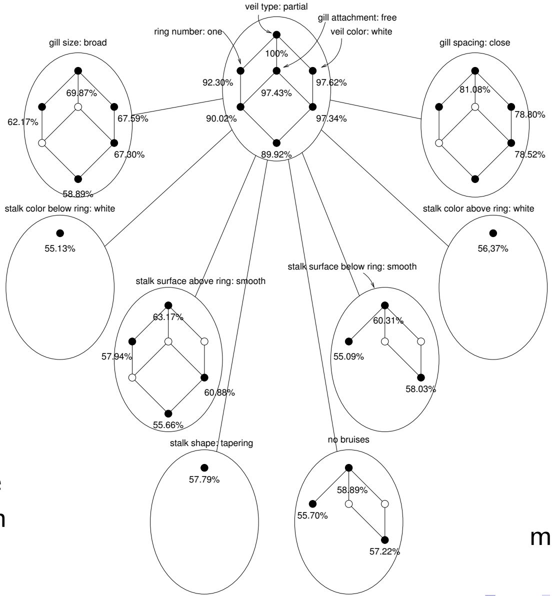
Iceberg Concept Lattices



With decreasing minimal support more information is revealed.

$\text{minsupp} = 55\%$

Iceberg Concept Lattices



In a nested line diagram we can read off implications.

minsupp $= 55\%$

Iceberg Concept Lattices: Support

Def.: The *support* of a set $X \subseteq M$ of attributes is defined as

$$\operatorname{supp}(X) := \frac{|X'|}{|G|}$$

Def.: The *iceberg concept lattice* of a formal context (G, M, I) for a given minimal support value $minsupp$ is the set

$$\{(A, B) \in \underline{\mathfrak{B}}(G, M, I) \mid \operatorname{supp}(B) \geqslant minsupp\}$$

The iceberg concept lattice can be computed using the TITANIC algorithm. (Stumme et al., 2001)

TITANIC Algorithm

TITANIC computes the closure system of all (*frequent*) concept intents using the *support* function $\operatorname{supp}(X) := \frac{|X'|}{|G|}$ (for a set $X \subseteq M$ of attributes).

frequent: only concept intents above a threshold $minsupp \in [0, 1]$

TITANIC Algorithm

TITANIC employs some simple properties of the support function:

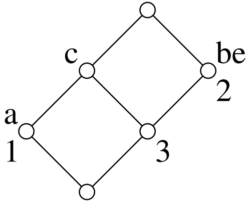
Lemma 4. Let $X, Y \subseteq M$.

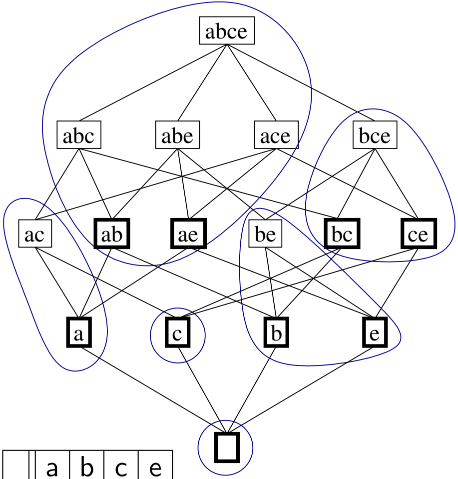
1. $X \subseteq Y \implies \text{supp}(X) \geqslant \text{supp}(Y)$
2. $X'' = Y'' \implies \text{supp}(X) = \text{supp}(Y)$
3. $X \subseteq Y \wedge \text{supp}(X) = \text{supp}(Y) \implies X'' = Y''$

TITANIC Algorithm

Lemma 4. Let $X, Y \subseteq M$.

1. $X \subseteq Y \implies \mathrm{supp}(X) \geqslant \mathrm{supp}(Y)$
2. $X'' = Y'' \implies \mathrm{supp}(X) = \mathrm{supp}(Y)$
3. $X \subseteq Y \wedge \mathrm{supp}(X) = \mathrm{supp}(Y) \implies X'' = Y''$





	a	b	c	e
1	×		×	
2		×		×
3		×	×	×

TITANIC Algorithm

TITANIC tries to optimize the following three questions:

1. How can we compute the closure of an attribute set using only the support values?
2. How can we compute the closure system such that we need to compute as few closures as possible?
3. How can we derive as many support values as possible from already known support values?

TITANIC Algorithm

1. How can we compute the closure of an attribute set using only the support values?

$$X'' = X \cup \{m \in M \setminus X \mid \text{supp}(X) = \text{supp}(X \cup \{m\})\}$$

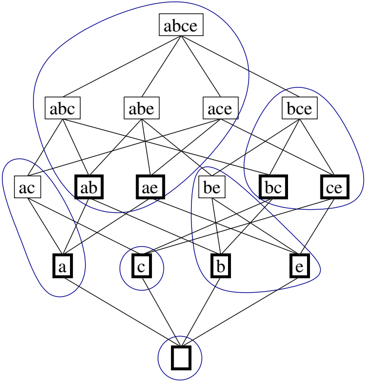
Example:
$\{b, c\}'' = \{b, c, e\}$, since

$\text{supp}(\{b, c\}) = \frac{1}{3}$

and

$\text{supp}(\{a, b, c\}) = \frac{0}{3}$
$\text{supp}(\{b, c, e\}) = \frac{1}{3}$

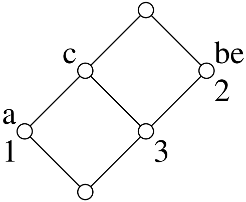
	a	b	c	e
1	×		×	
2		×		×
3		×	×	×

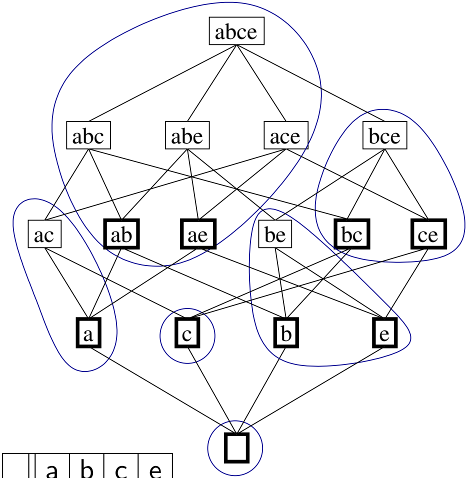


Titanic Algorithm

2. How can we compute the closure system such that we need to compute as few closures as possible?

We compute only the closures of the minimal generators.





	a	b	c	e
1	×		×	
2		×		×
3		×	×	×

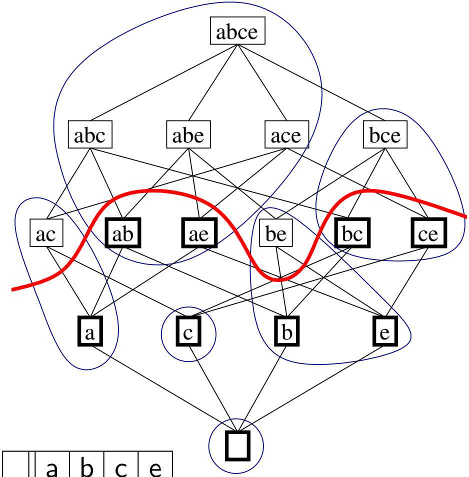
For this example Titanic needs two runs (Apriori four).

TITANIC Algorithm

2. How can we compute the closure system such that we need to compute as few closures as possible?

We compute only the closures of the minimal generators.

A set is a *minimal generator*, iff its support is unequal to the support of its lower covers.
The minimal generators form an order ideal (i.e., if a set is *not* a minimal generator, then none of its supersets is either)
→ approach similar to Apriori



	a	b	c	e
1	×		×	
2		×		×
3		×	×	×

For this example TITANIC needs two runs (Apriori four).

TITANIC Algorithm

TITANIC tries to optimize the following three questions:

1. How can we compute the closure of an attribute set using only the support values?

   $\rightarrow X'' = X \cup \{m \in M \backslash X \mid \text{supp}(X) = \text{supp}(X \cup \{m\})\}$

2. How can we compute the closure system such that we need to compute as few closures as possible?

   → approach similar to Apriori

3. How can we derive as many support values as possible from already known support values?

TITANIC Algorithm

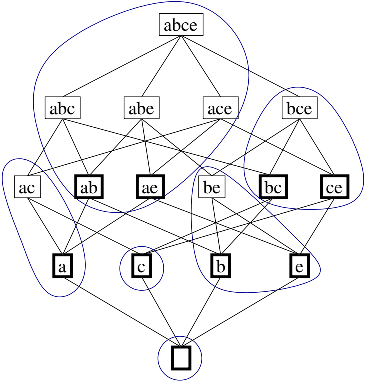
3. How can we derive as many support values as possible from already known support values?

Theorem: If X is *not* a minimal generator, then
$\text{supp}(X) = \min\{\text{supp}(K) \mid K \text{ is minimal generator}, K \subseteq X\}$

Example:
$\text{supp}(\{a,b,c\}) = \min\{\frac{0}{3}, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}\} = 0$
since the set is not a minimal generator and

$\text{supp}(\{a,b\}) = \frac{0}{3}$, $\text{supp}(\{b,c\}) = \frac{1}{3}$,
$\text{supp}(\{a\}) = \frac{1}{3}$, $\text{supp}(\{b\}) = \frac{2}{3}$,
$\text{supp}(\{c\}) = \frac{2}{3}$

Remark: It is sufficient, to check the largest minimal generators K with $K \subseteq X$, i.e., in this example $\{a,b\}$ and $\{b,c\}$.



	a	b	c	e
1	×		×	
2		×		×
3		×	×	×

TITANIC Algorithm

TITANIC tries to optimize the following three questions:

1. How can we compute the closure of an attribute set using only the support values?

   → $X'' = X \cup \{m \in M \setminus X \mid \operatorname{supp}(X) = \operatorname{supp}(X \cup \{m\})\}$

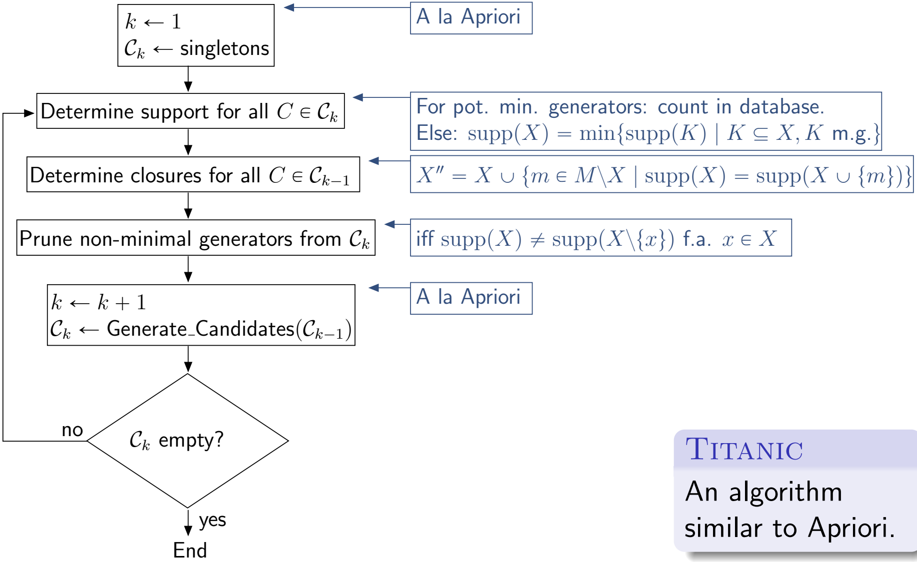
2. How can we compute the closure system such that we need to compute as few closures as possible?

   → *approach similar to Apriori*

3. How can we derive as many support values as possible from already known support values?

   → *If X is no minimal generator, then*
   $\operatorname{supp}(X) = \min\{\operatorname{supp}(K) \mid K \text{ *is minimal generator*}, K \subseteq X\}$

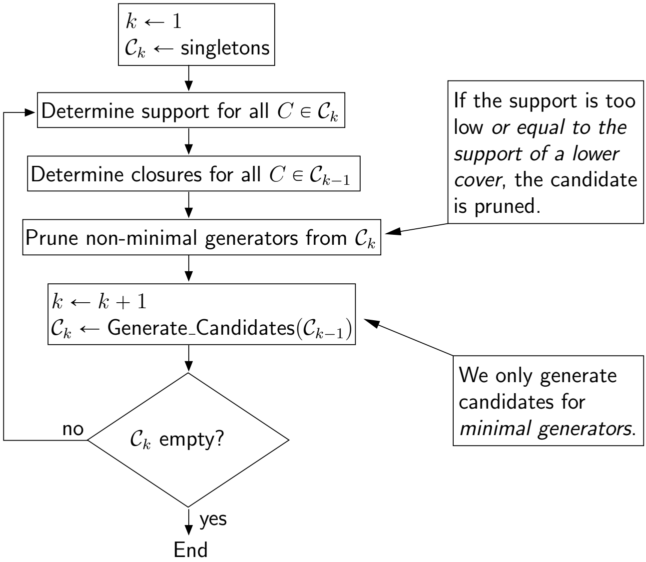
TITANIC Algorithm



TITANIC

An algorithm similar to Apriori.

TITANIC Algorithm: Compared to Apriori



TITANIC Algorithm

1) SUPPORT($\{\emptyset\}$);
2) $\mathcal{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) **forall** $m \in M$ **do** $\{m\}.p\_s \leftarrow \emptyset.s$;
5) $\mathcal{C} \leftarrow \{\{m\} \mid m \in M\}$;
6) **loop begin**
7) SUPPORT($\mathcal{C}$);
8) **forall** $X \in \mathcal{K}_{k-1}$ **do** $X.\text{closure} \leftarrow$ CLOSURE(X);
9) $\mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s\}$;
10) **if** $\mathcal{K}_k = \emptyset$ **then exit loop** ;
11) $k++$;
12) $\mathcal{C} \leftarrow$ TITANIC-GEN($\mathcal{K}_{k-1}$);
13) **end loop** ;
14) **return** $\bigcup_{i=0}^{k-1}\{X.\text{closure} \mid X \in \mathcal{K}_i\}$.

---

k	is the counter which indicates the current iteration. In the kth iteration, all key k-sets are determined.
$\mathcal{K}_k$	contains after the kth iteration all key k-sets K together with their support $K.s$ and their closure $K.\text{closure}$.
$\mathcal{C}$	stores the candidate k-sets C together with a counter $C.p\_s$ which stores the minimum of the supports of all $(k-1)$-subsets of C. The counter is used in step 9 to prune all non-key sets.

TITANIC Algorithm: TITANIC-GEN

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets K with their support $K.s$.

Output: $\mathcal{C}$, the set of candidate k-sets C
with the values $C.p\_s := \min\{\mathrm{supp}(C \setminus \{m\}) \mid m \in C\}$.

The variables p_s assigned to the sets $\{m_1, \ldots, m_k\}$ which are generated in step 1 are initialized by $\{m_1, \ldots, m_k\}.p\_s \leftarrow s_{\max}$.

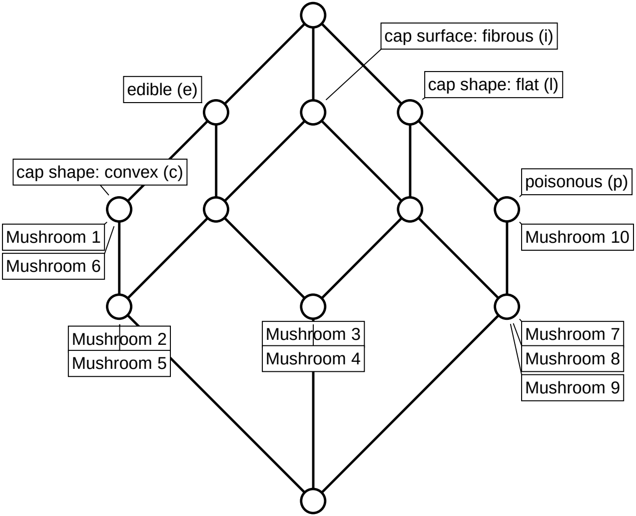
1) $\mathcal{C} \leftarrow \{\{m_1 < m_2 < \cdots < m_k\} \mid \{m_1, \ldots, m_{k-2}, m_{k-1}\}, \{m_1, \ldots, m_{k-2}, m_k\} \in \mathcal{K}_{k-1}\}$
2) **forall** $X \in \mathcal{C}$ **do begin**
3) **forall** $(k-1)$-subsets S of X **do begin**
4) **if** $S \notin \mathcal{K}_{k-1}$ **then begin** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{X\}$; **exit forall** ; **end**;
5) $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6) **end**;
7) **end**;
8) **return** $\mathcal{C}$.

TITANIC Algorithm: CLOSURE(X) for $X \in \mathcal{K}_{k-1}$

$Y \leftarrow X$;
forall $m \in X$ **do** $Y \leftarrow Y \cup (X \backslash \{m\})$.closure;
forall $m \in M \backslash Y$ **do begin**
  if $X \cup \{m\} \in \mathcal{C}$ **then** $s \leftarrow (X \cup \{m\}).s$
        else $s \leftarrow \min\{K.s \mid K \in \mathcal{K},\ K \subseteq X \cup \{m\}\}$;
  if $s = X.s$ **then** $Y \leftarrow Y \cup \{m\}$
end;
return Y.

TITANIC Algorithm: Example

	edible (e)	poisonous (p)	cap shape: convex (c)	cap shape: flat (l)	cap surface: fibrous (i)
Mushroom 1	×		×		
Mushroom 2	×		×		×
Mushroom 3	×			×	×
Mushroom 4	×			×	×
Mushroom 5	×		×		×
Mushroom 6	×		×		
Mushroom 7		×		×	×
Mushroom 8		×		×	×
Mushroom 9		×		×	×
Mushroom 10		×		×	



TITANIC Algorithm: Example

$k = 0$:

step 1		step 2
X	$X.s$	$X \in \mathcal{K}_k$?
$\emptyset$	1	yes

$k = 1$:

steps 4+5		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e\}$	1	6/10	yes
$\{p\}$	1	4/10	yes
$\{c\}$	1	4/10	yes
$\{l\}$	1	6/10	yes
$\{i\}$	1	7/10	yes

Step 8 returns: $\emptyset.\text{closure} \leftarrow \emptyset$

Then the algorithm repeats the loop for $k = 2, 3$, and 4:

	edible (e)	poisonous (p)	cap shape: convex (c)	cap shape: flat (l)	cap surface: fibrous (i)
Mushroom 1	×		×		
Mushroom 2	×		×		×
Mushroom 3	×			×	×
Mushroom 4	×			×	×
Mushroom 5	×		×		×
Mushroom 6	×		×		
Mushroom 7		×		×	×
Mushroom 8		×		×	×
Mushroom 9		×		×	×
Mushroom 10		×		×	

TITANIC Algorithm: Example

$k = 2$:

step 12		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e,p\}$	4/10	0	yes
$\{e,c\}$	4/10	4/10	no
$\{e,l\}$	6/10	2/10	yes
$\{e,i\}$	6/10	4/10	yes
$\{p,c\}$	4/10	0	yes
$\{p,l\}$	4/10	4/10	no
$\{p,i\}$	4/10	3/10	yes
$\{c,l\}$	4/10	0	yes
$\{c,i\}$	4/10	2/10	yes
$\{l,i\}$	6/10	5/10	yes

Step 8 returns:

$\{e\}.\text{closure} \leftarrow \{e\}$
$\{p\}.\text{closure} \leftarrow \{p,l\}$
$\{c\}.\text{closure} \leftarrow \{c,e\}$
$\{l\}.\text{closure} \leftarrow \{l\}$
$\{i\}.\text{closure} \leftarrow \{i\}$

$k = 3$:

step 12		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e,l,i\}$	2/10	2/10	no
$\{e,p,i\}$	0	0	no
$\{p,c,i\}$	0	0	no
$\{c,l,i\}$	0	0	no

Step 8 returns:

$\{e,p\}.\text{closure} \leftarrow \{e,p,c,l,i\}$
$\{e,l\}.\text{closure} \leftarrow \{e,l,i\}$
$\{e,i\}.\text{closure} \leftarrow \{e,i\}$
$\{p,c\}.\text{closure} \leftarrow \{e,p,c,l,i\}$
$\{p,i\}.\text{closure} \leftarrow \{p,l,i\}$
$\{c,l\}.\text{closure} \leftarrow \{e,p,c,l,i\}$
$\{c,i\}.\text{closure} \leftarrow \{e,c,i\}$
$\{l,i\}.\text{closure} \leftarrow \{l,i\}$

	edible (e)	poisonous (p)	cap shape: convex (c)	cap shape: flat (l)	cap surface: fibrous (i)
Mushroom 1	×		×		
Mushroom 2	×		×		×
Mushroom 3	×			×	×
Mushroom 4	×			×	×
Mushroom 5	×		×		×
Mushroom 6	×		×		
Mushroom 7		×		×	×
Mushroom 8		×		×	×
Mushroom 9		×		×	×
Mushroom 10		×		×	

Titanic Algorithm: Example

Since $\mathcal{K}_k$ is empty the loop is exited in step 10.

Finally the algorithm collects all concept intents (step 14):

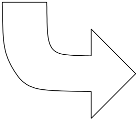
$$\emptyset,\ \{e\},\ \{p,l\},\ \{c,e\},\ \{l\},\ \{i\},\ \{e,p,c,l,i\},\{e,l,i\},\ \{e,i\},\ \{p,l,i\},\ \{e,c,i\},\ \{l,i\}$$

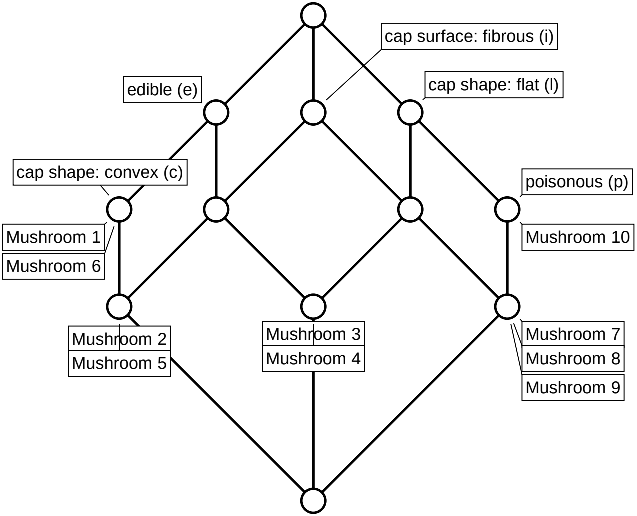
(which are exactly the intents of the concepts of the concept lattice on Slide 30). The algorithm determined the support of $5+10+3=18$ attribute sets in three passes of the database.

	edible (e)	poisonous (p)	cap shape: convex (c)	cap shape: flat (l)	cap surface: fibrous (i)
Mushroom 1	X		X		
Mushroom 2	X		X		X
Mushroom 3	X			X	X
Mushroom 4	X			X	X
Mushroom 5	X		X		X
Mushroom 6	X		X		
Mushroom 7		X		X	X
Mushroom 8		X		X	X
Mushroom 9		X		X	X
Mushroom 10		X		X	

TITANIC Algorithm: Example

$\emptyset$, $\{e\}$, $\{p, l\}$, $\{c, e\}$, $\{l\}$, $\{i\}$, $\{e, p, c, l, i\}$, $\{e, l, i\}$, $\{e, i\}$, $\{p, l, i\}$, $\{e, c, i\}$, $\{l, i\}$





TITANIC Algorithm: vs. NEXT CLOSURE

- NEXT CLOSURE uses almost no memory.
- NEXT CLOSURE can explicitly employ symmetries between attributes.
- NEXT CLOSURE can be used for knowledge discovery.
- TITANIC is much more performant, in particular on large datasets.
- TITANIC allows us to incorporate and employ minimal support constraints (next slide).

TITANIC Algorithm: Computing Iceberg Concept Lattices

- stop as soon as only *non-frequent* minimal generators are computed
- return only the closures of *frequent* minimal generators
- generate candidates only from the *frequent* minimal generators
- all subsets of candidates with $k-1$ elements must be *frequent*