Data mining

Roberto Innocente

inno@sissa.it

for the ICTP workshop on

Web enabling technologies

Introduction /1

Data mining also known as **Knowledge Discovery in Databases** or **KDD** (Piatesky-Shapiro 1991), is the process of extracting useful hidden information from very large databases in an unsupervised manner.

Introduction /2

Central themes of data mining are:

- Classification
- Cluster analysis
- Associations analysis
- Outlier analysis
- Evolution analysis

Classification/Prediction

Classification is a 2-step process :

- First a model is built based on a **training set** of tuples that produces a set of classification rules **:** e.g. If age = “20..30” and income = high then credit_rating=“excellent”
- Then the model is applied to a real data set and from this a **test-set** is extracted and the **accuracy** of the model computed

Cluster analysis/1

It is called so the process of grouping into classes objects that are similar in such a way to maximize the differences between objects in different classes and minimize those between objects in the same class.

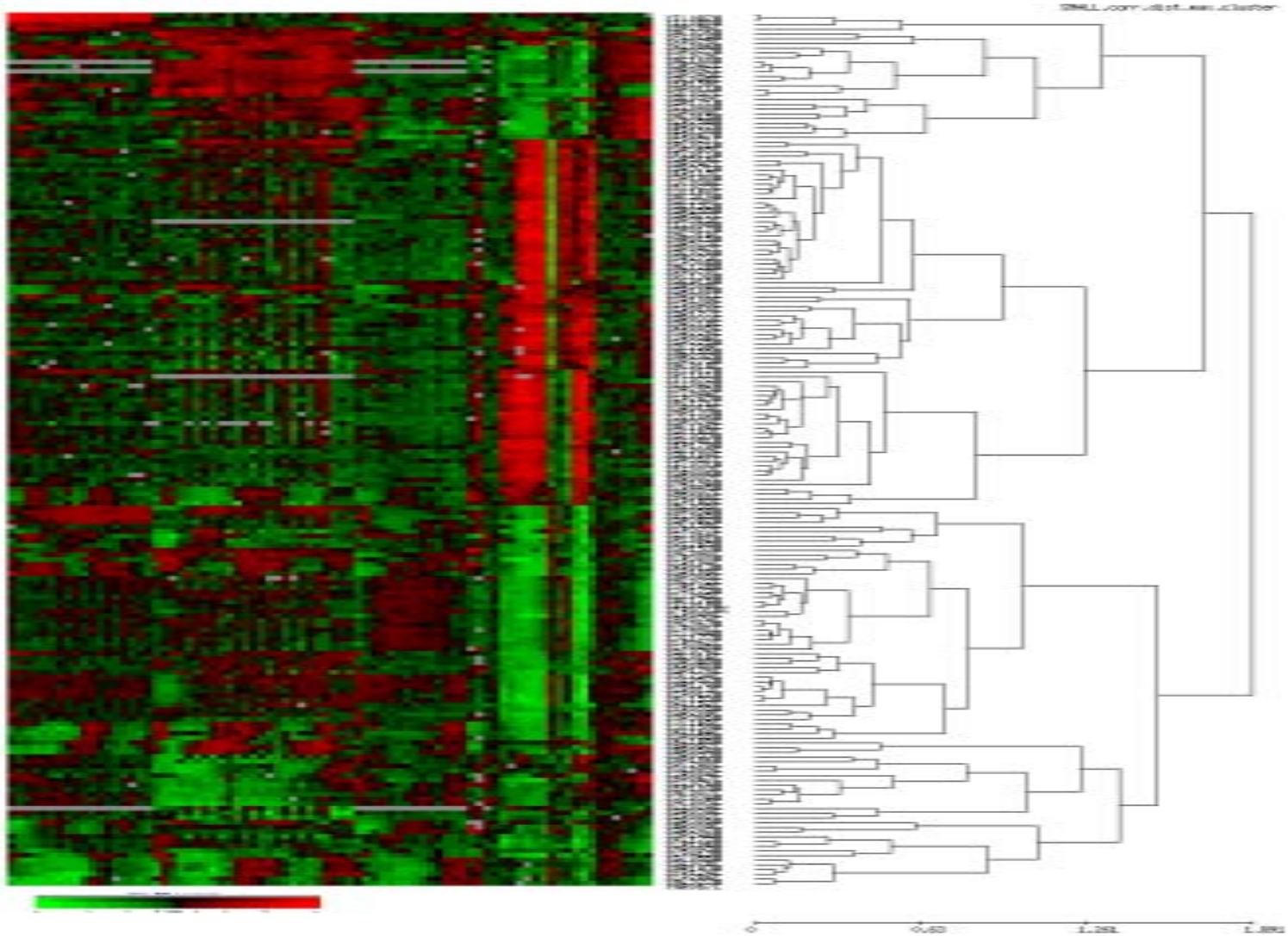
A typical algorithm used to do so in an unsupervised way is the *agglomerative hierarchical clustering* algorithm.

Cluster analysis/2

This algorithms starts putting each object in a separate class. Then using some distance function defined between objects and clusters of objects, merges the two less dissimilar clusters into a larger clusters until all objects belong to only 1 cluster.

This produces a binary tree also called a *dendrogram* with the final cluster being the root.

Cluster analysis/3



- dendrogram of a gene expression microarray

Outliers

Very often there are objects in a data set that do not comply to the general model of the data.

These object that are usually very different from the others are called *outliers*. In many cases for data mining it is extremely important to minimize the influence of outlier, eventually removing it from the data set.

ARM /1 (association rules mining)

- Formally introduced in 1993 by Agrawal, Imielinski and Swami (AIS) in connection with ***market basket analysis***
- Formalizes statements of the form:

What is the percentage of customers that together with cheese buy beer ?

ARM /2

- We have a set of items I={i1,i2,..}, and a set of transaction T={t1,t2..}. Each transaction (like a supermarket bill) is a set of items (or better as it is called an **itemset**)
- If U and V are disjoint itemsets, we call **support** of U=>V the fraction of transactions that contain U ∪ V and we indicate this with **s(U=>V)**
- We say that an itemset is **frequent** if its support is greater than a chosen threshold called **minsupp**.
- If A and B are disjoint itemsets, we call **confidence** of A=>B and indicate with **c(A=>B)**, the fraction of transactions containing A that contain also B. This is also called the Bayesian or conditional probability p(B|A).
- We say that a rule is **strong** if its confidence is greater than a threshold called **minconf.**

ARM /3

ARM can then be formulated as:

Given a set I of **items** and a set T of **transactions** over I, produce in an automated manner all association rules that are more than x% **frequent** and more than y% **strong**.

ARM /4

On the right we have 6 transactions T={1,2,3,4,5,6} on a set of 5 items I={A,B,C,D,E}

The itemset BC is present in the transactions {1,2,3,4,5} so its support s(B=>C) = 5/6

The confidence of B=>C, given that BC is present in 5 transactions and B is present in all 6 transactions, is c(B=>C) = s(B=>C)/s(B)=5/6

Trans	Itemset
1	BCDE
2	ABC
3	BCDE
4	ABCE
5	ABCDE
6	ABD

Horizontal format

Item	Transactions
A	2456
B	123456
C	12345
D	1356
E	1345

Vertical format

ARM /5

Another possible representation is the **matrix representation,** that can combine the properties of the so called horizontal and vertical format.

	Items				
Transaction	A	B	C	D	E
1		1	1	1	1
2	1	1	1		
3		1	1	1	1
4	1	1	1		1
5	1	1	1	1	1
6	1	1		1	

Matrix representation

ARM /6

All algorithms divide the search in two phases :

- Find frequent itemsets
- Find the strong association rules for each frequent itemset

ARM /7

The second phase can in principle be quite simple.

To find the strong association rules associated with an itemset U, simply :

```
for each proper subset A of U :
    If s(U)/s(A) is more than minconf then
        the rule A=>(U-A) is strong
```

For this reason, in what follows, only the search for frequent itemsets will be investigated.

Quantitative rules mining

It is possible to consider the case where an attribute is not boolean (true or false), but assumes a value. For example **age** in census data is such an attribute.

It is possible to reduce this case to the case of boolean attributes **binning** the range of the attribute value. For example age can be translated to the following boolean attributes:

- Young (age 0-30)
- Adult (31-65)
- Old (66-)

The expression level of a gene (0-255) can be represented by :

- Low (0-50)
- Medium(51-205)
- High(206-)

Sequential mining /1

In this case the database rows are **eventsets** with a timestamp:

- 110 A,B,E
- 150 E,F
- 160 A

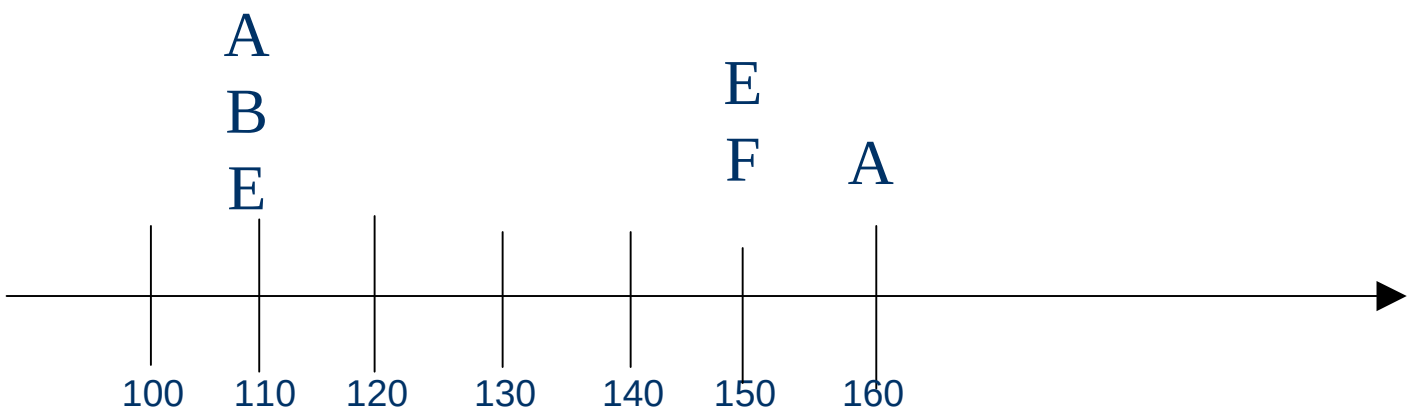
We are interested in frequent **episodes** (sequences of eventsets), like :

- (A,(B,E),F)

(where (B,E) is an eventset) occurring in a **time window:**

event A precedes the eventset (B,E) that precedes F.

Sequential mining /2



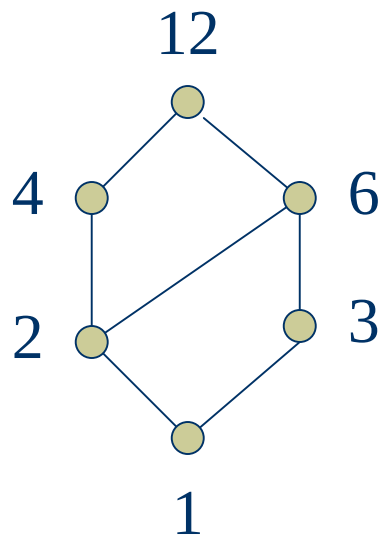
Sequential mining /3

- It’s been applied for example to the alarms of the finnish telephone network
- It can be applied to temporal series of gene expression

Posets /1

Given a set U, a binary relation ≤ reflexive, antisymmetric and transititve, is called a **partial order** (or an **order** *tout-court*), and (U,≤) is called a **partially ordered set** (or a **poset)**

A poset is frequently represented with a **Hasse diagram**, a diagram in which if a ≤ b, then there is an ascending path from a to b. The binary relation on **N,** *is divisor of* (usually represented with |) is a partial order on **N.**

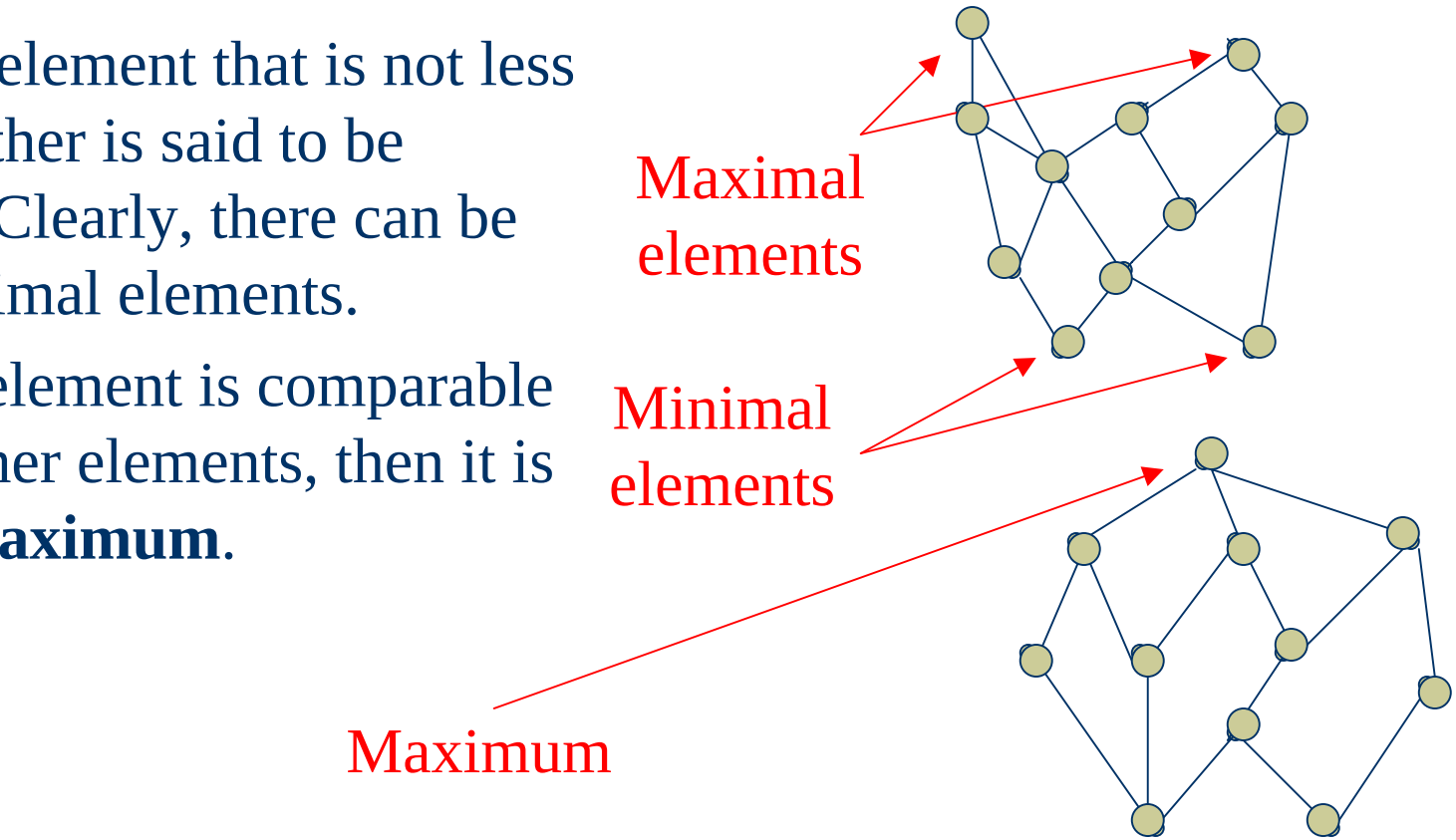


Hasse diagram of the divisors of 12

Posets /2

In a poset, an element that is not less than any other is said to be **maximal**. Clearly, there can be many maximal elements.

If a maximal element is comparable with all other elements, then it is the only **maximum**.



Posets /3

If (U, ≤) and (V, ≤) are two posets, a pair of functions f:U->V and g:V->U such that (u,u' ∈ U; v,v'∈ V) :

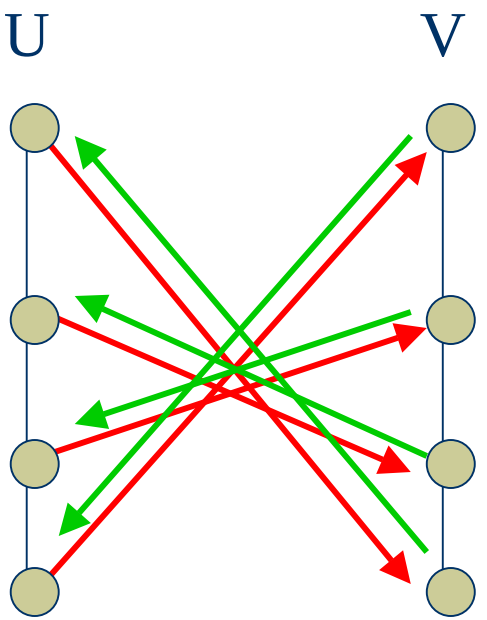
- if u ≤ u' then f(u') ≤ f(u)
- if v ≤ v' then g(v') ≤ g(v)
- u ≤ g(f(u))
- v ≤ f(g(v))

anti-homo morphism

are said to be a **Galois connection** between the two posets.

From the above properties we can deduce:

f(g(f(u))) = f(u) and g(f(g(v)))=g(v)



In this example U and V are linear orders

f → g ←

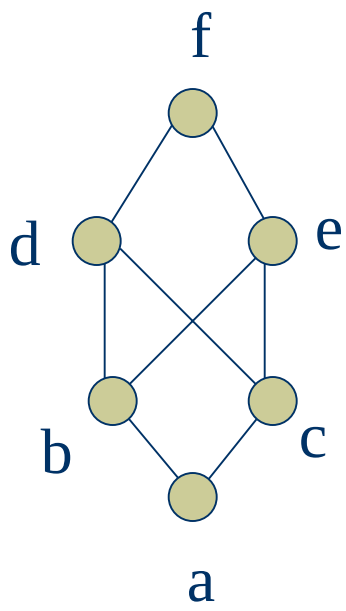
Lattices /1

A poset in which for each pair of elements u, v it exists an element z that is the **least upper bound** (or **join** or lub) and an element w that is the **greatest lower bound** (or **meet** or glb) is said to be a **lattice.**

This allows us to define 2 binary operators :

$z = \text{join}(u,v) = u \cup v$

$w = \text{meet}(u,v) = u \cap v$



This poset is not a lattice because there is no lub for (b,c)

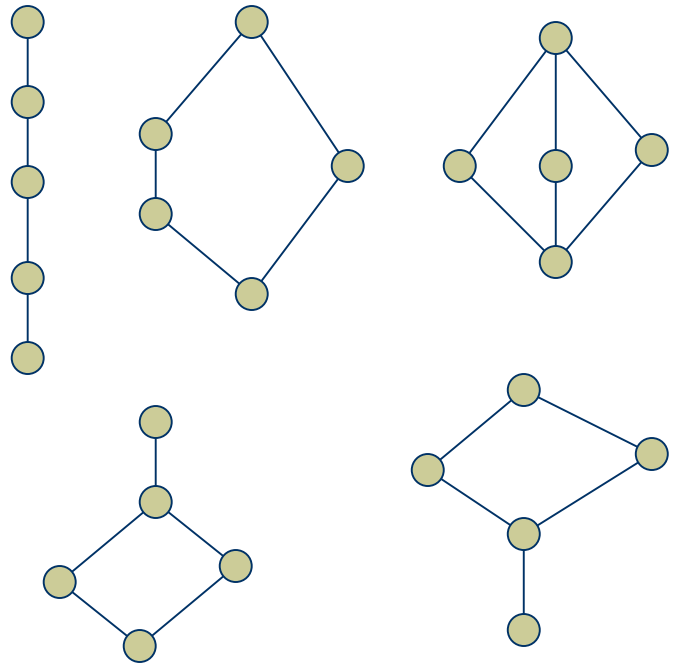
Lattices /2

We say that a lattice is **complete** if it has glb and lub for each subset. Every finite lattice is complete.

A poset is called a **join semilattice** (**meet semilattice**) if only the join (meet) exists.

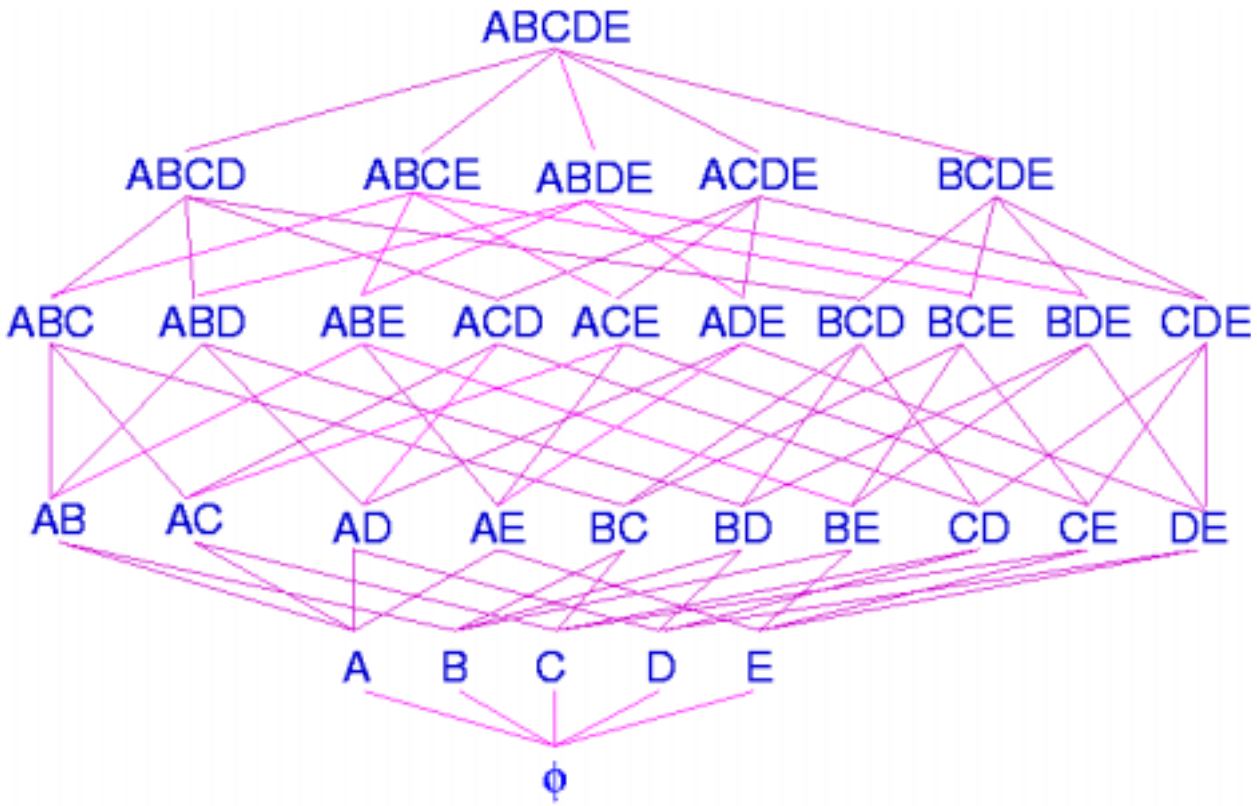
The powerset of a set ordered by inclusion is a complete lattice.

Frequent sets are only a meet-semilattice**.**



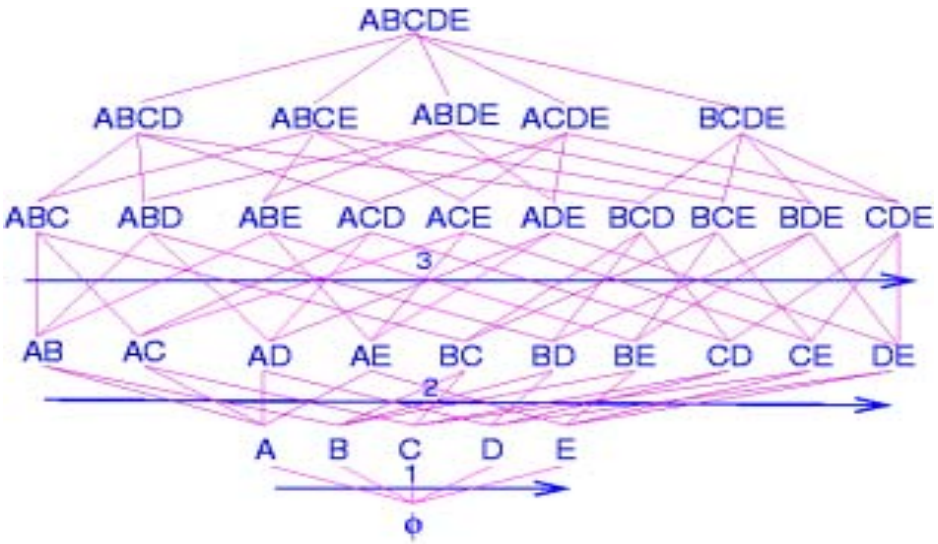
All lattices of order 5 (up to ismorphism)

Lattices /3

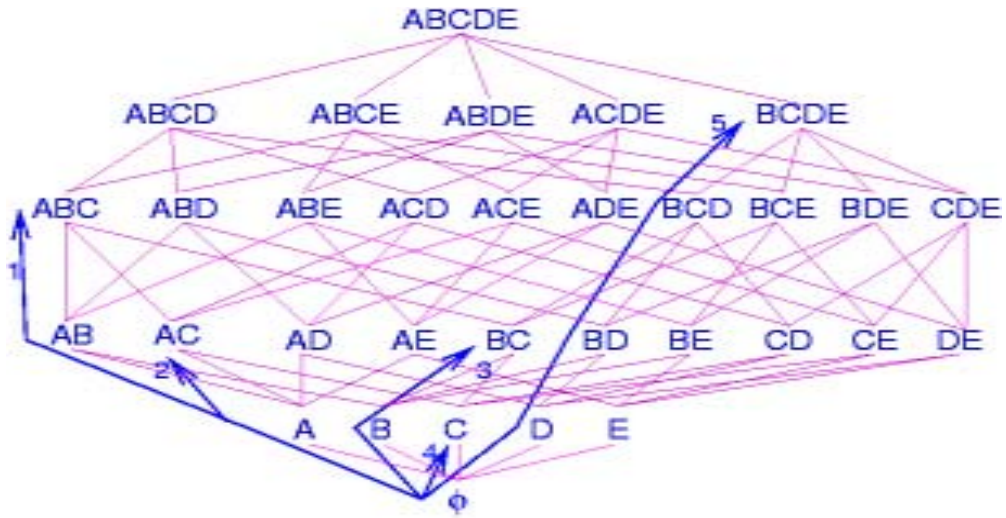


Hasse diagram of the powerset of {A,B,C,D,E} ordered by inclusion

Algorithmic families /1



Breadth first algorithms (or level wise)



Depth first algorithms

There are two ways in which you can run over the lattice of subsets in bottom-up order.

These ways correspond to two families of algorithms :

- *breadth first*
- *depth first*

Algorithmic families /2

Infrequent itemsets, have the property that all their supersets are also infrequent. Infrequent itmesets form a join semilattice. Minimal infrequent itemsets are sufficient to completely specify the semilattice.

So it makes sense to run over the lattice also in top-down order. Or better, as in **hybrid** algorithms, mixing bottom-up with top-down.

Furthermore the search can be performed for :

- *all frequent itemsets*
- *only maximal frequent itemsets*
- *closed frequent itemsets* (will be defined later)

Apriori /1

Presented by Agrawal and Srikant in 1994.

Fast algorithms for mining Association Rules

(IBM Almaden Research Center)

essentially based on the hereditary property that :

All subsets of a frequent itemset are also frequent

It performs a breadth first search.

If *l* is the maximum length of frequent itemsets then it performs *l* scans of the database.

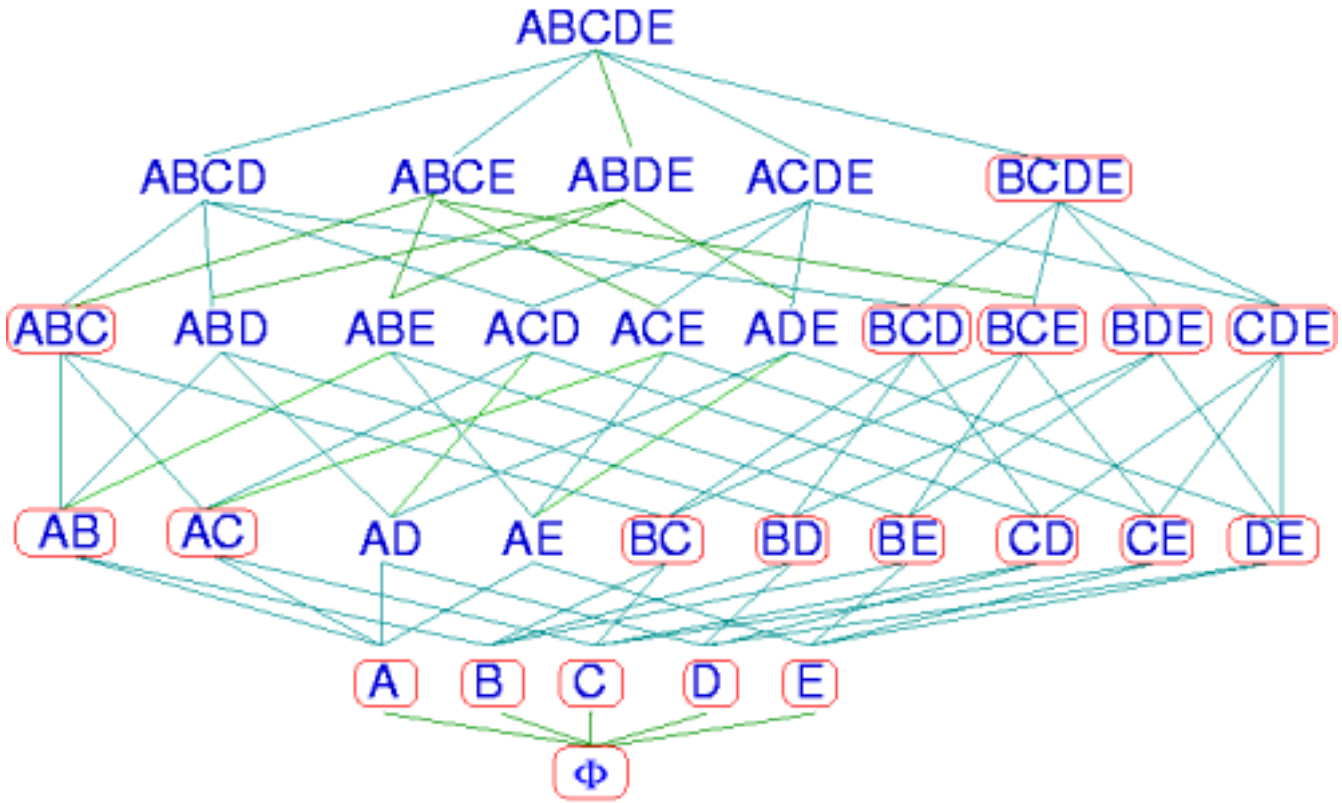
Apriori /2

```
F(1) = { frequent 1-itemsets}
for (k=2; F(k-1) not empty; k++) {
   C(k) = generate_candidates(F(k-1));
   forall transactions t in T {
      Ct = subset(C(k),t); // Ct are the C(k)
                           // candidates present in t
      forall candidates c in Ct  { c.count++: }
    }
   F(k) = {c in C(k) and c.count >= minsup}
}
Answer =  {all F(k) }
```

Apriori /3

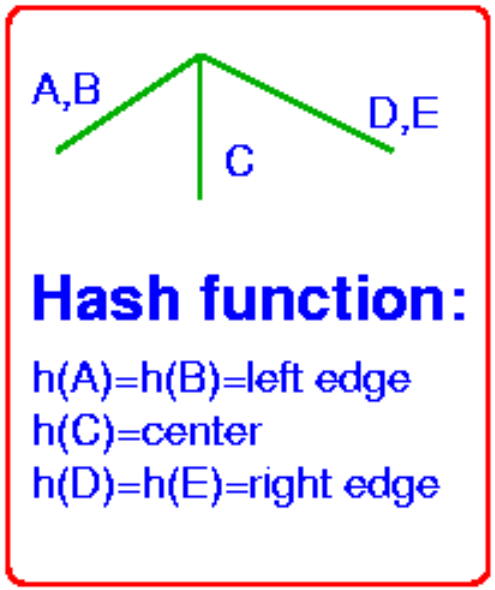
```
generate_candidates(F(k-1) {
    join  :
        for each pair l1,l2 in F(k-1){
          if l1 and l2 are (k-1)-itemsets pairs in F(k-1) that differ just in the last item then
                l1 ∪  l2 is a k-itemset candidate
         }
    pruning:
        foreach k-itemset candidate {
              if one of its (k-1)-subsets is not frequent then
                    prune it
         }
}
```

Apriori /4

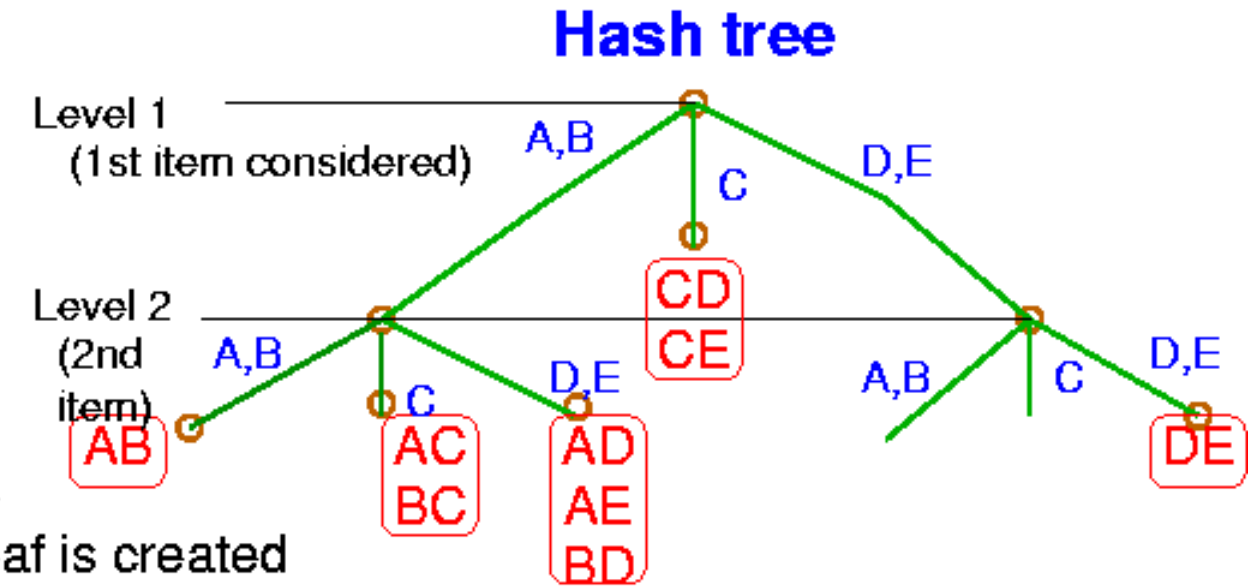


Frequent itemsets are marked in red

Apriori /5



C(k) candidates are kept in an Hash tree



When an interior node
is overflowed a new leaf is created
and candidates are redistributed
according to their l–th item

Apriori /6

- To find a frequent k-itemset it requires k passes over the database
- Frequent itemsets of over 50-60 items are not feasible. Apriori needs to run over all 2^60-1 frequent subsets
- Just a small example : find all itemsets with minsup=0.5 in the 2 transactions:

  (a1,a2,.....,a100)

  (a1,a2,.....,a100,a101,...,a200)

Partition /1

Presented by Savasere, Omiecinski, Navathe in VLDB conference 1995, it requires 2 passes over the database

- It partitions the database into a number of non-overlapping partitions
- Each partition is read and **vertical *tidlist*** (lists of transaction ids) are formed for each item
- Then all locally (local to the partition) frequent itemsets are generated via tidlist intersection
- After having scanned all partitions, all local frequent itemsets are merged to form the global candidates
- Itemsets frequent over all partitions are clearly frequent and so eliminated from the following pass
- A new scan of the database is performed, it is transformed in the tidlist format and counts of the global candidates are performed using tidlist intersection

Partition /2

This algorithm uses the vertical format (*tidlists*).

It performs only 2 scans of the database.

Partitions are calculated in such a way to be able to keep all their tidlists in memory.

FP-growth /1

Presented by Han, Pei, Yin in 2000.

This method does'nt require candidate generation, but stores in an efficient novel structure, an **FP-tree** (a Frequent Pattern tree**,** a version of a **prefix tree**), the transaction database.

It scans the database once to find frequent items. Frequent items F are then sorted in descending support count and kept in a list L.

Another scan of the databases is then performed, and for each transaction: infrequent items are suppressed and the remaining items are sorted in L-order and inserted in the FP-tree.

FP-growth /2

A null root node is created. Then for each *normalized* (w/o infrequent items and sorted in L-order) transaction t :

```
insert_tree(t,tree) {
    if tree has a child node equal to head(t) then
        increment the child count by 1
    else
        create a new child node and set its count to 1
    if rest(t) is non empty then
         insert_tree(rest(t),tree.child(head(t))
}
```

FP-growth /3

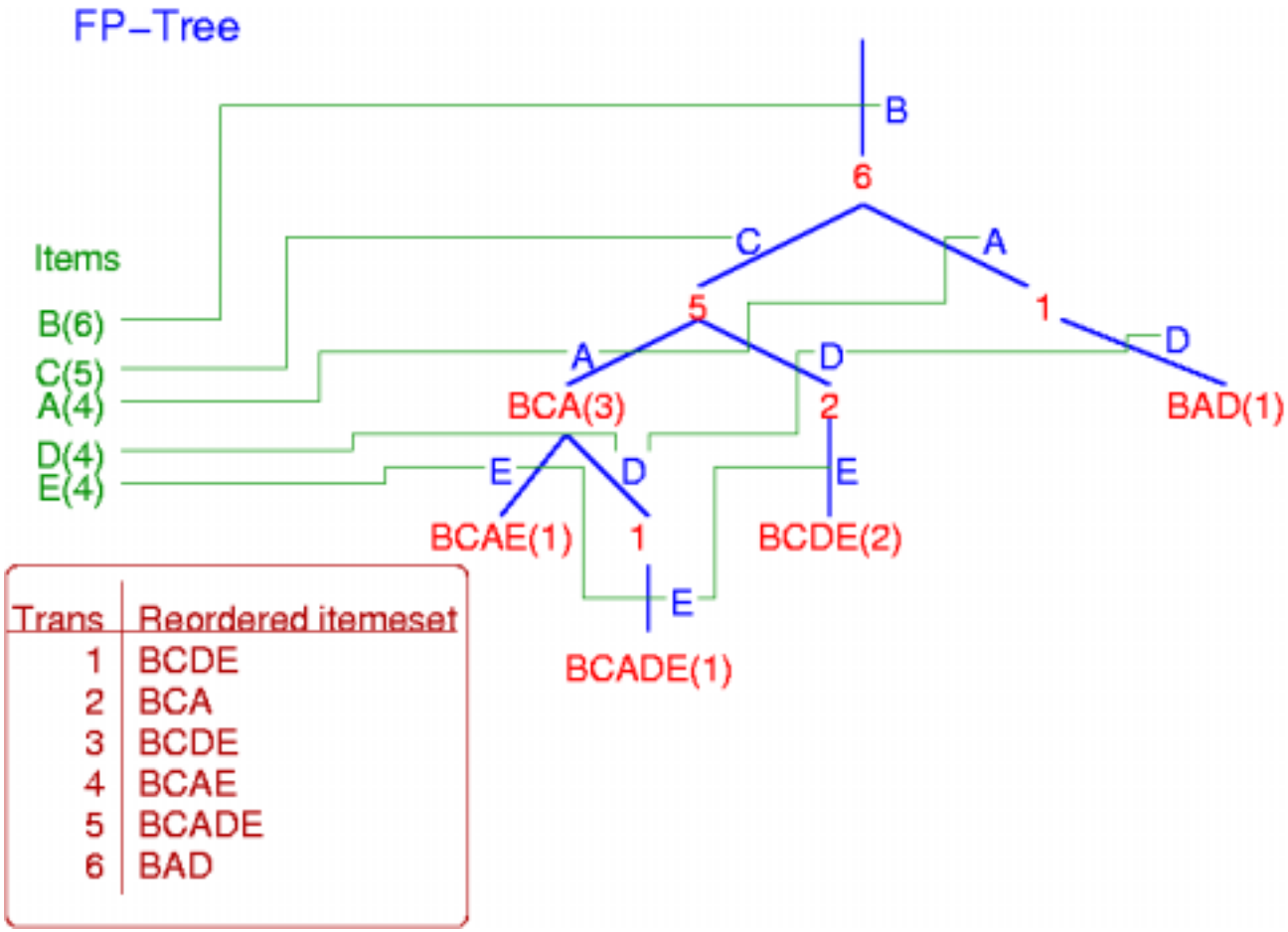
During the construction of the FP-tree, for each frequent item, a list linking all its presences in the FP-tree is kept updated.

Now all the information needed to mine frequent patterns is available in the FP-tree.

Having sorted the items inside transactions in L order, increases the probability to share prefixes.

It often happens that the FP-tree is much more compact than the original database.

FP-growth /4


FP–Tree
Items
B(6)
C(5)
A(4)
D(4)
E(4)
B
6
C
A
5
1
A
D
D
BCA(3)
2
BAD(1)
E
D
E
BCAE(1)
1
BCDE(2)
E
BCADE(1)
Trans
Reordered itemeset
1 BCDE
2 BCA
3 BCDE
4 BCAE
5 BCADE
6 BAD

FP-growth /5

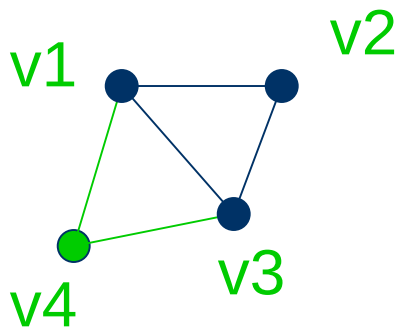
Now, for each frequent item alpha (in reversed L-order):

```
FP-growth(alpha,tree) {
 If tree has a single path P:
    for each combination U of nodes in P:
        form alpha ∪ U with support equal to the minsupport
              of items in U
  else:
    for each child c of tree:
        form beta = c ∪alpha with support supp(c)
        construct the tree of prefixes FP-tree(beta)
        if FP-tree(beta) is not empty:
              FP-growth(beta,FP-tree(beta)
}
```

Graphs /1

- A graph has a set of **vertices** V, and a set of **edges** E
- A **subgraph** G' of a graph G has part of the vertices of G and part of the edges G has between those vertices

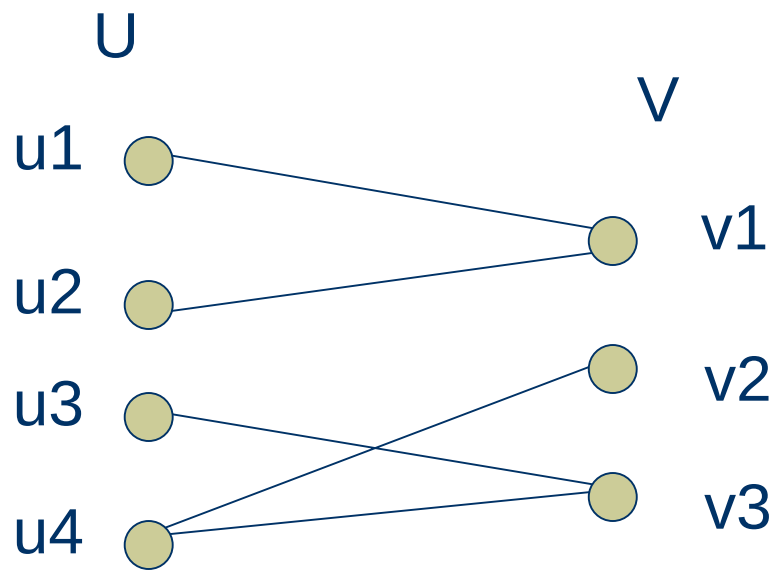
G=(V,E)
V={v1,v2,v3,v4}
E={(v1,v2),(v2,v3),(v3,v4),
(v4,v1),(v1,v3)}



G'=(V',E') (blue subgraph)
V'={v1,v2,v3)
E'={(v1,v2),(v2,v3),(v3,v1)}

Graphs /2

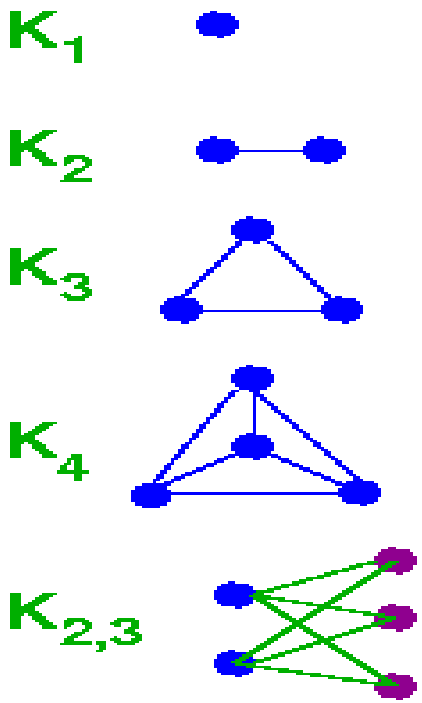
- A **bi-partite** graph is a graph in which the vertices can be partitioned into two sets U and V (with void intersection) and all the edges of the graph have a vertex in U and one in V (there are no edges between vertices in U or in V)



A bi-partite graph (U,V,E)

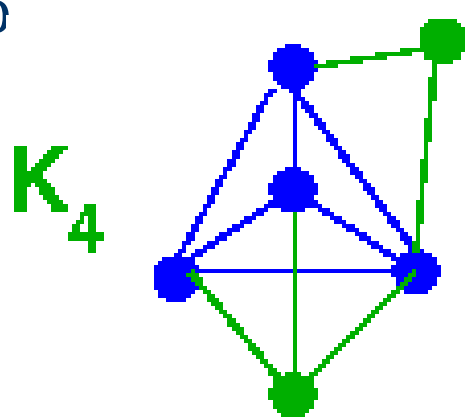
Graphs /3

- A graph is said to be **complete** if for each pair of vertices there is an edge connecting them
- Complete graphs are usually indicated by K
- A bi-partite graph is said to be complete if ...



Graphs /4

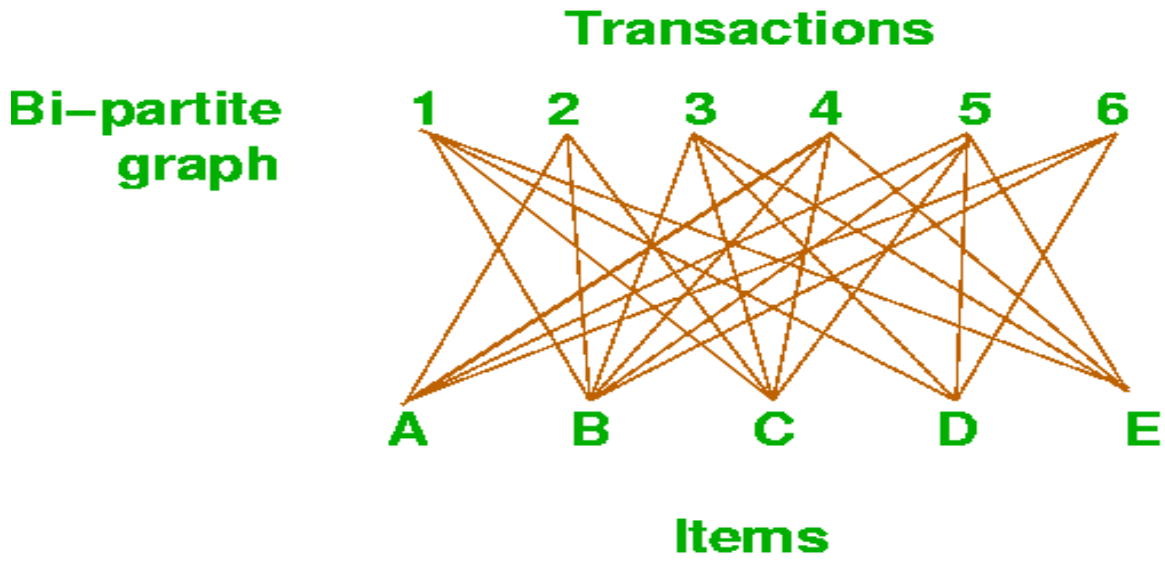
- A **complete** subgraph is said to be a **clique**
- A clique that is not contained in another is said to be a **maximal clique**



The blue subgraph is a maximal clique

Graphs /5

There is an edge between a transaction and an item if the item is present in the transaction.



Maximal cliques:

$K_{3,3} = (^{2}_{A}\ ^{4}_{B}\ ^{5}_{C})$

$K_{3,4} = (^{1}_{B}\ ^{3}_{C}\ ^{5}_{D}\ ^{}_{E})$

maximal frequent itemsets are maximal cliques of the bi–partite graph

Finding the maximal cliques in a bi–partite graph is a known NP–complete problem.

Max clique /1

Presented by M.Zaki, Parthasarathy, Ogihara, Li in 1997.

It computes frequent 1-itemsets and 2-itemsets as Apriori.

But then it tries to find only all **maximal frequent itemsets**.

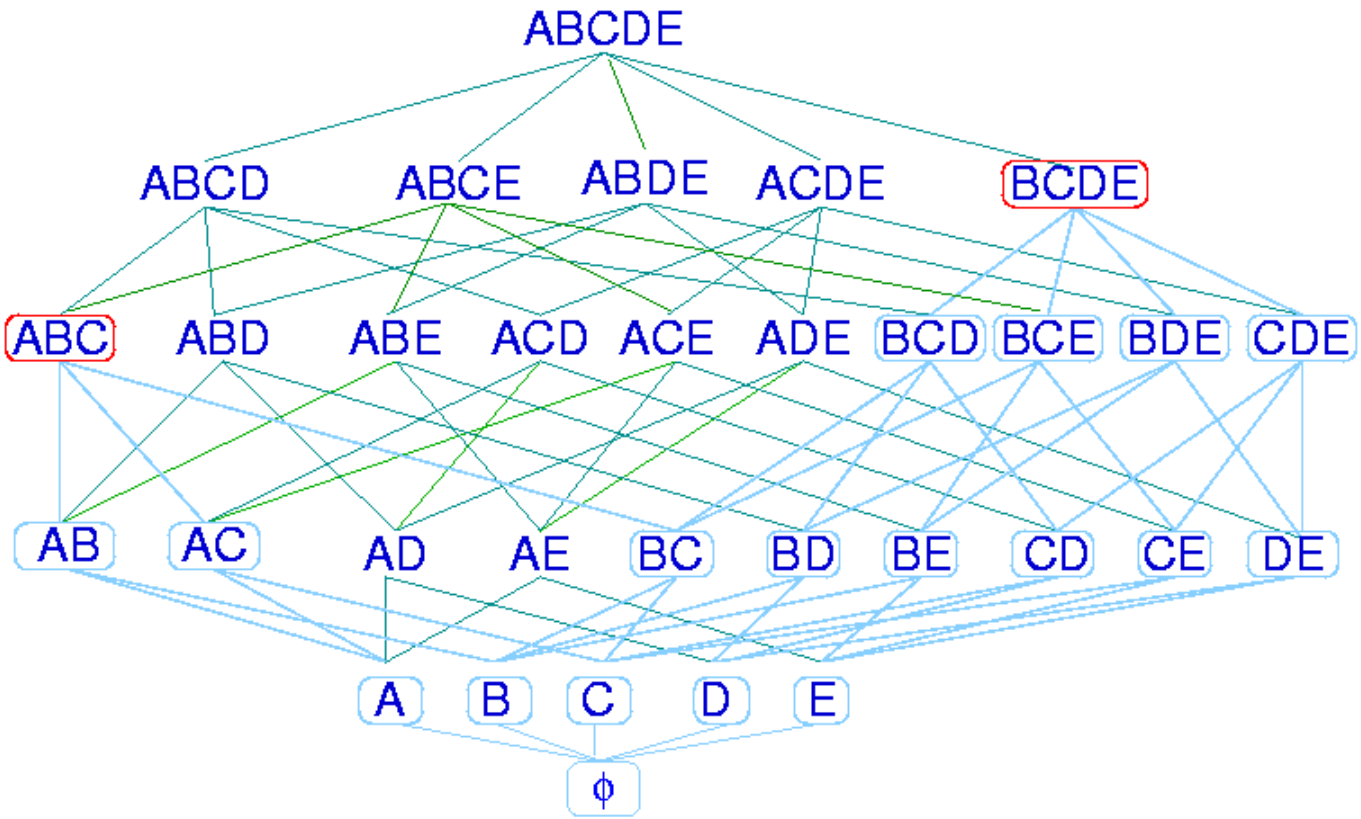
A maximal frequent itemset is a frequent itemset not contained in another frequent itemset.

All frequent itemsets are subsets of a maximal frequent itemset.

This algorithm is a depth-first algorithm.

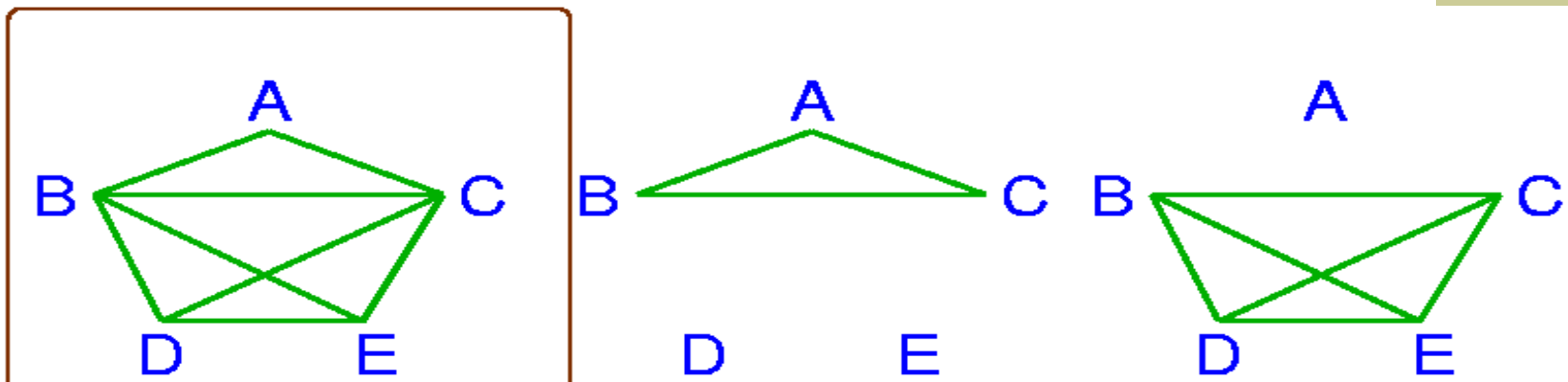
It uses the vertical format (*tidlists*)

Max clique /2



meet semilattice of frequent itemsets, with maximal frequent itemsets ABC and BCDE in red.

Max clique /3



Graph of frequent 2–itemsets :
two items are joined if the pair is frequent

Maximal cliques K_3 : **ABC**
K_4 : **BCDE**

Maximal cliques are candidate maximal frequent itemsets : in our case the 2 cliques are in effect maximal frequent itemsets

Max clique /4

Maximal clique generation algorithms are well known and the following can be used :

- Mulligan and Corneil,1972 (*modified Bierstone's) JACM*
- Bron & Kerbosch 1973,CACM
- Chiba & Nishizeki, 1985,SIAM JC
- Tsukiyama et al. 1977,SIAM JC

After having found the maximal cliques, their support is to be checked to verify that they are really frequent.

Closure operators/systems

If $(A, \leq)$ is a complete lattice and we have a function :

$cl: A \rightarrow A$

such that $(u,v \in A)$:

- if $u \leq v$ then $cl(u) \leq cl(v)$
- $u \leq cl(u)$
- $cl(cl(u)) = cl(u)$

we say that cl is a **closure operator** and cl(u) is said to be the **closure** of u.

If $u = cl(u)$, we say that u is **close.**

(Topological closure is a special case.)

For any Galois connection G=(f,g) between the complete lattices (P,≤) and (Q,≤), the mapping cl=f•g is a closure on P and cl'=g•f is a closure operator on Q.

The restriction of f to cl-closed elements is a bijection between cl-closed elements of P and cl'-closed elements of Q.

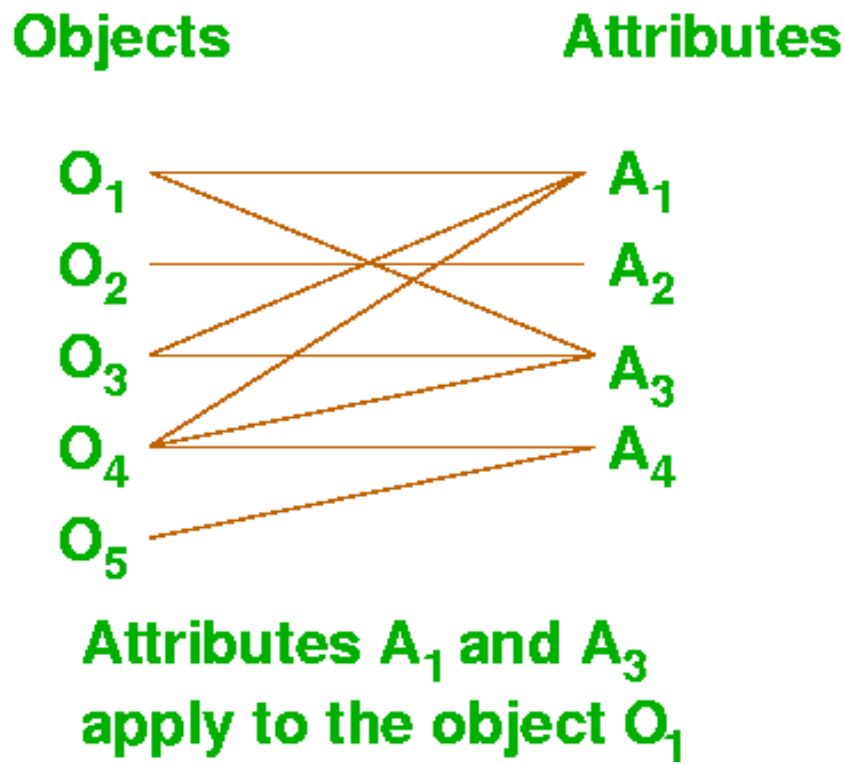
Formal Concept Analysis /1

- Introduced by Rudolf Wille around 1982, Darmstadt (Germany)
- It is an application of *lattice theory*
- Re-flourished in the last 6/7 years
- The name refers to the fact that the method applies mainly to the *analysis* of data, using a mathematical abstraction of *concept* (this is the reason behind the *formal* prefix)

Formal Concept Analysis /2

Given a set of **Objects O**, a set of **Attributes A**, and a binary relation **I ⊆ O x A**, we say that :

- **(O,A,I)** is a **formal context**



Formal Concept Analysis /3

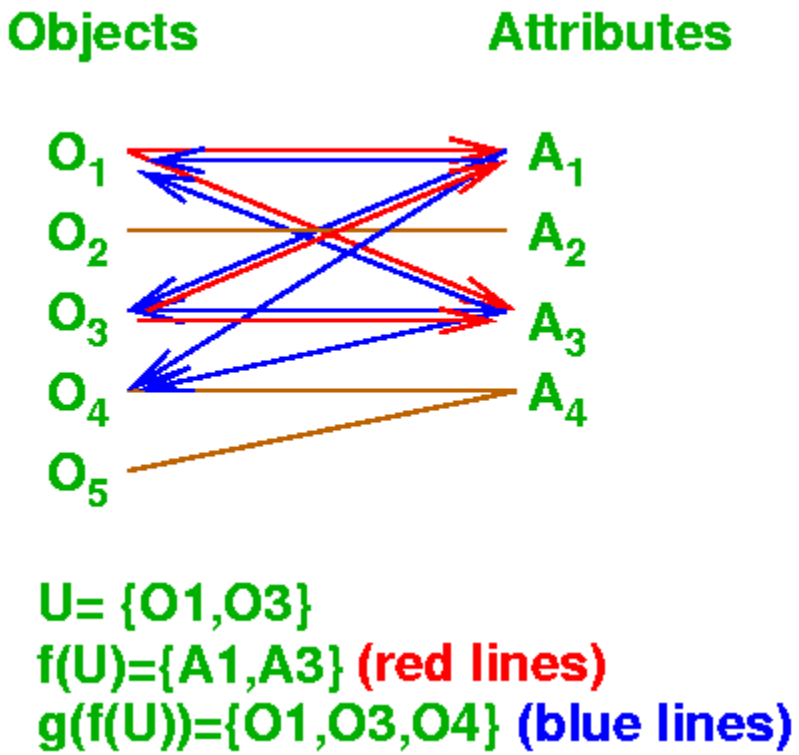
Through the binary relation $I \subseteq O \times A$ we can define two functions f and g such that for each subset U of O :

f(U) = {attributes that apply to all objects in U}

and conversely for each subset V of A :

g(V) = {objects for which all attributes in V apply}

The pair (f,g) is also called the **polarity** on O and A determined by I.



Formal Concept Analysis /4

It can be easily demonstrated that the polarity (f, g) of a relation is a **Galois connection** between the powerset of O and the powerset of A, ordered by inclusion.

Furthermore, called :

- h = g•f
- h' = f•g

we have that h is a **closure operator** on O and h’ is a **closure operator** on A.

The Galois connection establishes a duality between the two closure systems on O and A.

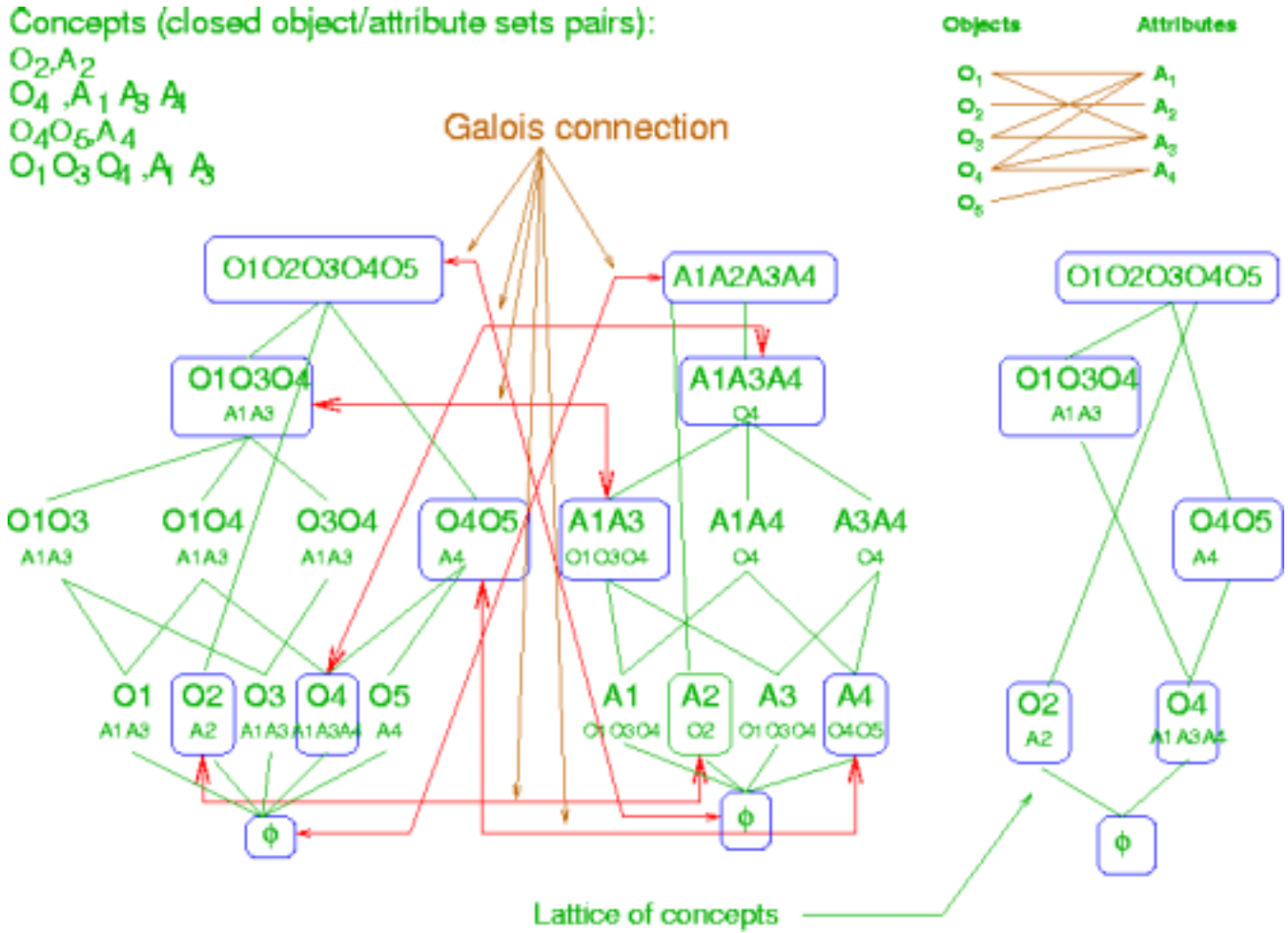
It is a bijective map between closed subsets of O and closed subsets of A.

Formal Concept Analysis /5

A **concept** is a pair (U,V) that comprise a closed set of objects U, together with a closed set of attributes connected by the Galois connection.

U is called the **extent** (or **extension**) of the concept and V is called the **intent** (or **intension**)of the concept.

Formal Concept Analysis /6



Formal Concept Analysis /7

Concepts form a lattice : the **lattice of concepts**.

If (U,V) is a concept, then you can't extend the **extension** U of the concept in such a way that all the attributes of V apply, and conversely, you can't extend the **intension** V such that it applies to all objects in U.

In the previous slide the sets of objects having an empty set of attributes, and the sets of attributes having an empty set of objects are not displayed. Their situation in the Hasse diagram is very simple: they are connected only to the empty set and to the complete set. Their closure is the complete set of objects/attributes.

A-Close /1

Proposed by Pasquier,Bastide, Taouil, Lakhal in 1999.

It is based on Formal Concept Analysis. Let us think to the set of items as the the set of objects and the set of transactions as the set of attributes. Then, an **itemset is closed** if there is no superset of it that appears in the same set of transactions.

Conversely, a set of transaction is closed if it's not contained in a superset of transactions containing the same items.

A-Close /2

What is important is that the support of an itemset is the same as that of its closure:

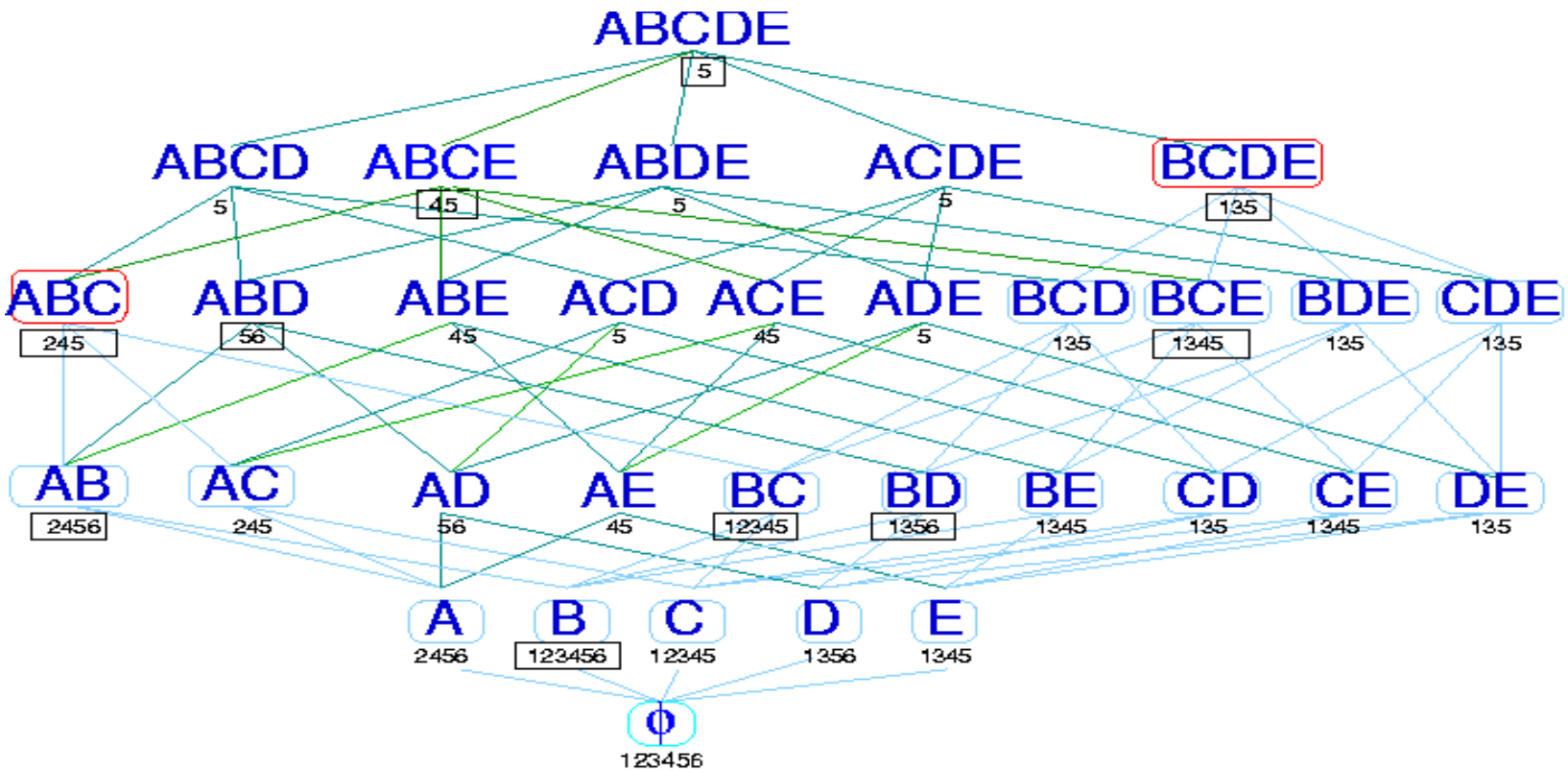
$$s(A) = s(cl(A))$$

Therefore, you need to *remember only the closed itemsets* and their support, to be able to count the support of any itemset.

Further, if you are interested only in frequent itemsets, then you need only to remember the support of frequent closed itemsets.

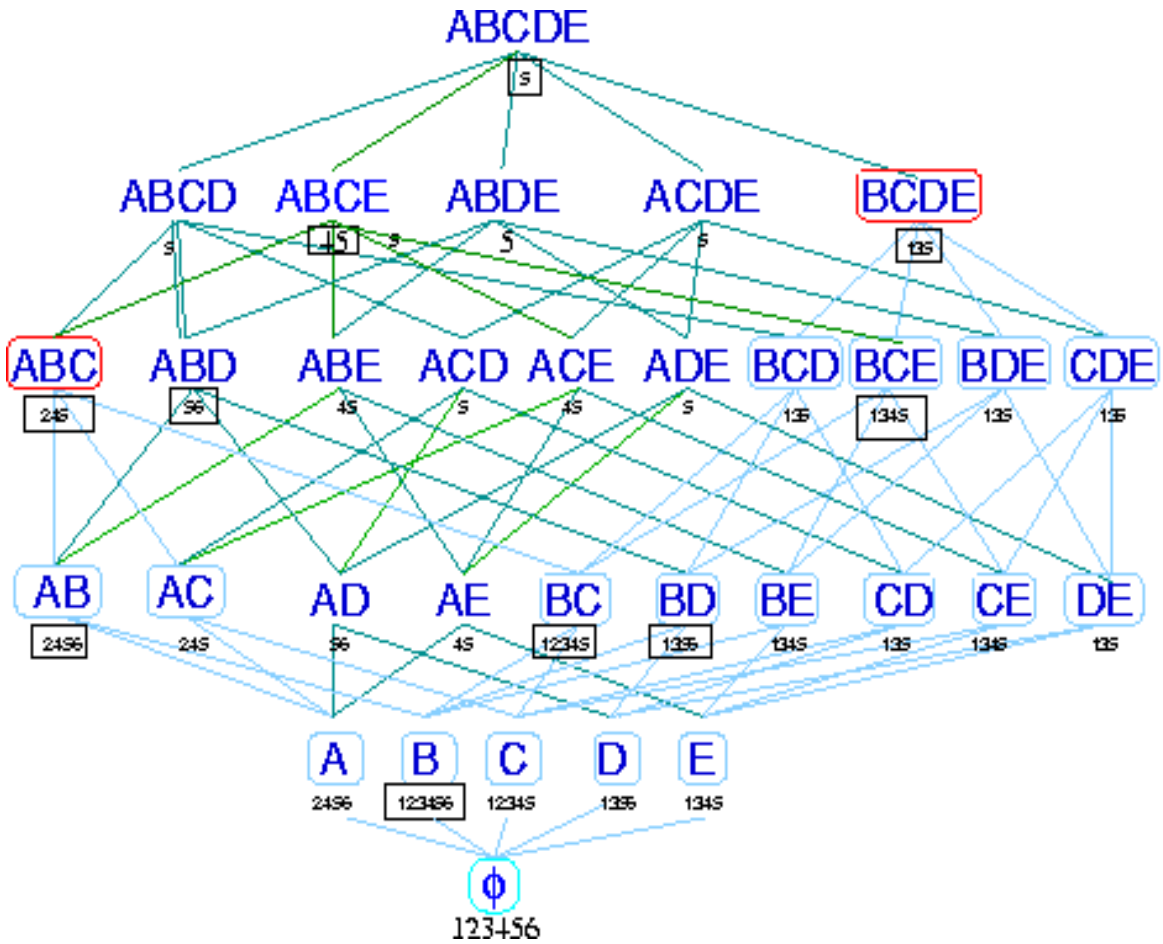
In fact maximal frequent itemsets are **closed**.

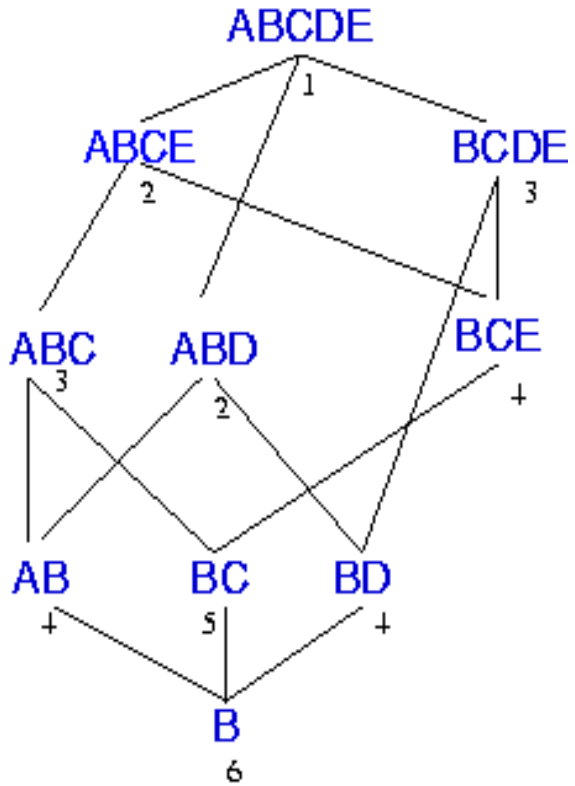
A-Close /3



Closed itemsets have their tidlist in a black box

A-Close /4





Lattice of closed itemsets and their support

A-Close /5

From the lattice of closed itemsets it is easy to count the support of every subset.

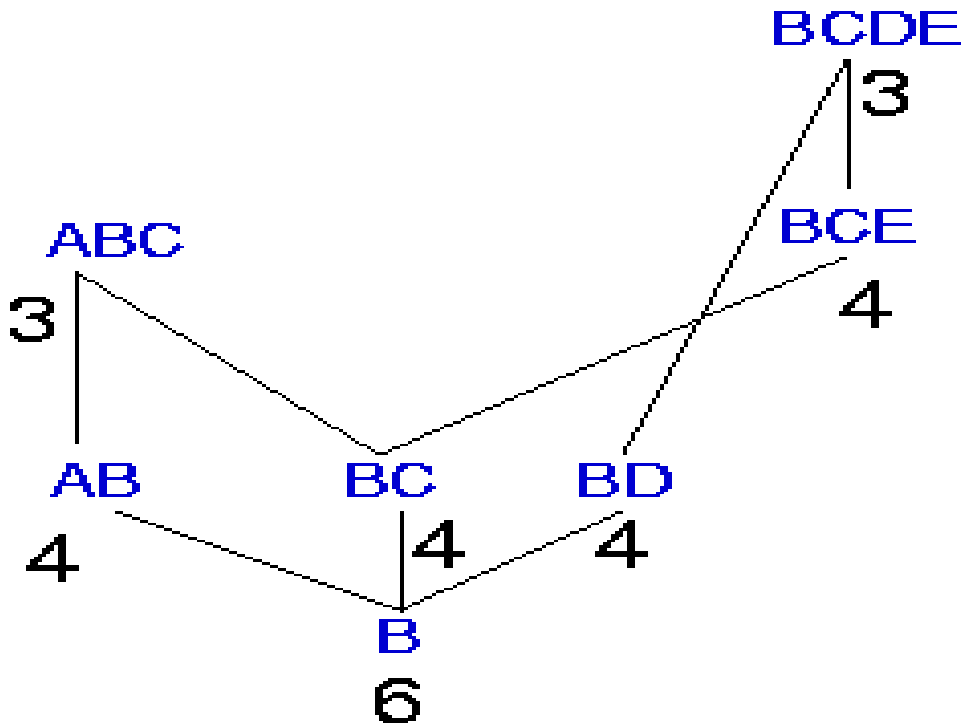
For example

s(A)=s(cl(A))=s(AB)=4

and

s(BDE)=s(cl(BDE))=s(BCDE)=3

A-Close /6



Closed frequent itemsets meet semilattice

CHARM

An efficient algorithm for mining closed frequent sets, was proposed by Zaki and Hsiao in 2001.

- It enumerates closed itemsets using a dual itemset/tidset search tree, and a hybrid search algorithm
- It uses *diffsets* to reduce memory requirements

Bibliography

- Agrawal R.,Imielinski T.,Swami A. : *Association rules between Sets of Items in large Databases*, SIGMOD 1993
- Agrawal R., Srikant R. : *Fast Algorithms for mining association rules*, VLDB 1994
- Savasere A., Omiecinski E., Navathe S.: *An efficient algorithm for mining association rules in large databases*
- Zaki M., Parthasarathy S., Ogihara M. : *New algorithms for fast discovery of association rules*, KDDM 1998
- Han J., Pei J., Yin Y. : *Mining frequent patterns without candidate generation*, SIGMOD 2000
- Ganter B., Wille R.: *Formal Concept Analysis:Mathematical Foundations*,Springer 1999
- Pasquier N., Bastide Y., Taouil R., Lakhal L.: *Discovering frequent closed itemsets for association rules*, ICDT 1999
- Zaki M., Hsiao C.: *CHARM : An Efficient Algorithm for closed Itemset Mining*, **2002**