CompSci 275, CONSTRAINT Networks

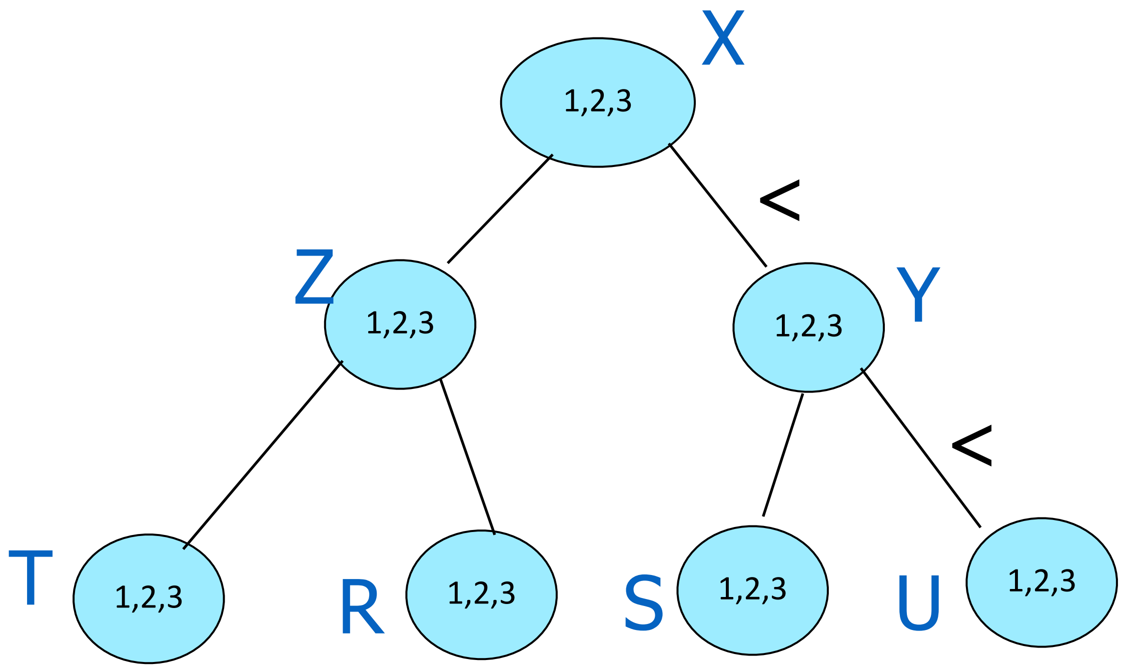
Rina Dechter, Fall 2022

# Tree-decomposition methods
# Chapter 9

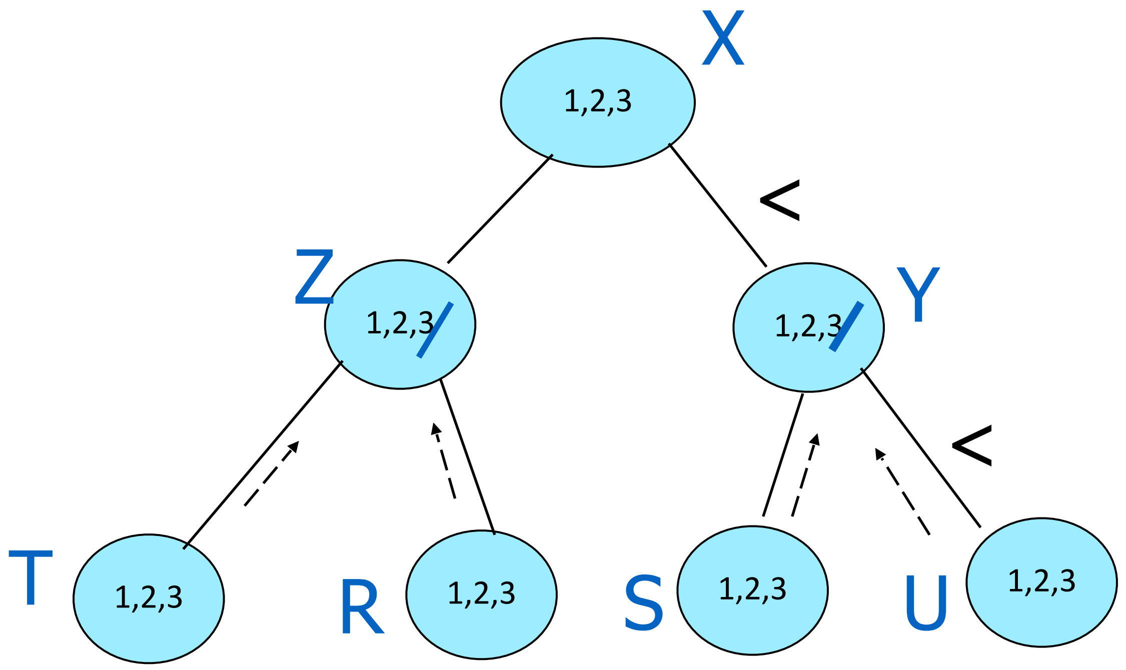
# Outline

- Acyclic networks
- Join-tree clustering
- Conditioning vs tree-clustering

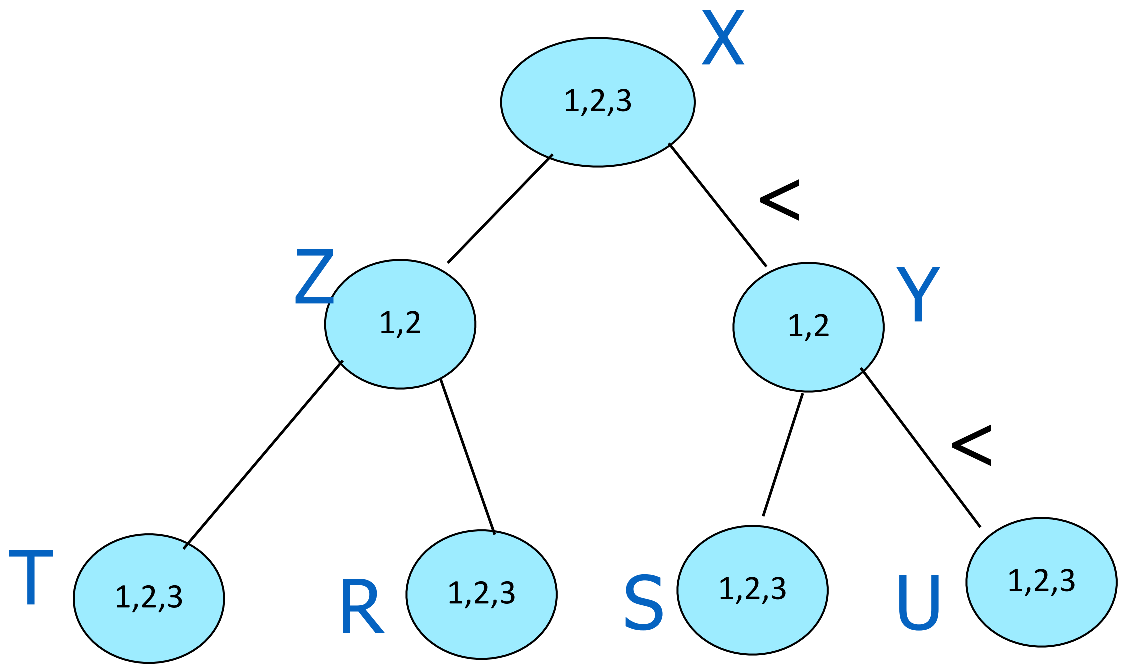
# Tree Solving is Easy



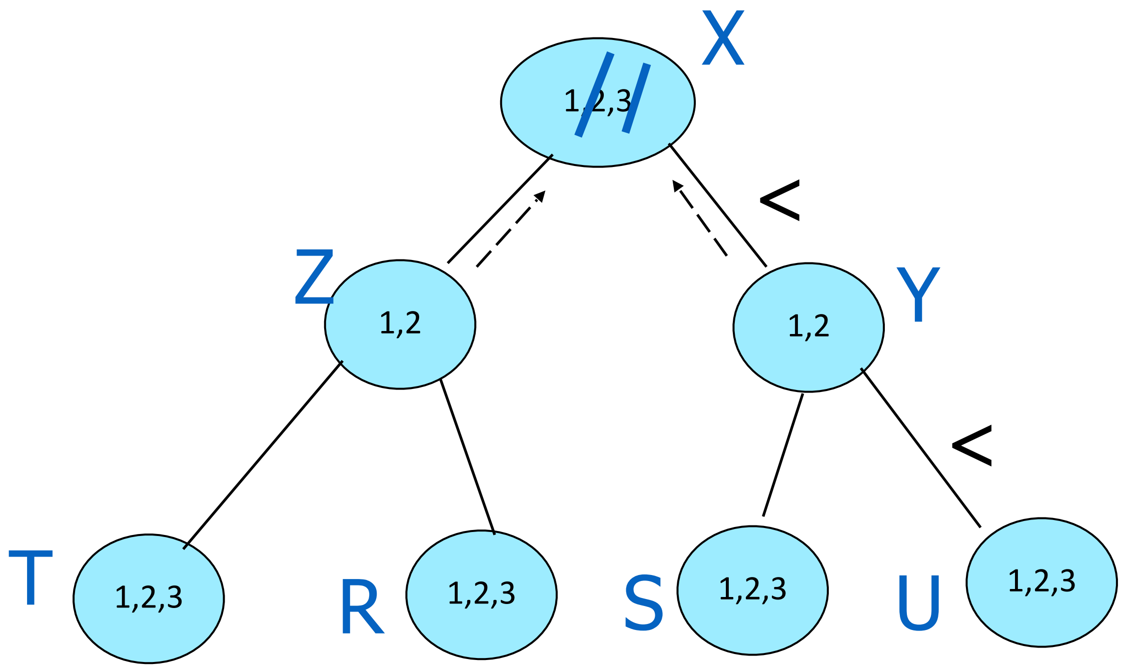
# Tree Solving is Easy



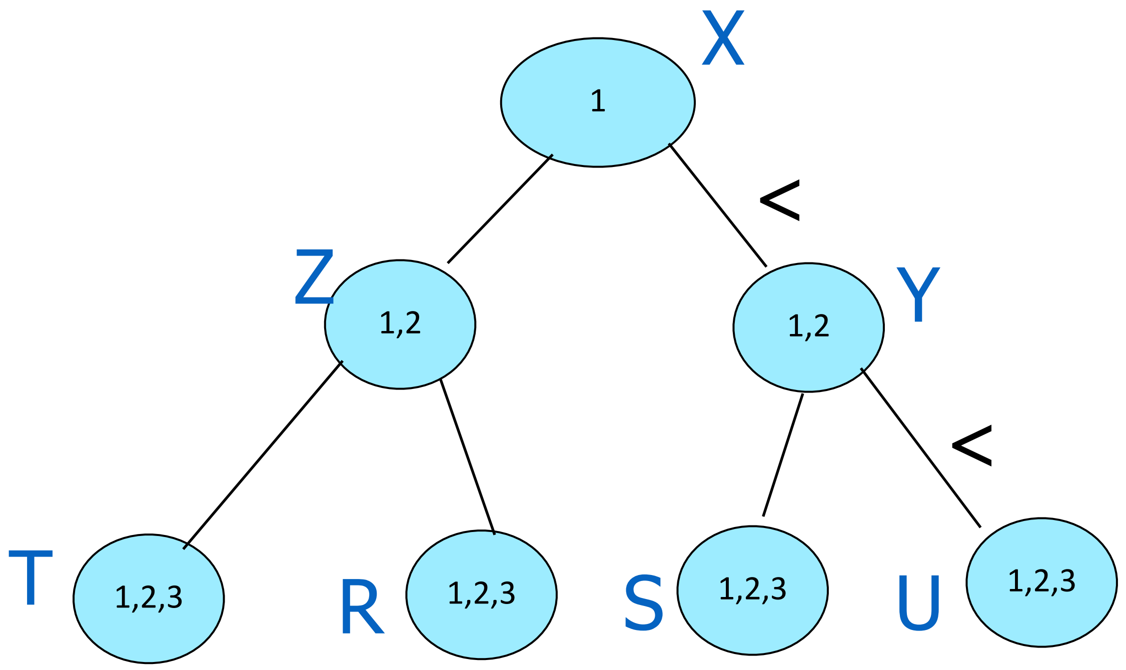
# Tree Solving is Easy



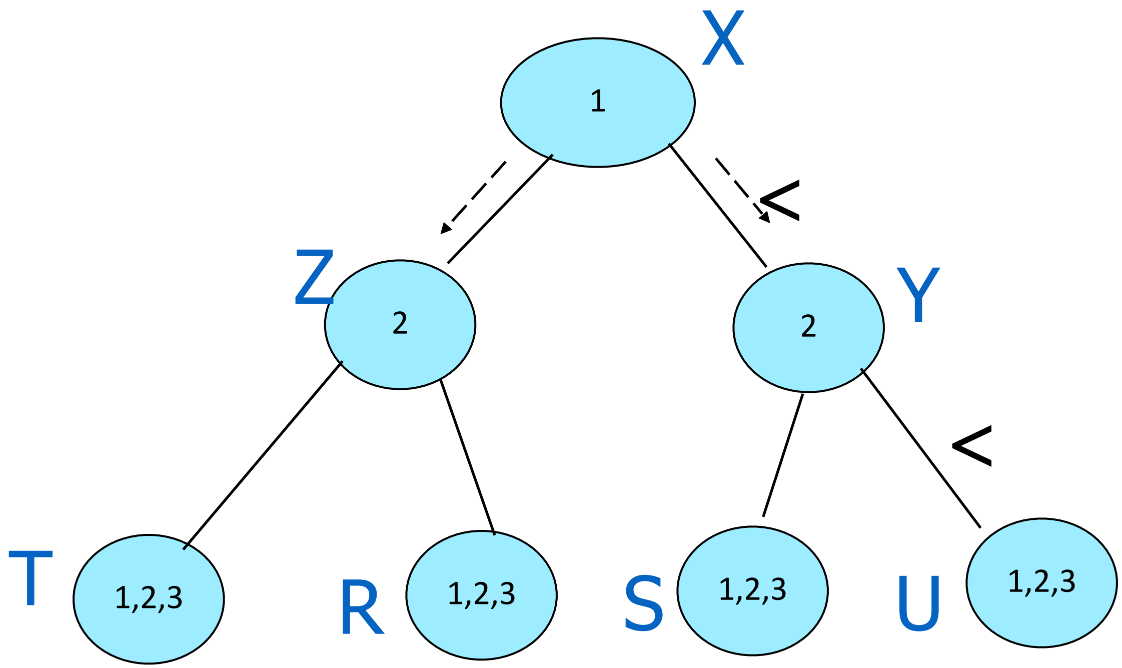
# Tree Solving is Easy



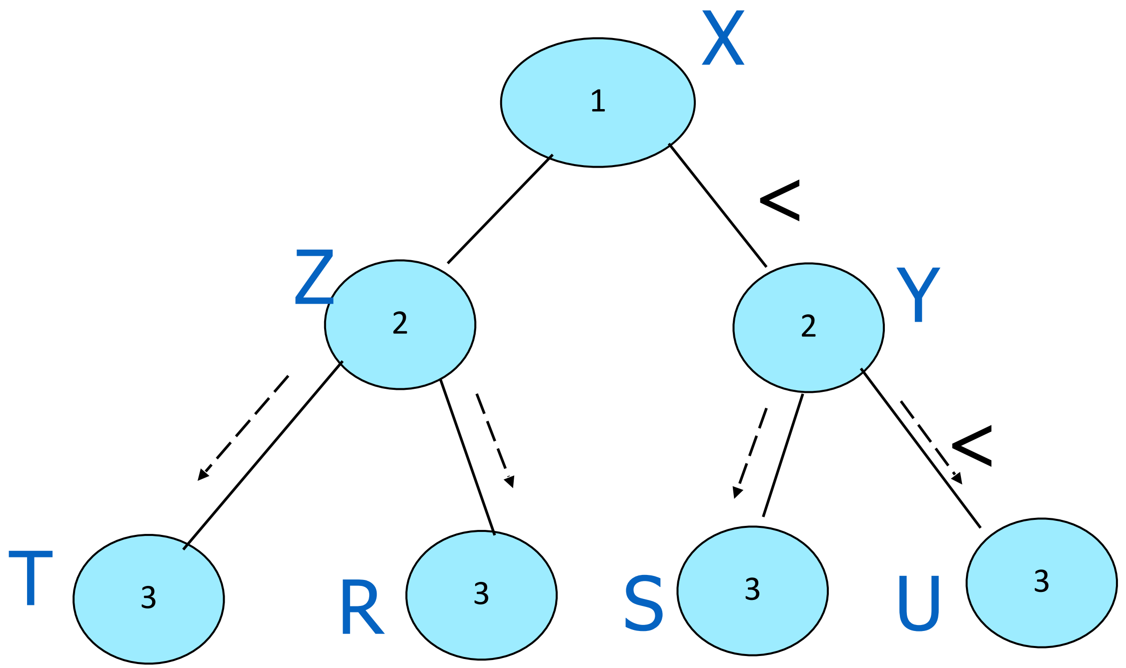
# Tree Solving is Easy



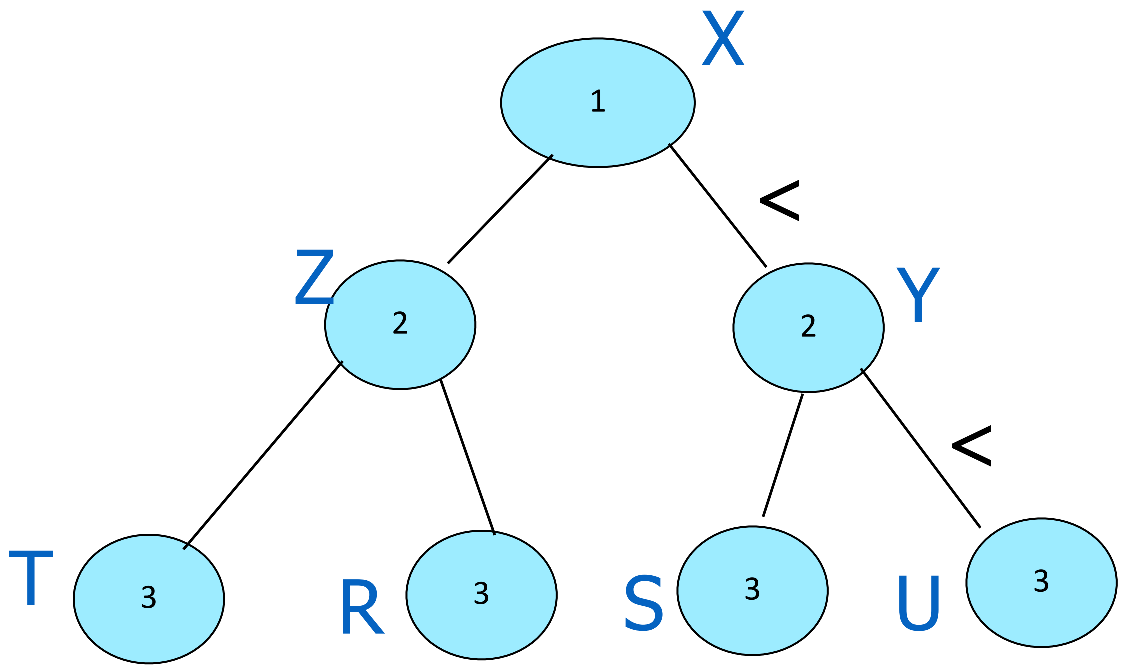
# Tree Solving is Easy



# Tree Solving is Easy



# Tree Solving is Easy

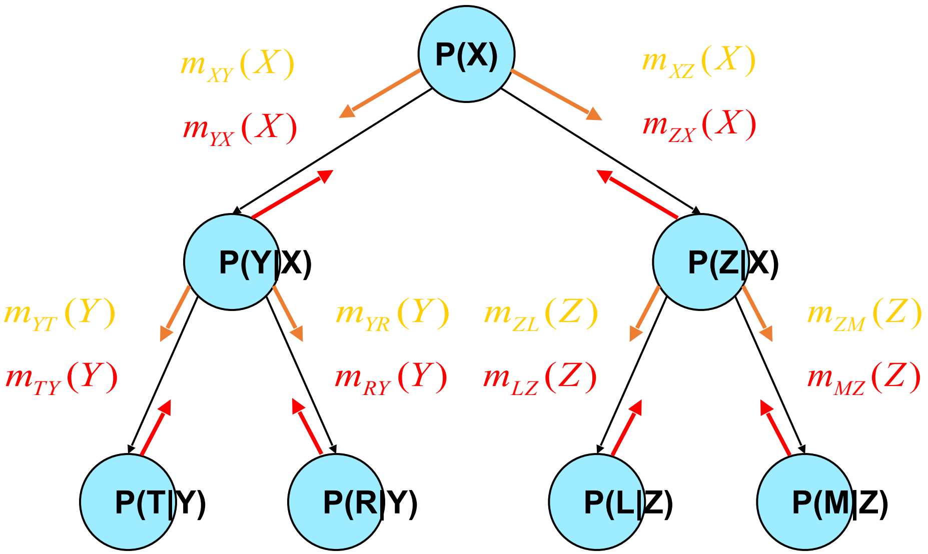



Constraint propagation
Solves trees in linear time

# Tree-solving

CSP – consistency (projection-join)

**Belief updating (sum-prod)**

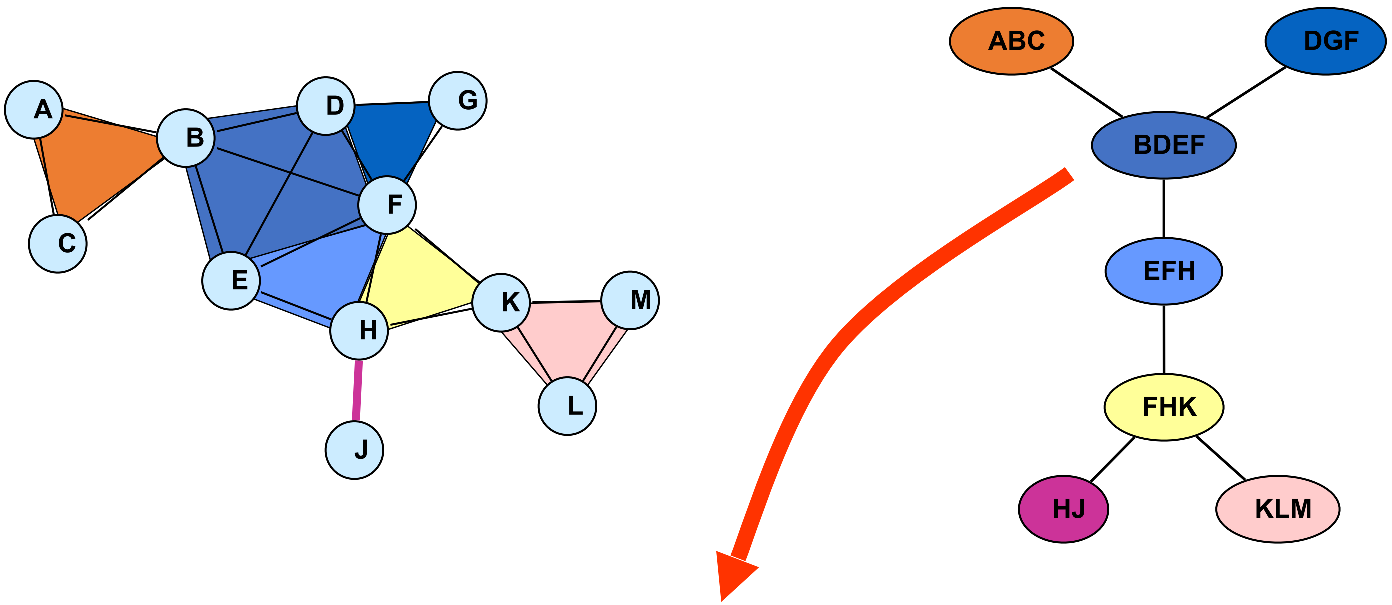



**MPE (max-prod)**

**#CSP (sum-prod)**

Trees are processed in linear time and memory

# Inference and Treewidth




*treewidth* = 4 - 1 = 3
treewidth = (maximum cluster size) - 1

# The dual problem/the acyclic problem

The dual graph of a constraint problem associates a node with each constraint scope and an arc for each two nodes sharing variables. This transforms a
non-binary constraint problem into a binary one, called the *dual problem:*

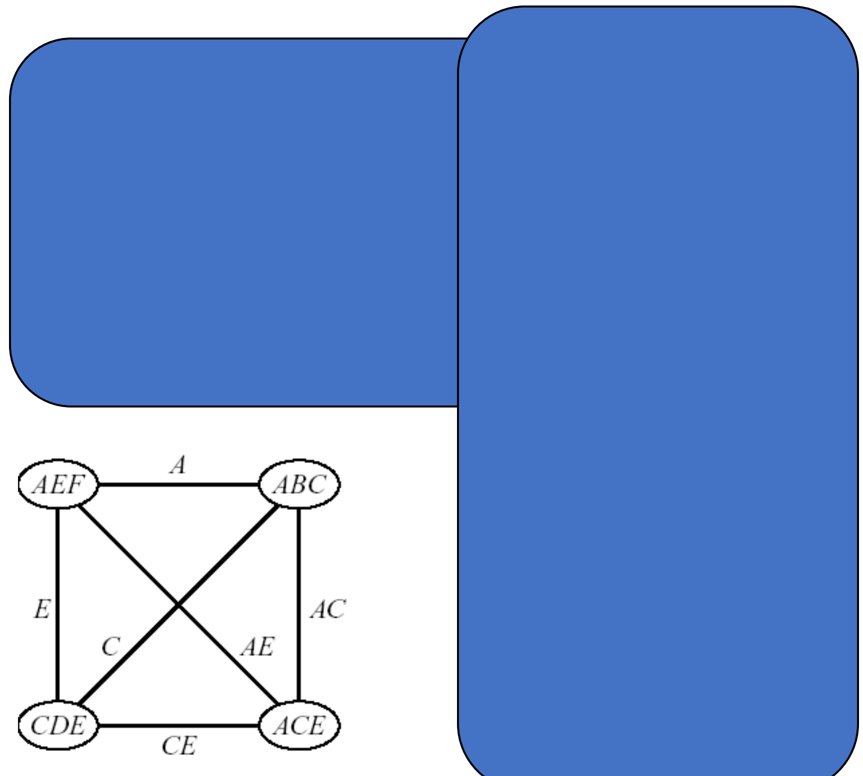
Variables: constraints,
Domains: legal tuples of the relation
Binary constraints between any two dual variables that share original variables, enforcing equality on the values assigned to the shared variables.

Therefore, if a problem's dual graph happens to be a tree, it can be solved by the tree-solving algorithm.

It turns out, however, that sometimes, even when the dual graph does not look like a

# Dual constraint problems

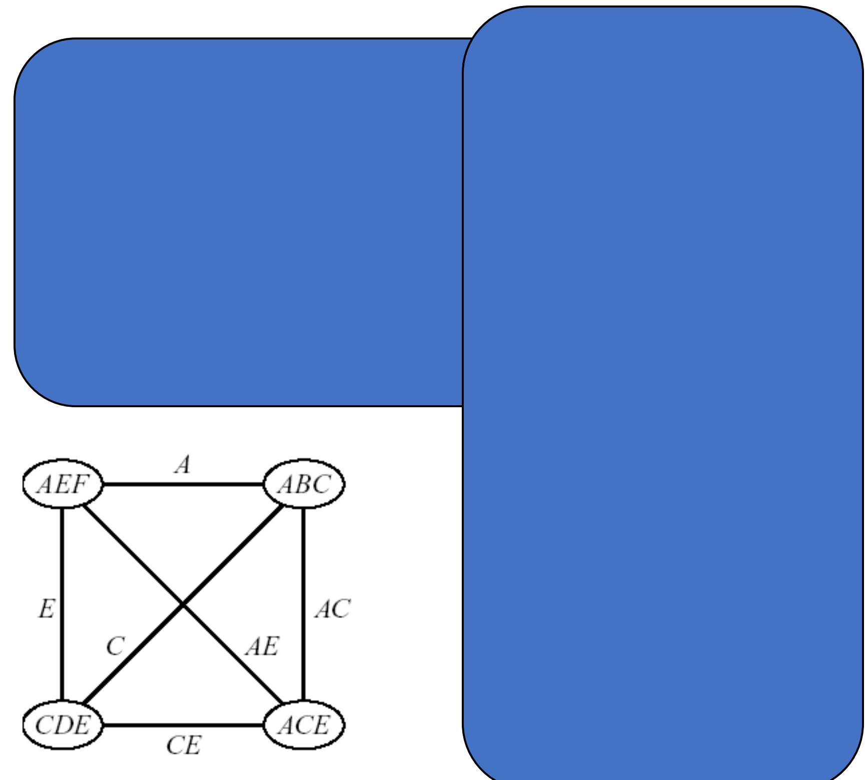
- Constraints can be: C= AVE
- F=AVE and so on…




(c) (d)

# Dual constraint problems

- Constraints can be: C= AVE
- F=AVE and so on…




(c) (d)

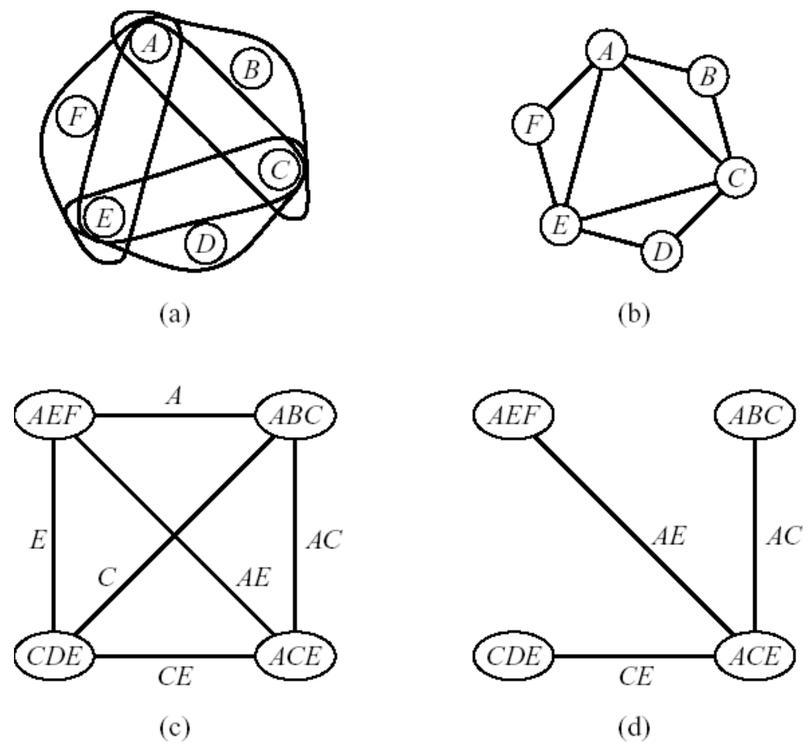
# Graph concepts reviews: Hyper graphs and dual graphs

- **A hypergraph** is H = (V,S) , V= {v_1,..,v_n} and a set of subsets **Hyperegdes**: S={S_1, ..., S_l }.

- **Dual graphs** of a hypergaph: The nodes are the hyperedges and a pair of nodes is connected if they share vertices in V. The arc is labeled by the shared vertices.

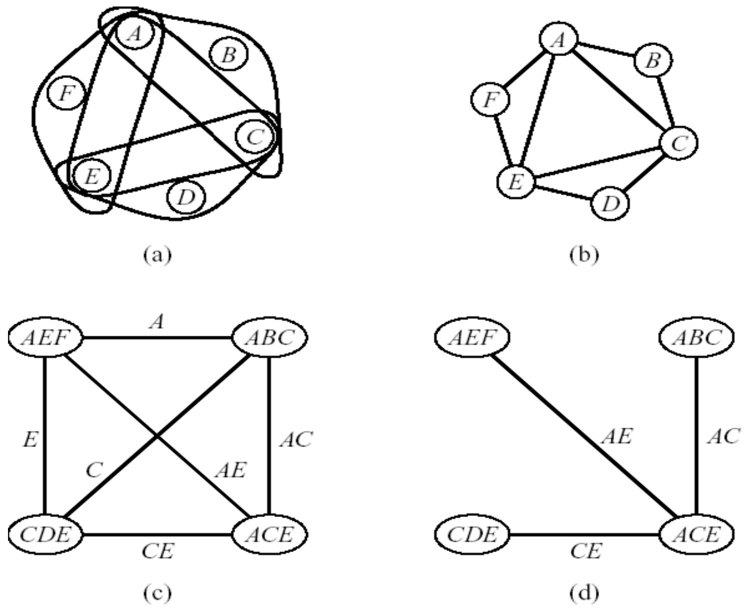
- **A primal graph** of a hypergraph H = (V,S) has V as its nodes, and any two nodes are connected by an arc if they appear in the same hyperedge.

- if all the constraints of a network R are binary, then its hypergraph is identical to its primal graph.

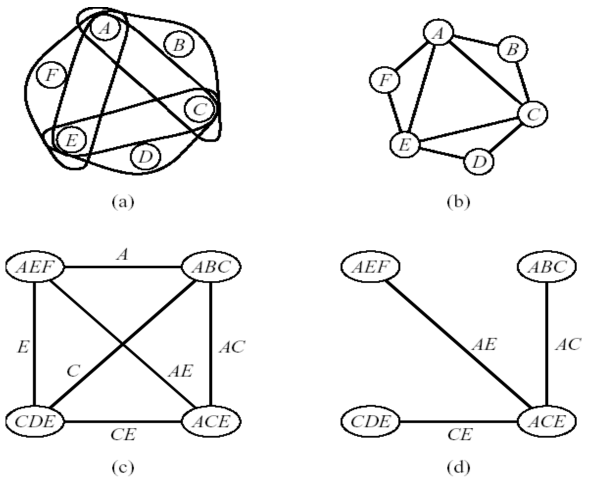


# Acyclic networks

- **The running intersection property (connectedness):** An arc can be removed from the dual graph if the variables labeling the arcs are shared along an alternative path between the two endpoints.
- **Join graph**: An arc subgraph of the dual graph that satisfies the connectedness property.
- **Join-tree:** a join-graph with no cycles
- **Hypertree:** A hypergraph whose dual graph has a join-tree.
- **Acyclic network:** is one whose hypergraph is a hypertree.



# Example




- Constraints are:
- $R_{ABC} = R_{AEF} = R_{CDE}$ = {(0,0,1) (0,1,0)(1,0,0)}
- $R_{ACE}$ = { (1,1,0) (0,1,1) (1,0,1) }.

- d= ($R_{ACE}$, $R_{CDE}$, $R_{AEF}$, $R_{ABC}$).
  - When processing $R_{ABC}$ its parent relation is $R_{ACE}$

$$R_{ACE} = \pi_{ACE}(R_{ACE} \otimes R_{ABC}) = \{(0,1,1)(1,0,1)\}$$

  - processing $R_{AEF}$ we generate relation

$$R_{ACE} = \pi_{ACE}(R_{ACE} \otimes R_{AEF}) = \{(0,1,1)\}$$

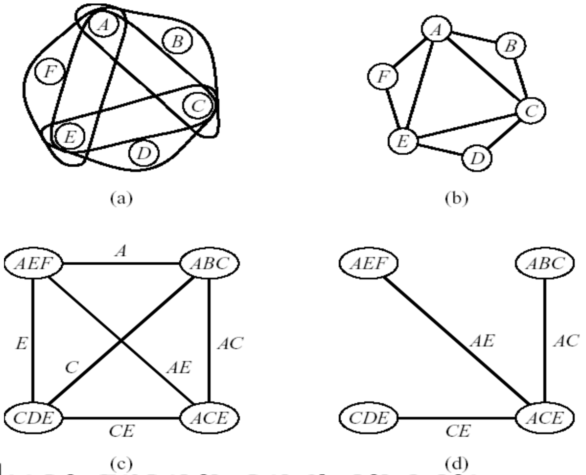
    - processing $R_{CDE}$ we generate:
    - $R_{ACE}$ = \pi_{ACE} ($R_{ACE}$ x $R_{CDE}$) = {(0,1,1)}.
- A solution is generated by picking the only allowed tuple for $R_{ACE}$, A=0,C=1,E=1, extending it with a value for D that satisfies $R_{CDE}$, which is only D=0, and then similarly extending the assignment to F=0 and B=0, to satisfy $R_{AEF}$ and $R_{ABC}$

# Solving acyclic networks

- Algorithm ***acyclic-solving*** applies a tree algorithm to the join-tree. It applies (a little more than) directional relational arc-consistency from leaves to root.

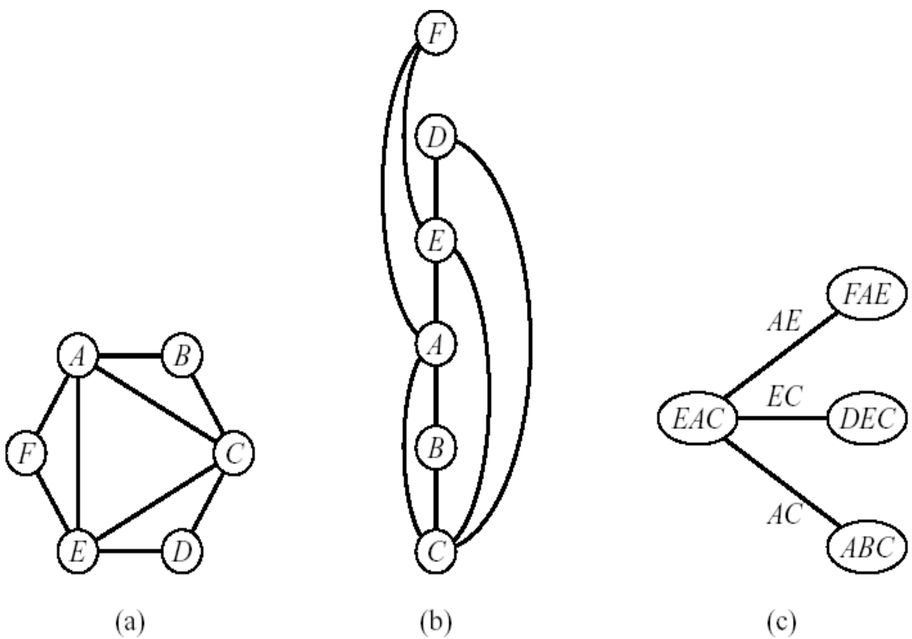
- **Complexity**: acyclic-solving is O(r l  log l) steps, where r is the number of constraints and l bounds the number of tuples in each constraint relation
- (It assumes that join of two relations when one's scope is a subset of the other can be done in linear time)

# Recognizing acyclic networks




- **Dual-based recognition**:
  - perform maximal spanning tree over the dual graph and check connectedness of the resulting tree.
  - Dual-acyclicity complexity is O($e^3$), e is the number of constraints.
- **Primal-based recognition**:
  - **Theorem** (Maier 83): A hypergraph has a join-tree iff its primal graph is chordal and conformal.
  - A chordal primal graph is **conformal** relative to a constraint hypergraph iff there is a one-to-one mapping between maximal cliques and scopes of constraints.

# Primal-based recognition




- Check cordality using max-cardinality ordering.
- Test conformality
- Create a join-tree: connect every clique to an earlier clique sharing maximal number of variables

# Outline

- Acyclic networks
- Join-tree clustering
- Conditioning vs tree-clustering

# Tree-based clustering

- Convert a constraint problem to an acyclic-one: group subsets of constraints to clusters until we get an acyclic problem.
- **Tree-decomposition**: Hypertree embedding of a hypergraph $H$ = (X,H) is a hypertree $S$ = (X, S) s.t., for every $\mathrm{h} \in H$ there is $h_1$ in S and h is included in $h_1$ .

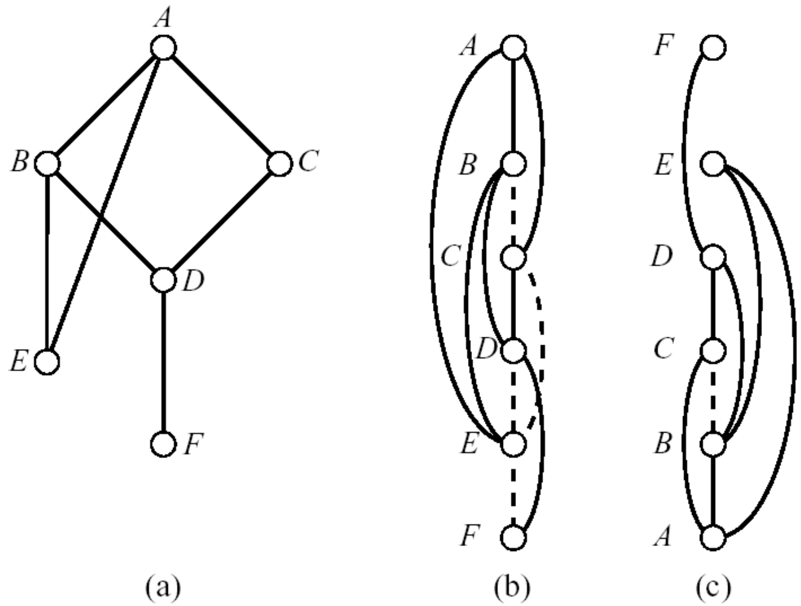
Join-tree clustering (JTC)

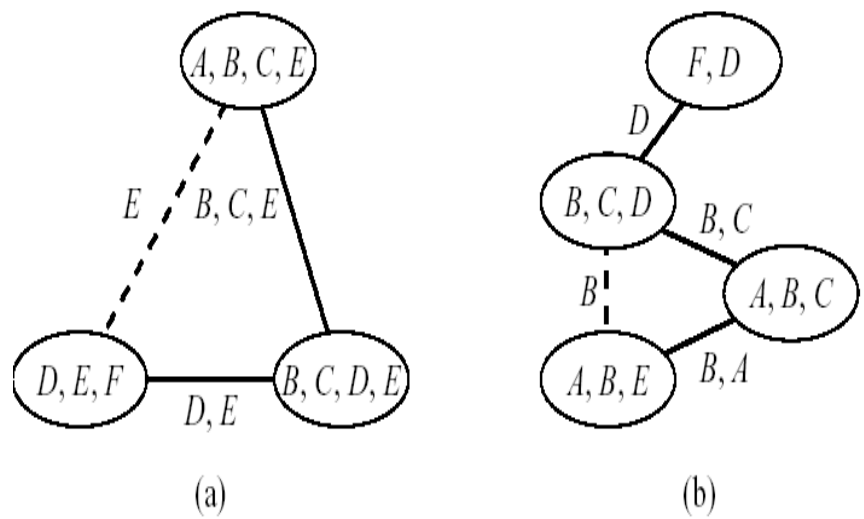
**Input:** A constraint problem $\mathcal{R} = (X, D, C)$ and its primal graph $G = (X, E)$.
**Output:** An equivalent acyclic constraint problem and its join-tree: $\mathcal{T} = (X, D, C\prime)$

1. Select a variable ordering, $d = (x_1, ..., x_n)$.
2. **Triangulation** (create the induced graph along $d$ and call it $G^*$):
   **for** $j = n$ to 1 by -1 do
   $E \leftarrow E \cup \{(i,k) | \ (i,j) \in E, (k,j) \in E\}$
3. **Create a join-tree of the induced graph** $G^*$:
   a. Identify all maximal cliques in the chordal graph (each variable and its parents is a clique). Let $C_1, ..., C_t$ be all such cliques, created going from last variable to first in the ordering.
   b. Create a tree-structure $T$ over the cliques: Connect each $C_i$ to a $C_j$ ($j < i$) with whom it shares largest subset of variables.
4. Place each input constraint in one clique containing its scope, and let $P_i$ be the constraint subproblem associated with $C_i$.
5. Solve $P_i$ and let $R'_i$ be its set of solutions.
6. Return $C' = \{R'_1, ..., R'_t\}$, the new set of constraints and their join-tree, $T$.

Figure 9.6: Join-tree clustering

# Example of tree-clustering



# Complexity of JTC

- **complexity of** JTC**:** *join-tree clustering is*
- *O*(*r* $k^{w*(d)+1}$ ) time and space, *where k is the maximum domain size and w*(d) is the induced width of the ordered graph.*
- *The complexity of acyclic-solving is O(n w*(d) (log k)* $k^{w*(d)+1}$)

# Unifying tree-decompositions

Let $R=<X,D,C>$ be a CSP problem. A tree decomposition for R is $<T,\chi,\psi>$, such that

- $T=(V,E)$ is a tree
- $\chi$ associates a set of variables $\chi(v)\subseteq X$ with each node v
- $\psi$ associates a set of functions $\psi(v)\subseteq C$ with each node v

such that

- 1. $\forall R_i \in C$, there is exactly one v such that $R_i \in \psi(v)$ and $scope(R_i) \subseteq \chi(v)$.
- 2. $\forall x \in X$, the set $\{v \in V | x \subseteq \chi(v)\}$ induces a connected subtree.

# HyperTree Decomposition

Let R=<X,D,C> be CSP problem. A tree decomposition is $<T,\chi,\psi>$, such that

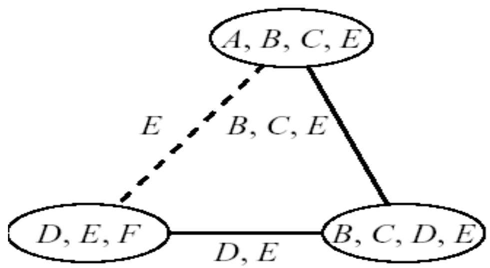
- T=(V,E) is a tree
- $\chi$ associates a set of variables $\chi(v) \subseteq X$ with each node
- $\psi$ associates a set of functions $\psi(v) \subseteq C$ with each node

such that

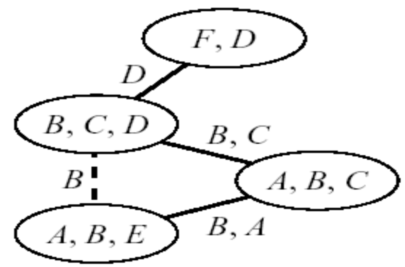
- 1. $\forall R_i \in C$, there is exactly one v such that $R_i \in \psi(v)$ and $scope(R_i) \subseteq \chi(v)$.
- 1a. $\forall v$, $\chi(v) \subseteq scope(\psi(v))$.
- 2. $\forall x \in X$, the set $\{v \in V | x \subseteq \chi(v)\}$ induces a connected subtree.

**w (tree-width) =** $\max_{v \in V} |\chi(v)|$
**hw (hypertree width) =** **max**$v \in V$ $|\psi(v)|$
**sep (max separator size) =** $\max_{(u,v)} |sep(u,v)|$

# Example of two join-trees again

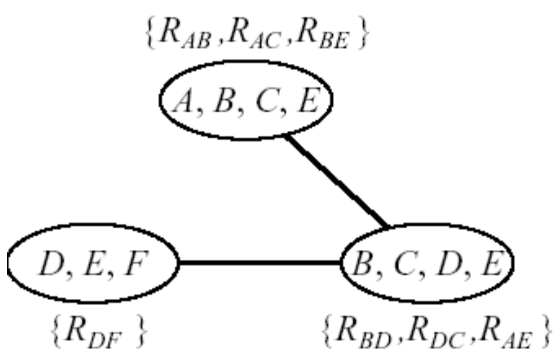



(a)

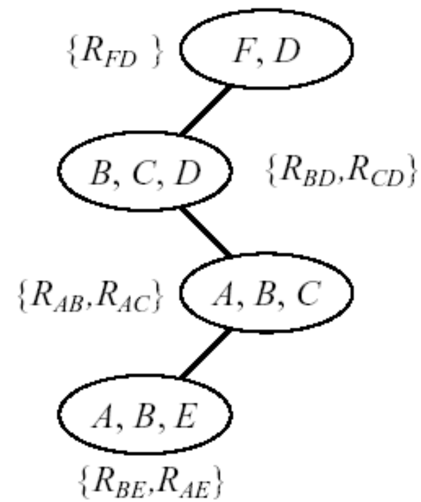



(b)

Tree decomposition




(a)




(b)

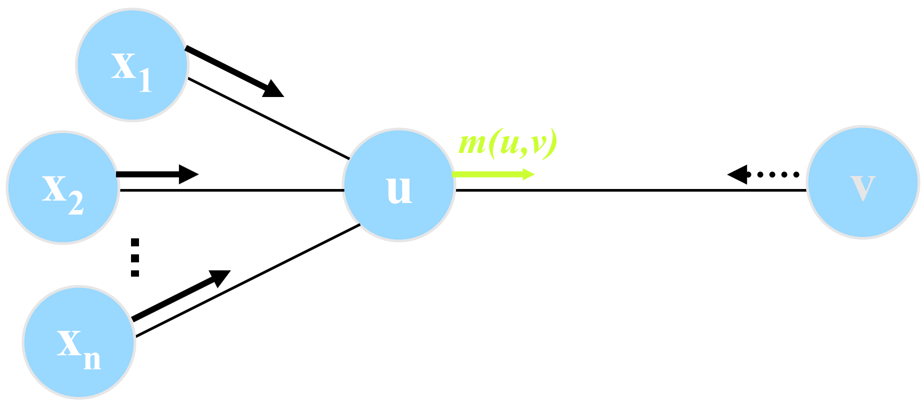
hyperTree-decomposition

# Cluster Tree Elimination

- Cluster Tree Elimination (CTE) works by passing messages along a tree-decomposition

- Basic idea:
  - Each node sends one message to each of its neighbors
  - Node *u* sends a message to its neighbor *v* only when *u* received messages from all its other neighbors

# Constraint propagation



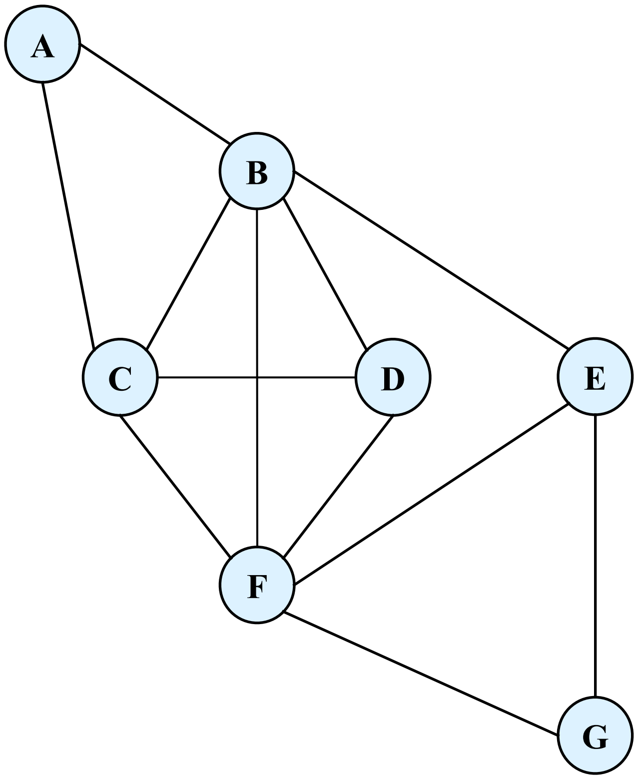

$$cluster(u) = \psi(u) \cup \{m(x_1,u), m(x_2,u), \ldots, m(x_n,u), m(v,u)\}$$

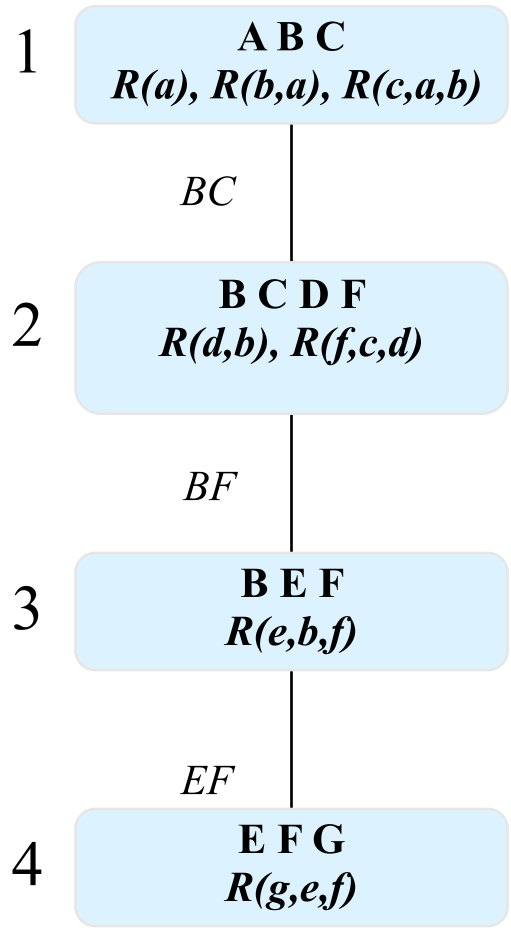
Compute the message :

$$m_{(u,v)} = \pi_{sep(u,v)}(\otimes_{R_i \in cluster(u)} R_i)$$

# Join-Tree Decomposition

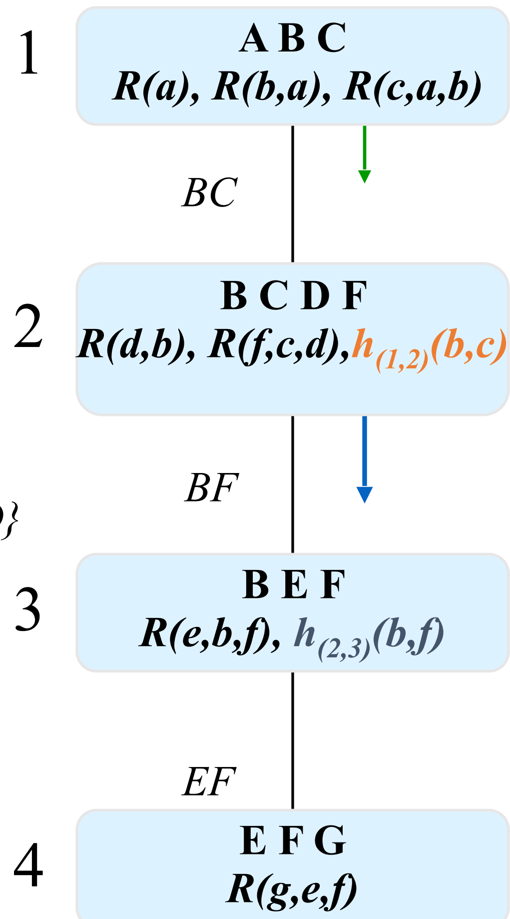
(Dechter and Pearl 1989)




- Each function in a cluster
- Satisfy running intersection property
- Tree-width: number of variables in a cluster-1
- Equals induced-width

# Cluster Tree Elimination




*sep(2,3)={B,F}*

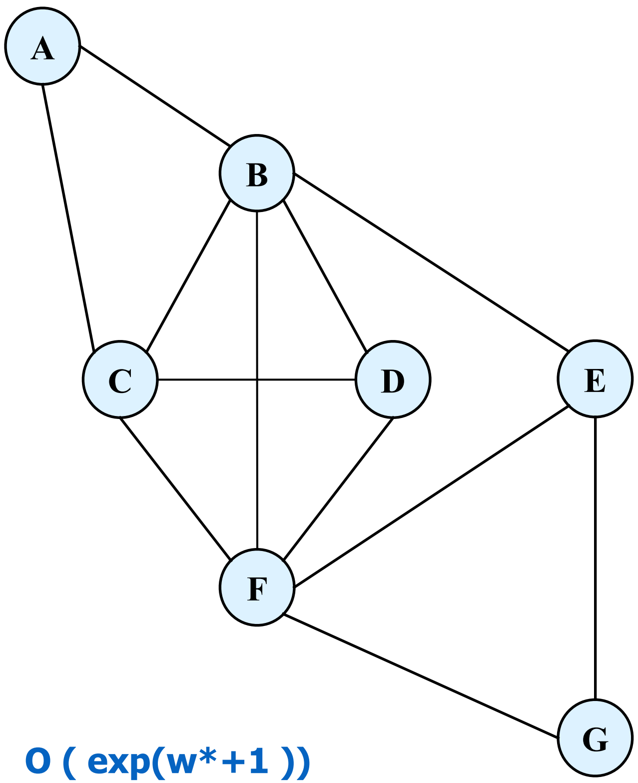
*elim(2,3)={C,D}*

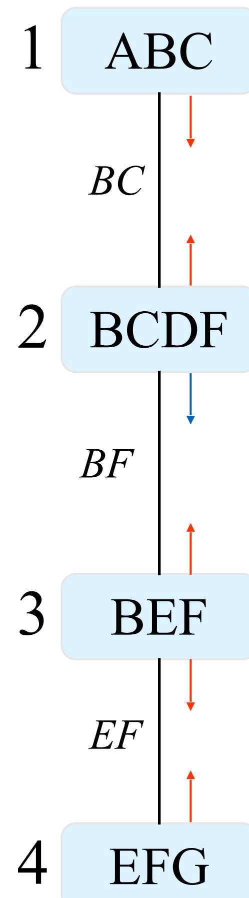
project join

$$h_{(1,2)}(b,c) = \Downarrow_a R(a) \otimes R(b,a) \otimes R(c,a,b)$$

$$h_{(2,3)}(b,f) = \Downarrow_{c,d} R(d,b) \otimes R(f,c,d) \otimes h_{(1,2)}(b,c)$$

# CTE: Cluster Tree Elimination




$$h_{(1,2)}(b,c) = \Downarrow_a R(a) \otimes R(b,a) \otimes R(c,a,b)$$

$$h_{(2,1)}(b,c) = \Downarrow_{d,f} R(d,b) \otimes R(f,c,d) \otimes h_{(3,2)}(b,f)$$

$$h_{(2,3)}(b,f) = \Downarrow_{c,d} R(d,b) \otimes R(f,c,d) \otimes h_{(1,2)}(b,c)$$

$$h_{(3,2)}(b,f) = \Downarrow_e R(e,b,f) \otimes h_{(4,3)}(e,f)$$

$$h_{(3,4)}(e,f) = \Downarrow_b R(e,b,f) \otimes h_{(2,3)}(b,f)$$

$$h_{(4,3)}(e,f) = \Downarrow_g R(g,e,f)$$

**Time:** O ( exp(w*+1 ))
**Space:** O ( exp(sep))
**Time:** O(exp(hw)) (Gottlob et. Al., 2000)

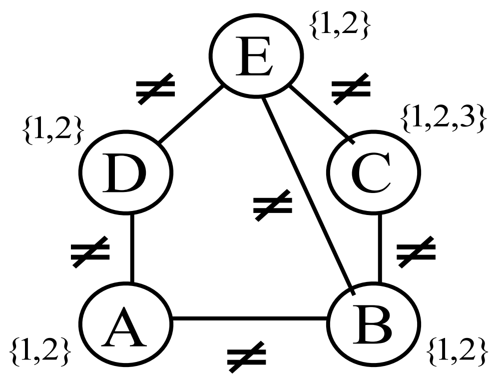
# Adaptive-consistency as tree-decomposition

- Adaptive consistency is a message-passing along a bucket-tree

- **Bucket trees**: each bucket is a node and it is connected to a bucket to which its message is sent.
  - The variables are the clicue of the triangulated graph
  - The functions are those placed in the initial partition

# Bucket Elimination

## Adaptive Consistency (Dechter and Pearl, 1987)



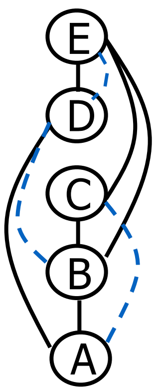

$Bucket(E): E \neq D,\ E \neq C,\ E \neq B$

$Bucket(D): D \neq A$ // $R_{DCB}$

$Bucket(C): C \neq B$ // $R_{ACB}$

$Bucket(B): B \neq A$ // $R_{AB}$

$Bucket(A):$ $R_A$



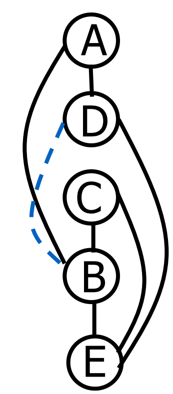

$Bucket(A): A \neq D,\ A \neq B$

$Bucket(D): D \neq E$ // $R_{DB}$

$Bucket(C): C \neq B,\ C \neq E$

$Bucket(B): B \neq E$ // $R^D_{BE}$, $R^C_{BE}$

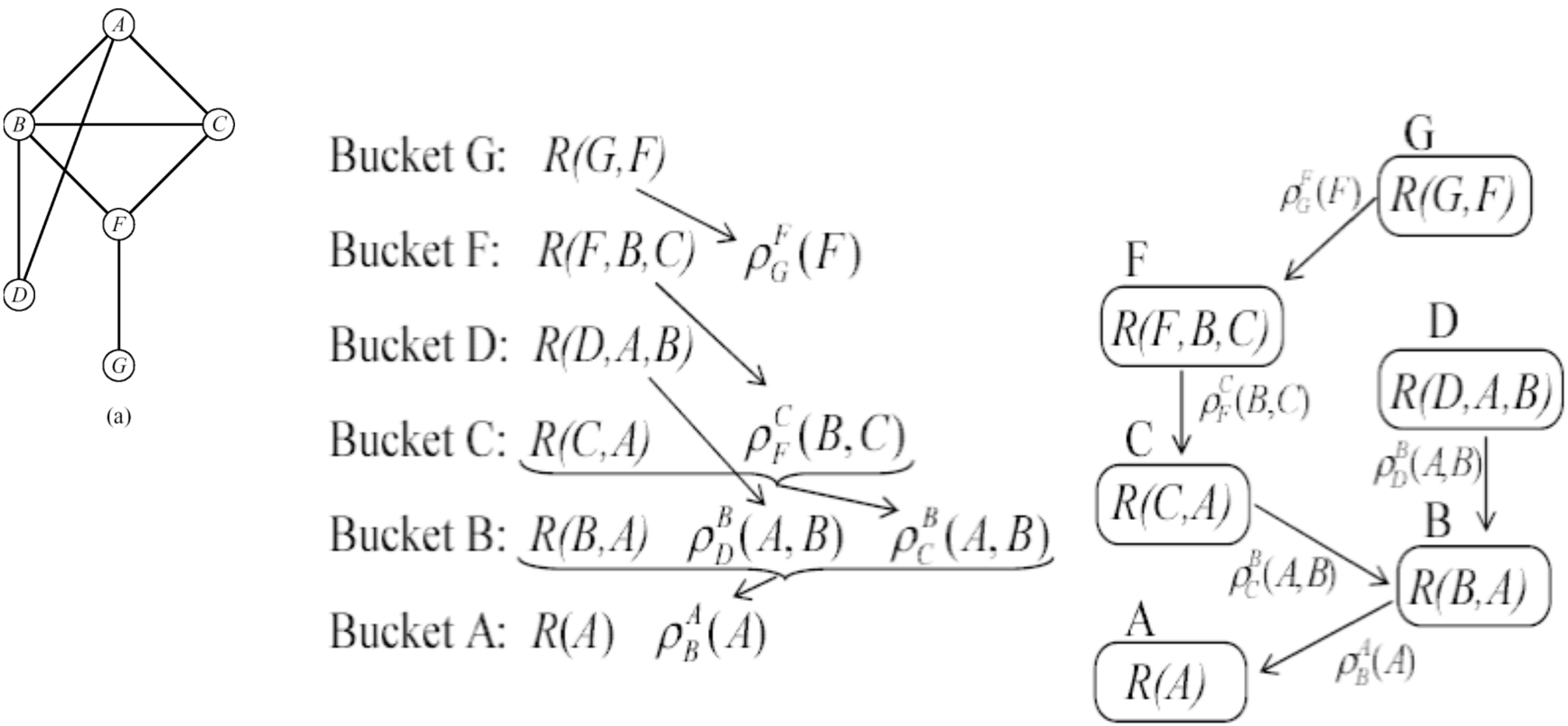
$Bucket(E):$ // $R_E$



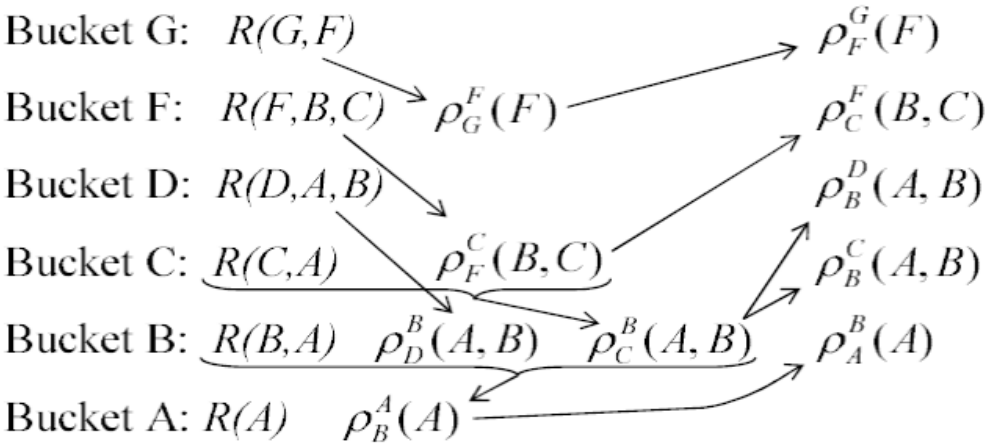

**Complexity :** $O(n \exp(w^*(d)))$,

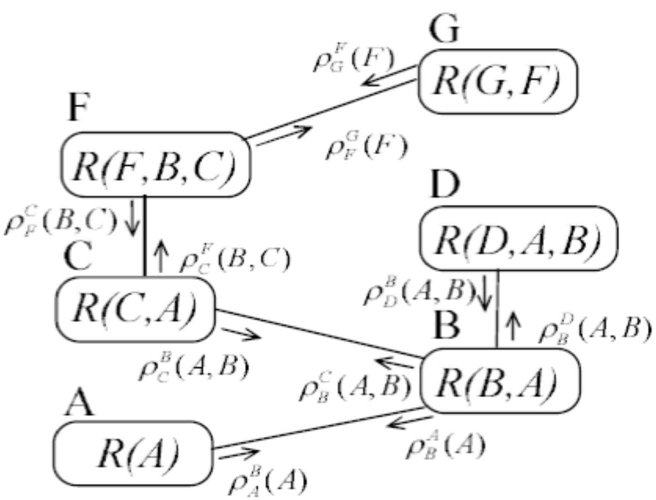
$w^*(d)$ - ***induced width along ordering d***

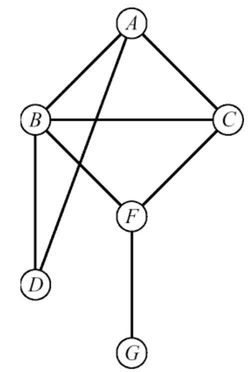
# From bucket-elimination to bucket-tree propagation



# The bottom up messages



(a)

# Adaptive-tree-consistency (ATC) as tree-decomposition

- Adaptive consistency is a message-passing along a bucket-tree

- **Bucket trees**: each bucket is a node and it is connected to a bucket to which its message is sent.

- **Theorem**: A bucket-tree is a tree-decomposition therefore, CTE adds a bottom-up message passing to bucket-elimination.

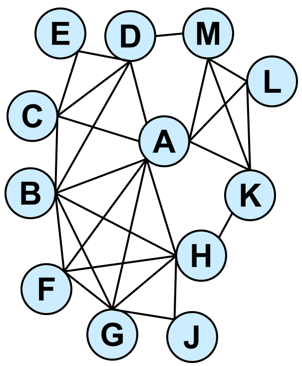
- The complexity of ATC is O(r deg $k^{w*+1}$) time and O(n $k^{w*}$) space.

# Outline

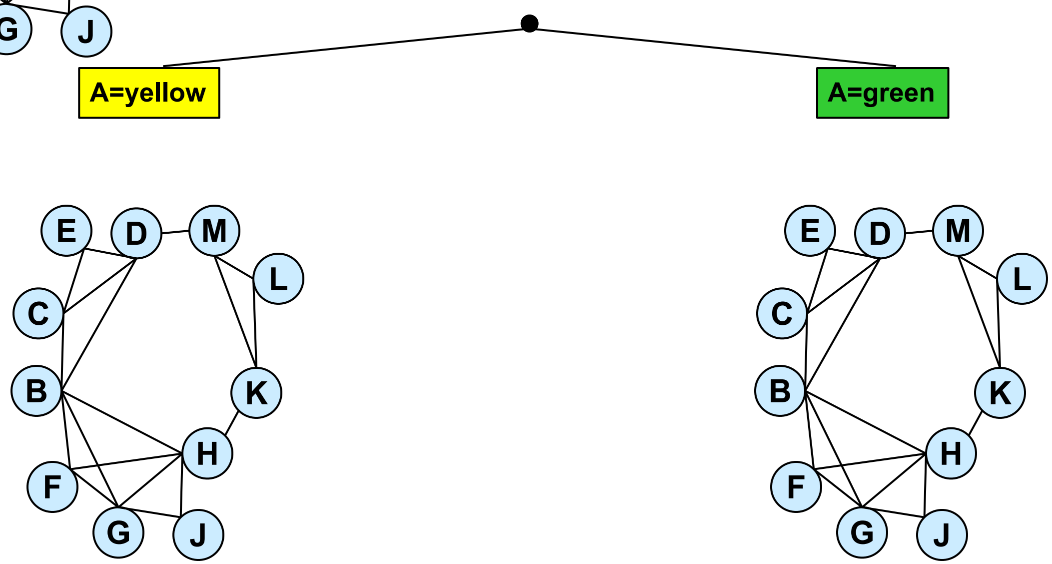
- Acyclic networks
- Join-tree clustering
- Conditioning vs tree-clustering

# Conditioning

Graph
Coloring
problem

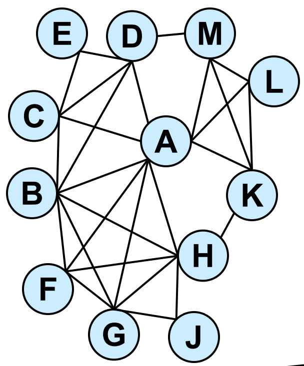



- Inference may require too much memory
- **Condition** on some of the variables

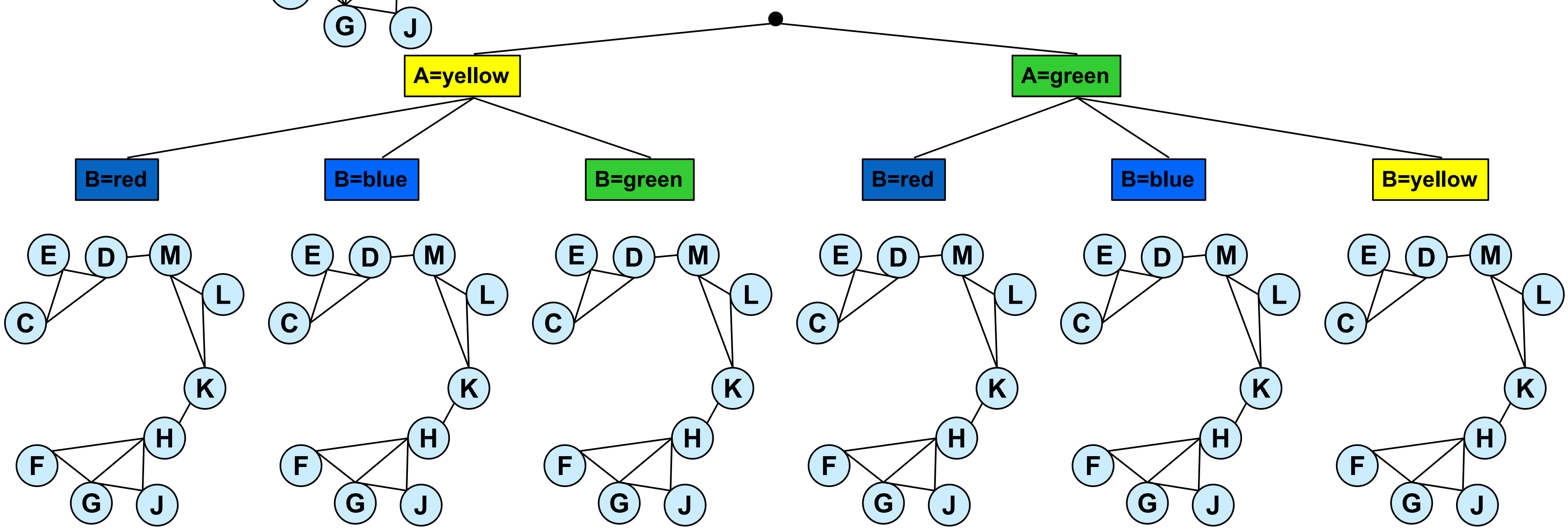


# Conditioning

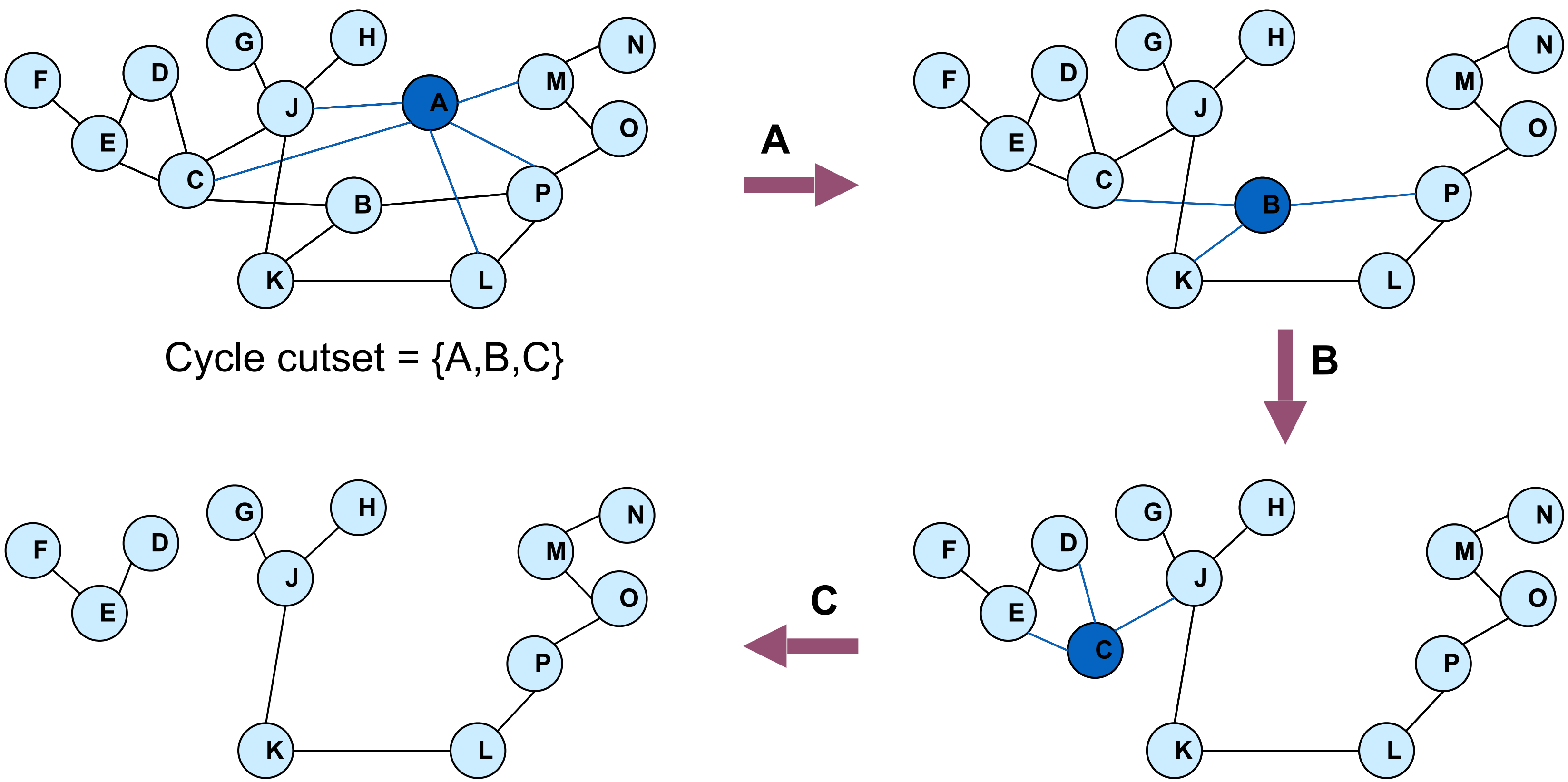
Graph Coloring problem




- Inference may require too much memory
- **Condition** on some of the variables



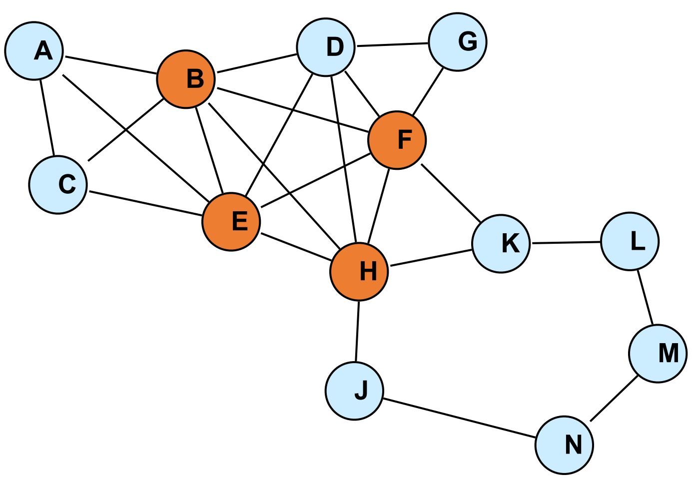
# Cycle cutset

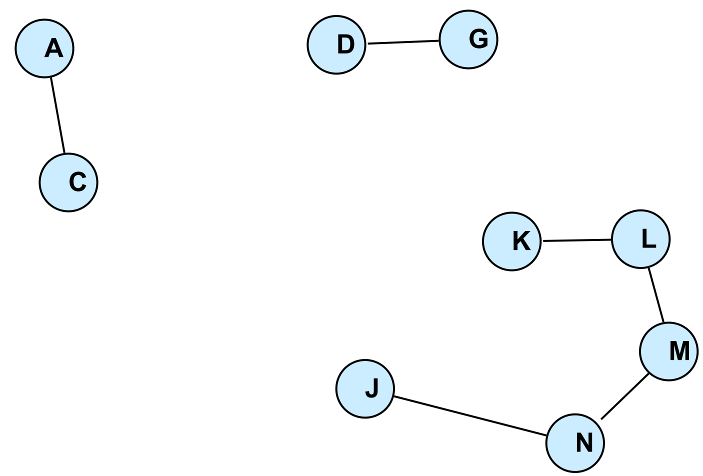


# Transforming into a tree

- **By Inference**
  - Time and spacde exponential in tree-width
- **By Conditioning-search**
  - Time exponential in the cycle-cutset

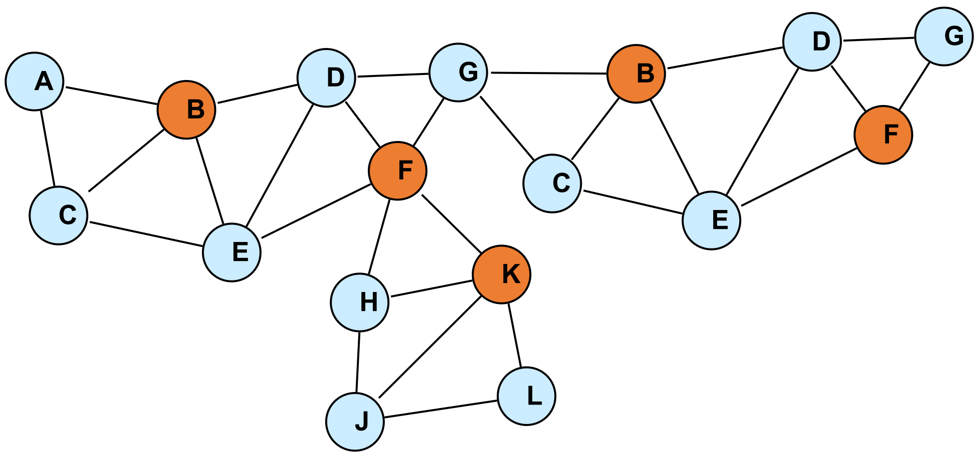
# Treewidth equals cycle cutset




treewidth = cycle cutset = 4

# Treewidth smaller than cycle cutset




treewidth = 2

cycle cutset = 5

# Outline

- Acyclic networks
- Join-tree clustering
- Conditioning vs tree-clustering

# Cluster Tree Elimination - properties

- Correctness and completeness: Algorithm CTE is sound and complete for generating minimal subproblems over chi(v) for every v: i.e. the solution of each subproblem is minimal.
- Time complexity: $O\ (\ deg \times (r+N) \times k^{w^*+1}\ )$
- Space complexity: $O\ (\ N \times d^{\ sep})$

  where *deg* = the maximum degree of a node
  r = number of of CPTs
  *N* = number of nodes in the tree decomposition
  *k*= the maximum domain size of a variable
  *w** = the induced width
  *sep* = the separator size

- JTC is $O\ (\ r \times k^{\ w^*+1}\ )$ *time and space*

# Cluster-Tree Elimination (CTE)

$$m_{(u,w)} = \Downarrow_{sep(u,w)} [\otimes_j \{m_{(v_j,u)}\} \otimes \psi(u)]$$