The Algorithms of Graphical Models

Javier Larrosa

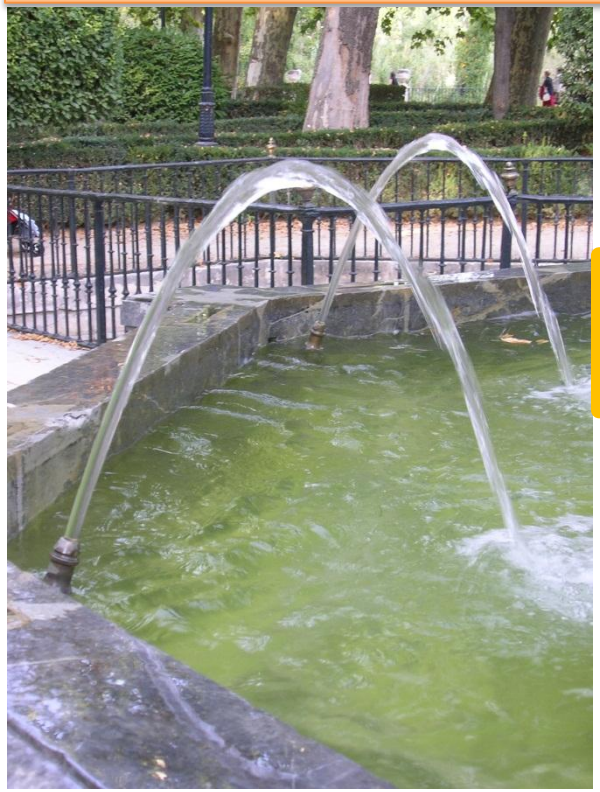
Emma Rollón

{larrosa, erollon}@lsi.upc.edu

Outline

- Introduction
- Exact algorithms
  - DFS, BB, And/Or, Variable Elim., Cluster Decomp.
- Approximate algorithms
  - EPT, relaxations

Mathematical Models



$$y = ax^2 + bx + c$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Data Structures

- Graphical Models are a data structure
  - They compactly store information about an exponentially large number of scenarios

- Factorization:

  $61\,917\,364\,224 = 3^{10}\,2^{20}$

Definition of GM

- Set of variables: $\mathrm{X} = \{x_1, x_2, \ldots, x_n\} \quad (x_i \in D)$
- Global function: $F(X)\colon D^n \longrightarrow U$

$$F(X) = f_1(X^1) \otimes f_2(X^2) \quad \otimes \cdots \otimes f_e(X^e)$$

- Query: $\bigoplus_X F(X)$

Valuation Structure $(U, \otimes, \oplus)$

- Valuation Set: U
- Aggregator: $\otimes : U^2 \longrightarrow U$
- Query operator: $\oplus : U^2 \longrightarrow U$

- $(U, \otimes, \oplus)$ is a commutative semiring
  - The operations are commutative and associative
  - Each operation has an identity element

$$a \oplus 0 = a \qquad\qquad a \otimes 1 = a$$

  - The distributive law holds:

$$(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$$

Graphical Models

- Bayesian Networks

$$P(x_1, x_2, \ldots, x_n) = \prod_i P(x_i | p(x_i))$$

- Markov Networks

$$P(x_1, x_2, \ldots, x_n) = \prod_i \Phi_i(X^i)$$

- Influence Diagrams
  - Chance and decision variables
  - utility functions

Graphical Models

– Constraint Satisfaction Networks (CSPs)

$$F(x_1, x_2, \ldots, x_n) = R_1(X^1) \bowtie \ldots \bowtie R_e(X^e)$$

– Weighted Constr. Sat. Networks (WCSPs)

$$F(x_1, x_2, \ldots, x_n) = \sum_i f_i(X^i)$$

(More) Graphical Models

- Boolean Satisfiability Problems (SAT)

$$F(x_1, x_2, \dots, x_n) = C_1 \wedge \dots \wedge C_e$$

Clauses

- Pseudo-boolean optimization

$$F(x_1, x_2, \dots, x_n) = \sum_i \alpha_i \left(\prod_j l_j\right)$$

Non-linear expr.

- Integer programming

$$F(x_1, x_2, \dots, x_n) = \sum_i c_i x_i$$

$$\text{subject to } AX \leq B$$

Linear expression

- …

Example: pseudo-boolean

$$F(x_1, \ldots, x_4) =$$
$$= x_1 + 4x_1x_2 + 4x_1(1 - x_2)(1 - x_3)$$
$$+ 7x_1x_2x_4 + 4(1 - x_2)x_3 + 9x_3(1 - x_4)$$

Example: pseudo-boolean

$$F(x_1, \ldots, x_4) = \\ = x_1 + 4x_1x_2 + 4x_1(1 - x_2)(1 - x_3) \\ + 7x_1x_2x_4 + 4(1 - x_2)x_3 + 9x_3(1 - x_4)$$

$$F = x_1 + 4x_1x_2 + 4x_1x'_2x'_3 + 7x_1x_2x_4 \\ + 4x'_2x_3 + 9x_3x'_4$$

IP as GM

- Standard IP formulation

min $40x_1 + 60x_2 + 15y$
subject to
$x_1 + x_2 + x_3 = 1$
$100x_1 + 200x_2 >= 90z_1$
$80x_1 + 150x_2 >= 50z_2$
$40x_1 + 20x_2 >= 20z_3$
$10x_1 >= 2z_4$
$z_1 + z_2 + z_3 + z_4 >= 3$
$x_2 <= y$
$z_i = 0$ or 1 i=1,2,3,4
$y = 0$ or 1
$x_i >= 0$ i=1,2,3

- GM formulation

min $F(X) = 40x_1 + 60x_2 + 15y +$
$f(x_1, x_2, x_3) +$
$g(x_1, x_2, z_1) +$
....

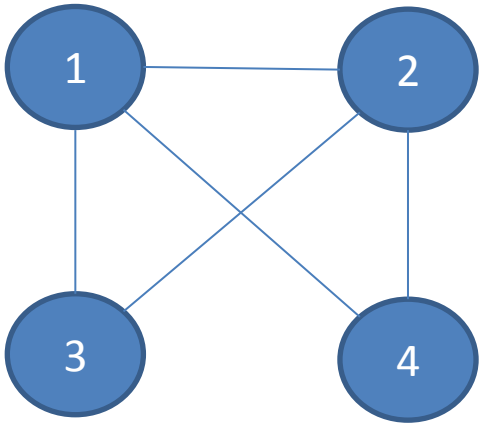
Canonical Queries

1. Satisfaction ($\exists_X F(X)=1$)
   - SAT, CSP, Determinism,…
   - Complexity: NP-complete
2. Optimization ($min_X F(X)$)
   - Max-SAT, Weighted CSP, MPE,…
   - Complexity: NP hard
3. Marginalization ($\sum_X F(X)$)
   - #SAT, #CSP, MAP
   - Complexity: #P

Interaction Graph

- $F(X) = f_1(X^1) \otimes f_2(X^2) \otimes \cdots \otimes f_e(X^e)$
- $G = (V, E)$ s.t.
  - $V = X$
  - $(x_i, x_j) \in E$ iff both variables appear in some factor

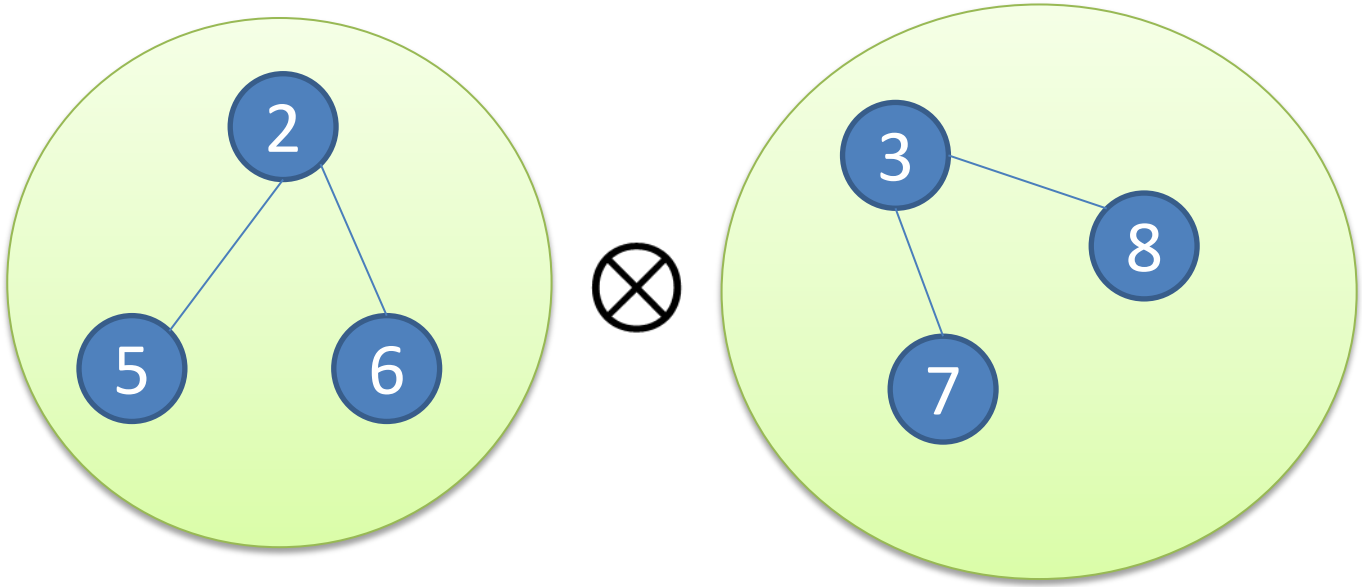
- $F = x_1 + 4x_1x_2 + 4x_1x'_2x'_3 + 7x_1x_2x_4 + 4x'_2x_3$



“Graphical” Properties: Independence

Each connected component of the problem can be solved independently:

$$\bigoplus_X F(X) = \left(\bigoplus_Y G(Y)\right) \otimes \left(\bigoplus_Z H(Z)\right)$$



“Graphical” Properties: Ciclycity

Dificulty increases along with graph ciclycity

- Measure of ciclycity (*width, w*)
  - Minimum ciclycity (*w=1*) → trees
  - …
  - Maximum ciclycity (*w=n*)→ cliques

- Fixed-parameter complexity w.r.t. ciclycity

Algorithms for GM

- Task: $F^* := \bigoplus_X F(X)$

- Exact Algorithms $Solve(F) = F^*$
- Approximate Algorithms:
  - Lower bounds $\mathrm{LB}(F) \leq F^*$
  - Upper bounds $\mathrm{UB}(F) \geq F^*$
  - Unbounded $\mathrm{AP}(F) \approx F^*$

Let's get started

- We will assume:

$$min_{x_1,x_2,\ldots,x_n} F(x_1, x_2, \ldots, x_n) = \sum_i f_i(X^i)$$

with boolean variables and positive costs

Operations on factors: conditioning

x_i	x_j	f
0	0	α
0	1	β
1	0	δ
1	1	η

$f(x'_i)$

x_j	g
0	α
1	β

Operations on factors: conditioning

x_i	x_j	f
0	0	α
0	1	β
1	0	δ
1	1	η

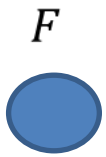
$f(x'_i)$

x_j	g
0	α
1	β

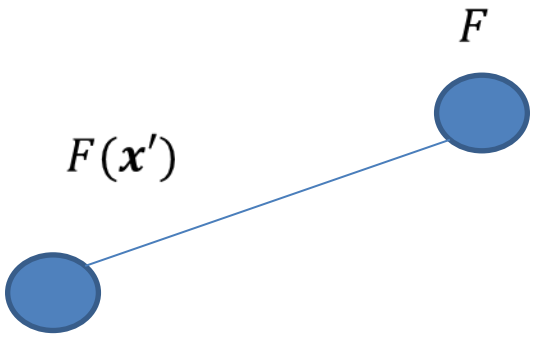
$\mathbf{g}(x_j)$

h
β

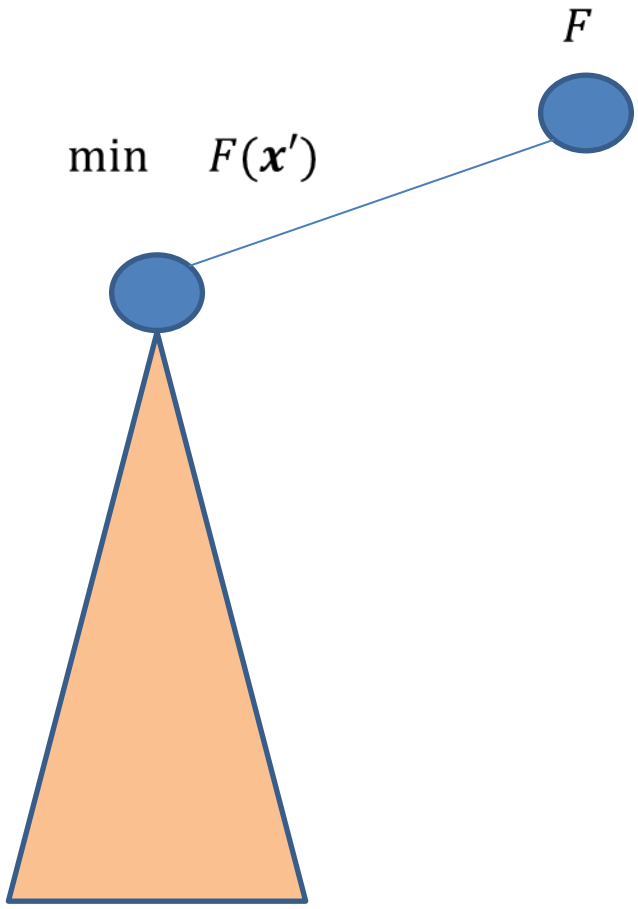
Depth-First Search



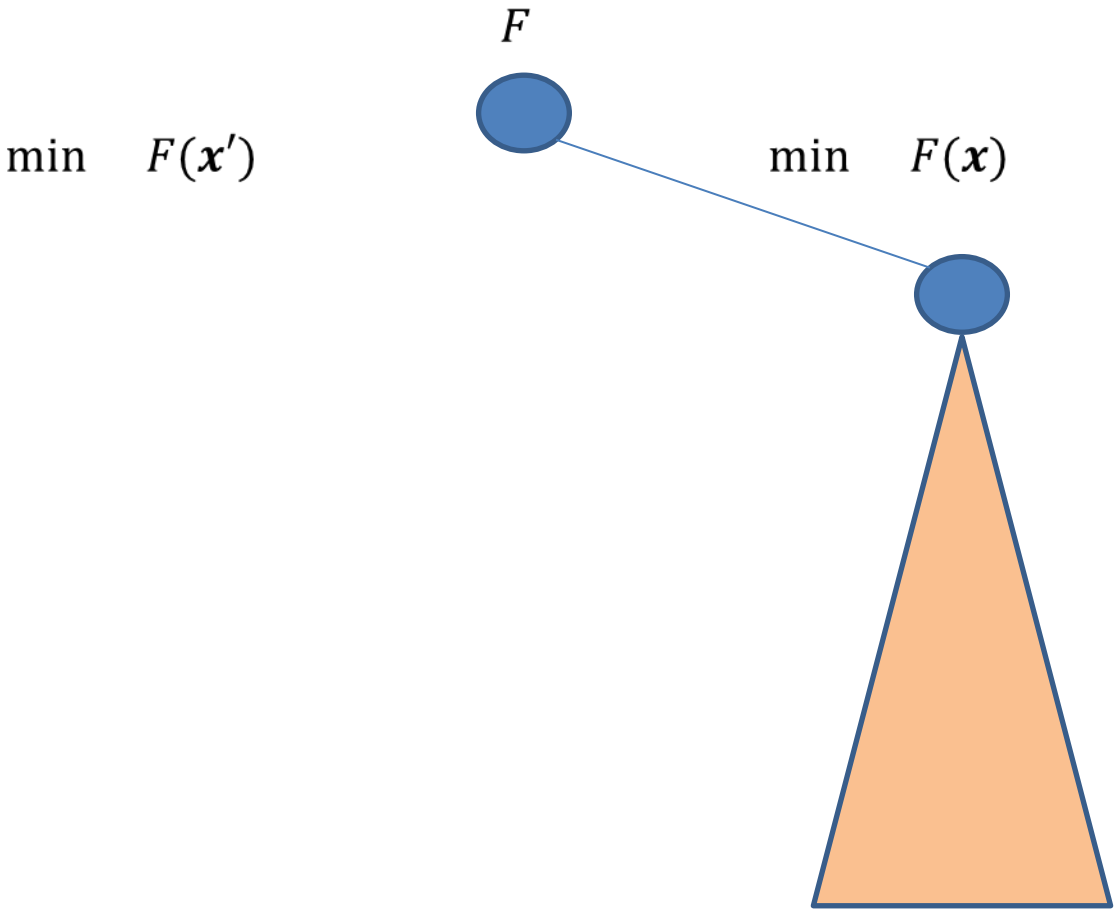
Depth-First Search



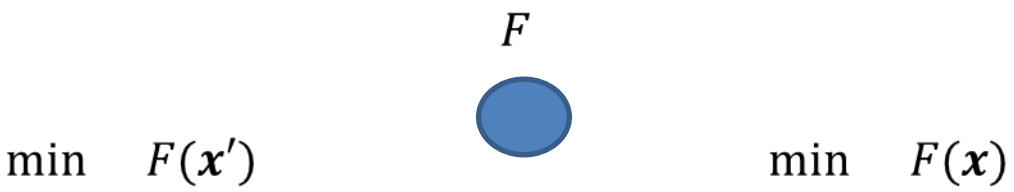
Depth-First Search



Depth-First Search



Depth-First Search



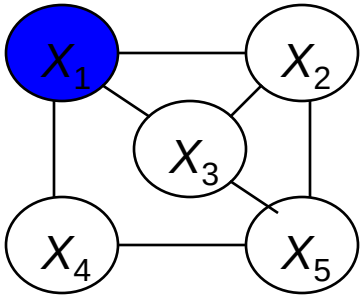
Depth-First Search

$$\min F := \min\{\min F(\boldsymbol{x}'), \min F(\boldsymbol{x})\}$$

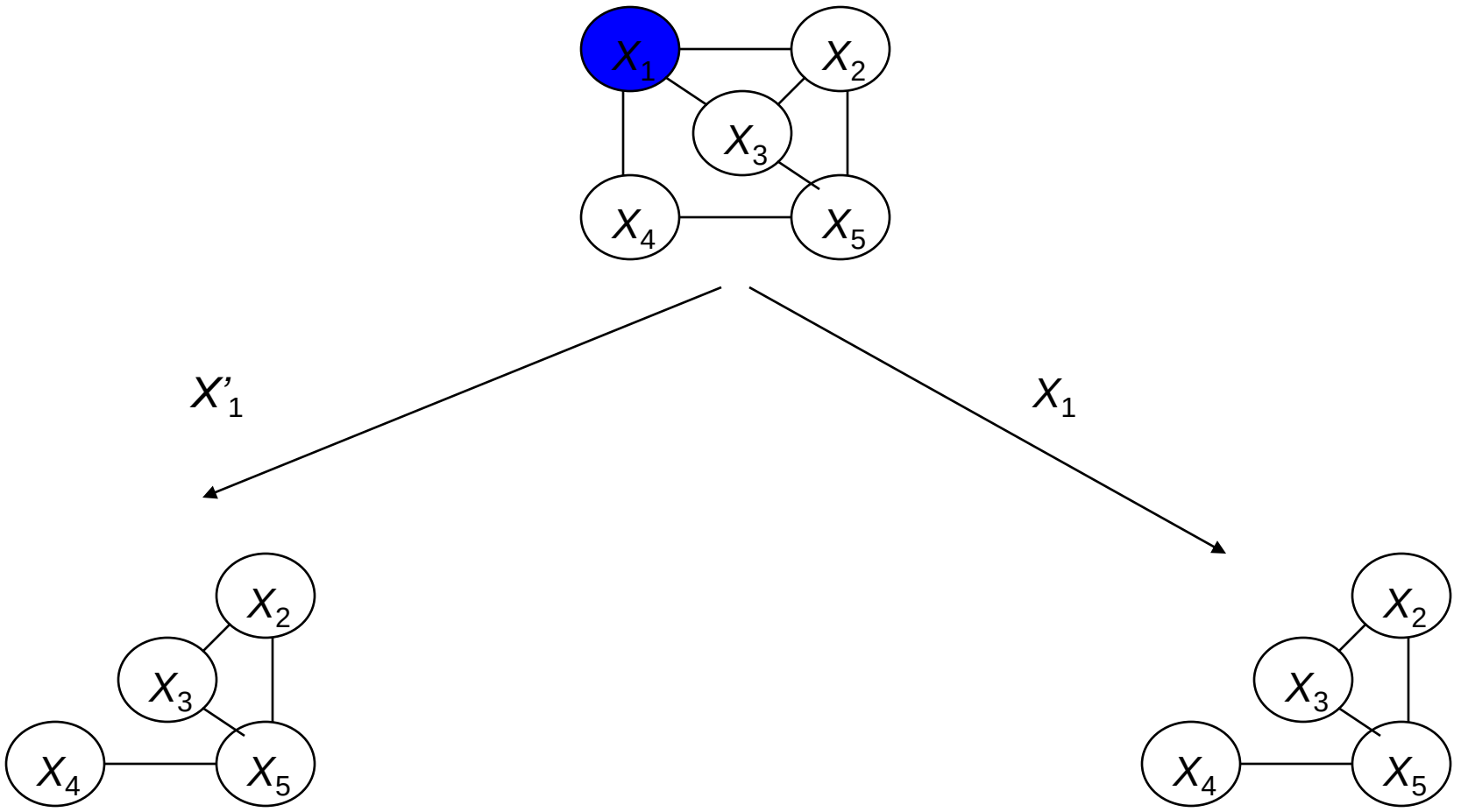
$$\min F(\boldsymbol{x}') \qquad \min F(\boldsymbol{x})$$

“graphically”

- Select a variable



“graphically”



Depth-First Search

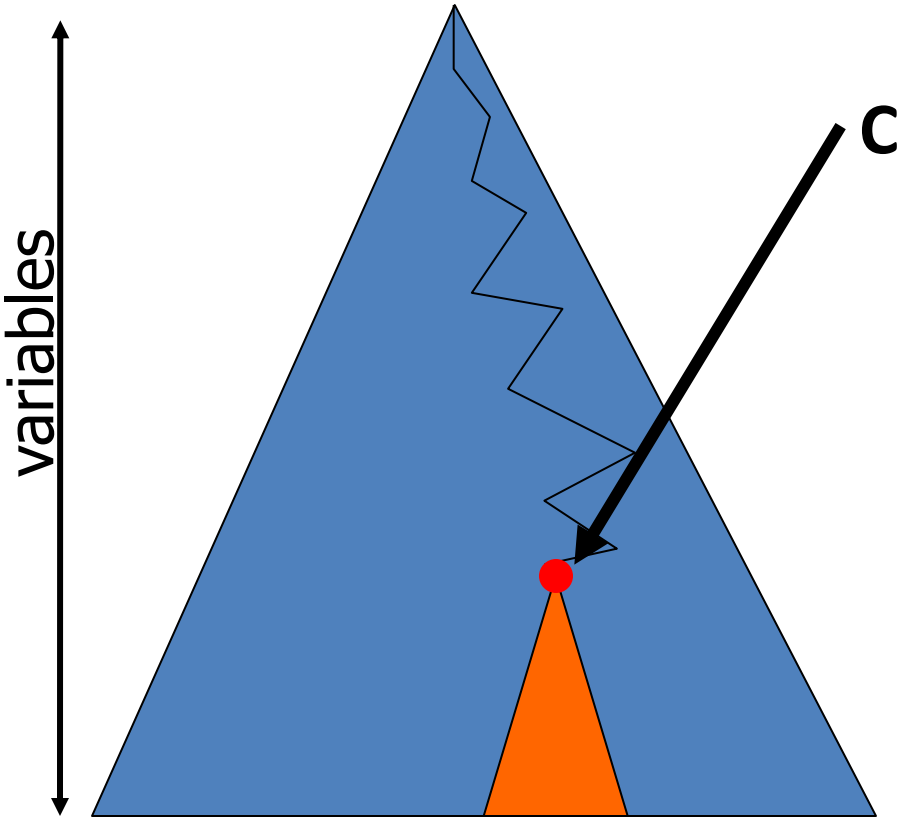
```
function Solve(F)
    if Constant(F) then return F
    x := SelectVar(F);
    return min{Solve(F(x')), Solve(F(x))};
endfunction
```

Cost $O(2^n)$

Algorithmic aspects: incrementality

```
function Solve(F)
    if Constant(F) then return F
    x := SelectVar(F);
    return min{Solve(F(x')), Solve(F(x))};
endfunction
```

Branch and Bound (for optimiz. query only)



(UB) Upper Bound

= best solution so far

(LB) Lower Bound

under estimation of the best solution in the sub-tree

If LB ≥ UB then prune

DFS+BB

```
function Solve(F, ub)
    if Constant(F) then return min{F,ub};
    if (LB(F)≥ ub) return ub;
    x:= SelectVar(F);
    ub:=Solve(F(x'),ub);
    return Solve(F(x),ub);
endfunction
```

Initial call: Solve(*F*, UB(F))

Cost $O(2^n)$

Observation

- $0 \leq LB(F) \leq Solve(F)$
  - If $LB(F) = 0$ then BB visits 2^n nodes
  - If $LB(F) = Solve(F)$ then BB visits n nodes

- The better the lower bound is the higher the pruning takes place …
- … but it is more costly

Algorithmic aspects: encourage pruning

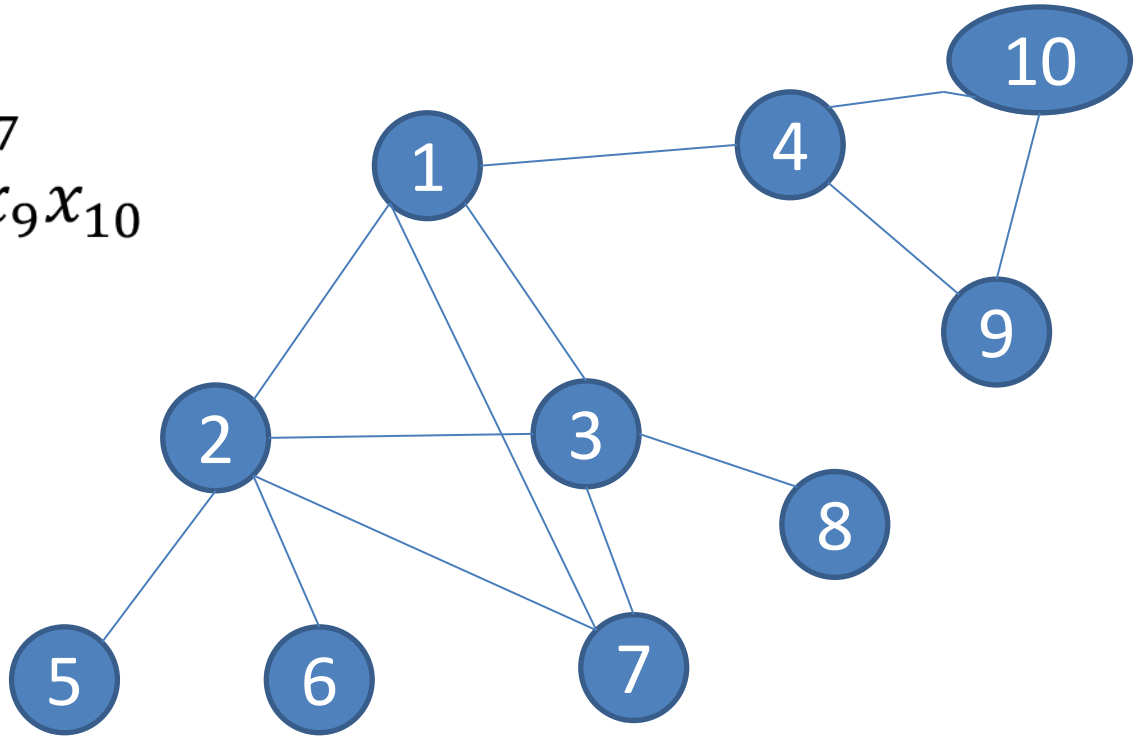
```
function Solve(F, ub)
    if Constant(F) then return min{F,ub};
    if (LB(F)≥ ub) return ub;
    x:= SelectVar(F);
    ub:=Solve(F(x'),ub);
    return Solve(F(x),ub);
endfunction
```

Initial call: Solve(*F*, UB(F))

Cost $O(2^n)$

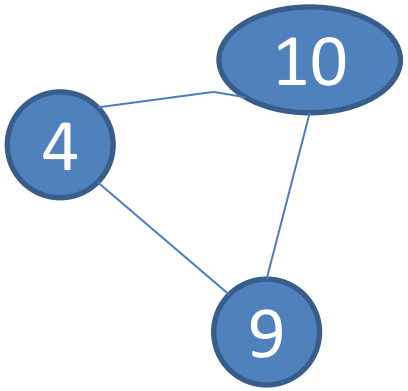
And/Or search ($(\otimes/\oplus)$-search)

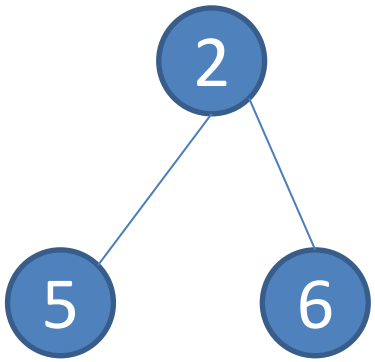
$$F = x_1x_2x_3x_7 + x'_1x_3 + x_1x_4 + x_2x_5 + x'_2x_6 + x_3x_7 + x'_3x'_8 + x_4x_9x_{10}$$

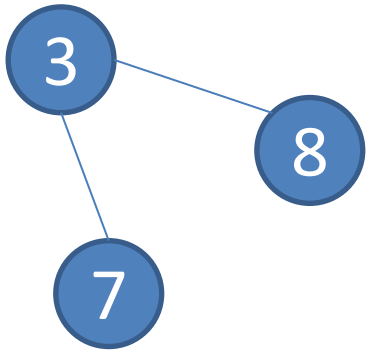


And/Or

$$F(x'_1) = x_3 + x_2x_5 + x'_2x_6 + x_3x_7 + x'_3x'_8 + x_4x_9x_{10}$$

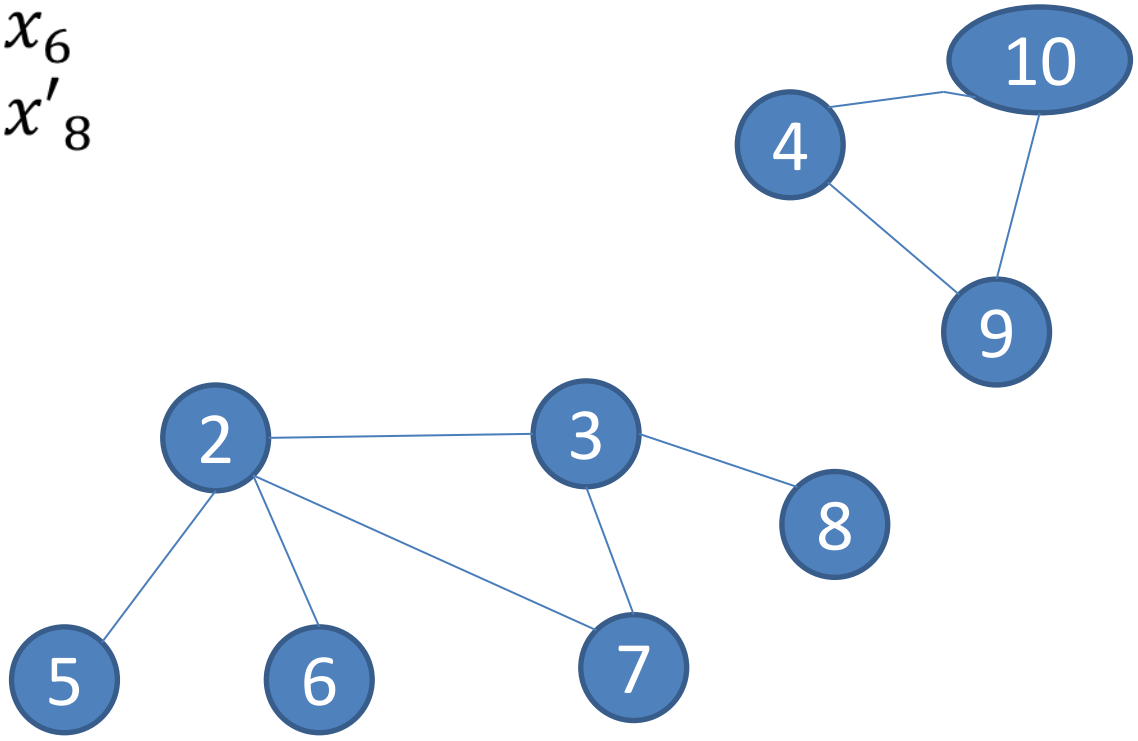




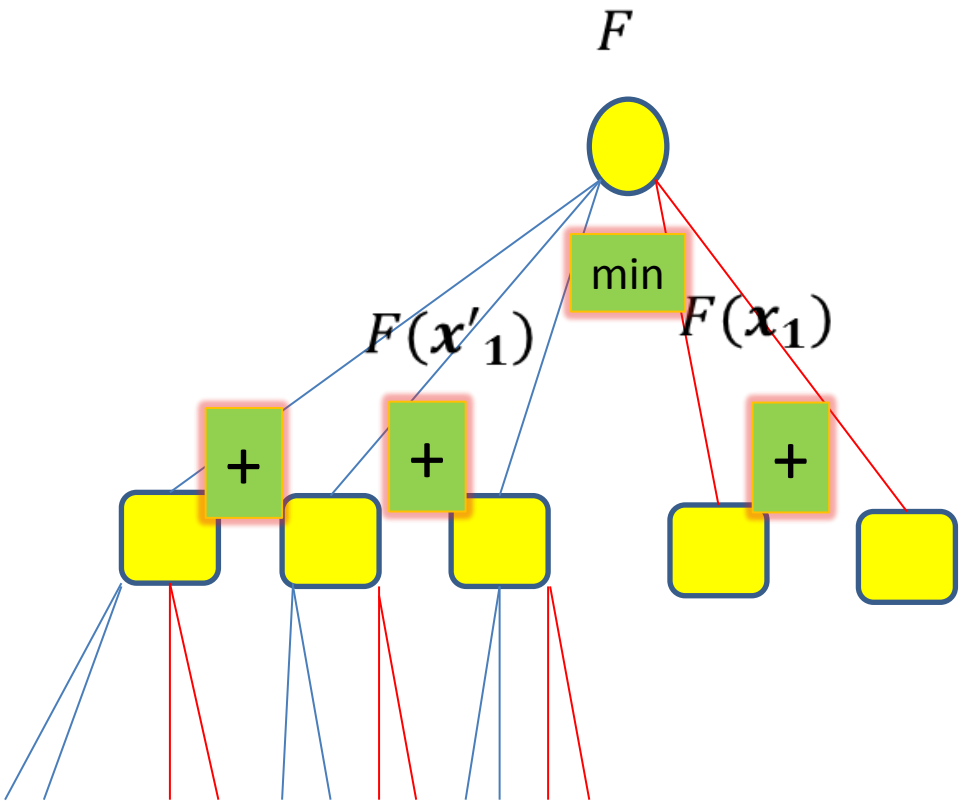


And/Or

$$F(\boldsymbol{x_1}) = x_2x_3x_7 + x_4 + x_2x_5 + x'_2x_6 + x_3x_7 + x'_3x'_8 + x_4x_9x_{10}$$



And/Or Search Tree



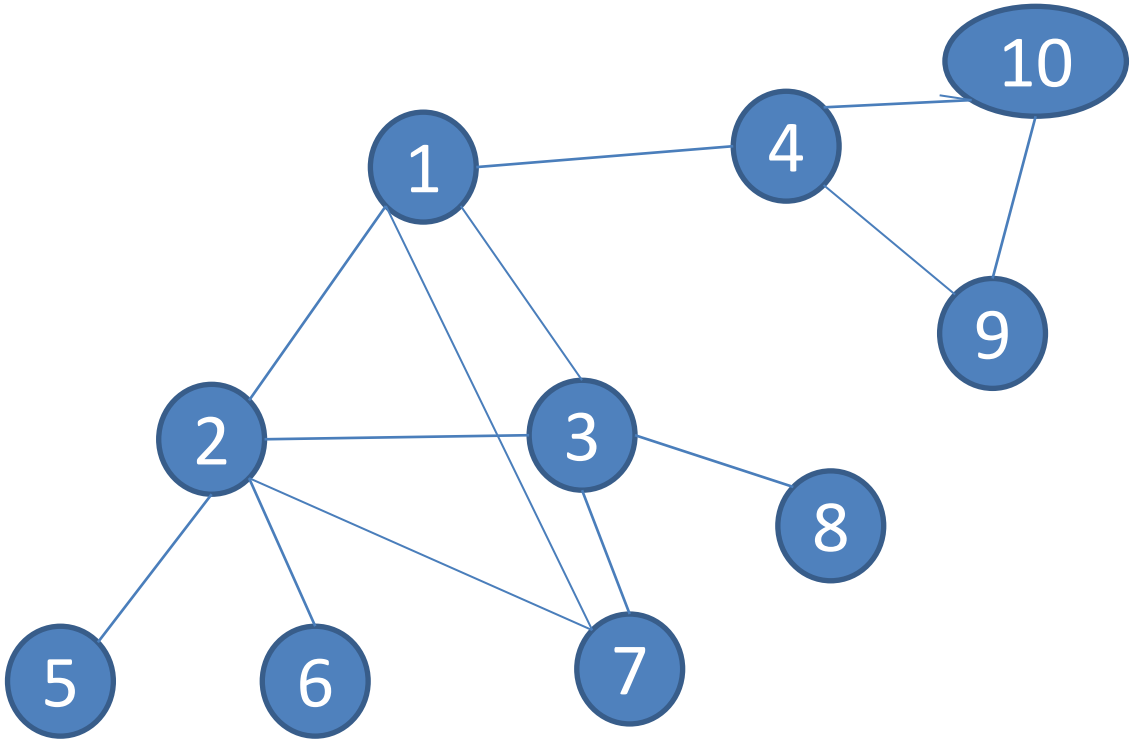
And/Or

function Solve(*F*)
  if Constant(*F*) **then return** *F*
  x := SelectVar(*F*);
  return min {IndepSolve(*F*(***x***'')),IndepSolve(*F*(***x***))}
endfunction

function IndepSolve(*F*)
  $\{F_1, F_2, \ldots, F_k\}$:=CC(*F*);
  return Σ_i Solve(F_i);
endfunction

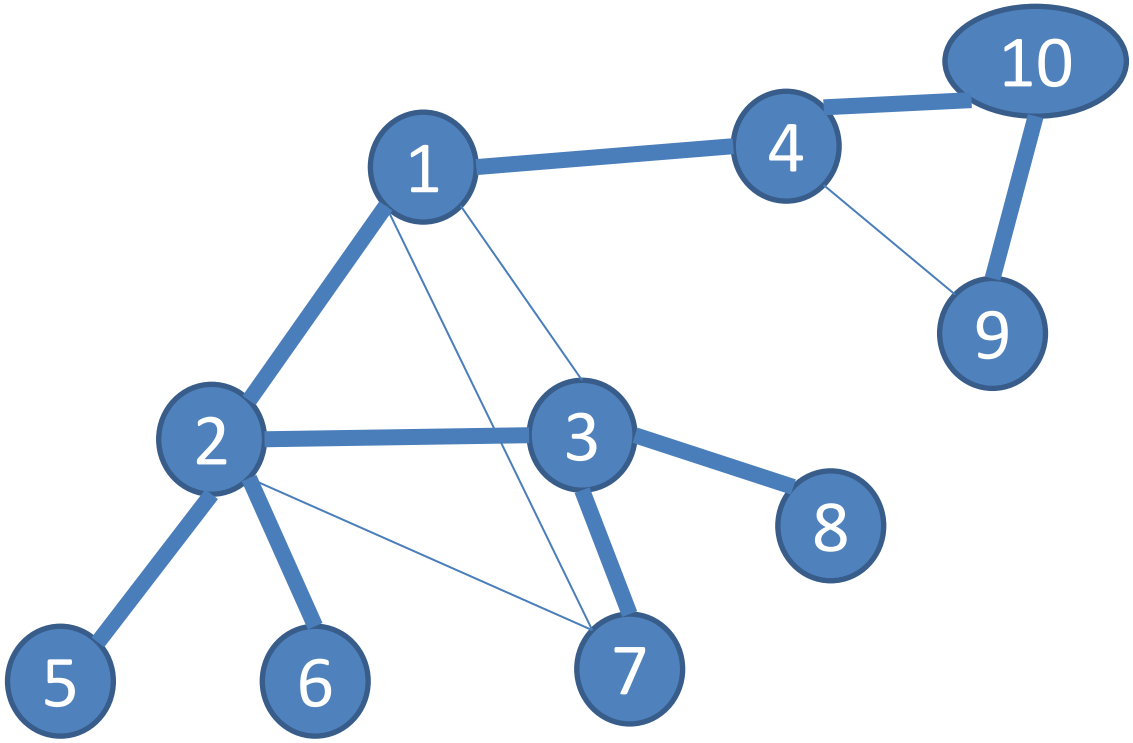
Complexity

- Pseudo-tree



Complexity

- Pseudo-tree

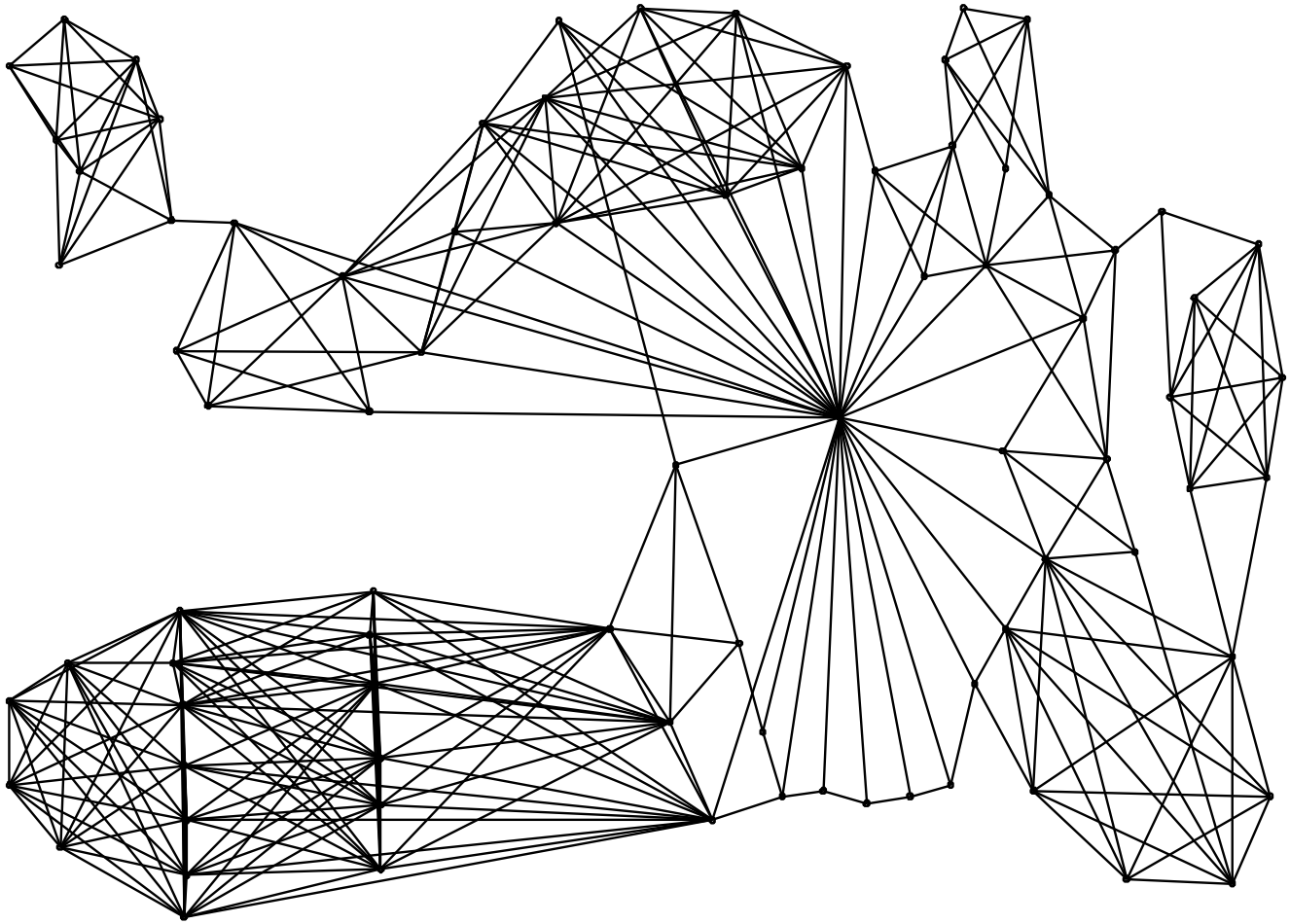


Complexity

- *h* pseudo-tree height
- Cost: $O(2^h)$
- Property: $h = O(w \log n)$

- This is true iff the variable ordering of the algorithm is consistent with the pseudotree

Real example (CELAR SCEN06)



Branch and Bound And/Or

function Solve(*F*,*ub*)
  if Constant(*F*) **then return** min{*F*,ub}
  x := SelectVar(*F*);
  ub :=IndepSolve(*F*(***x'***),*ub*);
  return IndepSolve(*F*(***x***),*ub*);
endfunction

function IndepSolve(*F*,*ub*)
  $\{F_1, F_2, \ldots, F_k\}$:=CC(*F*);
  if Σ_i LB(F_i)≤ *ub* **then** *ub*:= min{*ub*, Σ_i Solve(F_i,*ub*)};
  return *ub*;
endfunction

Algoritmic aspects

function Solve(*F*,*ub*)
  if Constant(*F*) **then return** min{*F*,ub}
  x := SelectVar(*F*);
  ub :=IndepSolve(*F*(***x'***),*ub*);
  return IndepSolve(*F*(***x***),*ub*);
endfunction

This is not a good upper bound

function IndepSolve(*F*,*ub*)
  $\{F_1, F_2, \ldots, F_k\}$:=CC(*F*);
  if Σ_i LB(F_i)≤ *ub* **then** *ub*:= min{*ub*, Σ_i Solve(F_i,*ub*)};
  return *ub*;
endfunction

Algoritmic aspects

function Solve(*F*,*ub*)
  if Constant(*F*) **then return** min{*F*,ub}
  x := SelectVar(*F*);
  ub :=IndepSolve(*F*(***x'***),*ub*);
  return IndepSolve(*F*(***x***),*ub*);
endfunction

Replace ub by UB(F_i)

function IndepSolve(*F*,*ub*)
  $\{F_1, F_2, \ldots, F_k\}$:=CC(*F*);
  if Σ_i LB(F_i)≤ *ub* **then** *ub*:= min{*ub*, Σ_i Solve(F_i,*ub*)};
  return *ub*;
endfunction

Algoritmic aspects

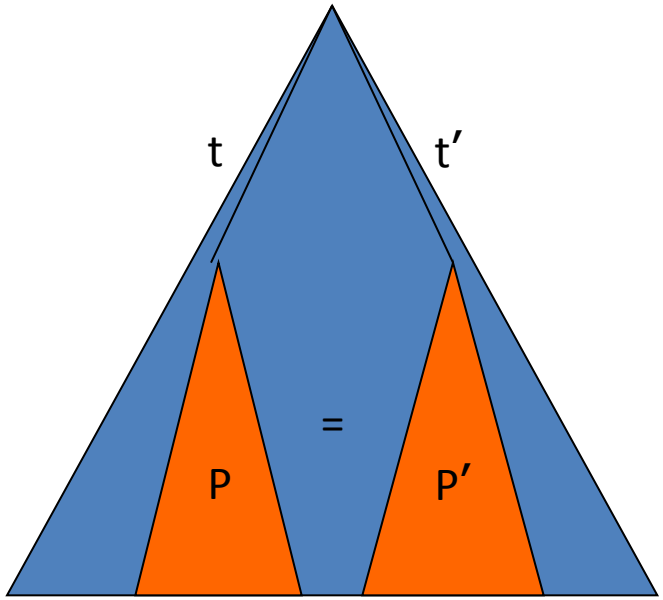
function Solve(*F*,*ub*)
  if Constant(*F*) **then return** min{*F*,ub}
  x := SelectVar(*F*);
  ub :=IndepSolve(*F*(***x'***),*ub*);
  return IndepSolve(*F*(***x***),*ub*);
endfunction

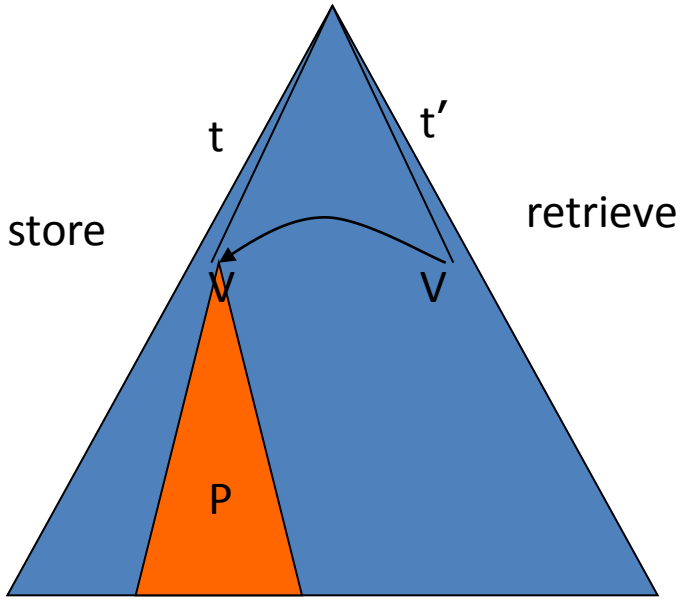
function IndepSolve(*F*,*ub*)
  $\{F_1, F_2, \ldots, F_k\}$:=CC(*F*);
  if Σ_i LB(F_i)≤ *ub* **then** *ub*:= min{*ub*, Σ_i Solve(F_i,*ub*)};
  return *ub*;
endfunction

Repeat the test replacing LBs by exact values
As they are available

Memoization

Different nodes, Same subproblem





Memoization

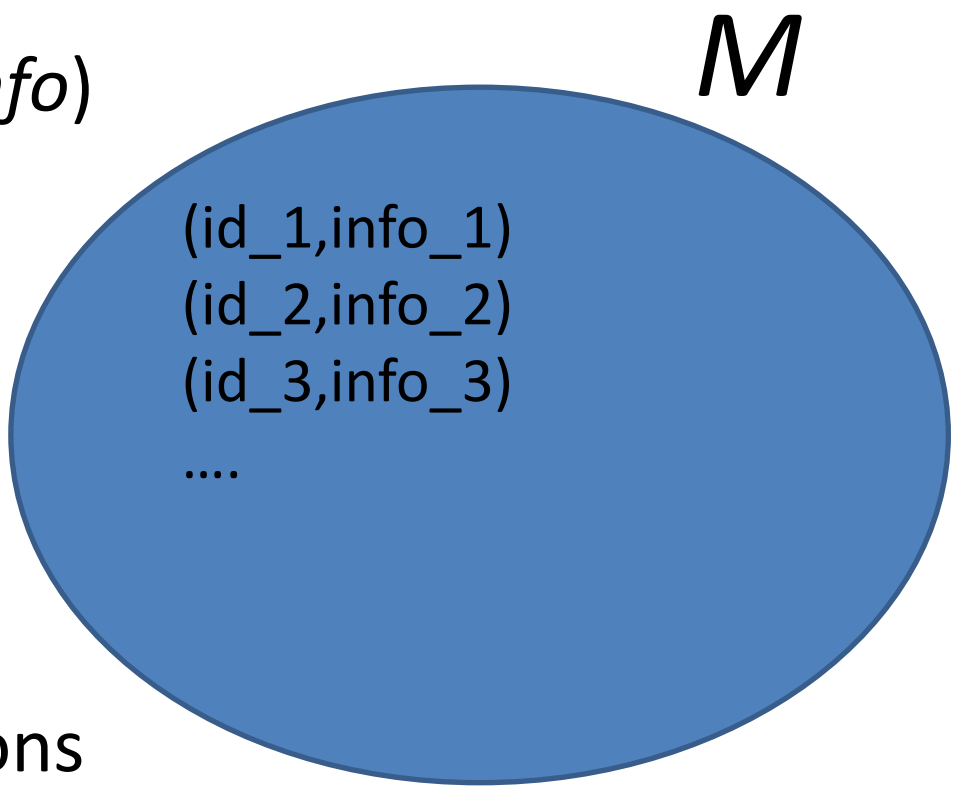
- Associative Memory:
  stores set of pairs (*id,info*)

- Two operations:

  M.Add(*id,info*)

  M.Check(*id*)

- Efficient implementations



Memoization

```
function Solve(F, M)
    if Constant(F) then return F
    if (M.Check(F)≠NIL) then return M.Check(F)
    x:= SelectVar(F);
    m:=min{Solve(F(x'),M), Solve(F(x),M)};
    M.Add(F,m);
   return m;
```

Algorithmic aspects

- Drawback: *M* exponentially big
- Bounded memoization: forget subproblems rarely retrieved
- Smart memoization:
  - identify as equal isomorphic problems
  - Store independent subproblems
- Memoization combines well with BB
- Memoization combines well with and/or:
  - And/or: $O(2^{w \log n})$
  - And/or + Memoiz: $O(2^{w})$

Memoization + BB

```
function Solve(F, ub, M)
    if Constant(F) then return min{F,ub}
    if (M.Check(F)≠NIL) then
            return min{M.Check(F),ub}
    if (LB(F)≥ ub) return ub;
    x:= SelectVar(F);
    ub:=Solve(F(x'),ub); ub:=Solve(F(x),ub);
    M.Add(F,ub);
   return ub;
```

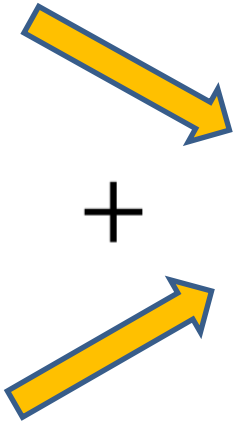
Variable Elimination (Bucket Elimination, BE)

- It is based on Dynamic programming
- Idea: Eliminate variables while preserving query result

Operations on factors: combination ($\otimes$)

x_i	f
0	α
1	β

x_j	g
0	δ
1	η



x_i	x_j	$f+g$
0	0	α+δ
0	1	α+η
1	0	β+δ
1	1	β+η

Combination property

$$f_1(X^1) + \cdots + f_i(X^i) + f_k(X^k) + \cdots + f_e(X^e)$$

is equivalent to

$$f_1(X^1) + \cdots + (f_i + f_k)(X^i \cup X^k) + \cdots + f_e(X^e)$$

Operations on factors: marginalization ($\oplus$)

x_i	x_j	f
0	0	α
0	1	β
1	0	δ
1	1	η

$min_{x_j} f$

x_i	$min_{x_j} f$
0	min{α,β}
1	min{δ,η}

Marginalization property

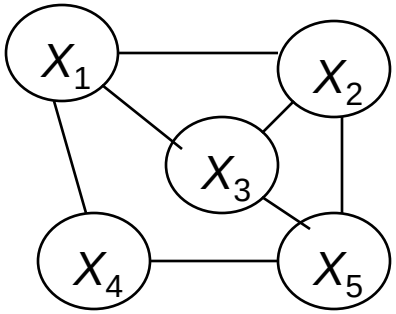
If f_i is the only factor mentioning x_k in

$$f_1(X^1) + \cdots + f_i(X^i) + \cdots + f_e(X^e)$$

Then it is equivalent to

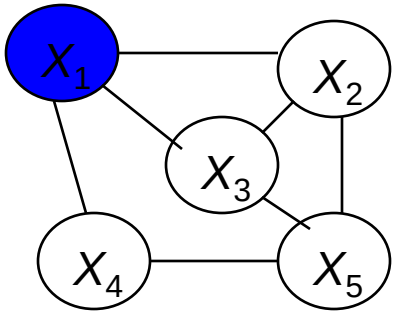
$$f_1(X^1) + \cdots + min_{x_k}(f_i(X^i)) + \cdots + f_e(X^e)$$

Variable Elimination



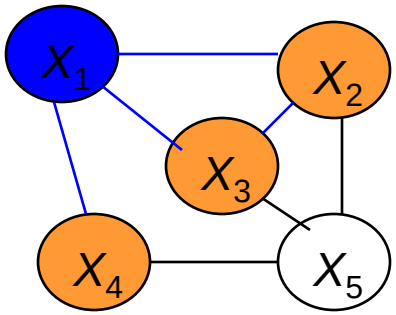
Variable Elimination

- Select a variable



Variable Elimination

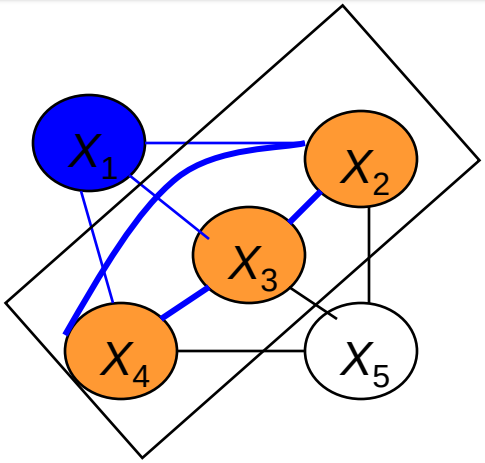
- Identify set of factors that *mention* the variable (bucket)



Variable Elimination

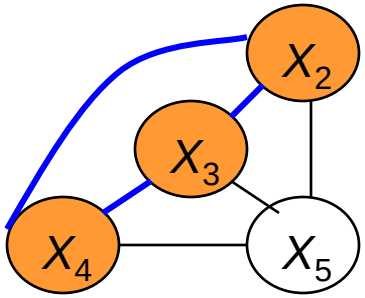
- Compute new factor

$$g := min_x\Big(\sum_{f \in bucket} f\Big)$$



Variable Elimination

- Replace bucket by new factor



Variable Elimination

function Solve(*F*)
    if Constant(*F*) **then return** *F*
    x := *SelectVar*(*F*);
    return Solve(Elim(x,F);
Endfunction

function Elim(*x*,*F*)
    B := factors with *x* in their scope;
    F':= replace *B* by $min_x(\sum_{f \in B} f)$ in *F*
    return *F'*;
endfunction

VE: complexity (time and space)

- Cost of eliminating x: $O(2^{degree(x)})$
- Cost of VE: $O(2^{w})$

- Highly dependent on the elimination ordering
  - Finding the best ordering: NP-complete
  - Heuristics are usually used

VE-BB(k)

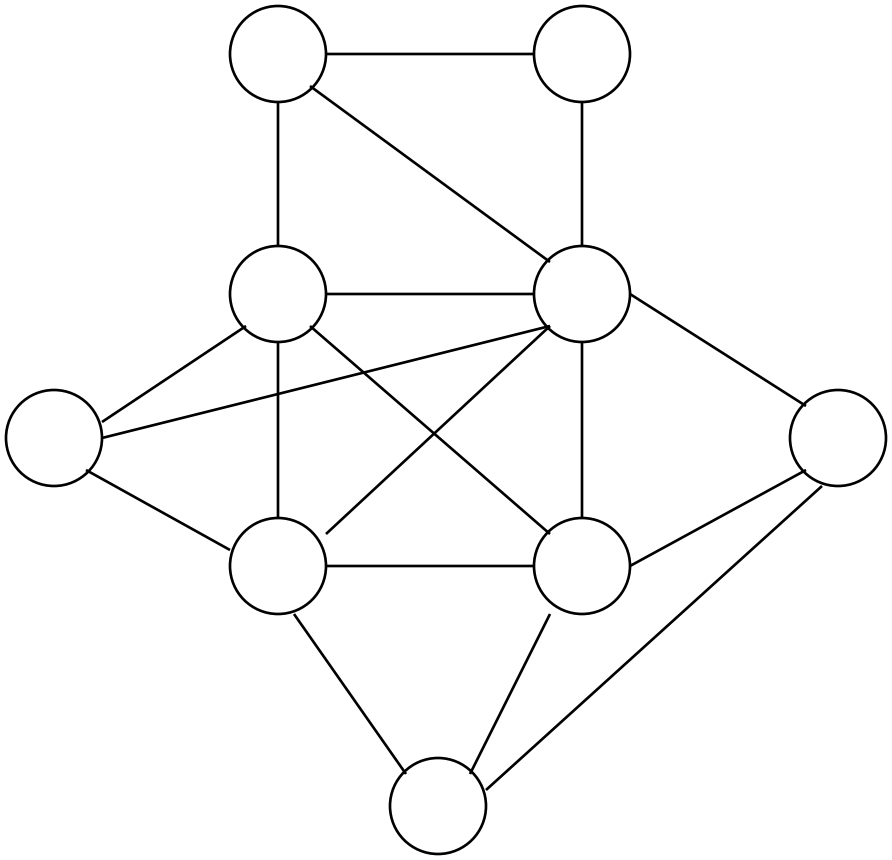
- At each node:

  $x_i \leftarrow$ select variable

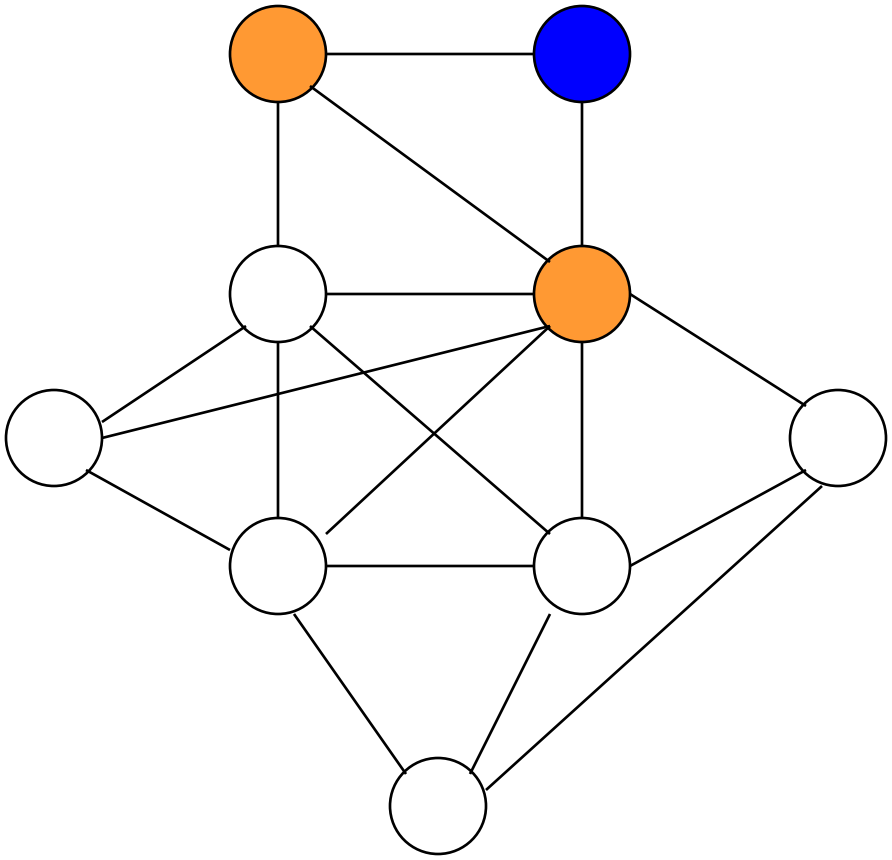
  if $dg(x_i) \leq k$ then eliminate x_i

  else branch on the values of x_i

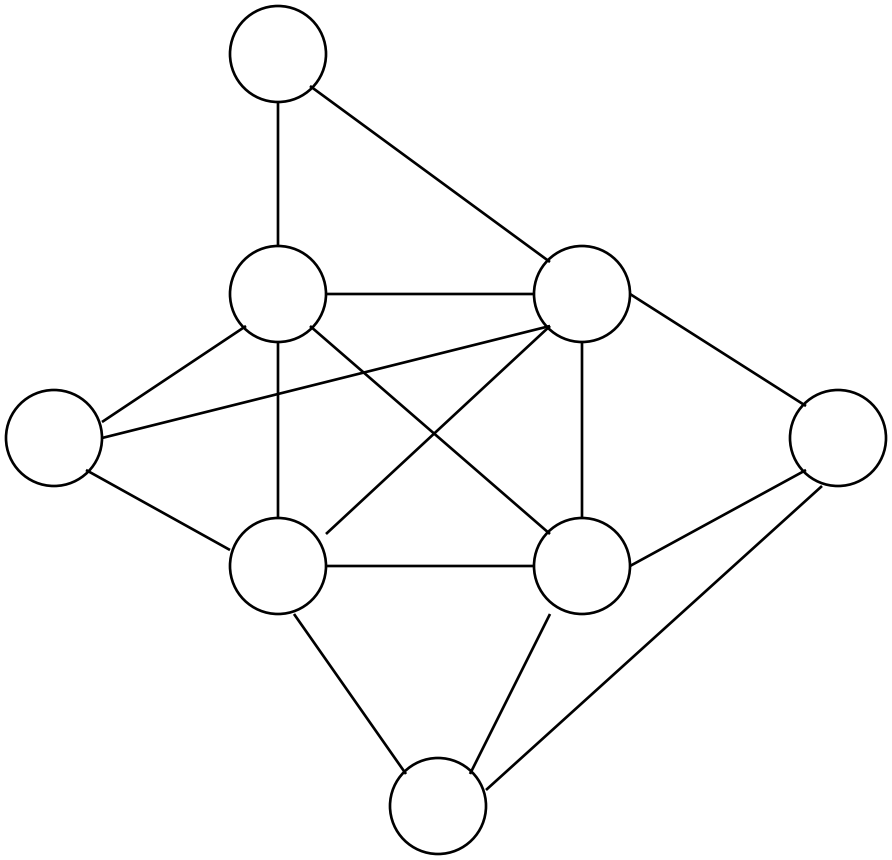
VE-BB(2): example



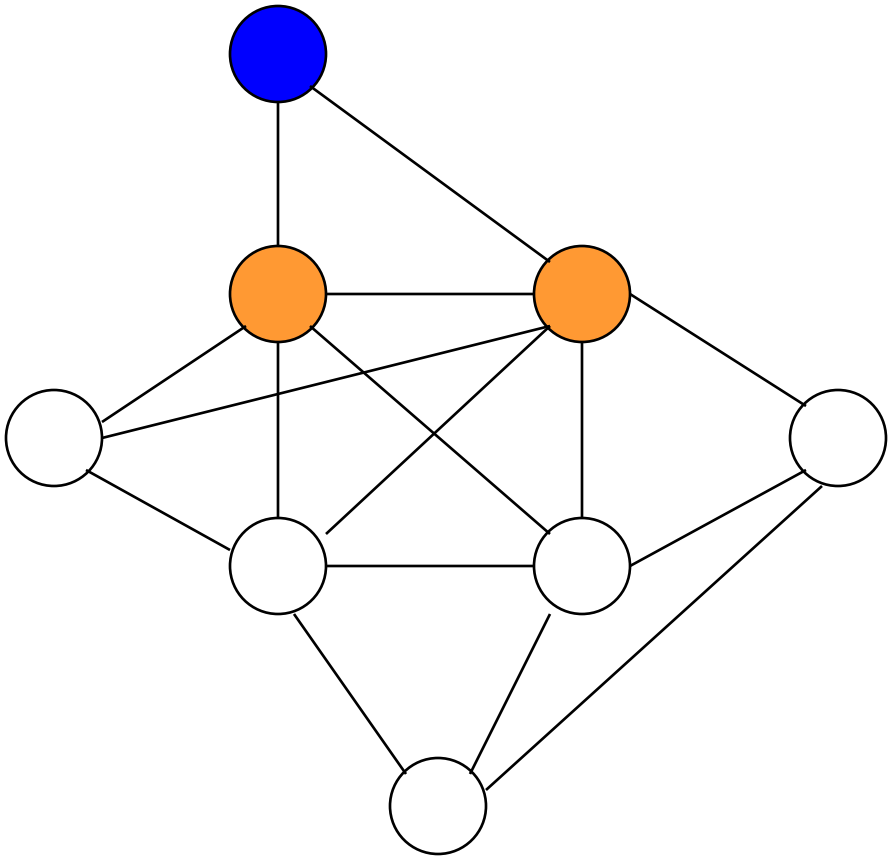
VE-BB(2): example



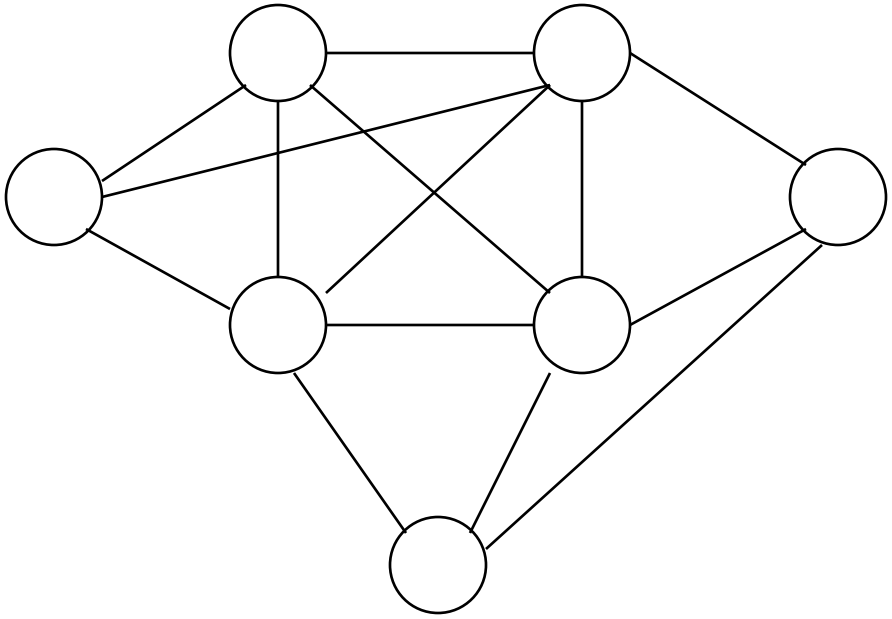
VE-BB(2): example



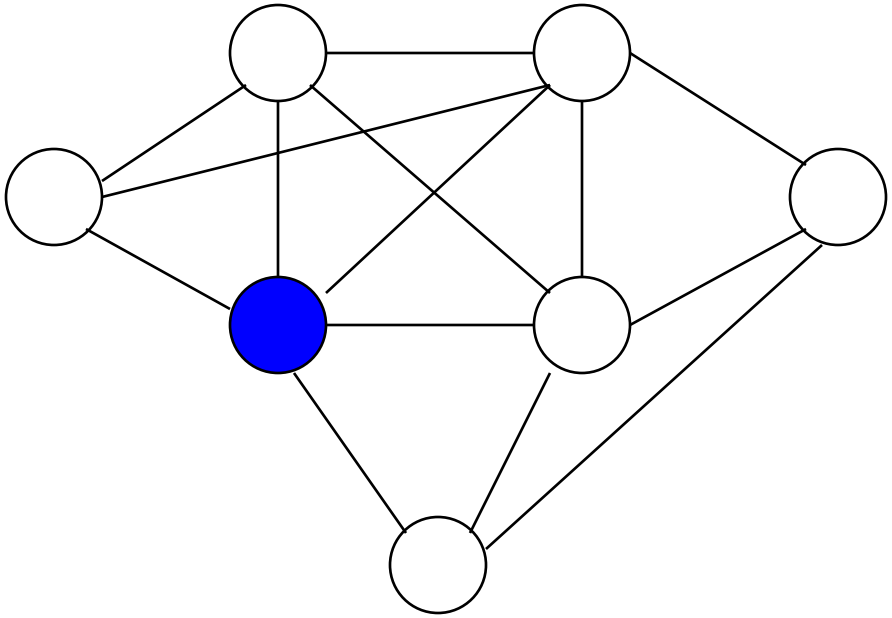
VE-BB(2): example



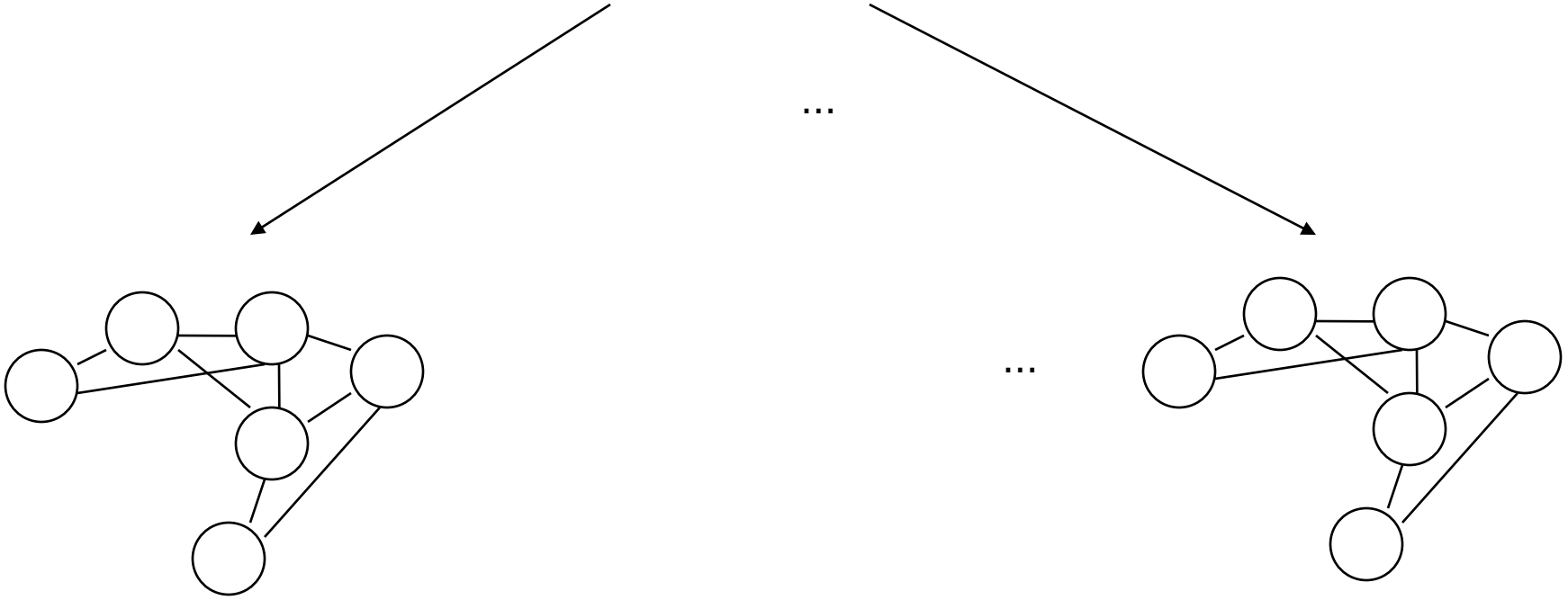
VE-BB(2): example



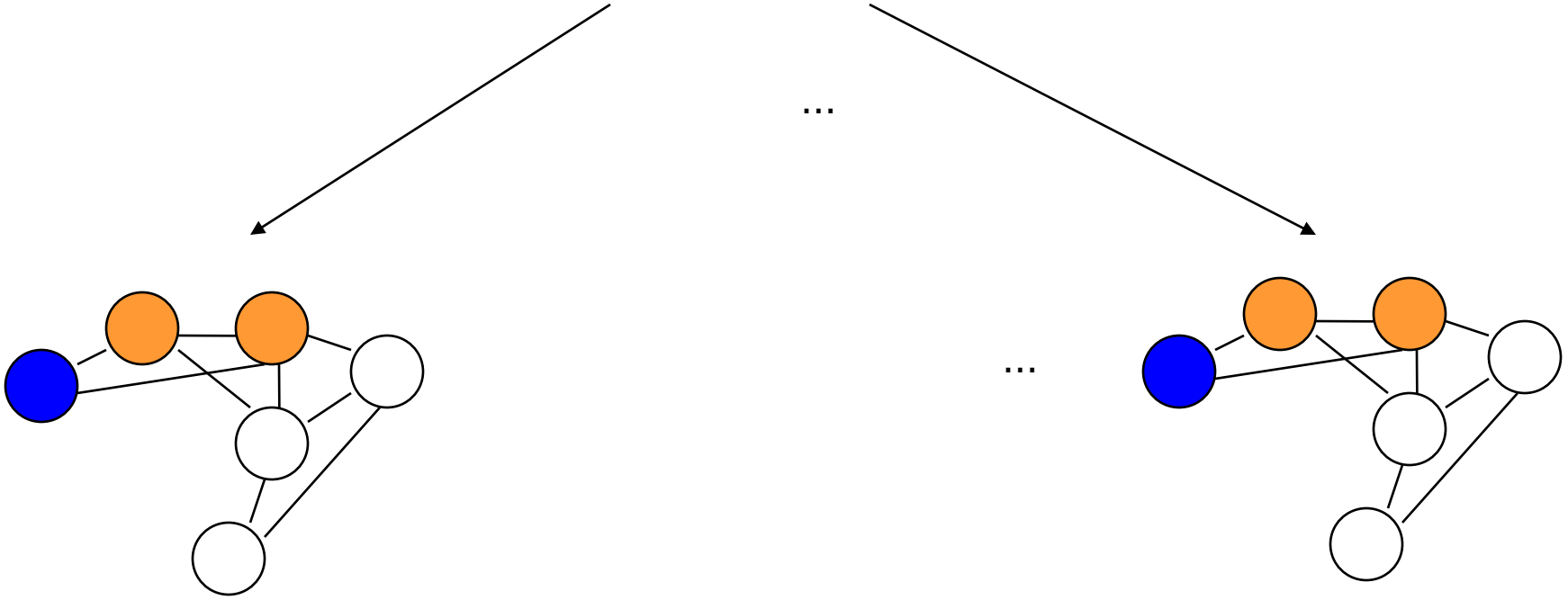
VE-BB(2): example



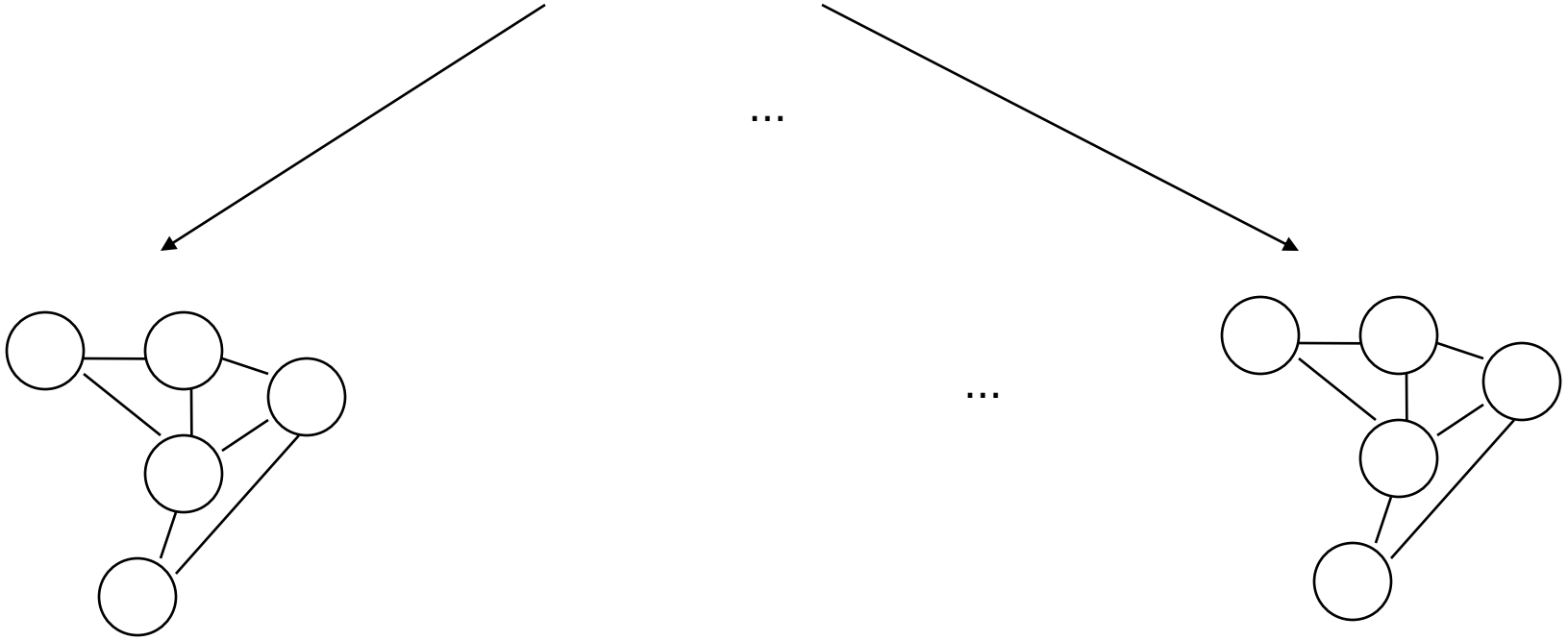
VE-BB(2): example



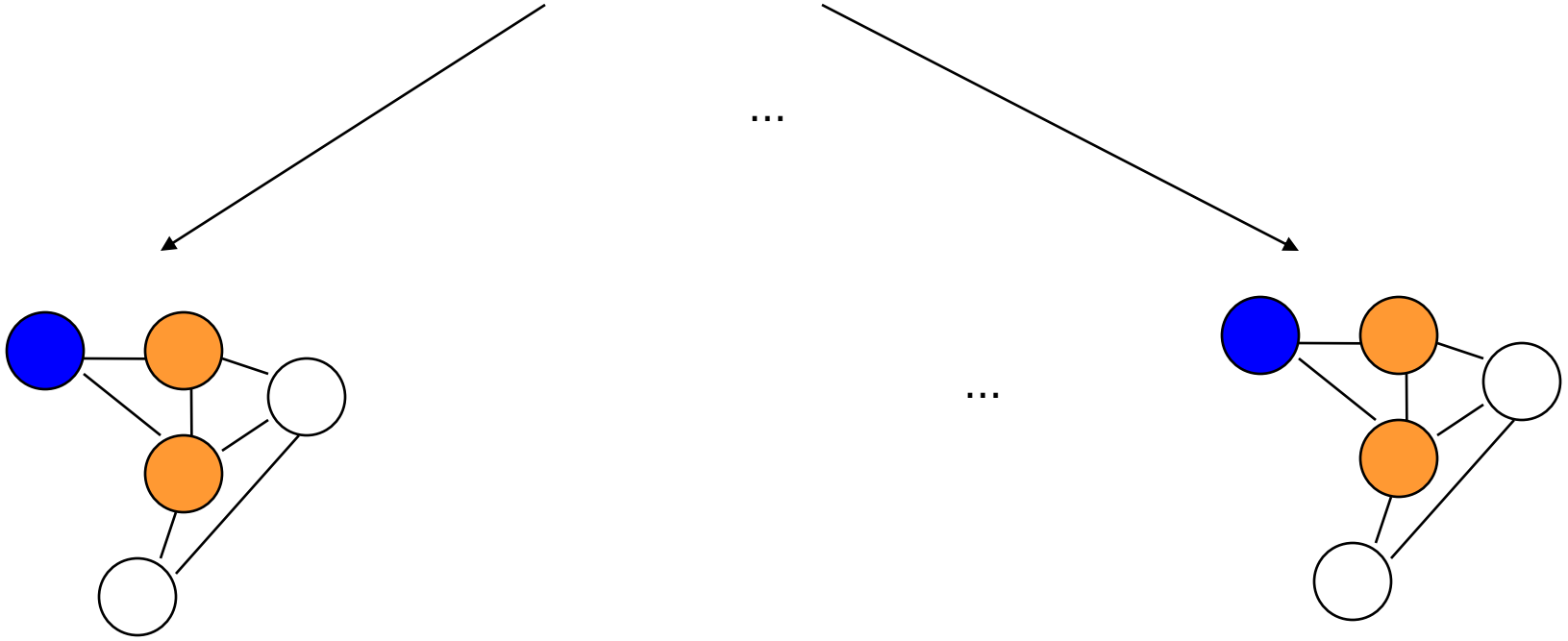
VE-BB(2): example



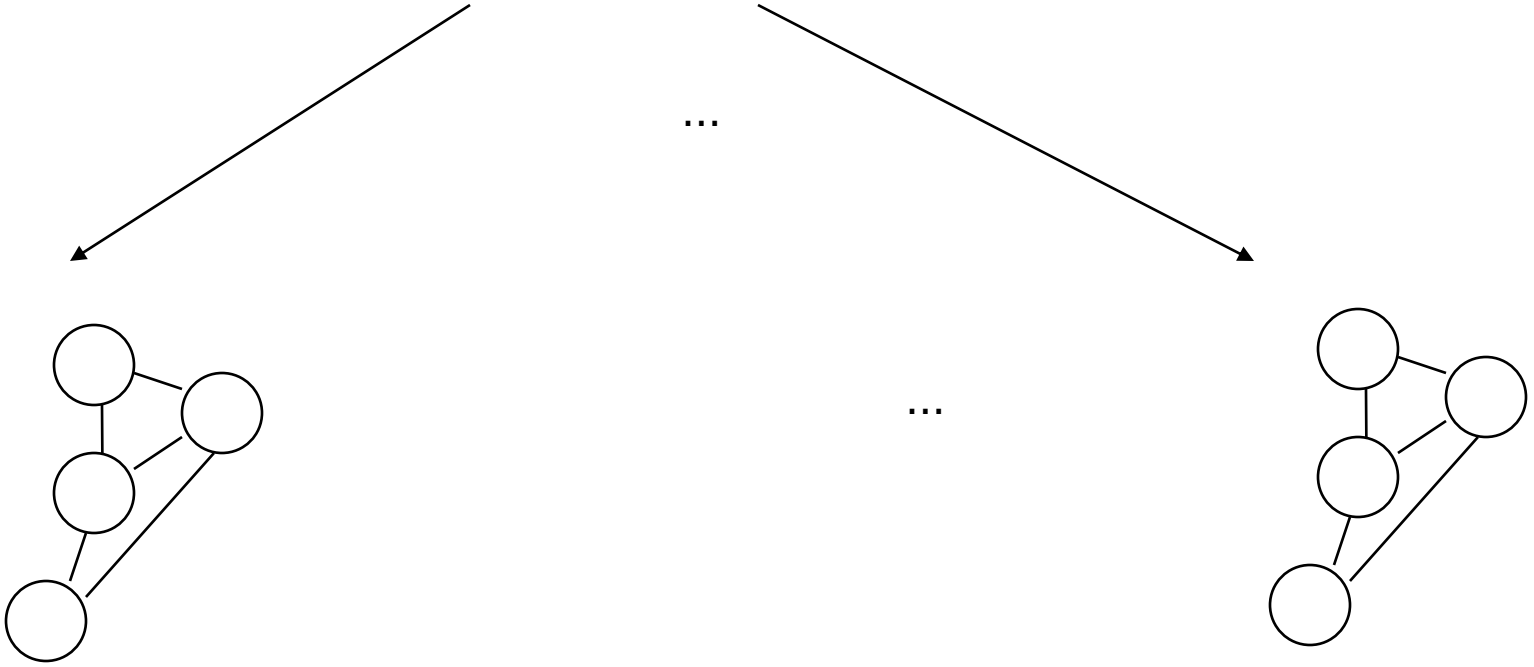
VE-BB(2): example



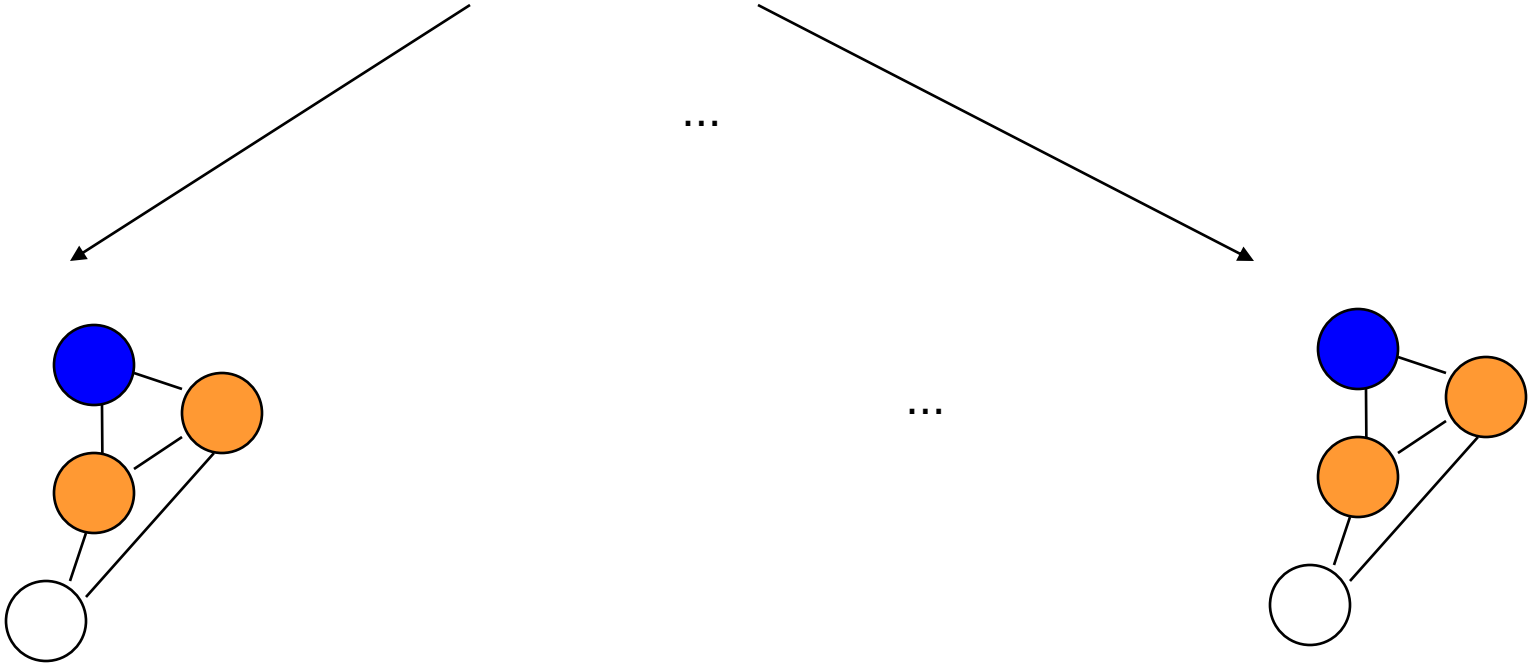
VE-BB(2): example



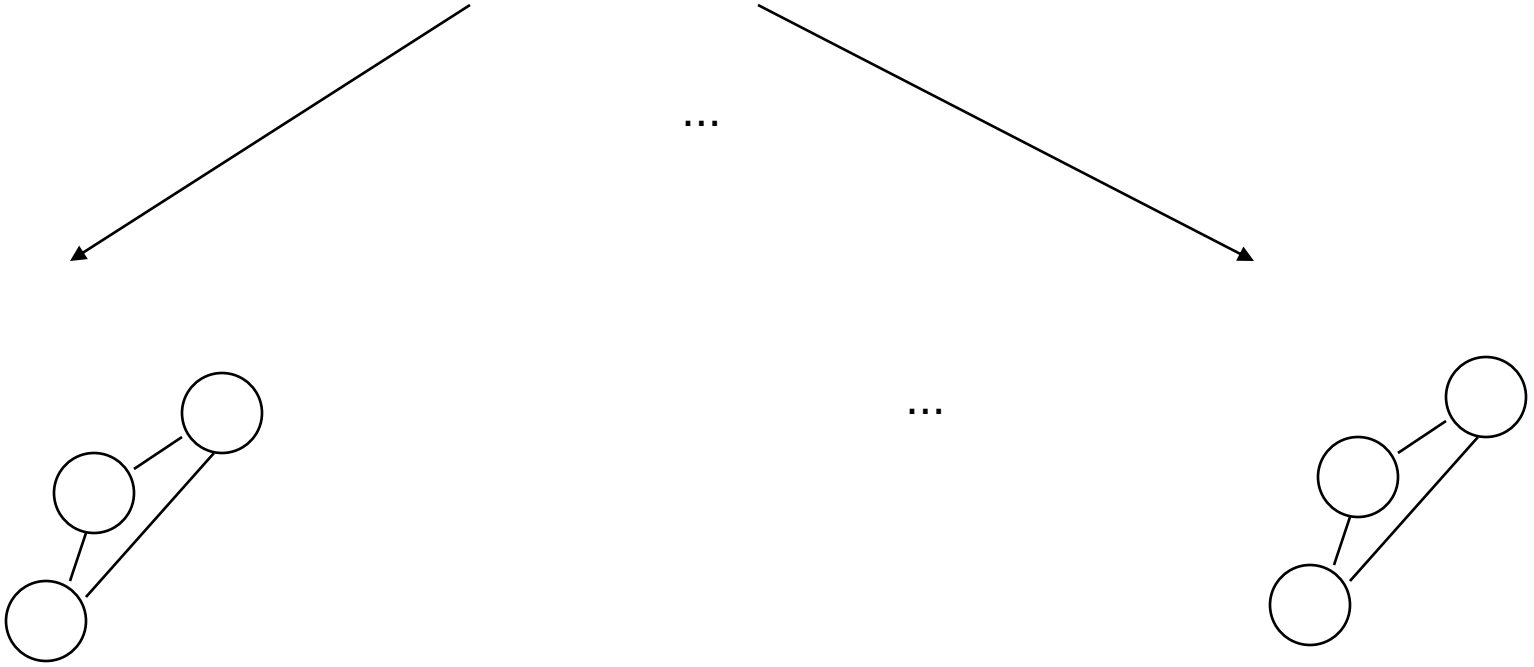
VE-BB(2): example



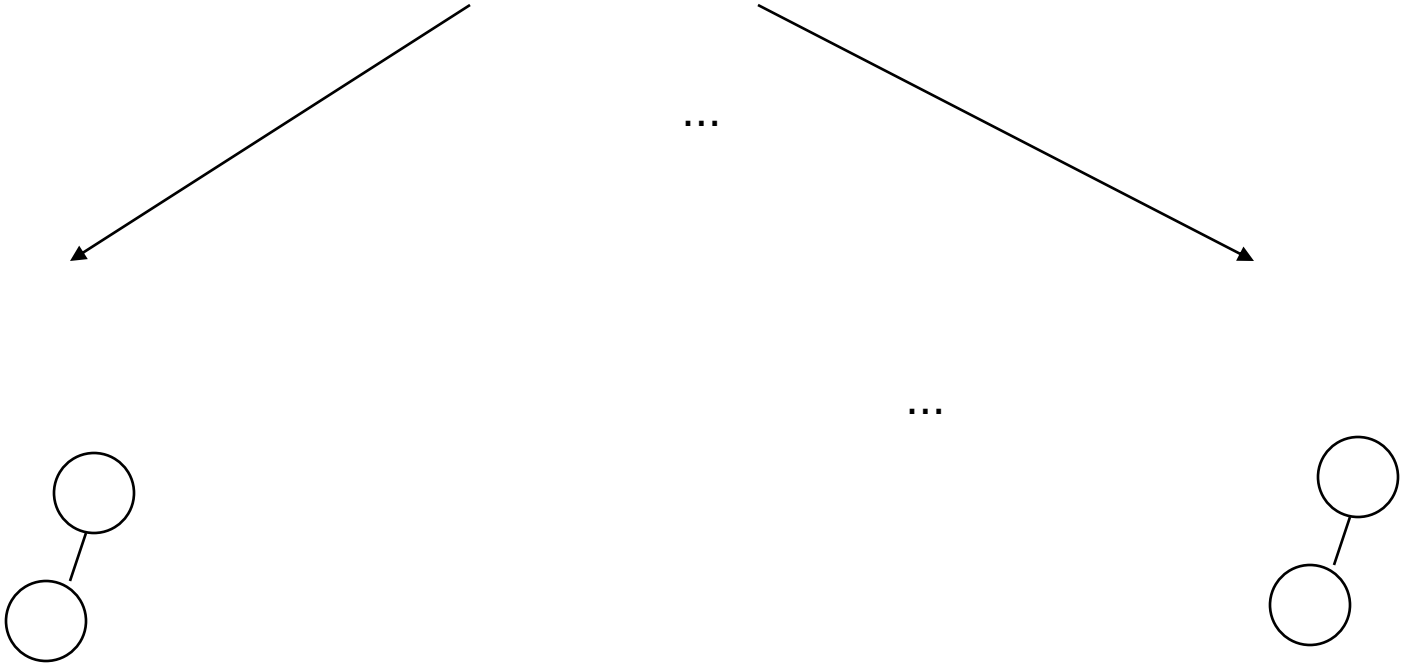
VE-BB(2): example



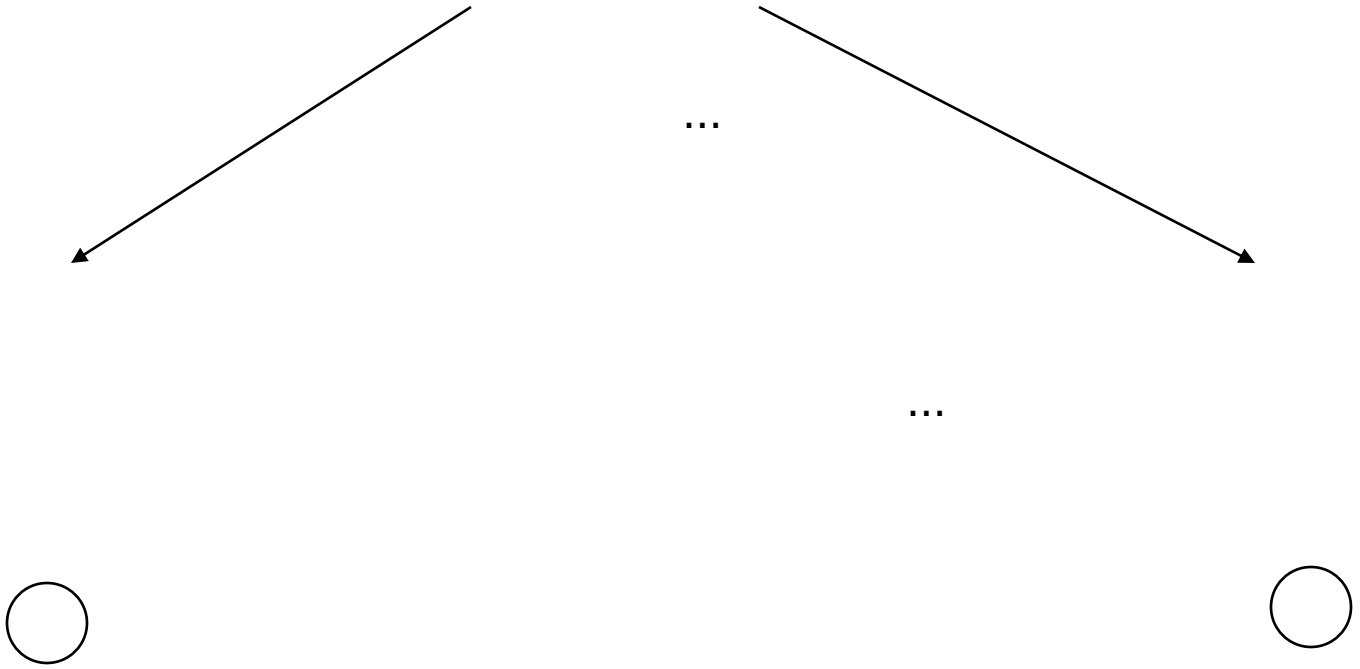
VE-BB(2): example



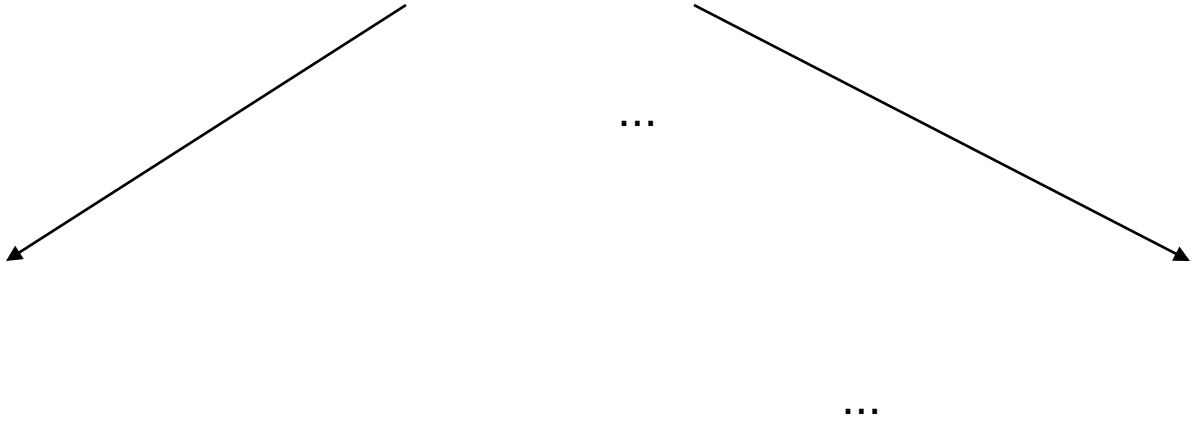
VE-BB(2): example



VE-BB(2): example



VE-BB(2): example



VE-BB

function Solve(*F*)

if Constant(*F*) **then return** *F*

x := *SelectVar*(*F*);

if deg(*x*)>*k* **return** min{Solve(*F*(***x'***)), Solve(F(***x***))};

return Solve(Elim(*x,F*);

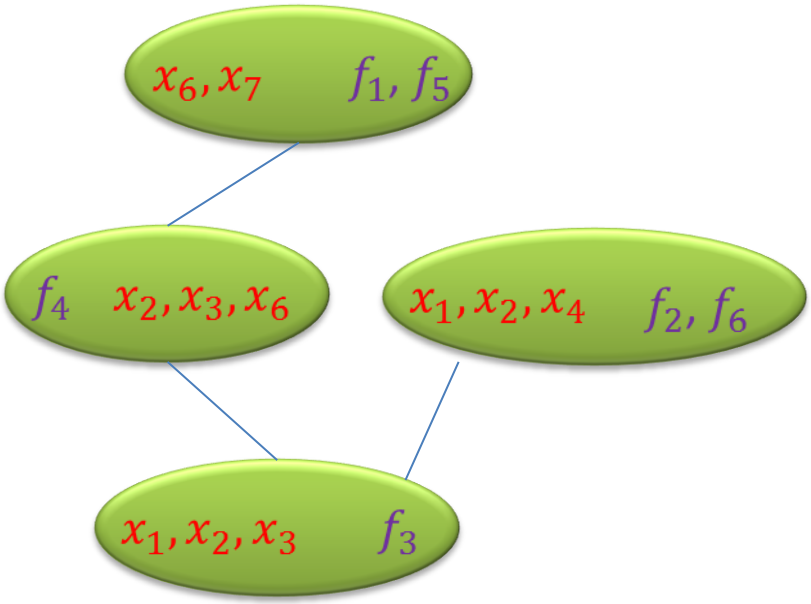
endfunction

Tree Decomposition

- Distributed/assincronous message-passing algorithm
- Idea:
  - Group factors into clusters (forming a tree decomposition)
  - Each cluster computes local information and sends it to neighbor clusters
  - After completion each cluster has received all the information that is relevant and it does not have locally

Tree Decomposition

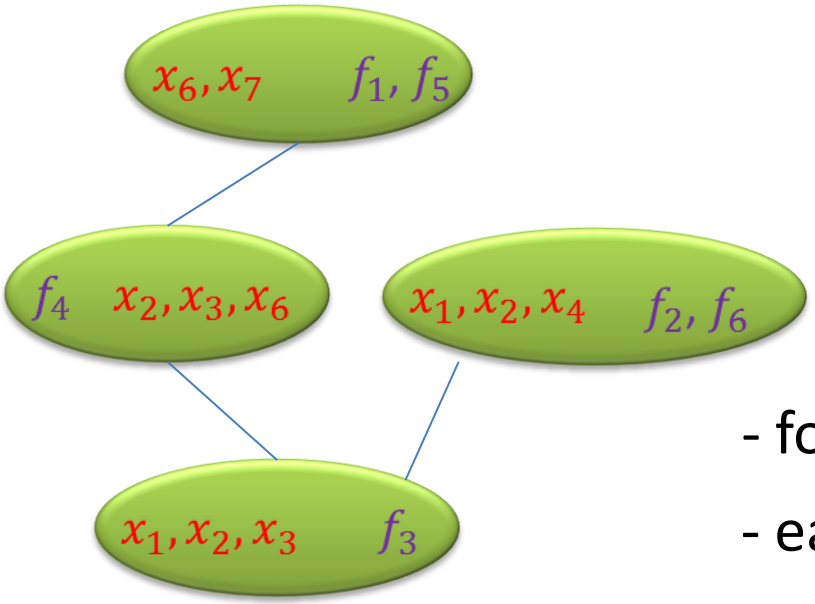
- $F(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = f_1 + f_2 + f_3 + f_4 + f_5 + f_6$



- Each node *v*, "contains" some variables ($\chi(v)$) and some factors ($\psi(v)$)

Tree Decomposition

- $F(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = f_1 + f_2 + f_3 + f_4 + f_5 + f_6$



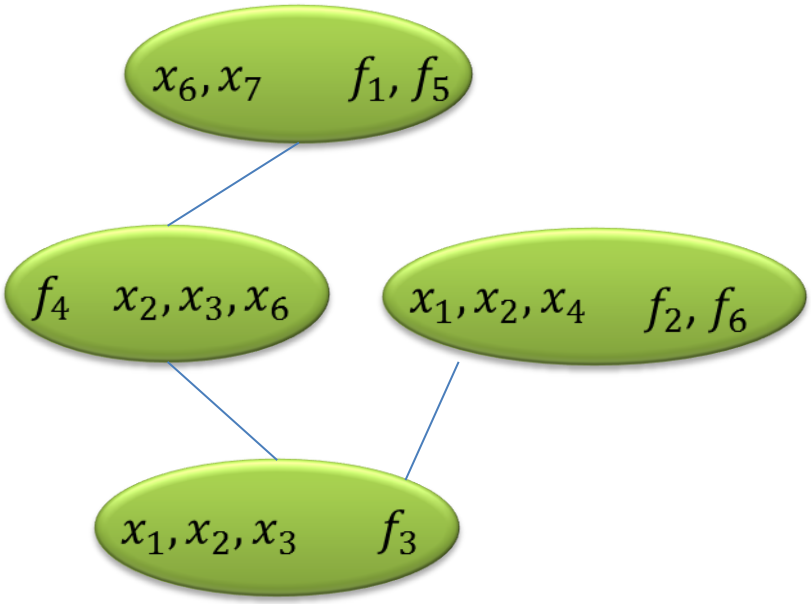
- Each node *v*, "contains" some variables ($\chi(v)$) and some factors ($\psi(v)$)

- for each node *v*, scopes of $\psi(v) \subseteq \chi(v)$

- each factor appears once in the TD

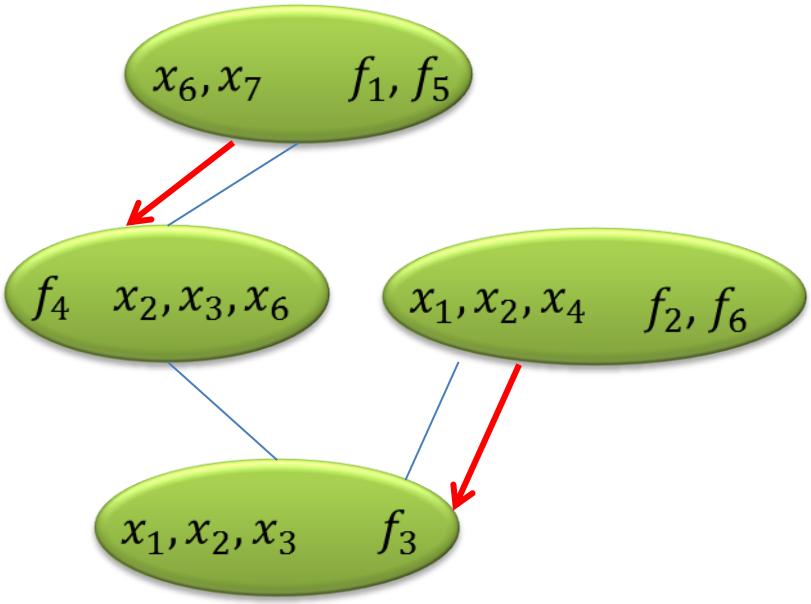
- nodes including variable *x*, form a connected subtree

Tree Decomposition



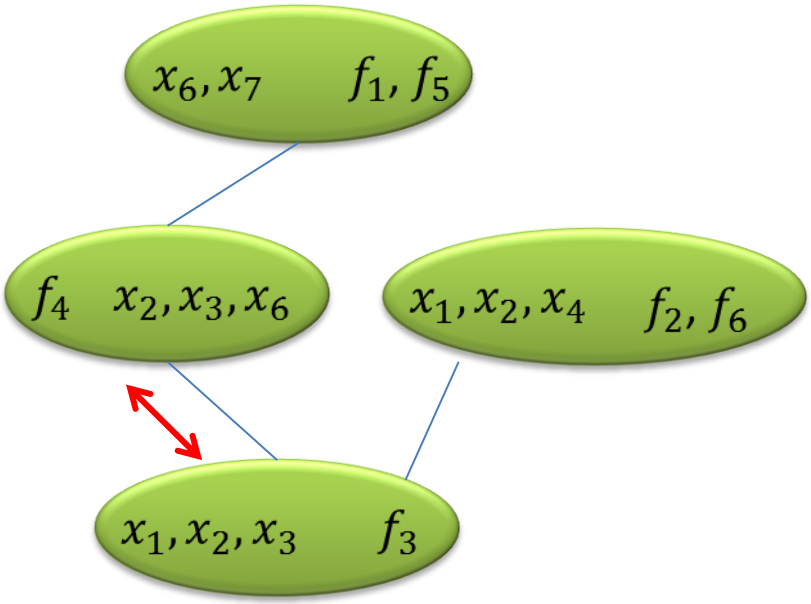
- node *v* sends a message to neighbor *w* (M_{vw}) iff it has already received messages from all other neighbors

Tree Decomposition



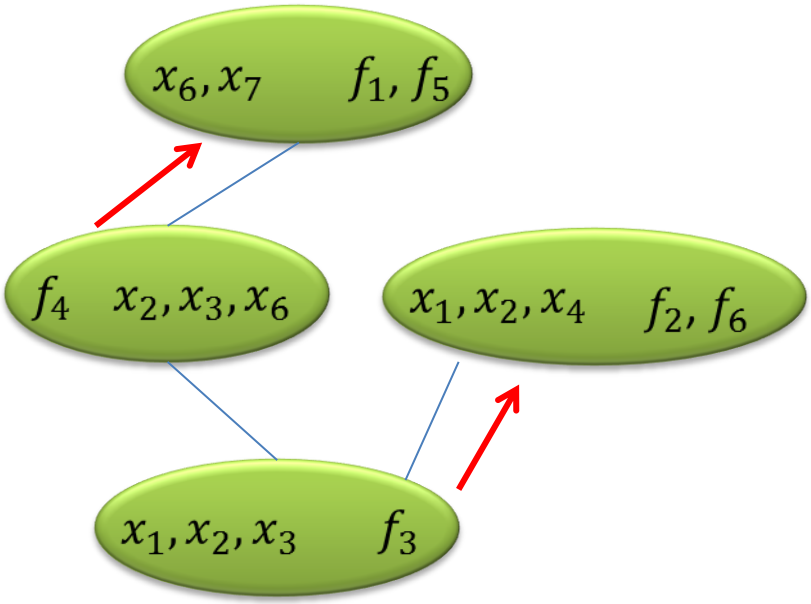
- node *v* sends a message to neighbor *w* (M_{vw}) iff it has already received messages from all other neighbors

Tree Decomposition



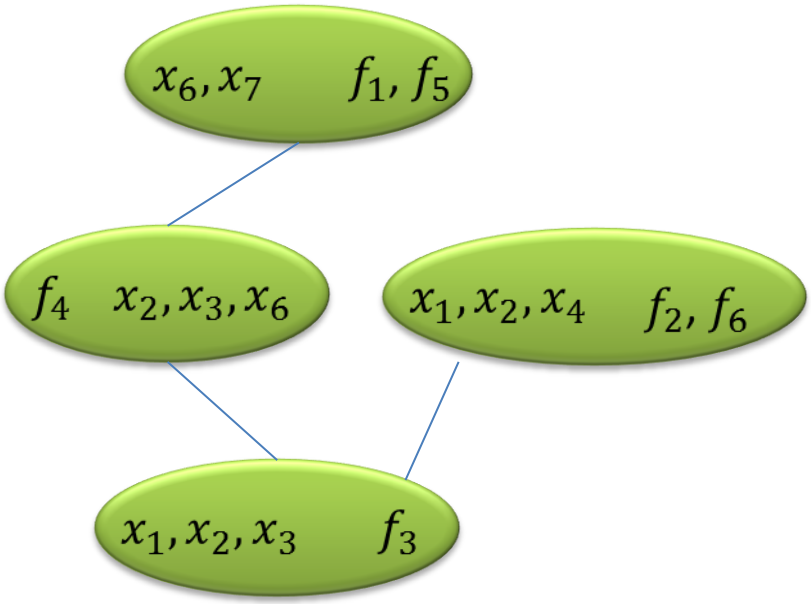
- node *v* sends a message to neighbor *w* (M_{vw}) iff it has already received messages from all other neighbors

Tree Decomposition



- node *v* sends a message to neighbor *w* (M_{vw}) iff it has already received messages from all other neighbors

Tree Decomposition



- After completion, each node has enough information to answer the query

Tree Decomposition

function Solve(*F*)

<*T*, χ , ψ>:= Tree-decomposition(F)

repeat

(v,w):= select next message

$M_{vw} := min_{\chi(v)-\chi(w)}(\Sigma_{f \in \psi(v)} f + \Sigma_{k \in ne(v),\, k \neq w} M_{kv})$

send message M_{vw}

until all messages have been sent

endfunction

Tree Decomposition

- Time complexity: exponential on the largest $\chi(v)$
  - It is a ciclycity measure
- Space complexity: exponential on the largest $\chi(\upsilon)-\chi(\omega)$
  - Largest message

Tree Decomposition vs Variable Elimination

- VE is an instantiation of Tree Decomposition
- The variable elimination order implicitely defines a Cluster Tree