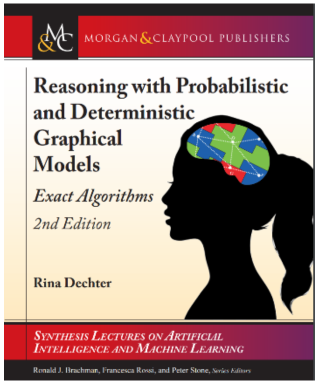
Algorithms for Reasoning with graphical models

Slides Set 9:

AND/OR search for Probabilistic Networks

Rina Dechter

(Dechter1 chapter 6 and 7)



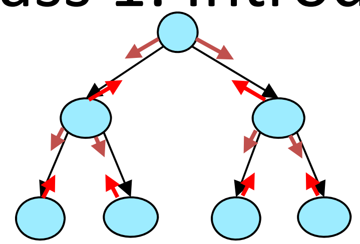
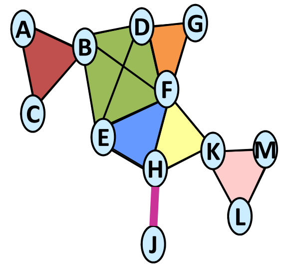
Algorithms for Reasoning with graphical models

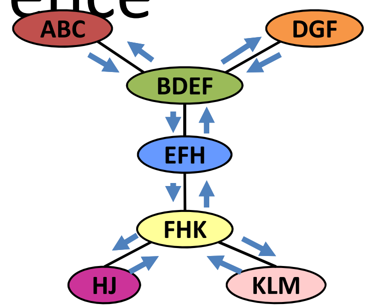
Class6: AND/OR search for Probabilistic Networks

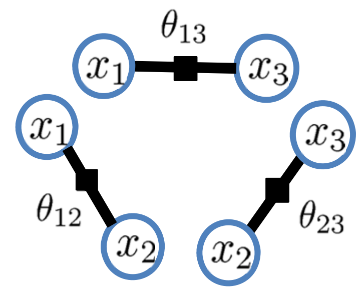
Rina Dechter

Overall Perspective

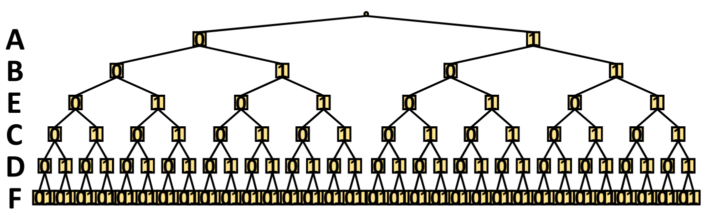
- Class 1: Introduction and Inference

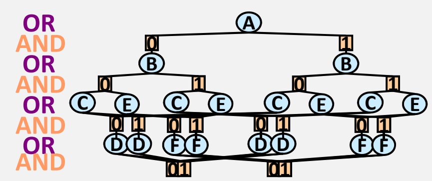







- Class 2: Search

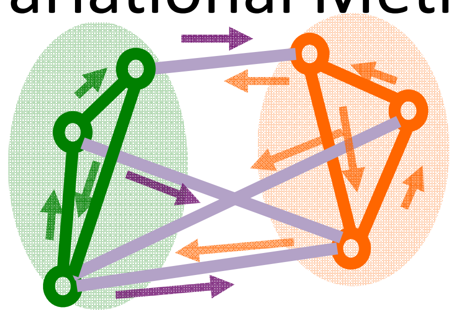
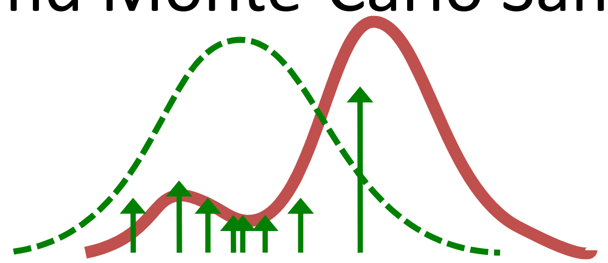




Context minimal AND/OR search graph

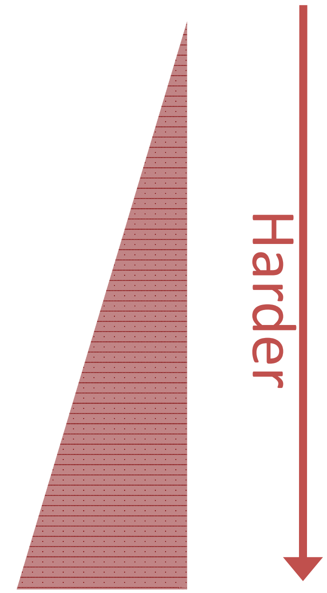
18 AND nodes

- Class 3: Variational Methods and Monte-Carlo Sampling

Types of queries

▸ Max-Inference	$f(\mathbf{x}^*) = \max_{\mathbf{x}} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$
▸ Sum-Inference	$Z = \sum_{\mathbf{x}} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$
▸ Mixed-Inference	$f(\mathbf{x}_M^*) = \max_{\mathbf{x}_M} \sum_{\mathbf{x}_S} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$

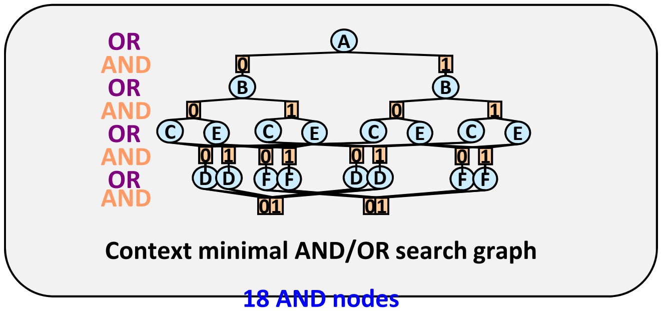


- **NP-hard**: exponentially many terms
- We will focus on **approximation** algorithms
  - **Anytime**: very fast & very approximate ! Slower & more accurate

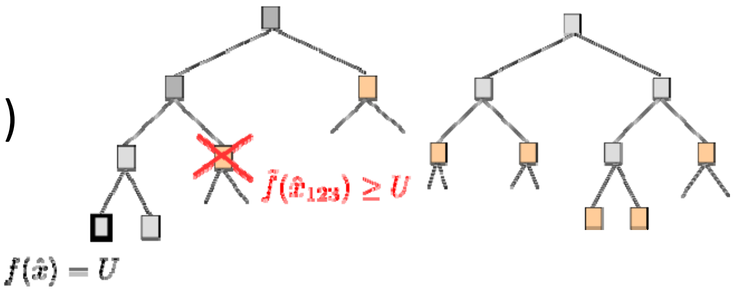
Outline: Search for Graphical Models

- Review Graphical Modes

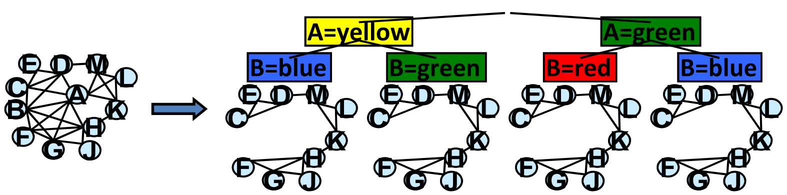
- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - Generating good pseudo-trees
  - Brute-force AND/OR



- Heuristic search (HS) for AND/OR spaces
  - Basic Heuristic search (Depth and Best)
  - AND/OR Depth-first HS (branch and bound)
  - AND/OR Best-first heuristic search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)

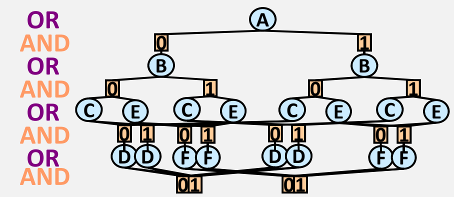


- Hybrids of search and Inference
- Summary and Class 2



Outline: Search for Graphical Models

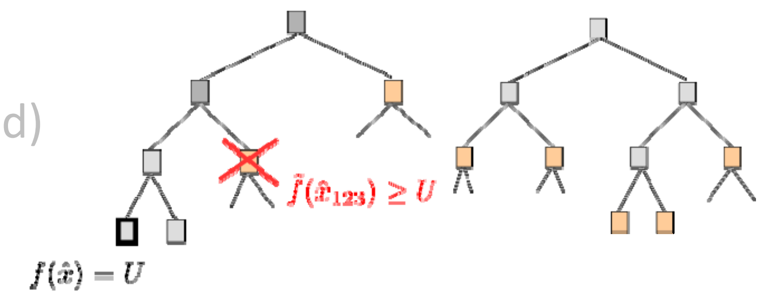
- Review Graphical Modes
- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - Generating good pseudo-trees
  - Brute-force AND/OR



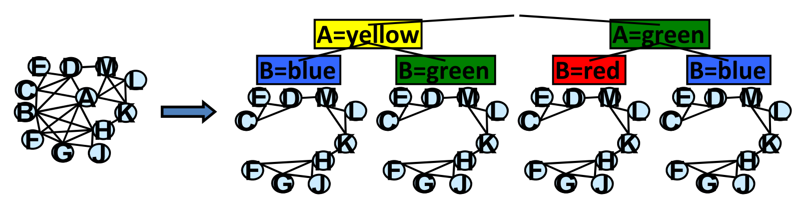
Context minimal AND/OR search graph

18 AND nodes

- Heuristic search (HS) for AND/OR spaces
  - Basic Heuristic search (Depth and Best)
  - AND/OR Depth-first HS (branch and bound)
  - AND/OR Best-first heuristic search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)



- Hybrids of search and Inference
- Summary and Class 2



Conditioning - the Probability Tree

$$P(a,e=0) = P(a)\sum_{b} P(b \mid a)\sum_{c} P(c \mid a)\sum_{b} P(d \mid a,b)\sum_{e=0} P(e \mid b,c)$$

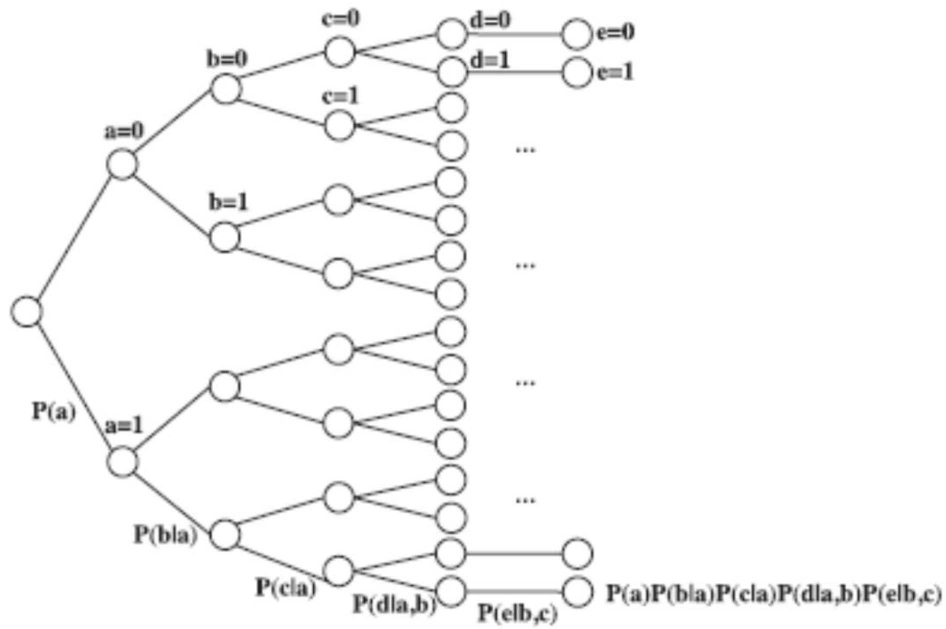
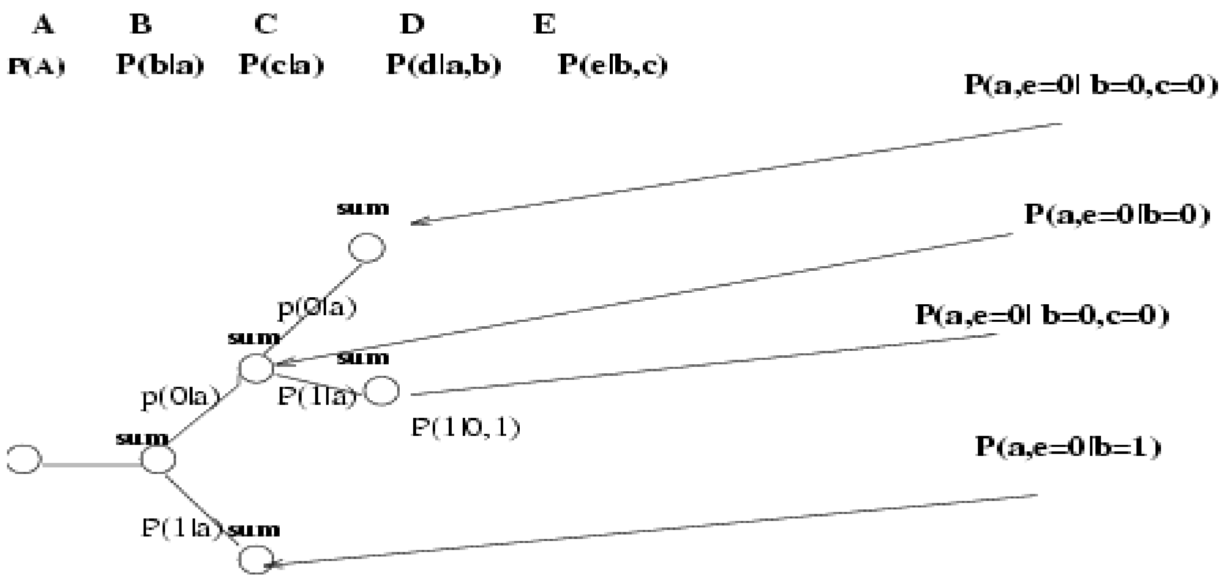


Figure 6.1: Probability tree for computing P(d=1,g=0).

Complexity of conditioning: exponential time, linear space

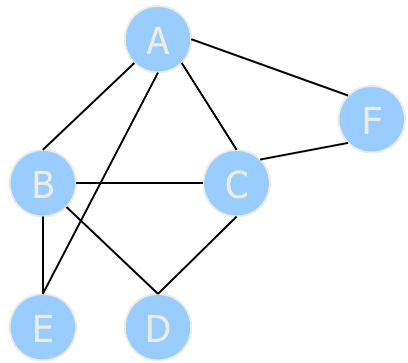
Conditioning+Elimination

$$P(a,e=0) = P(a)\sum_b P(b \mid a)\sum_c P(c \mid a)\sum_d P(d \mid a,b)\sum_{e=0} P(e \mid b,c)$$

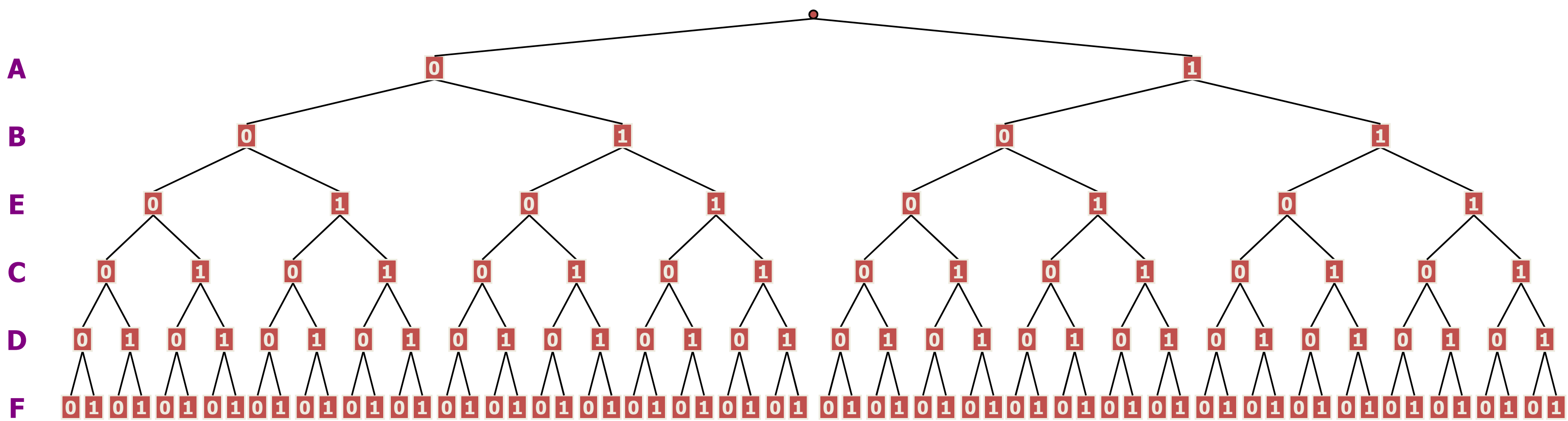


Idea: conditioning until w^* of a (sub)problem gets small

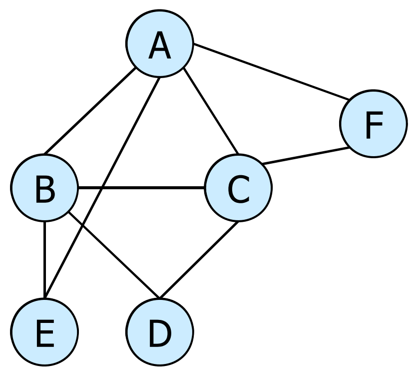
The Classic OR Search Space



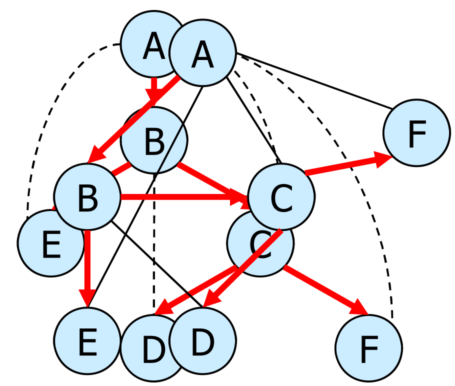
Ordering: A B E C D F



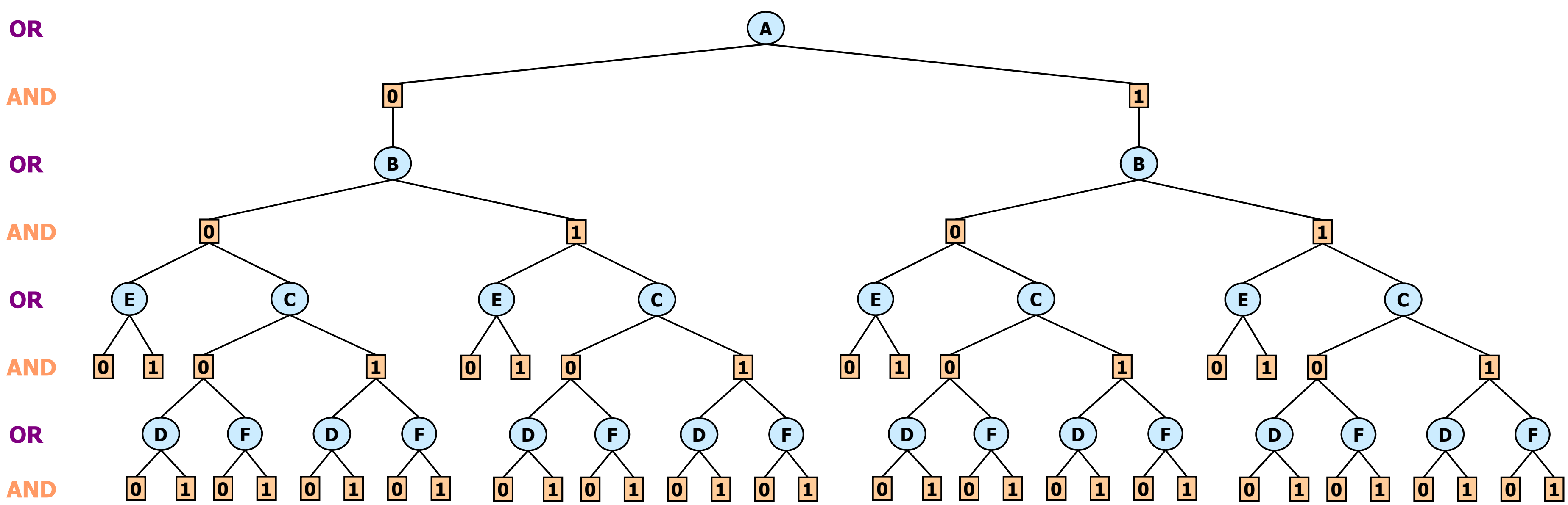
AND/OR Search Space



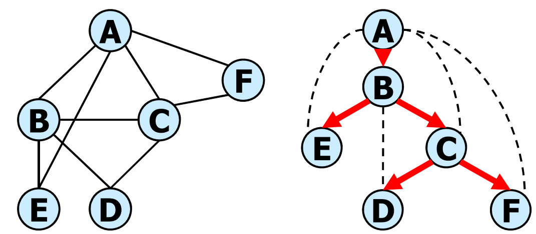
Primal graph

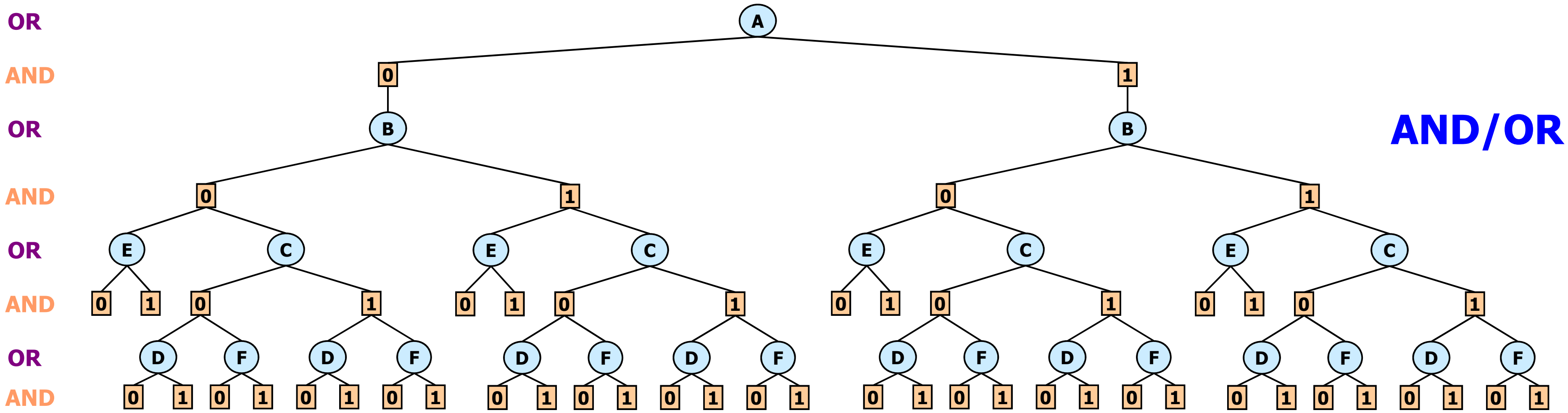


DFS tree

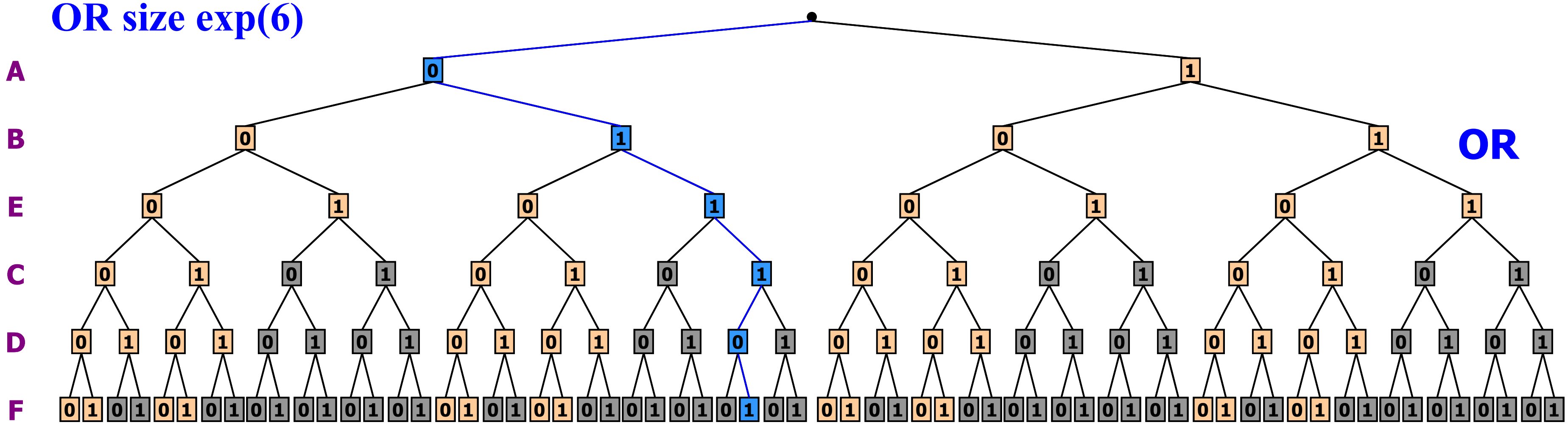


AND/OR vs. OR

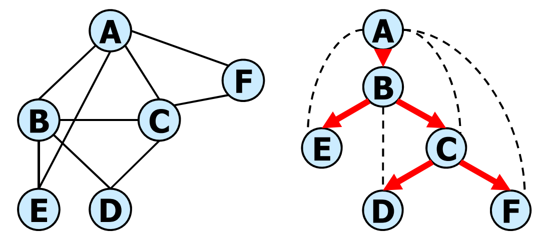


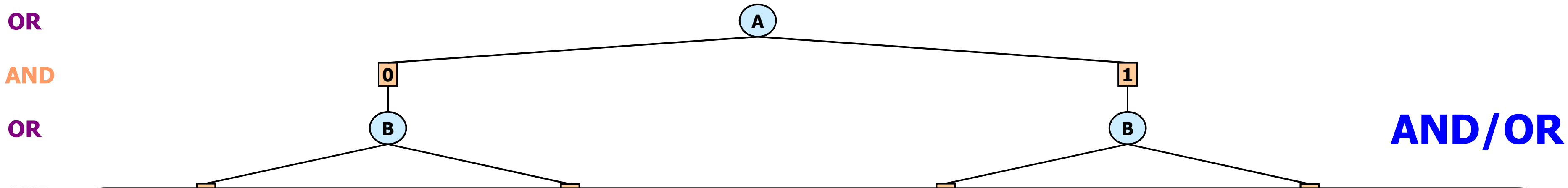


AND/OR size: exp(4),
OR size exp(6)

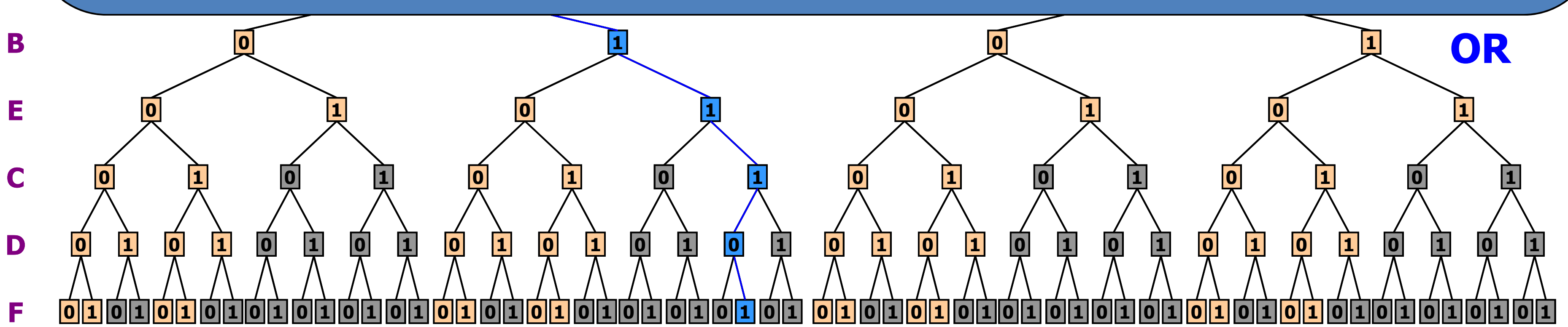


AND/OR vs. OR



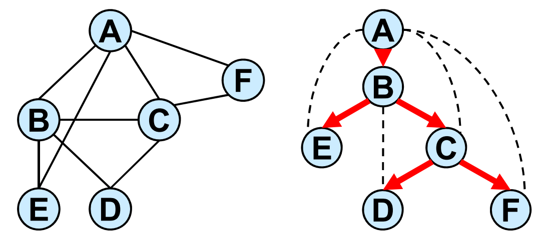


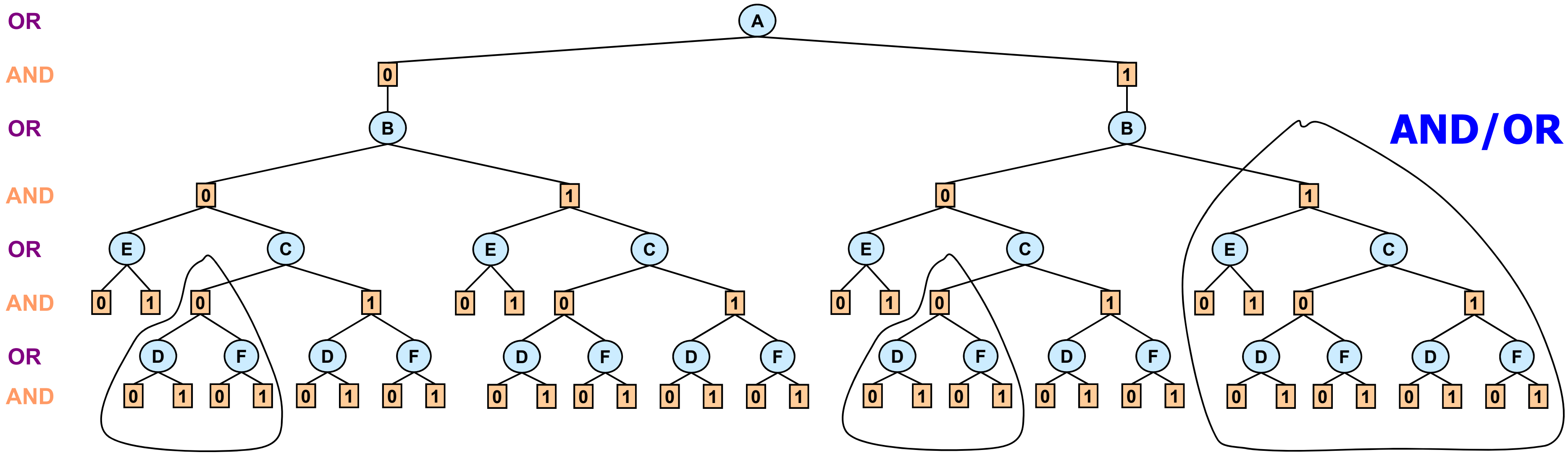
- *Size of tree O(nk^h)*
- *Can be traversed in*
  - *Time O(nk^h)*, Space O(n)
- All solution trees = all configurations

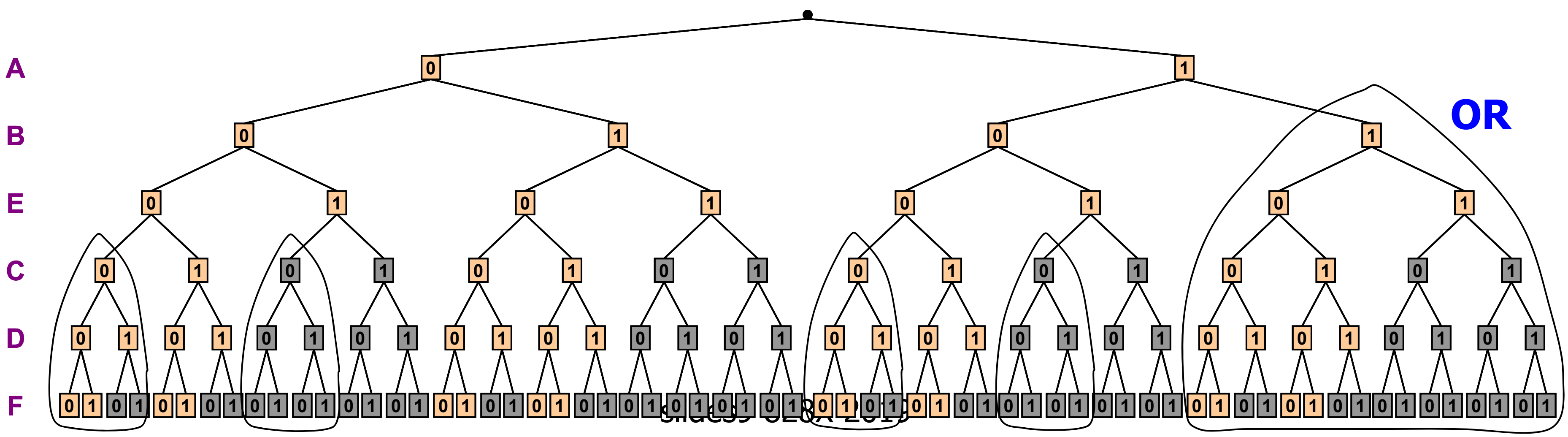


AND/OR vs. OR

No-goods
(A=1,B=1)
(B=0,C=0)

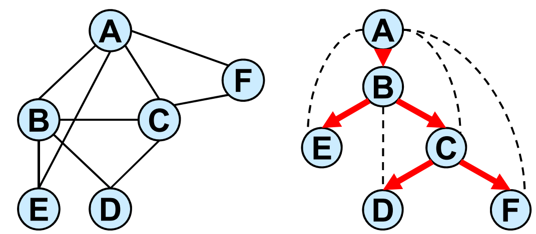


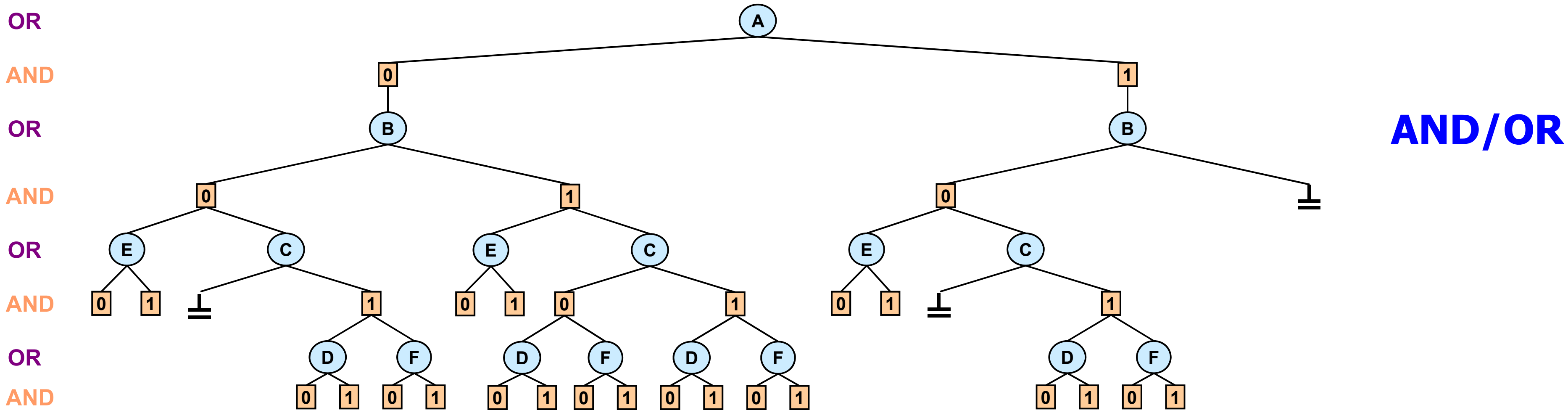


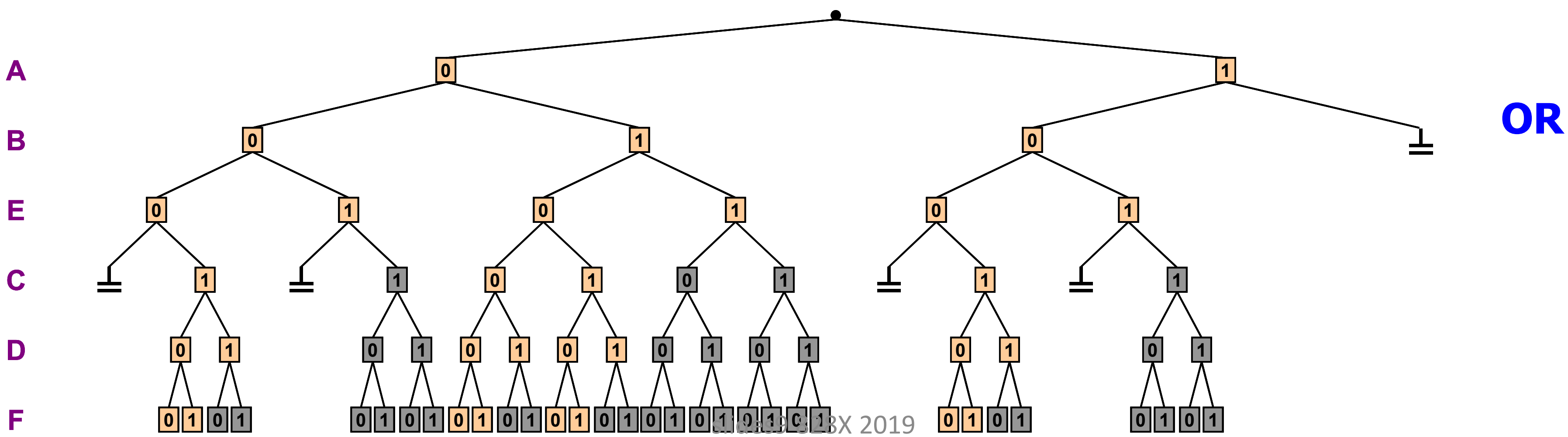


AND/OR vs. OR

(A=1,B=1)
(B=0,C=0)







Arc weights
Cost of a solution tree
The value function

Arc Weights for AND/OR Trees

$P(E \mid A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

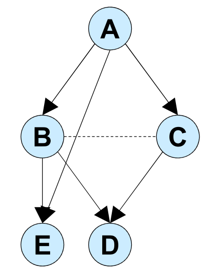
A	B=0	B=1
0	.4	.6
1	.1	.9

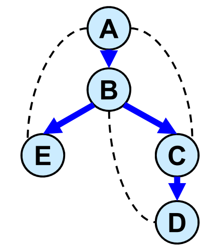
$P(C \mid A)$

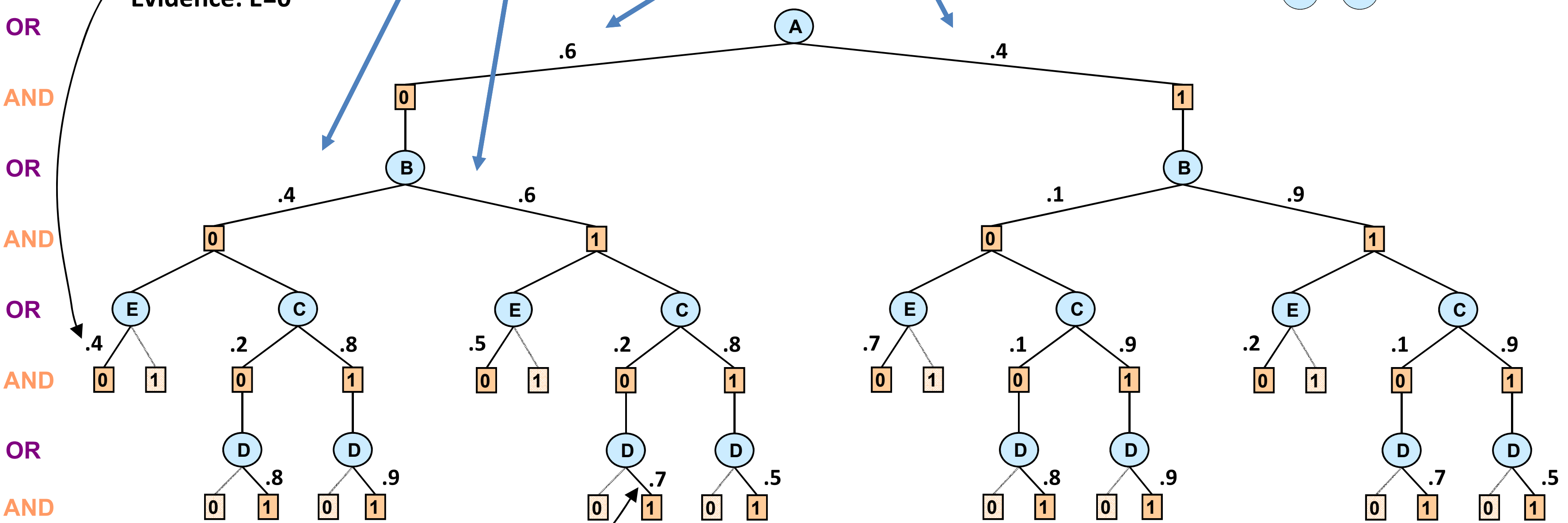
A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4







$P(D \mid B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1

OR to AND arc weight <X,x> is the product of factors that all their arguments are just assigned at AND node X=x but not before

Cost of a Solution Tree

$P(E \mid A, B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

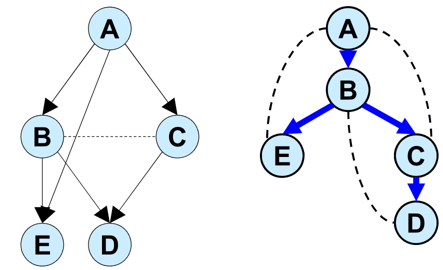
A	B=0	B=1
0	.4	.6
1	.1	.9

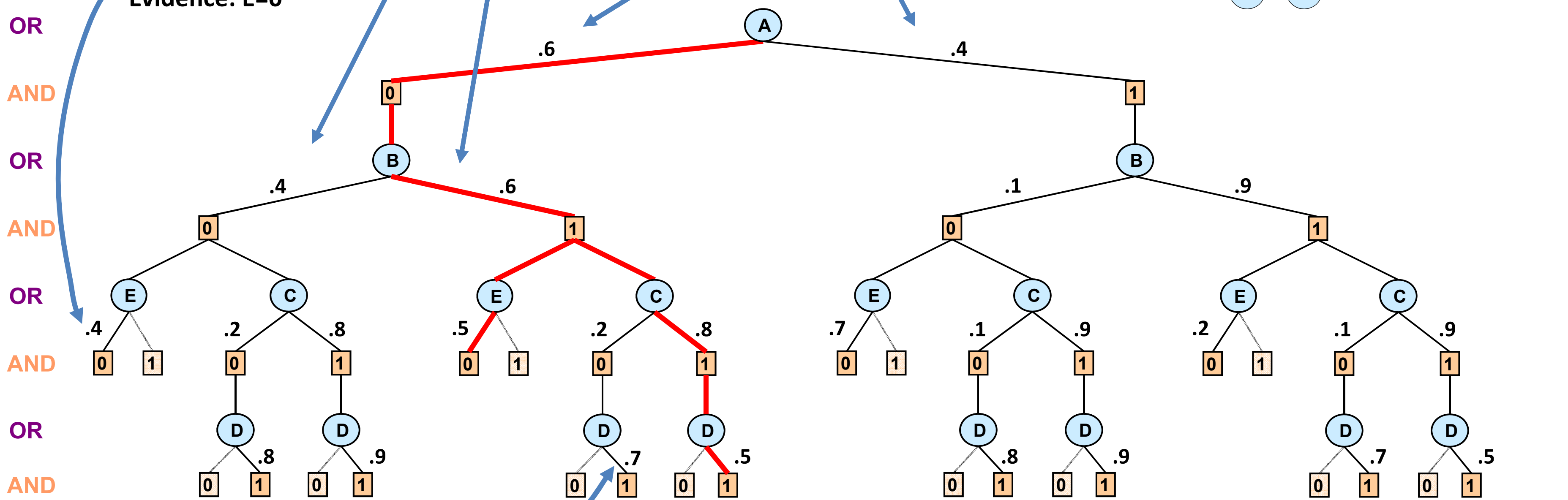
$P(C \mid A)$

A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4





$P(D \mid B, C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1

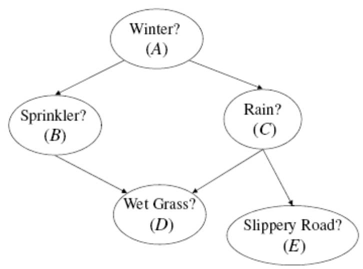
A solution tree includes the root and has a single child for any OR node, and all children of any of its AND nodes

Cost of the solution tree: the product of weights on its arcs

Cost of (A=0,B=1,C=1,D=1,E=0) = 0.6 · 0.6 · 0.5 · 0.8 · 0.5 = 0.0720

Arc Weights for AND/OR Trees

A Bayesian Network



A	Θ_A
true	.6
false	.4

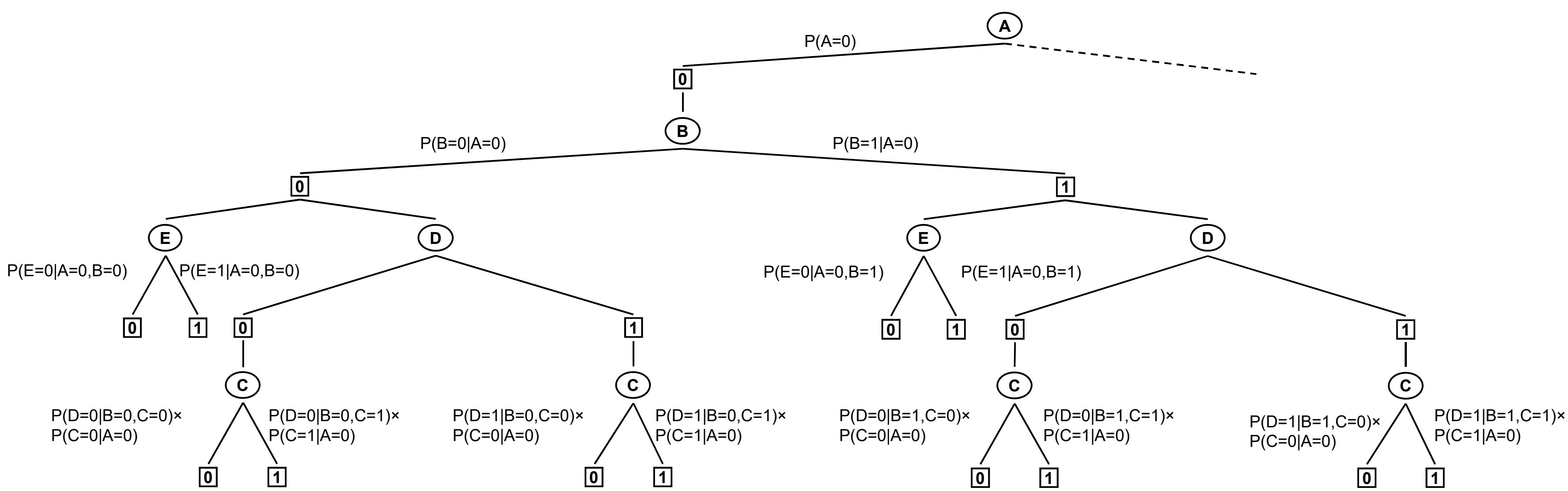
A	B	$\Theta_{B\|A}$
true	true	.2
true	false	.8
false	true	.75
false	false	.25

A	C	$\Theta_{C\|A}$
true	true	.8
true	false	.2
false	true	.1
false	false	.9

B	C	D	$\Theta_{D\|BC}$
true	true	true	.95
true	true	false	.05
true	false	true	.9
true	false	false	.1
false	true	true	.8
false	true	false	.2
false	false	true	0
false	false	false	1

C	E	$\Theta_{E\|C}$
true	true	.7
true	false	.3
false	true	0
false	false	1

OR to AND arc weight (X,x) is the product of factors f that are instantiated at the AND node X=x but not before



The Value Function for (Probability of Evidence)

$P(E \mid A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

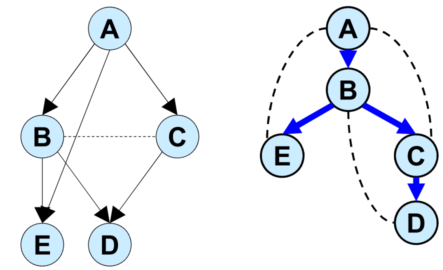
A	B=0	B=1
0	.4	.6
1	.1	.9

$P(C \mid A)$

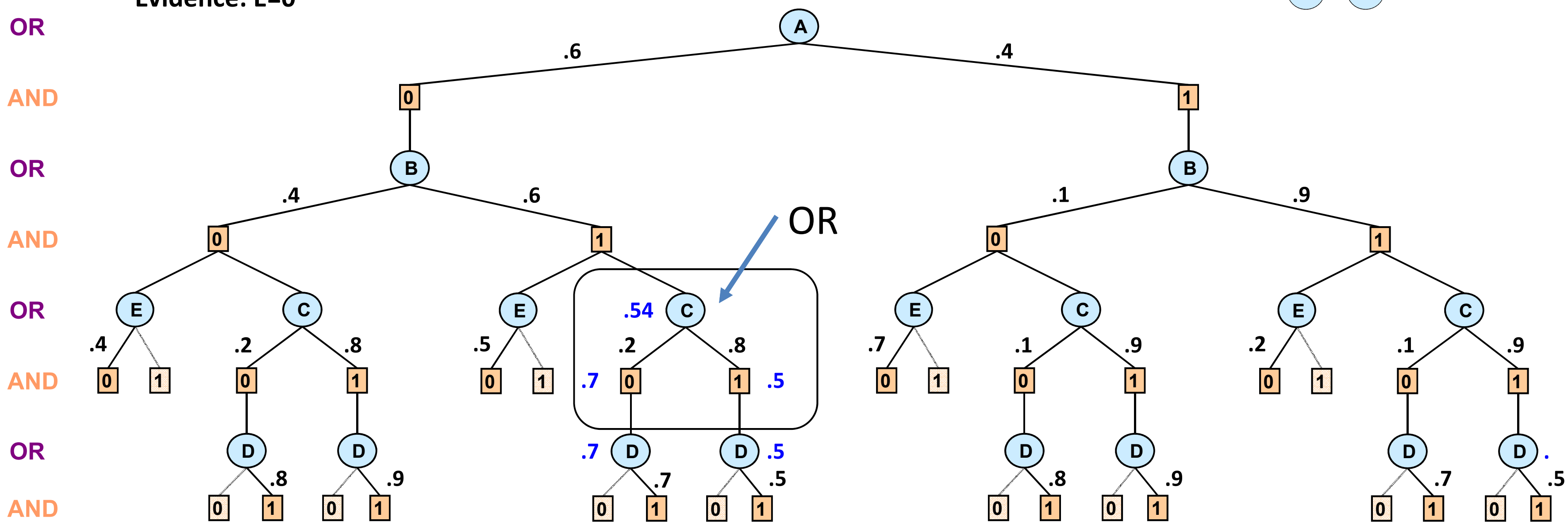
A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4



P(D=1,E=0)=?



$P(D \mid B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1,E=0

Value of node = updated belief for sub-problem below

AND node: product $\prod_{n' \in children(n)} v(n')$

OR node: Marginalization by summation $\sum_{n' \in children(n)} w(n,n')\, v(n')$

The Value Function (Probability of Evidence)

$P(E \mid A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

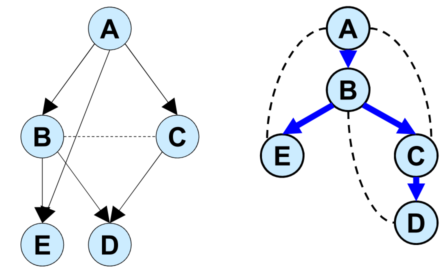
A	B=0	B=1
0	.4	.6
1	.1	.9

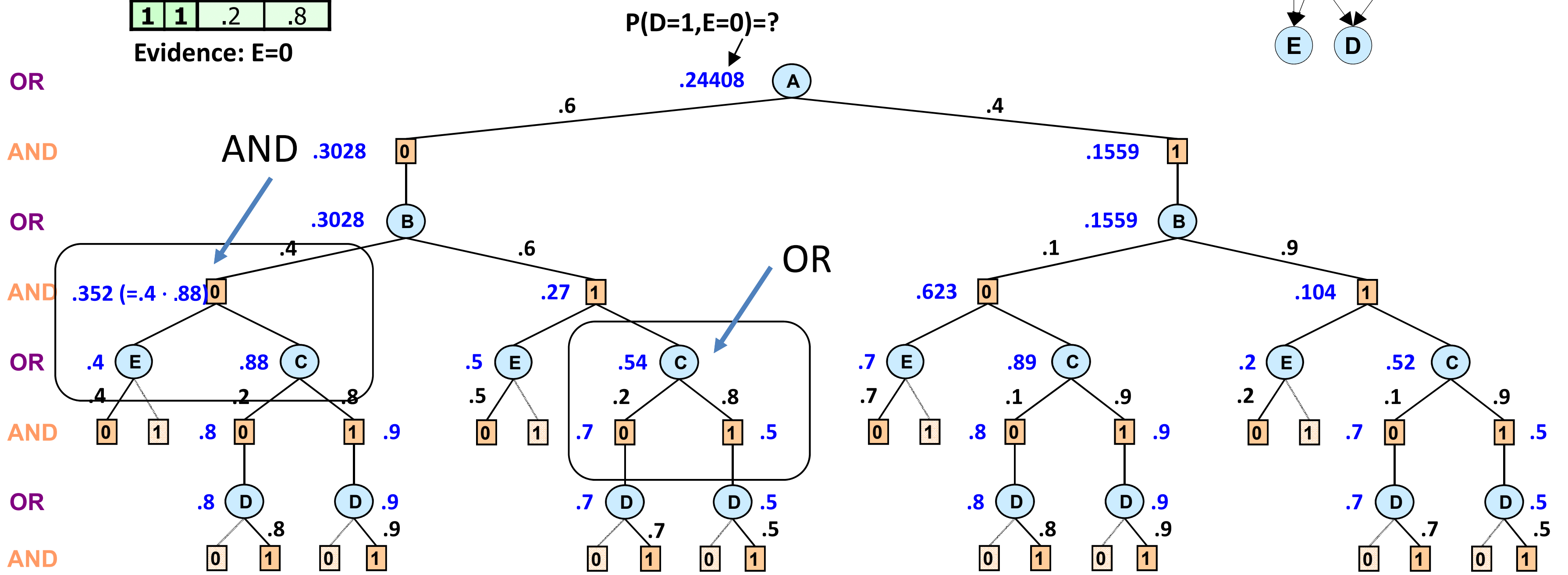
$P(C \mid A)$

A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4





$P(D \mid B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1

Value of node = updated belief for sub-problem below

AND node: product $\prod_{n' \in children(n)} v(n')$

OR node: Marginalization by summation $\sum_{n' \in children(n)} w(n,n')\, v(n')$

The Value Function

$P(E \mid A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

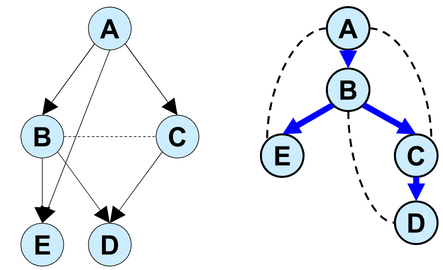
A	B=0	B=1
0	.4	.6
1	.1	.9

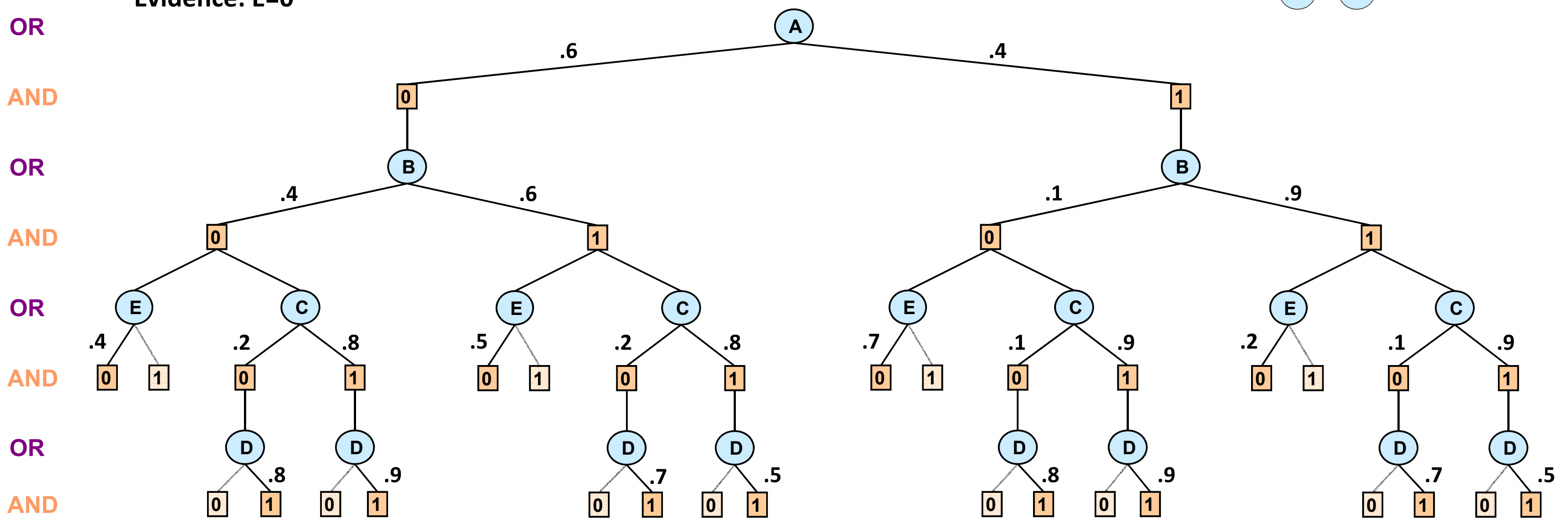
$P(C \mid A)$

A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4





$P(D \mid B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1,E=0

- V(n) is dictated by the query of interest
- V(n) the value of the sub-problem represented by T(n)
- For sum-inference it is the probability mess below n
- Can be computed recursively based on child values.

The Value Function

$P(E \mid A, B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

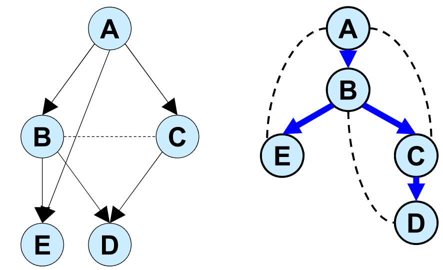
A	B=0	B=1
0	.4	.6
1	.1	.9

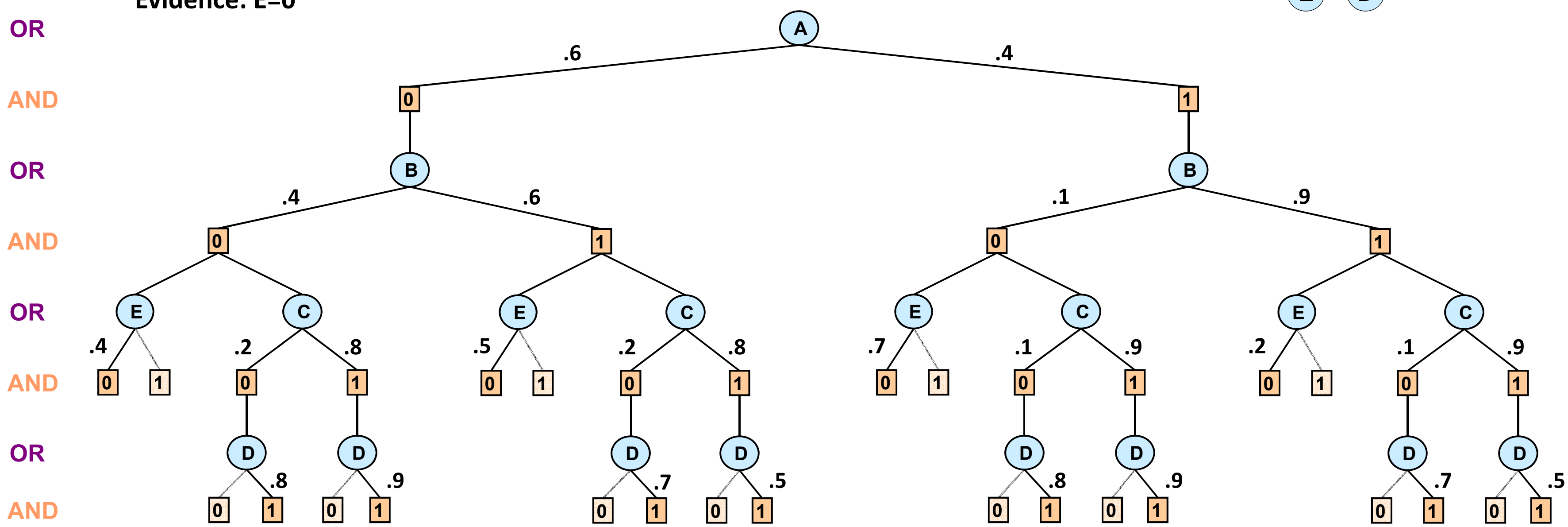
$P(C \mid A)$

A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4





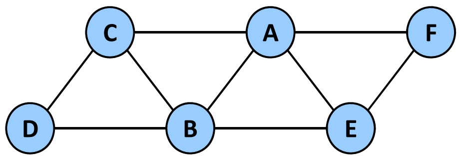
$P(D \mid B, C)$

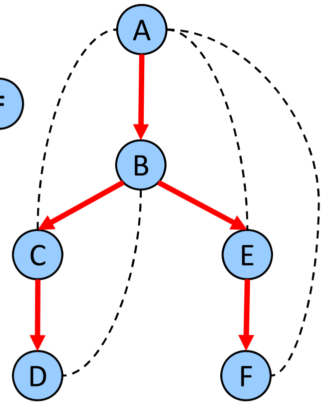
B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1

- V(n) is dictated by the query of interest
- V(n) the value of the sub-problem represented by T(n)
- For sum-inference it is the probability mess below n
- Can be computed recursively based on child values.

The Value Function for Optimization





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

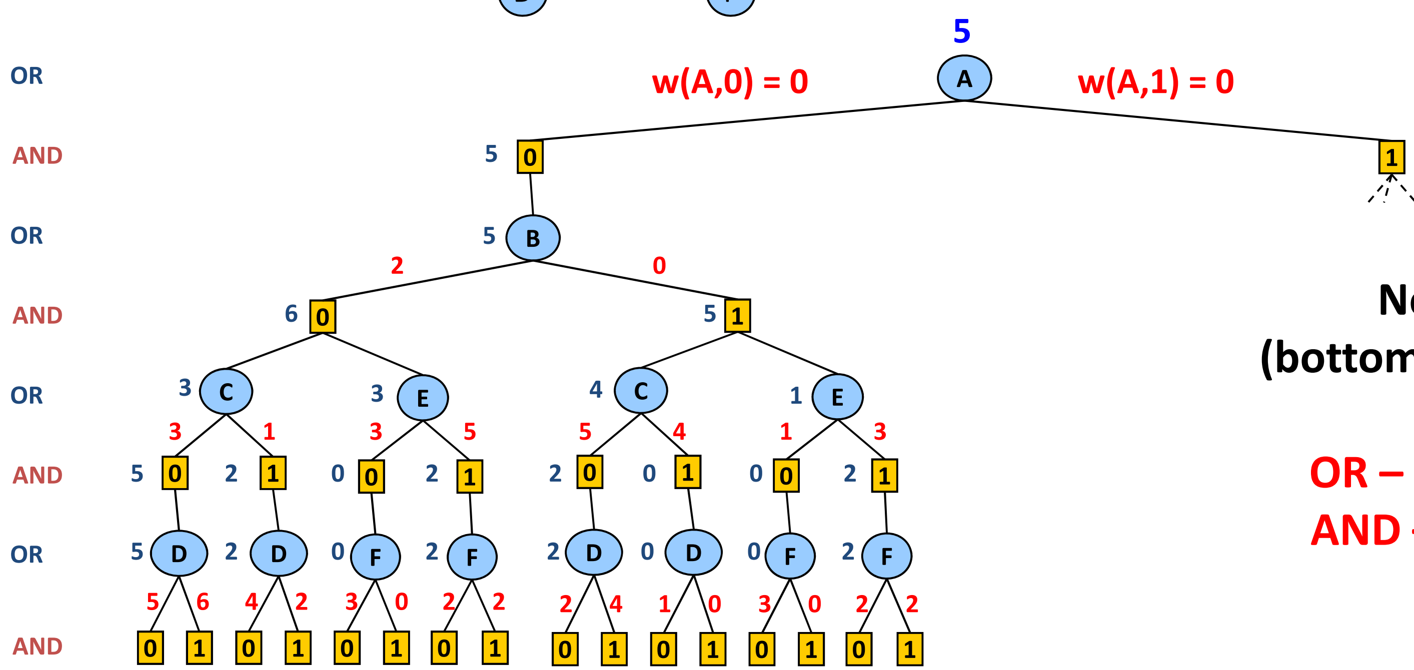
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

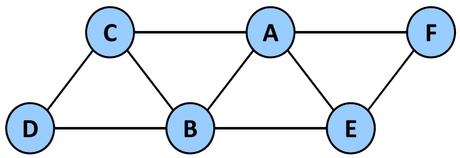
Objective function: $F^* = \min_x \sum_\alpha f_\alpha(x_\alpha)$

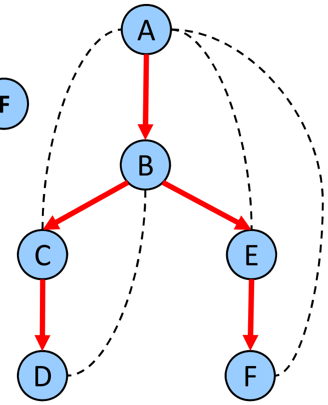


Node Value
(bottom-up evaluation)

OR – minimization
AND – summation

The Value Function for Optimization





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

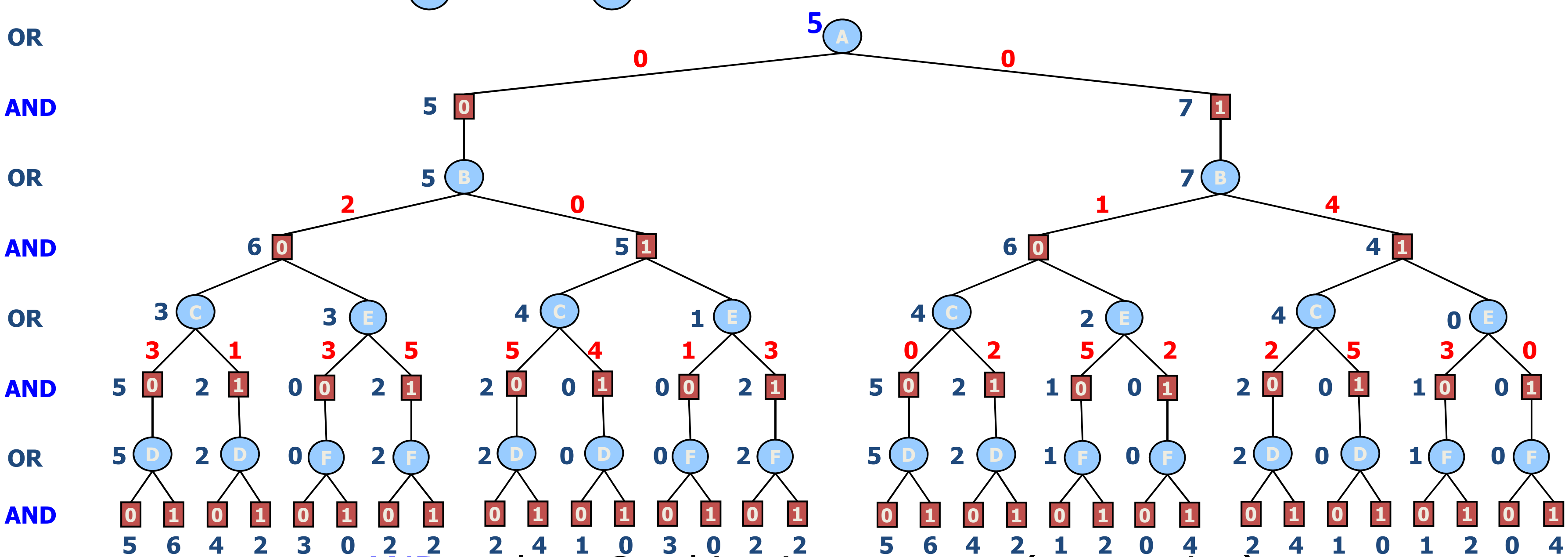
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

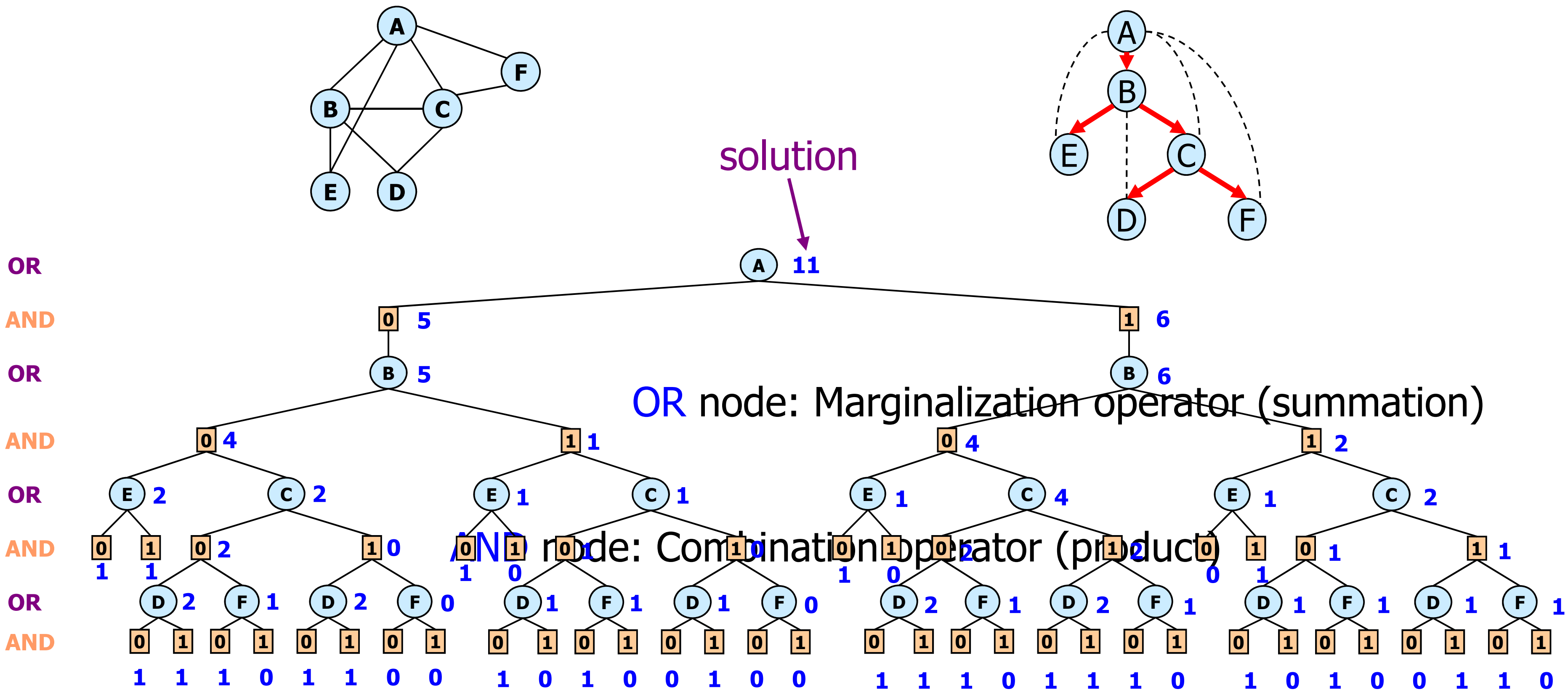
Objective function: $F^* = \min_x \sum_\alpha f_\alpha(x_\alpha)$



AND node = Combination operator (summation)

OR node = Marginalization operator (minimization)

The AND/OR Counting Value(#CSP)



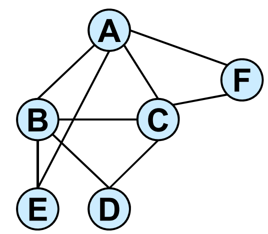
OR node: Marginalization operator (summation)

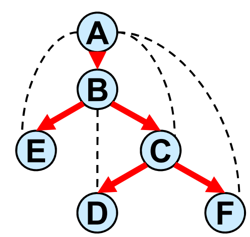
AND node: Combination operator (product)

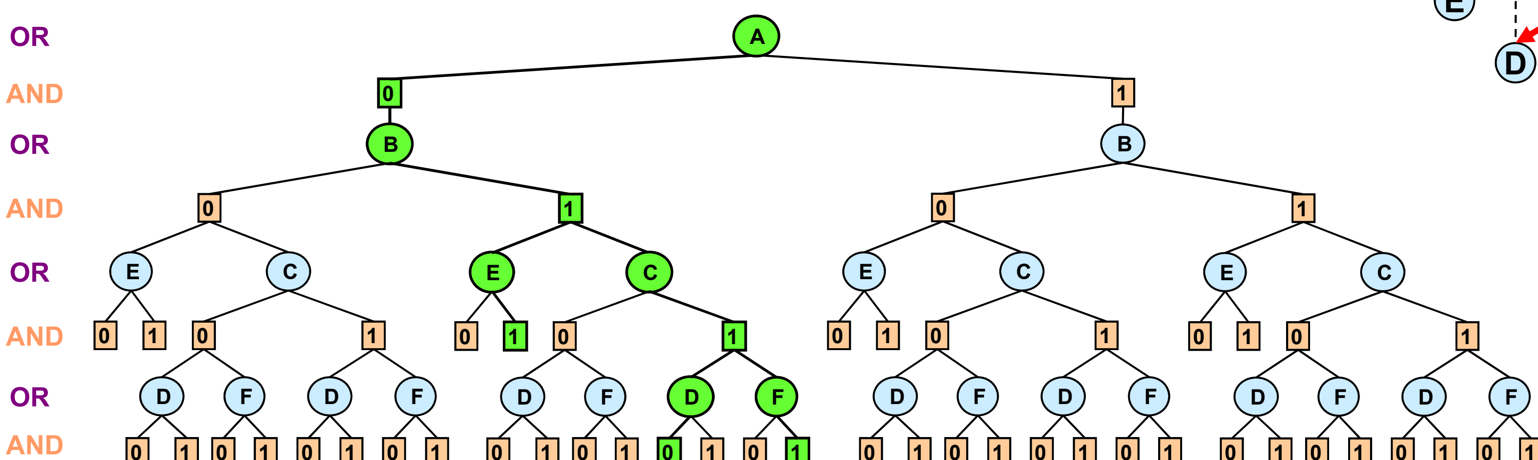
Value of node = number of solutions below it

Summary: AND/OR Search Tree for GMs

- **The AND/OR search tree of R relative to a pseudo-tree, T, has:**
  - **Alternating levels of: OR nodes (variables) and AND nodes (values)**
- **Successor function:**
  - **The successors of OR nodes X are all its consistent values along its path**
  - **The successors of AND <X,v> are all X child variables in T**
  - **Arc-weight are assigned from the model factors**
- **A solution is a consistent subtree. Its cost, the product of the weights.**
- **Query: compute the value of the root node**







Size and Traversal of AND/OR Search Tree

	AND/OR tree	**OR tree**
Space	O(n)	O(n)
Size= Time	$O(n\ k^h)$ $O(n\ k^{w^* \log n})$ (Freuder & Quinn85), (Collin, Dechter & Katz91), (Bayardo & Miranker95), (Darwiche01)	$O(k^n)$

k = domain size
h= height of pseudo-tree
n = number of variables
w*= treewidth

$h \leq w^* \log n$

AND/OR vs. OR Spaces

width	height	OR space		AND/OR space		
		Time (sec.)	Nodes	Time (sec.)	AND nodes	OR nodes
5	10	3.15	2,097,150	0.03	**10,494**	5,247
4	9	3.13	2,097,150	0.01	**5,102**	2,551
5	10	3.12	2,097,150	0.03	**8,926**	4,463
4	10	3.12	2,097,150	0.02	**7,806**	3,903
5	13	3.11	2,097,150	0.10	**36,510**	18,255

Random graphs with 20 nodes, 20 edges and 2 values per node

Pseudo-trees

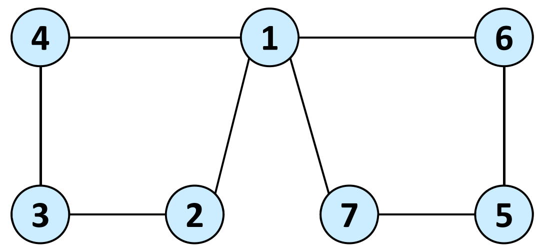
PseudoTree Definition

Given undirected graph G = (V, E), a directed rooted tree T = (V, E′) defined on all its nodes is a pseudo tree if any arc of G which is not included in E′ is a back-arc in T , namely it connects a node in T to an ancestor in T . The arcs in E′ may not all be included in E.

Given a pseudo tree T of G, the extended graph of G relative to T includes also the arcs in E′ that are not in E: as GT = (V, E ∪ E′).

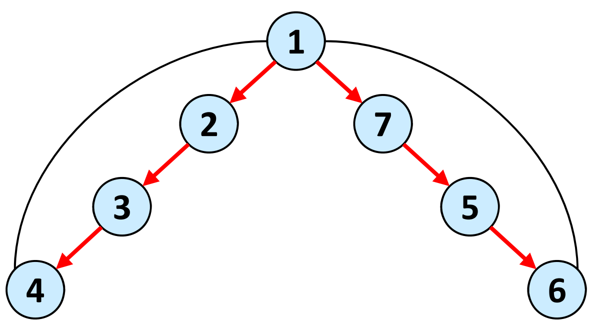
Pseudo Trees

A psedo-tree of a graph is a tree spanning its nodes, where all arcs in the graph not in the tree are back-arcs

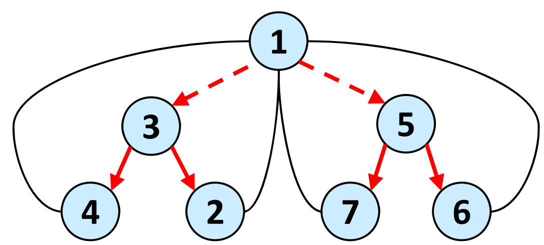


(a) Graph

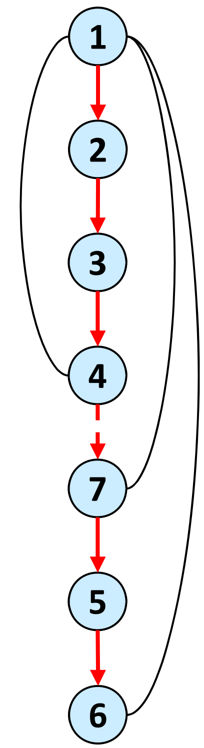
$$h \leq w^* \log n$$



(b) DFS tree
height=3



(c) Pseudo tree
height=2



(d) Chain
height=6

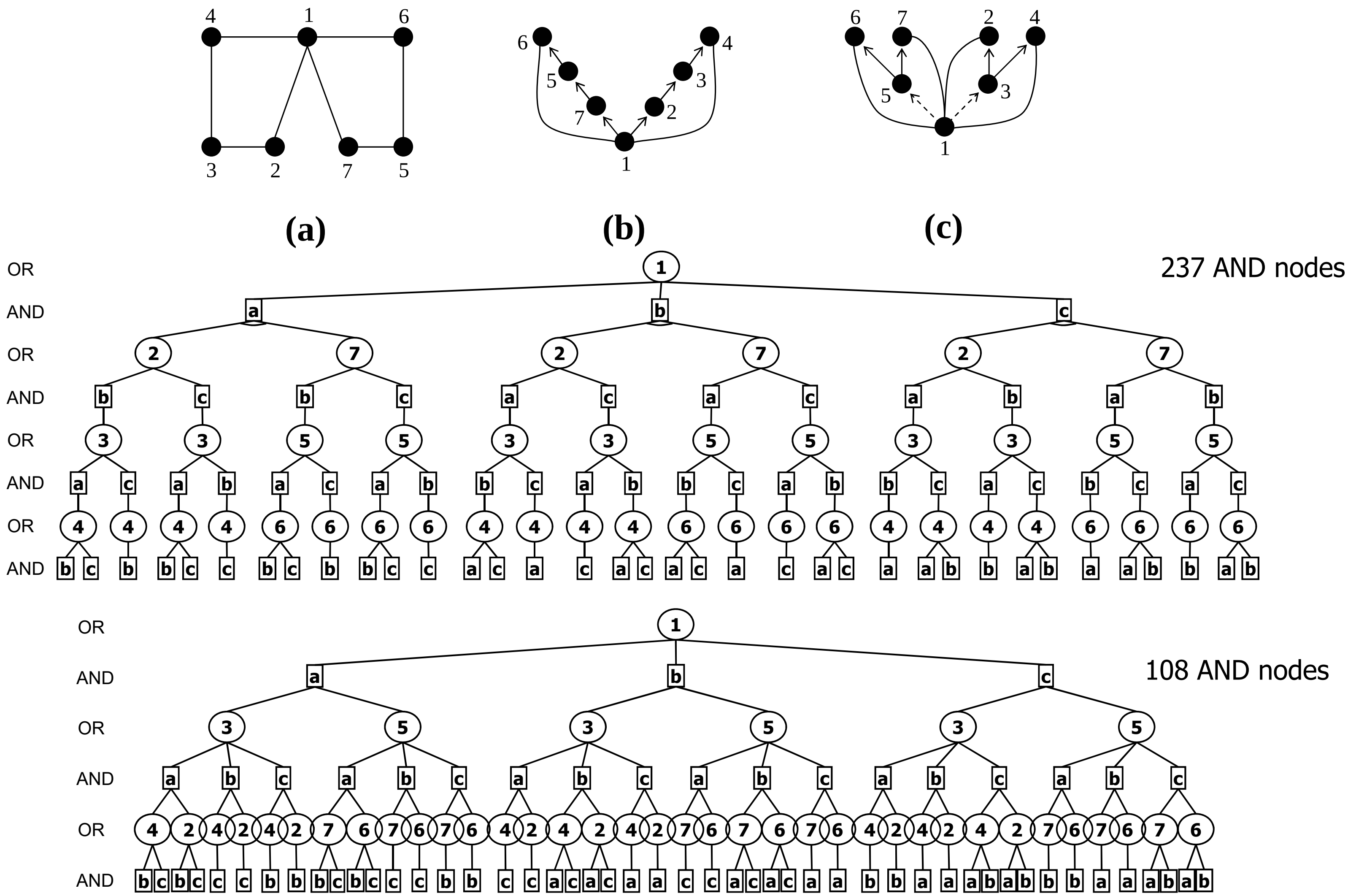
Extended Graphs

Definition 6.11 Pseudo tree, extended graph. Given an undirected graph $G = (V, E)$, a directed rooted tree $\mathcal{T} = (V, E')$ defined on all its nodes is a *pseudo tree* if any arc in E which is not in E' is a back-arc in $\mathcal{T}$, namely, it connects a node in $\mathcal{T}$ to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E. Given a pseudo tree $\mathcal{T}$ of G, the *extended graph* of G relative to $\mathcal{T}$ includes also the arcs in E' that are not in E. That is, the extended graph is defined as $G^{\mathcal{T}} = (V, E \cup E')$.

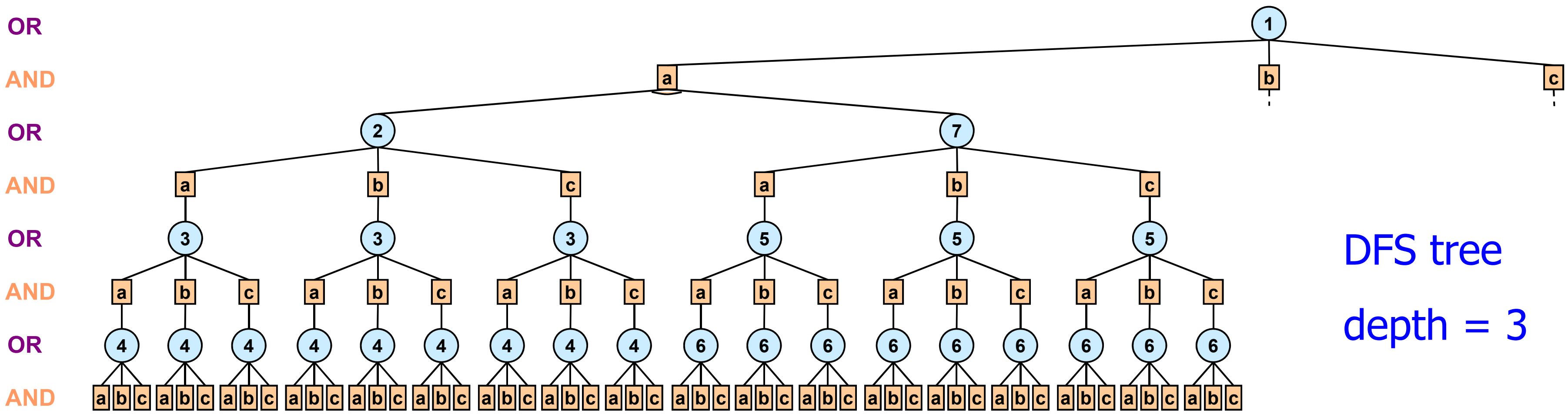
Theorem 6.14 Size of AND/OR search tree. *Given a graphical model $\mathcal{M}$, with domains size bounded by k, having a pseudo tree $\mathcal{T}$ whose height is h and having l leaves, the size of its AND/OR search tree $S_{\mathcal{T}}(\mathcal{M})$ is $O(l \cdot k^h)$ and therefore also $O(nk^h)$ and $O((bk)^h)$ when b bounds the branching degree of $\mathcal{T}$ and n bounds the number of nodes. The size of its OR search tree along any ordering is $O(k^n)$ and these bounds are tight. (See Appendix for proof.)*

Question: given, n,k,w,h,b develop and expression that study the size of the AND/OR search tree
As a function of these parameters, which are not independent of each other

From DFS-Trees to Pseudo-Trees

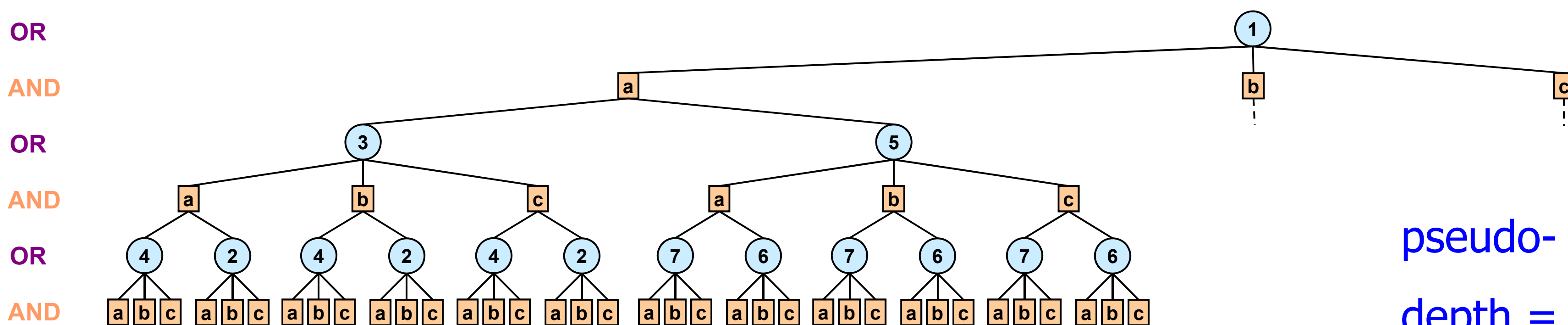


From DFS to Pseudo Trees



DFS tree

depth = 3



pseudo- tree

depth = 2

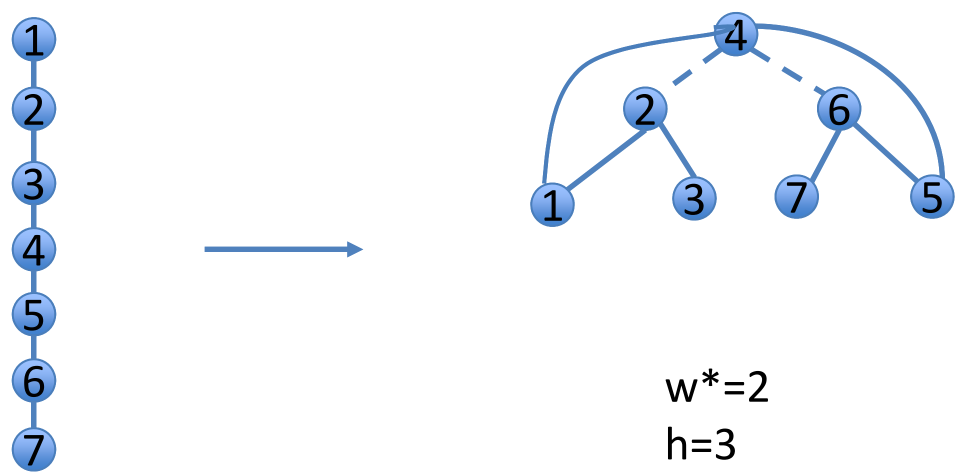
Finding Min-Depth Pseudo-Trees

- Finding min depth DFS, or pseudo tree is NP-complete, but:
- Given a tree-decomposition whose treewidth is w^*, there exists a pseudo -tree T of G whose depth, satisfies $h <= w^* \log n$,

Finding Min-height Pseudo-Trees

- Finding a min height pseudo-tree is NP-complete, but:
- Given a tree-decomposition with treewidth w*, there exists a pseudo-tree whose height satisfies
  - h <= w* log n
- Optimality of h and w* cannot be achieved at once.

W*=1
h=7



w*=2
h=3

AND/OR Search-tree properties

(k **= domain size,** ***h*** **= pseudo-tree height.** ***n*** **= number of variables)**

- **Theorem:** Any AND/OR search tree based on a pseudo-tree is sound and complete (expresses all and only solutions)

- **Theorem:** Size of AND/OR search tree is $O(n\ k^h)$

  Size of OR search tree is $O(k^n)$

- **Theorem:** Size of AND/OR search tree can be bounded by $O(\exp(w^* \log n))$

- When the pseudo-tree is a chain we get an OR space

Summary: Queries and Value of Nodes

- V(n) is the value of the tree T(n) for the task:
  - Counting: v(n) is number of solutions in T(n)
  - Consistency: ~~v(n) is 0 if T(n) inconsistent, 1 othewise.~~
  - Max-Inference: v(n) is the optimal solution in T(n)
  - Sum-Inference: v(n) is probability of evidence in T(n).
  - Mixed-Inference: v(n) is the marginal map in T(n).
- Goal: compute the value of the root node recursively traversing the AND/OR tree.

Complexity of searching depth-first is

- Space: $O(n)$
- Time: $O(nk^h)$
- Time: $O(k^{w*logn})$

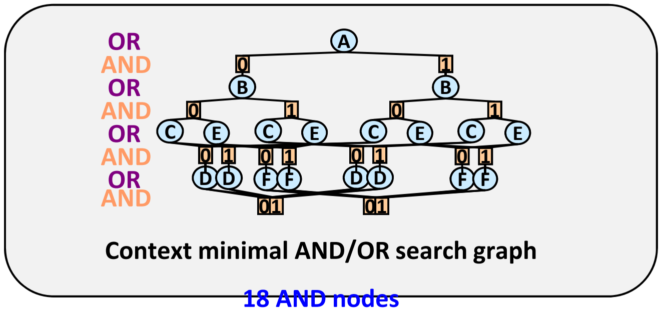
Outline: Search

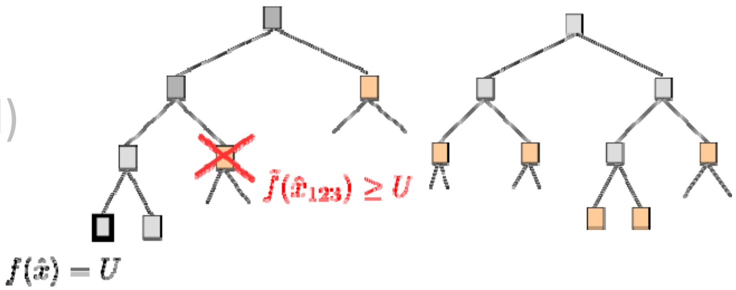
- Review Graphical Modes

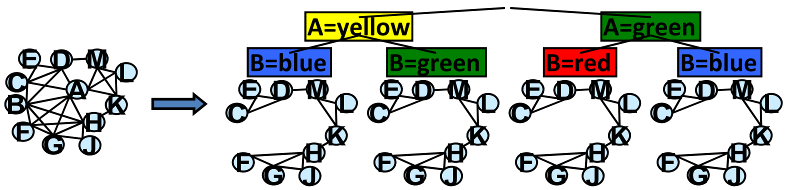
- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - **AND/OR search graphs**
  - Generating good pseudo-trees
  - Brute-force AND/OR

- Heuristic search (HS) for AND/OR spaces
  - Basic Heuristic search (Depth and Best)
  - AND/OR Depth-first HS (branch and bound)
  - AND/OR Best-first heuristic search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)

- Hybrids of search and Inference
- Summary and Class 2

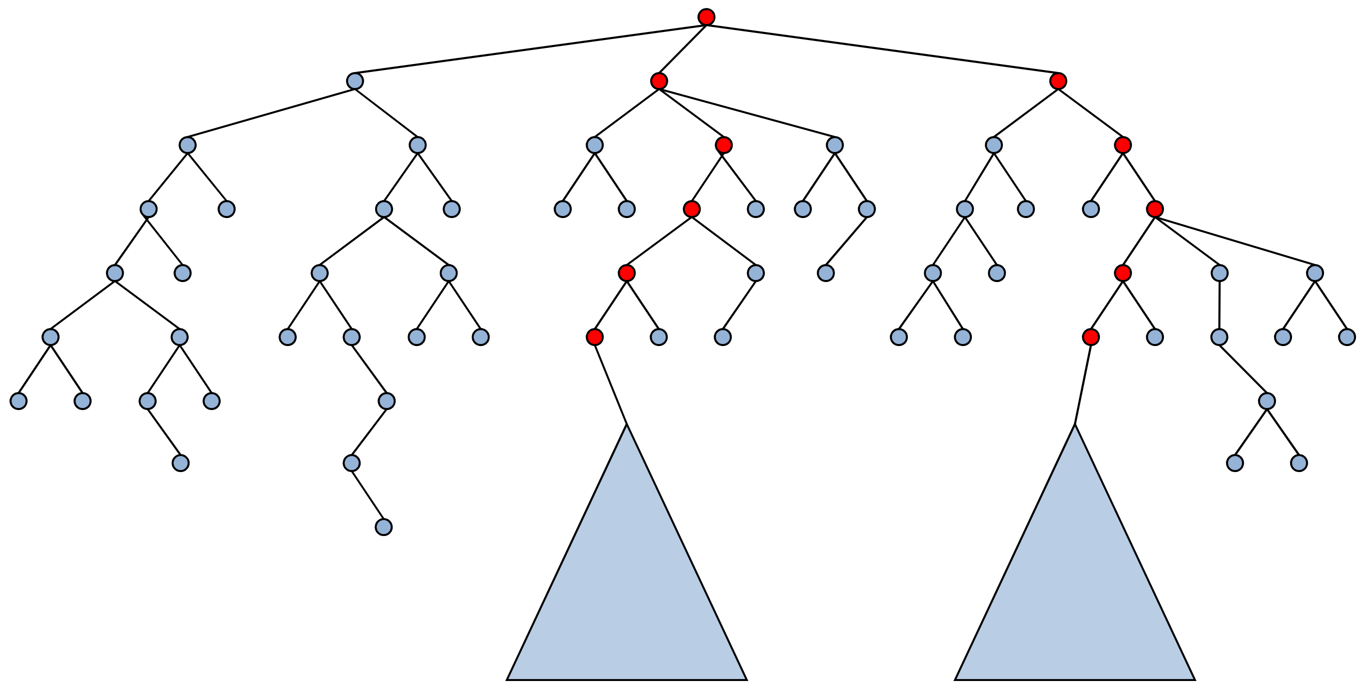






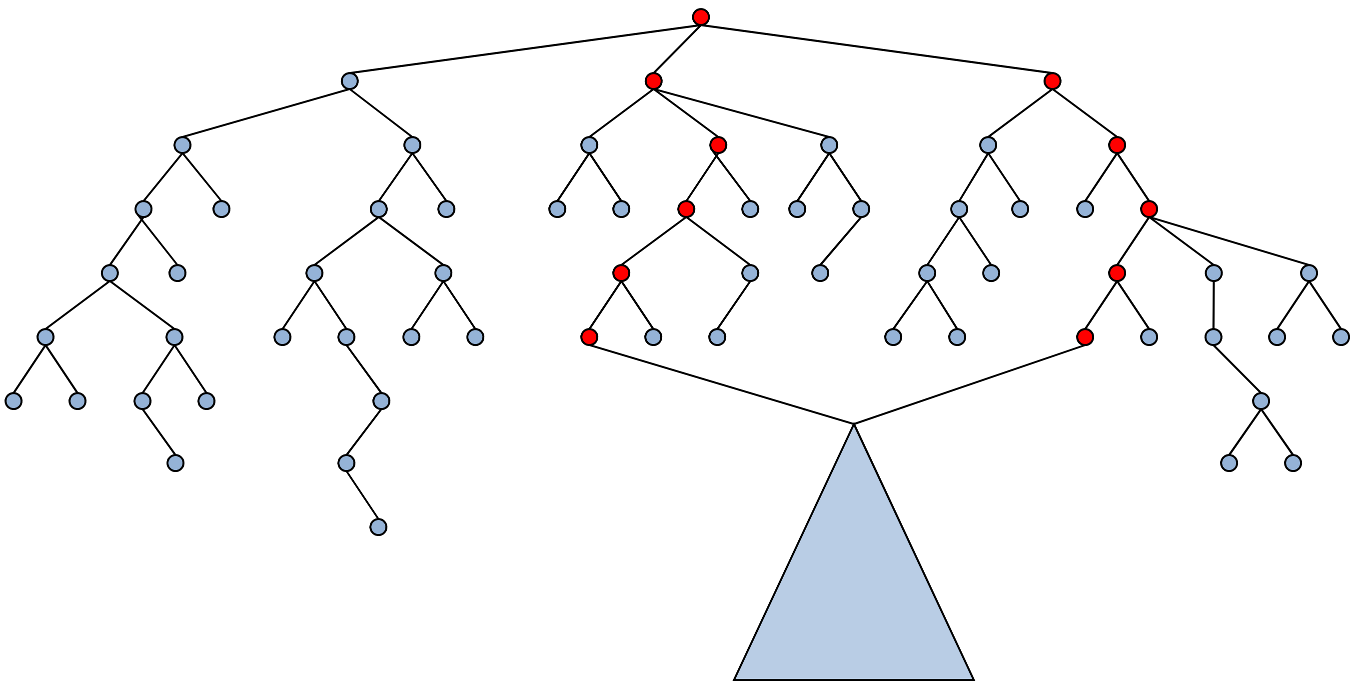
From Search Trees to Search Graphs

- Any two nodes that root identical subtrees or subgraphs can be merged

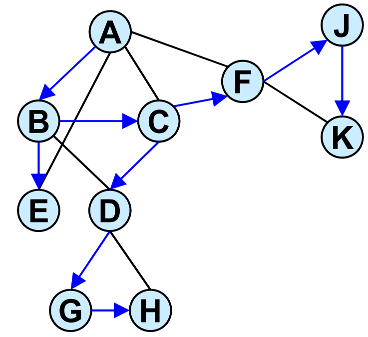


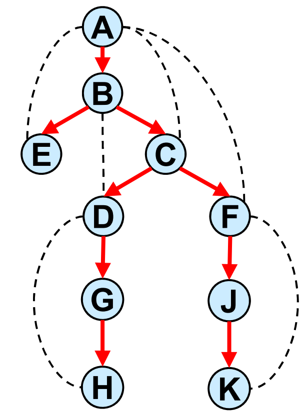
From Search Trees to Search Graphs

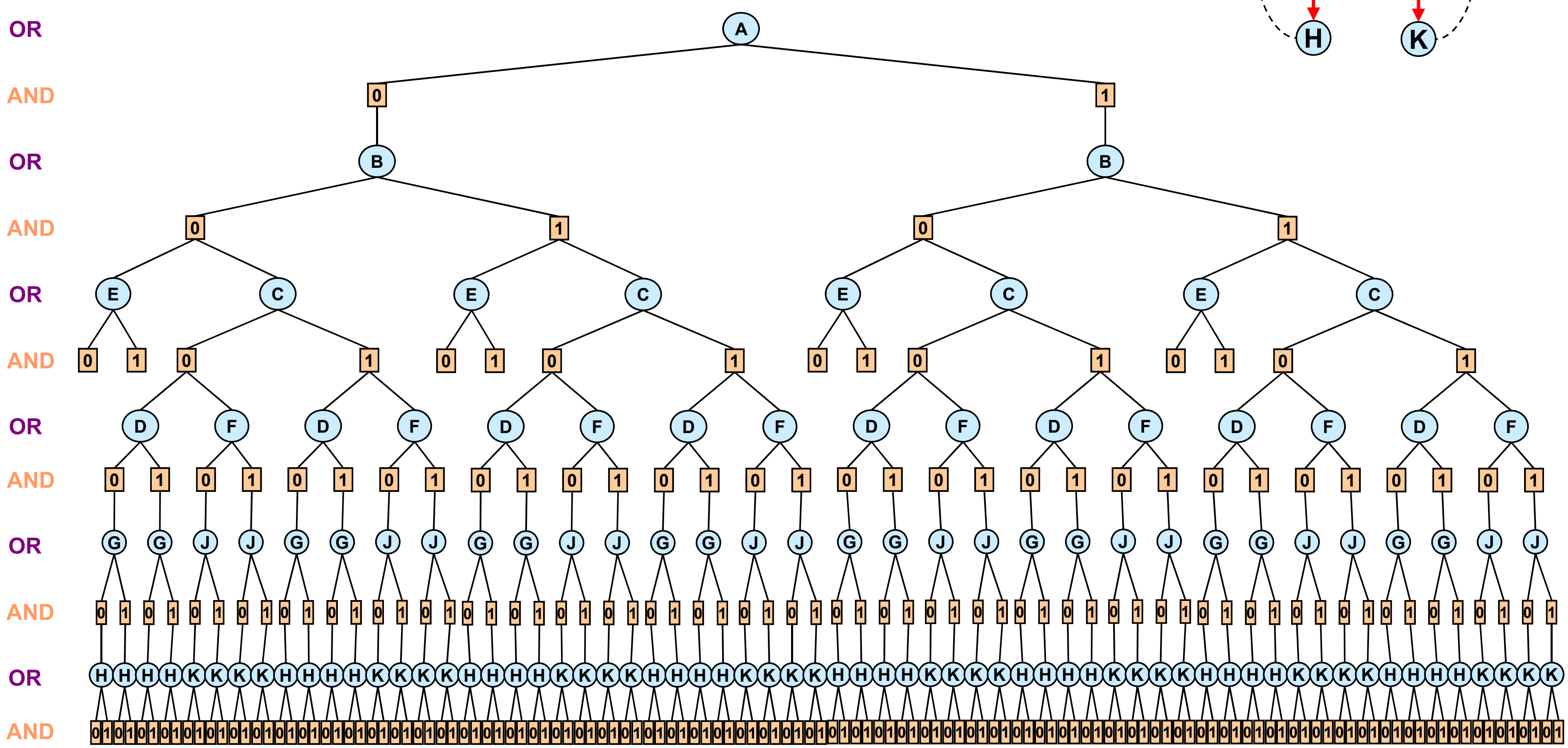
- Any two nodes that root identical subtrees or subgraphs can be merged



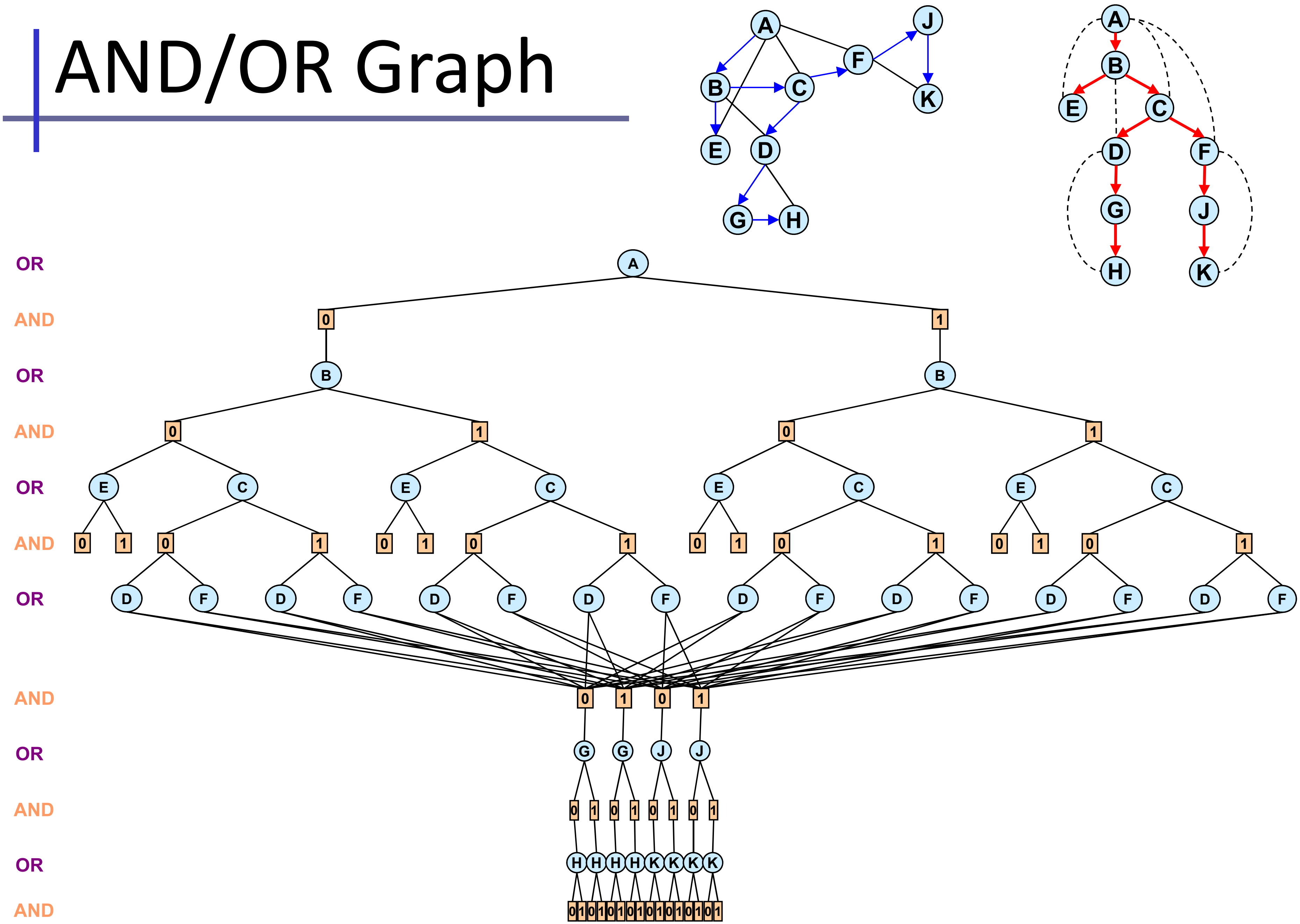
AND/OR Tree


A
B
C
D
E
F
G
H
J
K


A
B
E
C
D
F
G
J
H
K

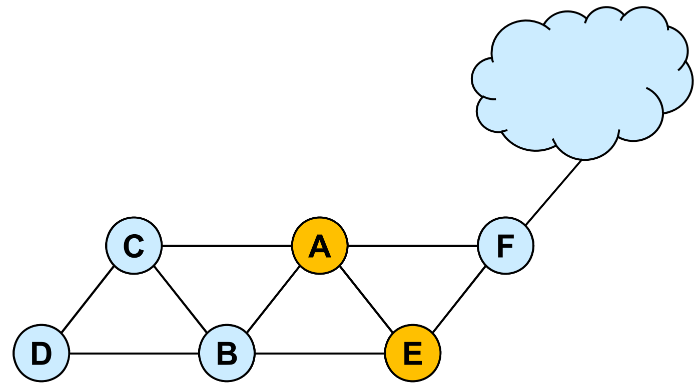

OR
AND
OR
AND
OR
AND
OR
AND
OR
AND
OR
AND

AND/OR Graph

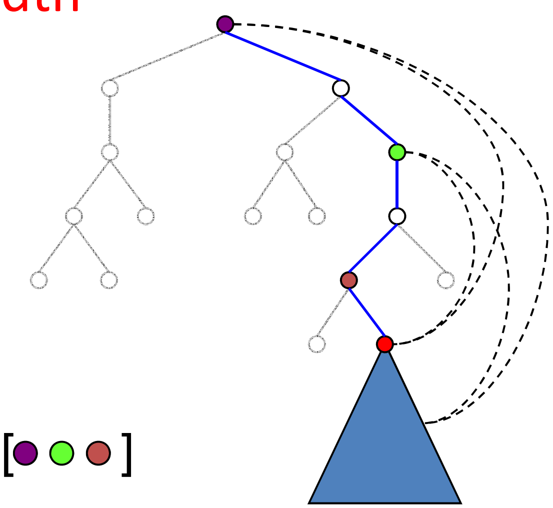


Merging Based on Context

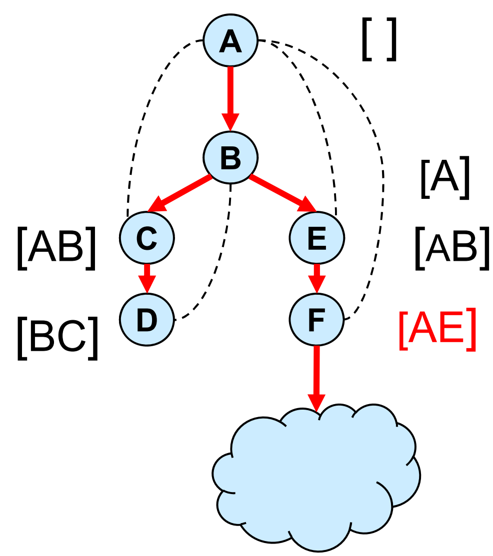
- context (X) = ancestors of X in pseudo tree, connected to X, or to descendants of X
- context (X) = parents in the induced graph
- max |context| = induced width = treewidth



pseudo tree



context(●) = [● ● ●]



Context-Based Minimal AND/OR Search Graph

Definition 7.2.13 (context minimal AND/OR search graph) *The AND/OR search graph of M guided by a pseudo-tree T that is closed under context-based merge operator, is called the* context minimal *AND/OR search graph and is denoted by* $C_T(R)$.

AND/OR Tree DFS Algorithm (Value=Sum-Product)

$P(E \mid A, B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

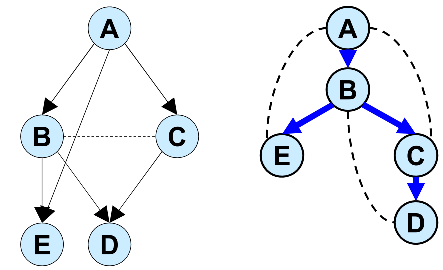
A	B=0	B=1
0	.4	.6
1	.1	.9

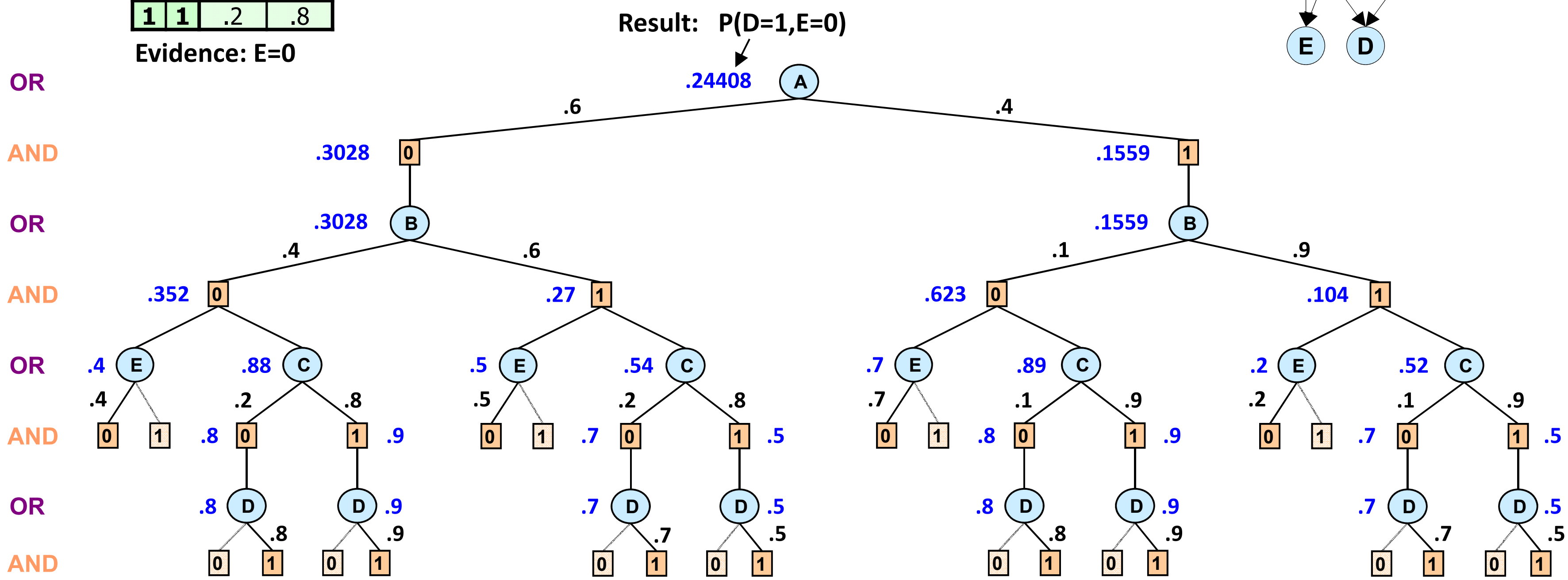
$P(C \mid A)$

A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4





$P(D \mid B, C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1,E=0

AND/OR Search Graph (Value=Sum-Product)

$P(E \mid A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

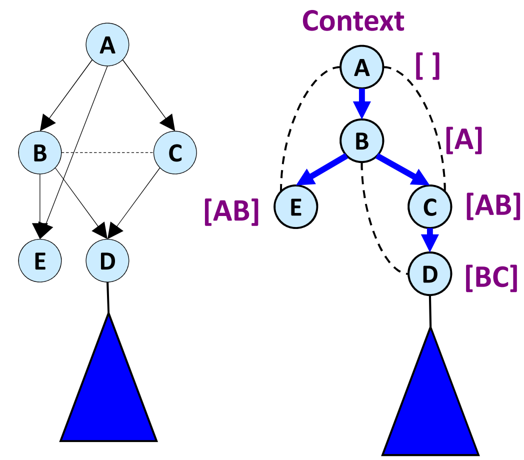
A	B=0	B=1
0	.4	.6
1	.1	.9

$P(C \mid A)$

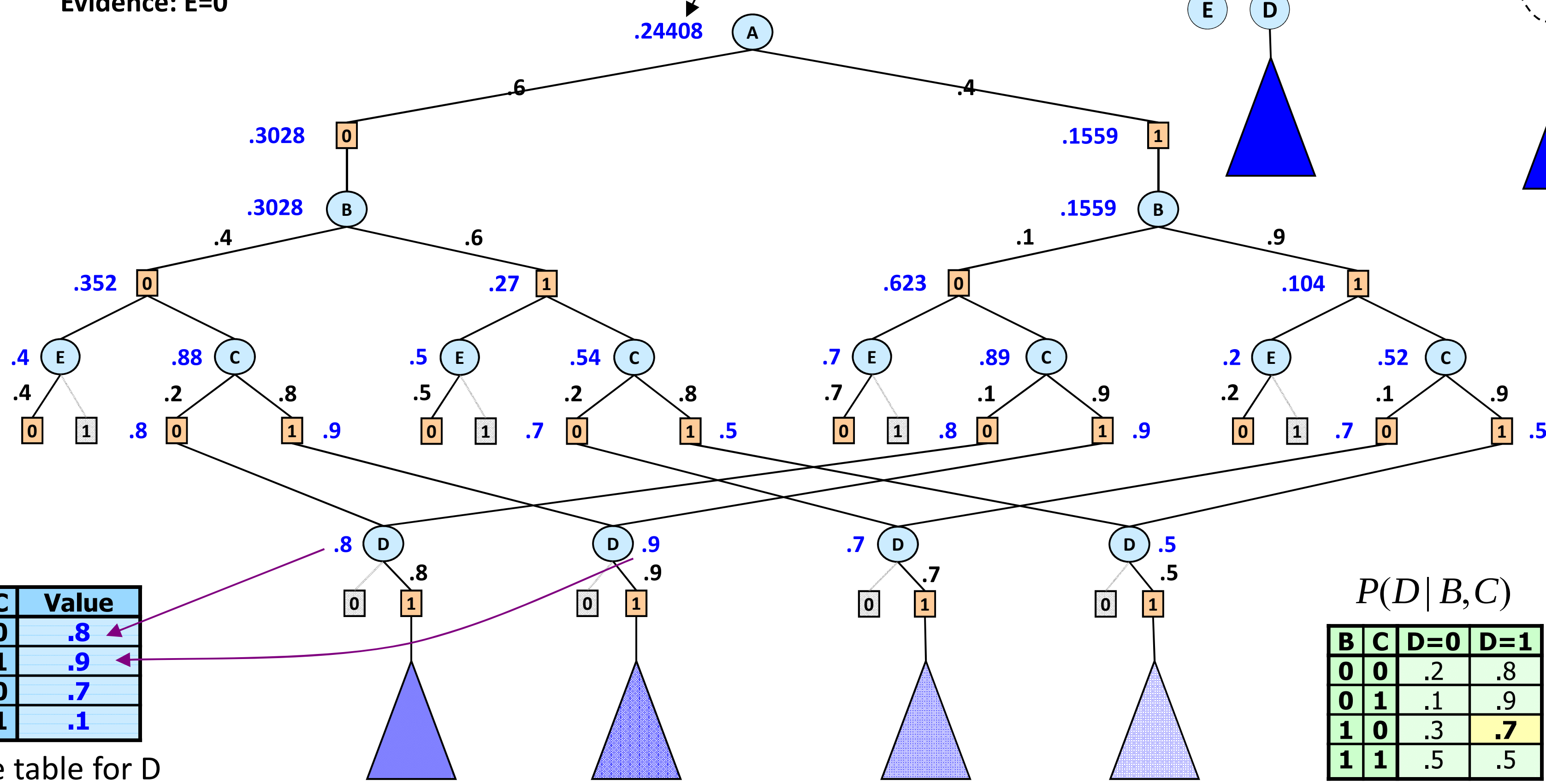
A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4



Result: P(D=1,E=0)



B	C	Value
0	0	.8
0	1	.9
1	0	.7
1	1	.1

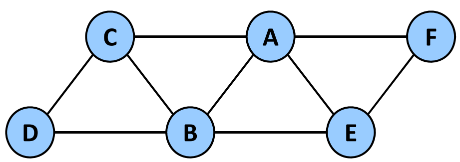
Cache table for D

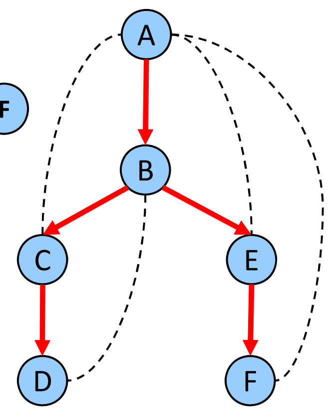
$P(D \mid B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1,E=0

AND/OR Search Graph (Optimization)





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

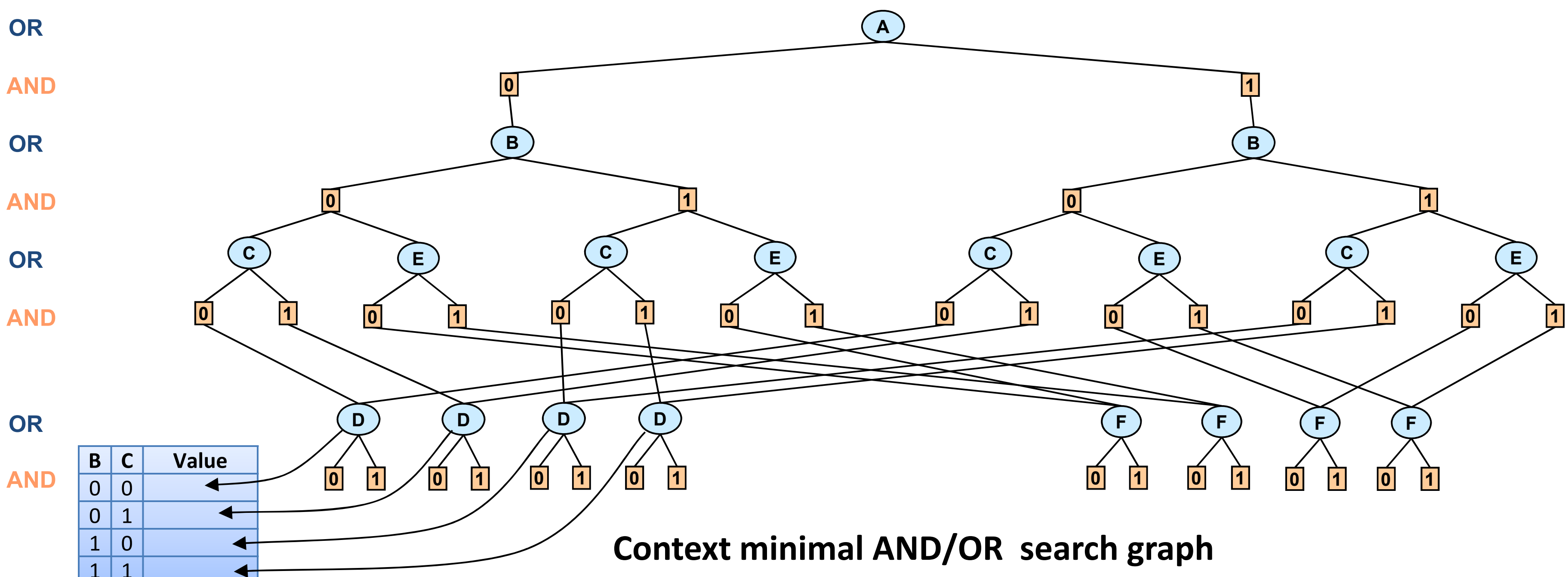
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

Objective function: $F^* = \min_x \sum_\alpha f_\alpha(x_\alpha)$



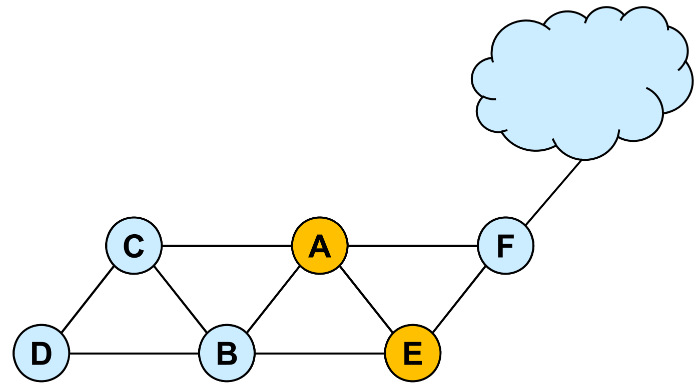
B	C	Value
0	0	
0	1	
1	0	
1	1	

Cache table for **D**

Context minimal AND/OR search graph

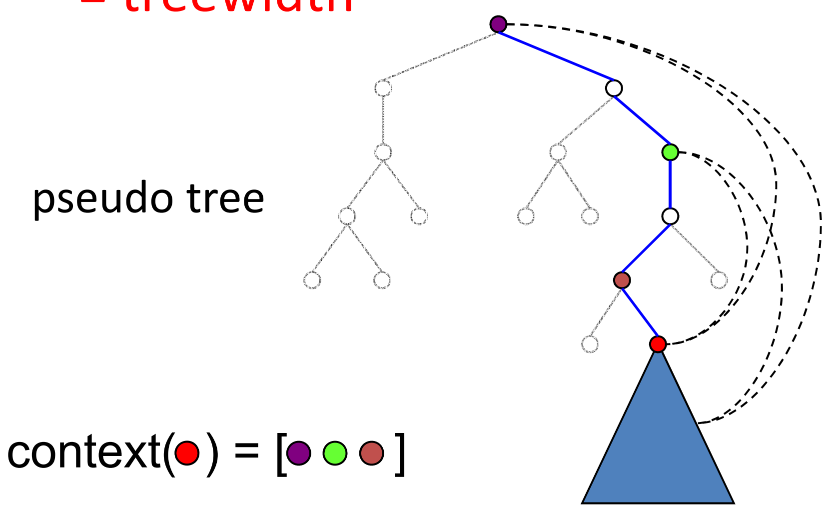
Merging Based on Context

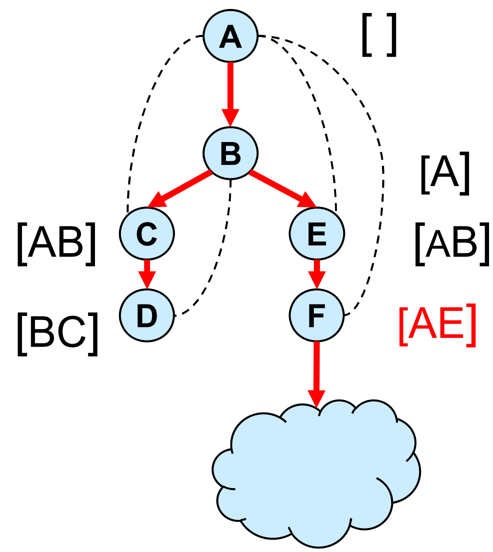
- context (X) = ancestors of X in pseudo tree, connected to X, or to descendants of X



- context (X) = parents in the induced graph

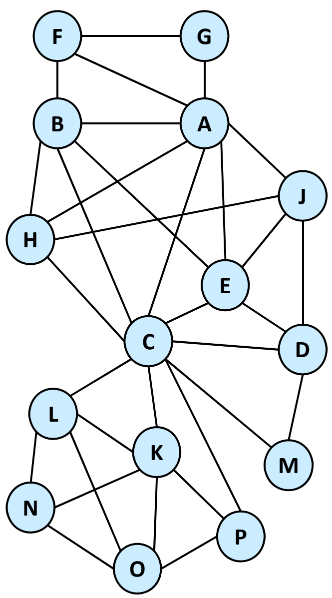
- max |context| = induced width = treewidth

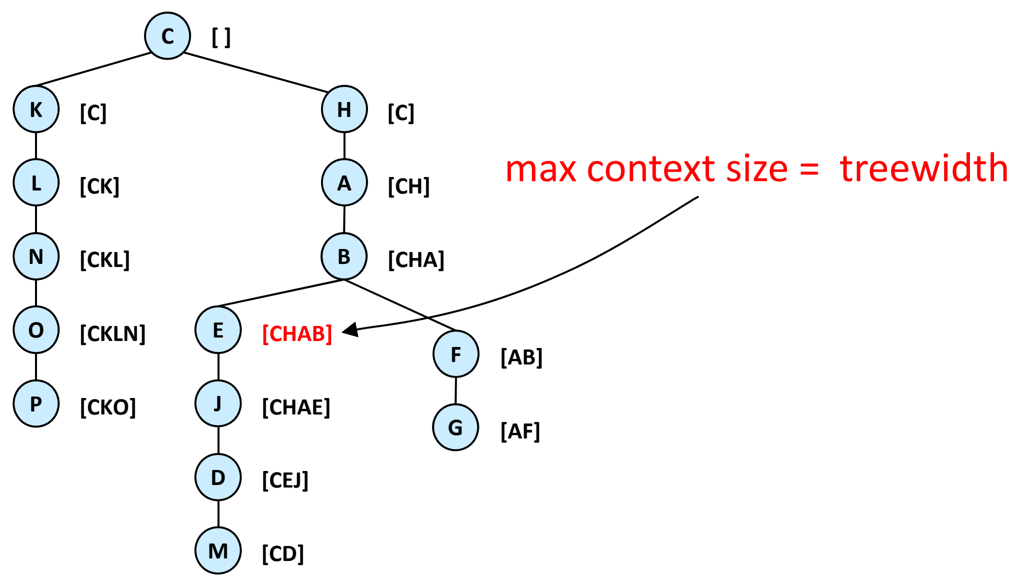




How Big Is The Context?

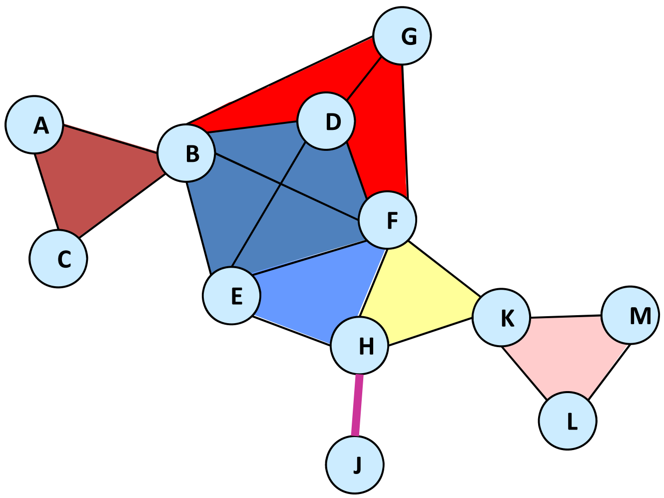
- Theorem: The maximum context-size of a pseudo-tree equals the treewidth along the pseudo tree.

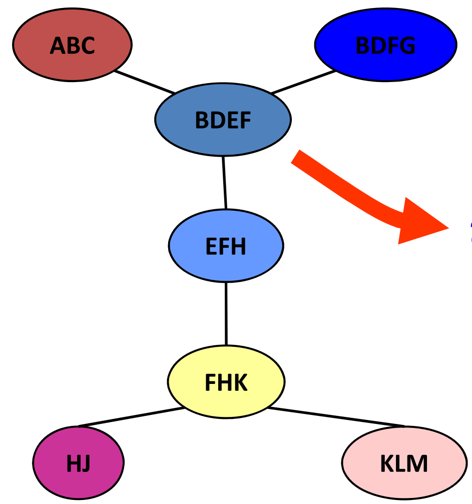




(C K H A B E J L N O D P M F G)

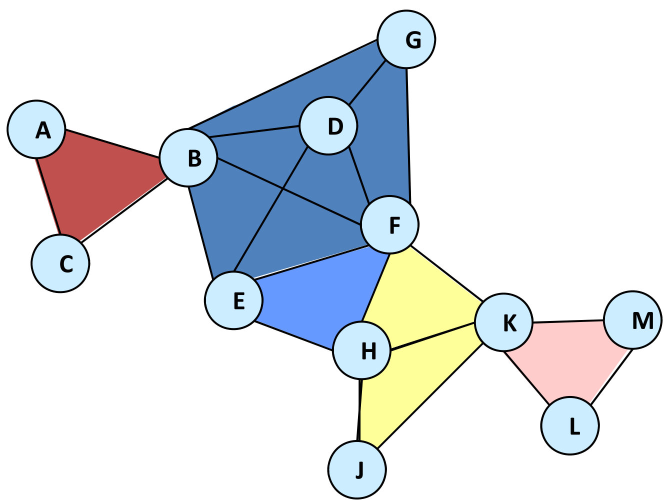
Treewidth vs. Pathwidth

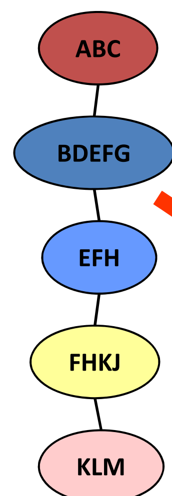




***treewidth* = 3**

= (max cluster size) - 1

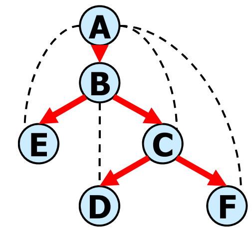


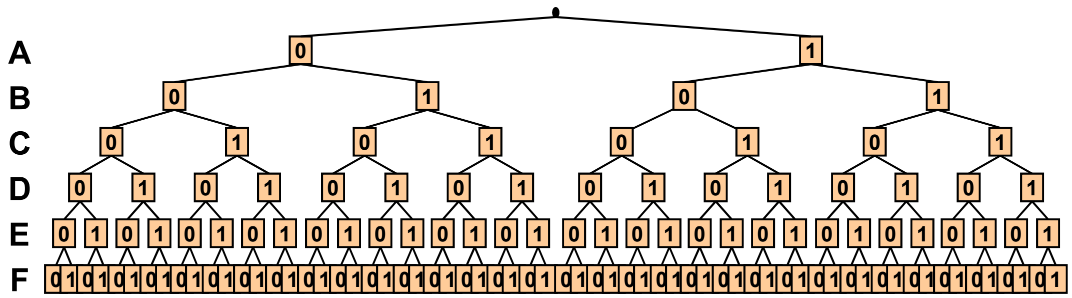


***pathwidth* = 4**

= (max cluster size) - 1

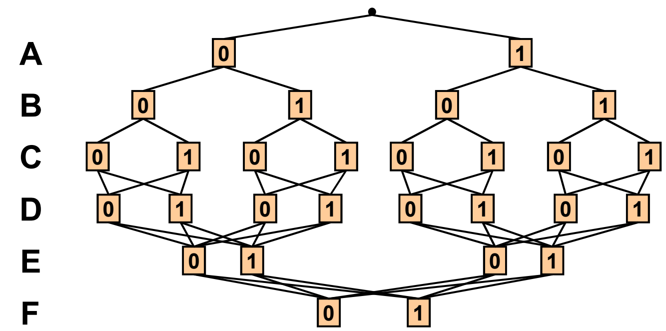
All Four Search Spaces





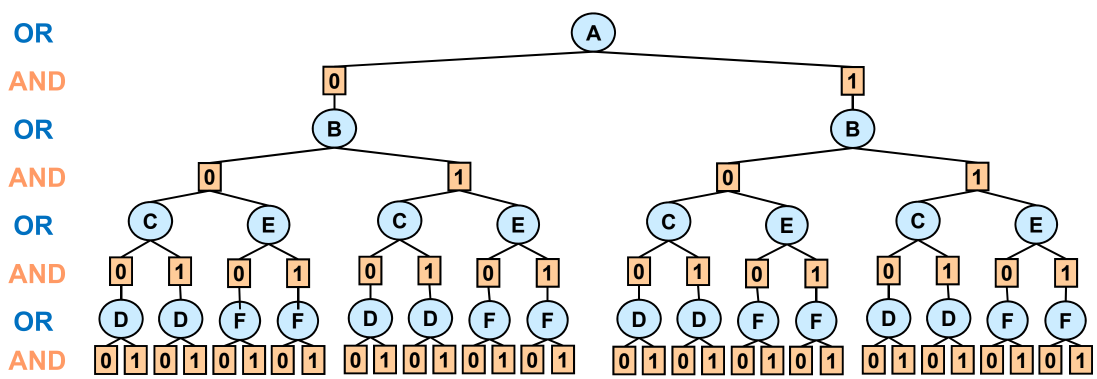
Full OR search tree

126 nodes



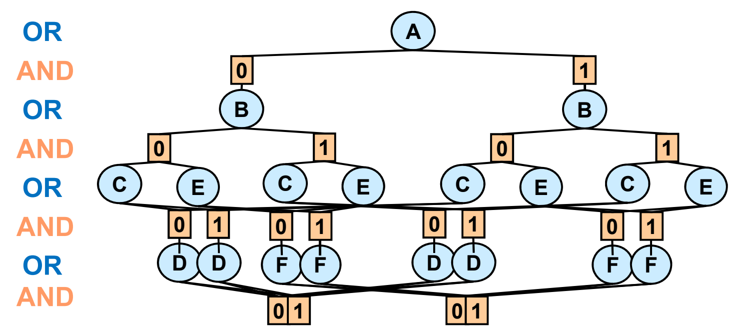
Context minimal OR search graph

28 nodes



Full AND/OR search tree

54 AND nodes



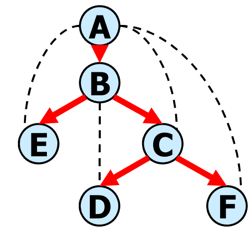
Context minimal AND/OR search graph

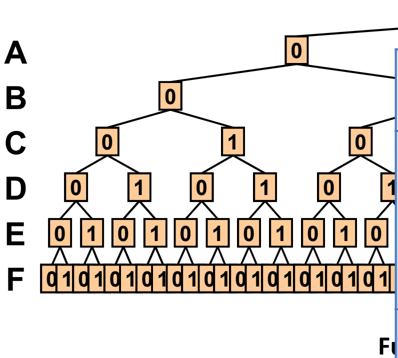
18 AND nodes

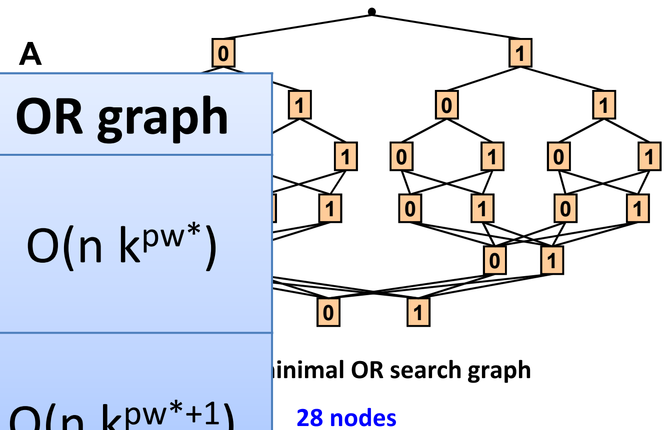
k = domain size
n = number of variables
w*= treewidth
pw*= pathwidth

Any query is best computed over the context-minimal AND/OR space

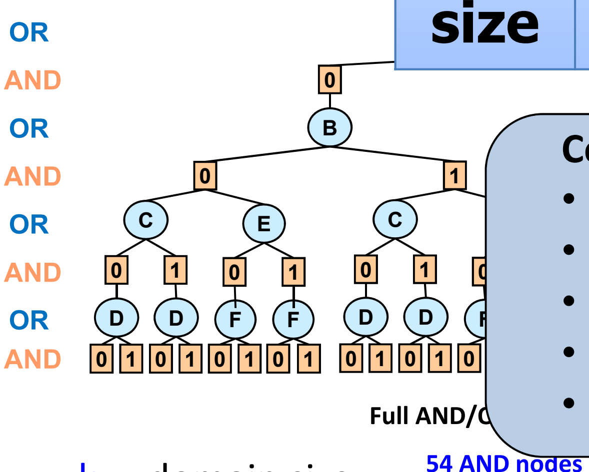
All Four Search Spaces





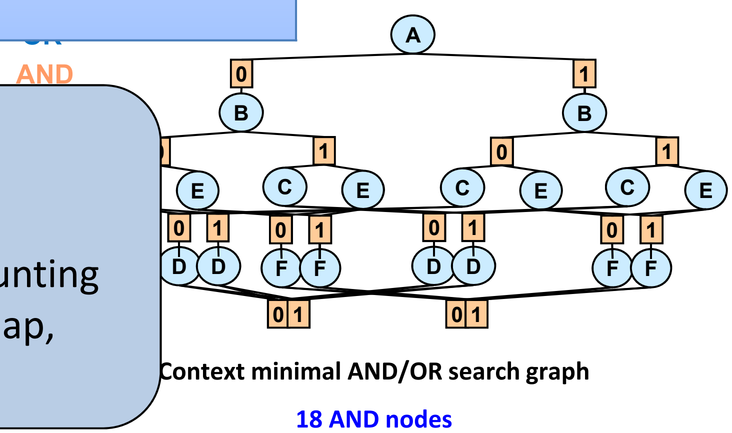


	AND/OR graph	OR graph
Space	$O(n\ k^{w^*})$	$O(n\ k^{pw^*})$
Time size	$O(n\ k^{w^*+1})$	$O(n\ k^{pw^*+1})$



Computes any query:

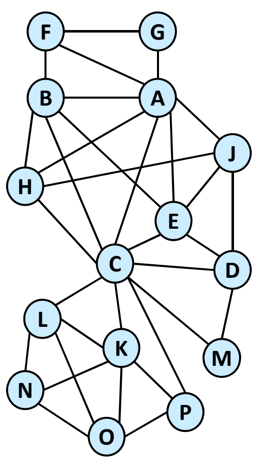
- Constraint satisfaction
- Max-Inference: Optimization
- Sum-Inference: Weighted counting
- Mixed-Inference: Marginal Map,
- Maximum expected utility

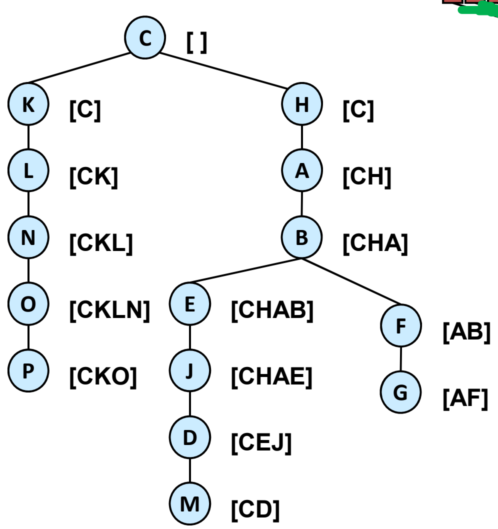


k = domain size
n = number of variables
w^*= treewidth
pw^*= pathwidth

Any query is best computed over the context-minimal AND/OR space

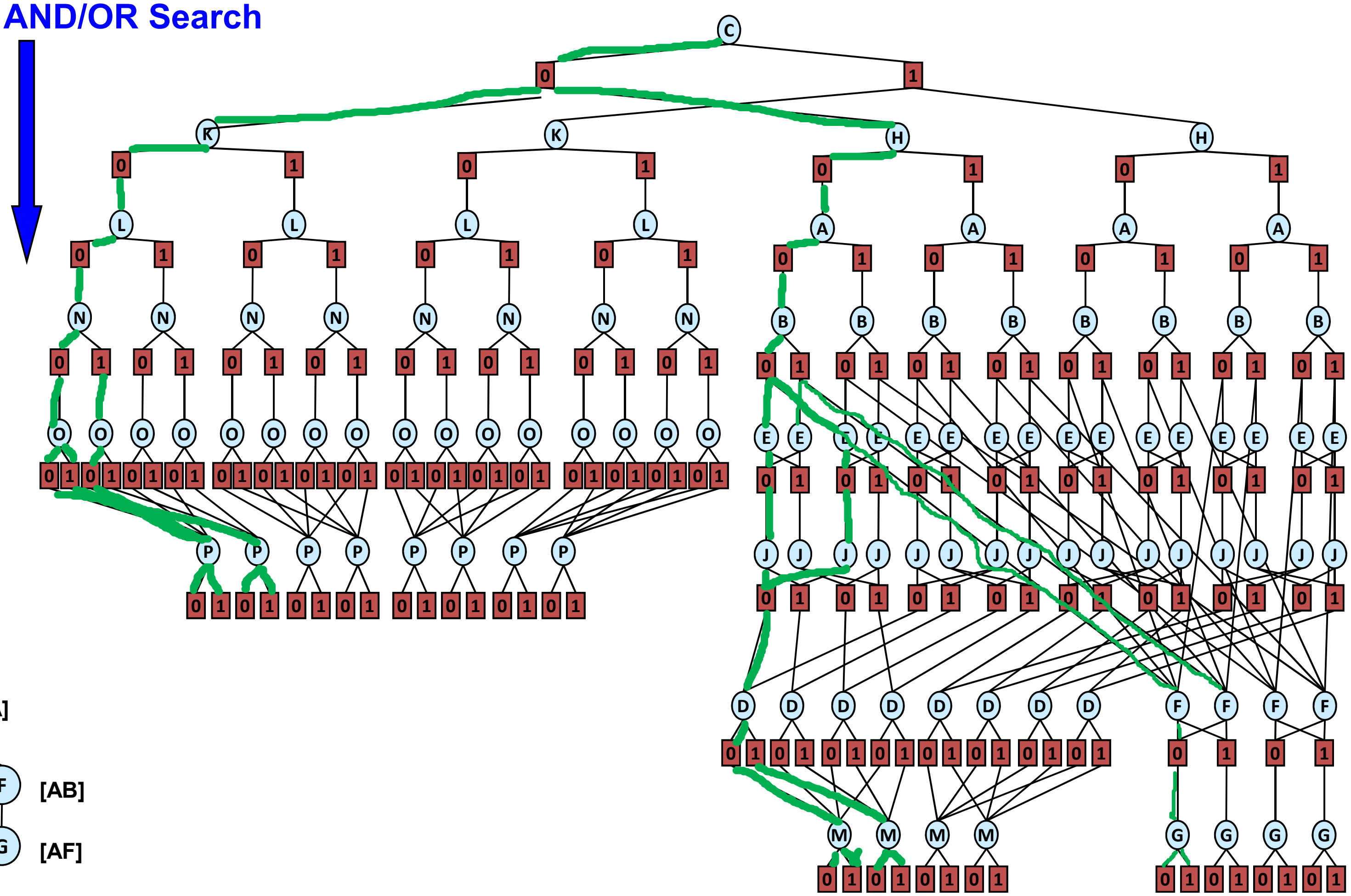
AND/OR Search and Variable Elimination



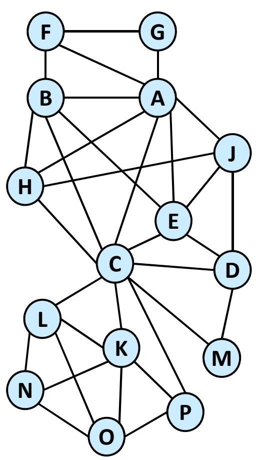


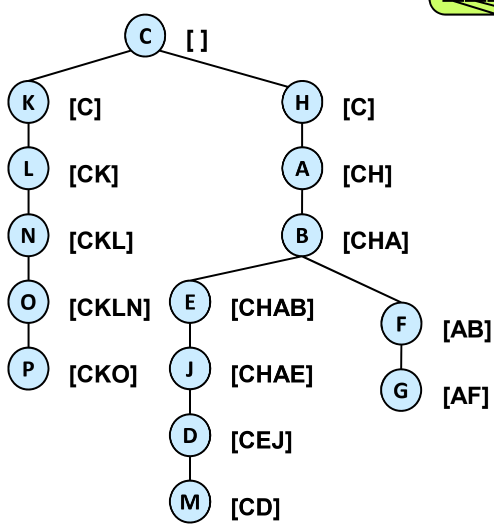
(C K H A B E J L N O D P M F G)

AND/OR Search

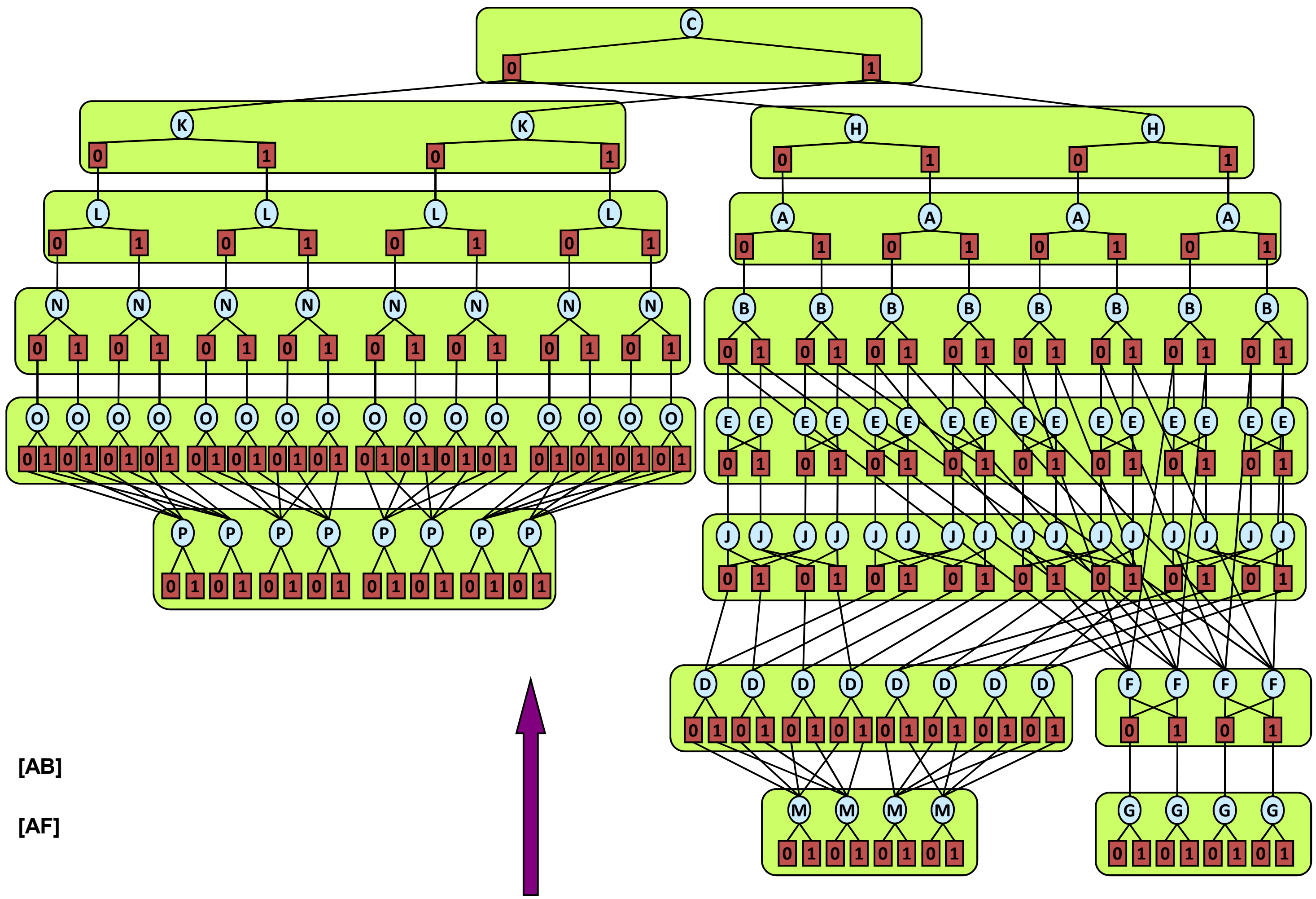


AND/OR Search and Variable Elimination



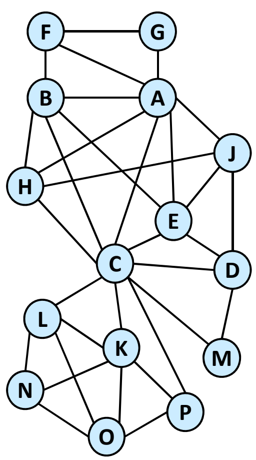


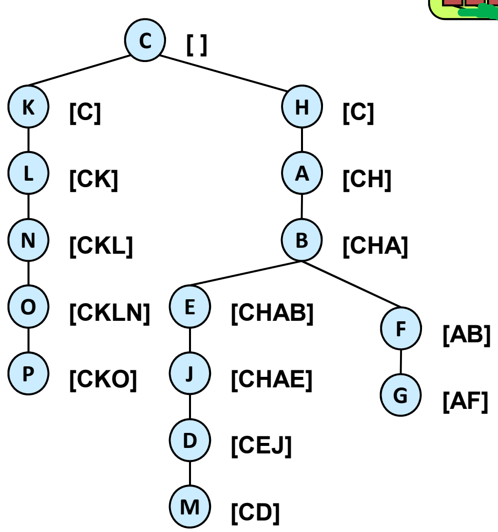
(C K H A B E J L N O D P M F G)



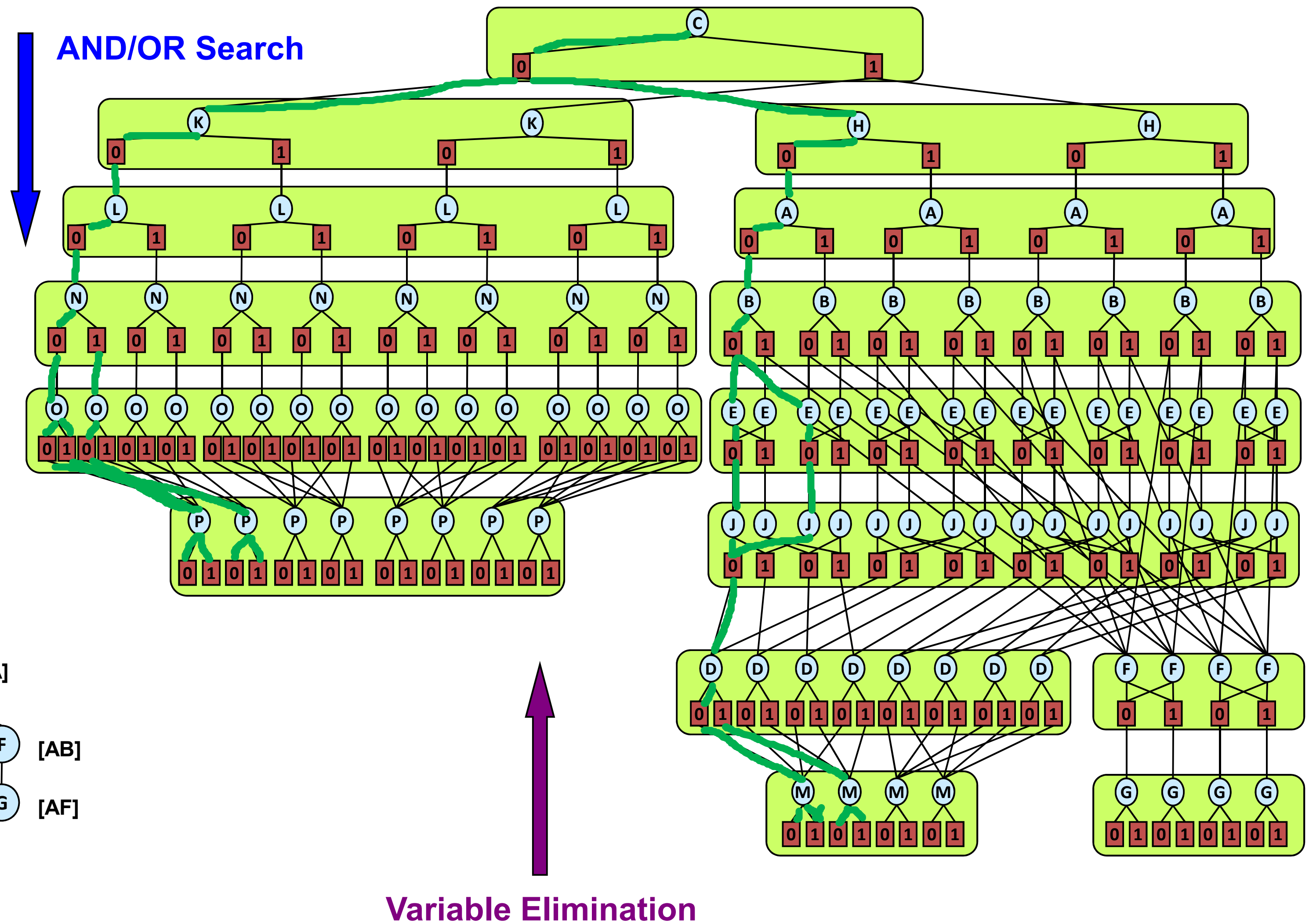
Variable Elimination

AND/OR Search and Variable Elimination





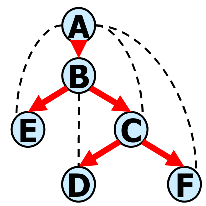
(C K H A B E J L N O D P M F G)

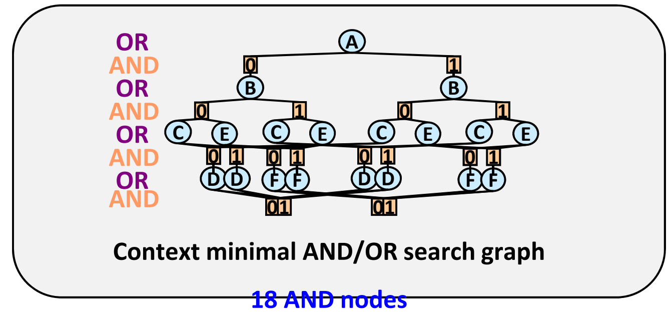


Road Map: Search

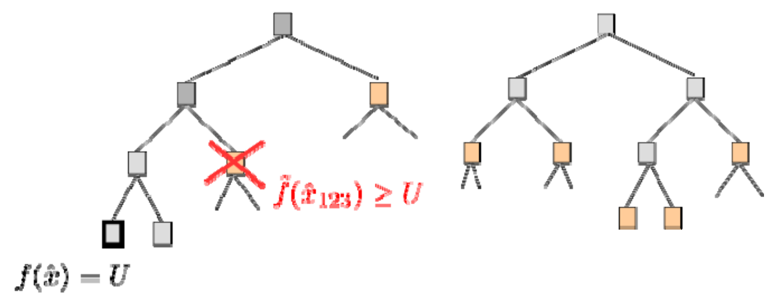
- Review Graphical Modes

- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - **Generating good pseudo-trees**
  - Brute-force AND/OR

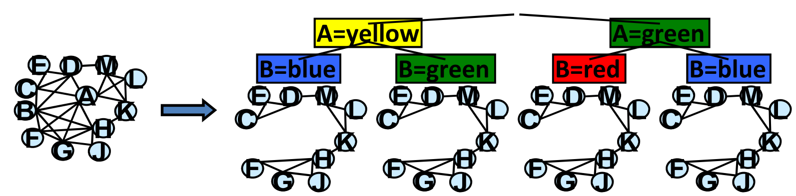




- Heuristic search for AND/OR spaces
  - Basic Heuristic search (Depth and Best)
  - Depth-first AND/OR branch and bound
  - Best-first AND/OR search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)



- Hybrids of search and Inference
- Summary and Class 2



Finding Good Pseudo-Trees

Finding Min-Height Pseudo-Trees

- Finding min height pseudo tree is NP-complete, but:
- Given a tree-decomposition whose tree-width is w^*, there exists a pseudo-tree T of G whose depth, satisfies $h \le w^* \log n$,

Constructing Pseudo-Trees

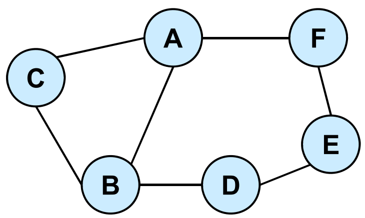
- **Min-Fill** [Kjaerulff, 1990]
  - Depth-first traversal of the induced graph obtained along the min-fill elimination order
  - Variables ordered according to the smallest "fill-set"

- **Hypergraph Partitioning** [Karypis and Kumar, 2000]
  - Functions are vertices in the hypergraph and variables are hyperedges
  - Recursive decomposition of the hypergraph while minimizing the separator size at each step
  - Using state-of-the-art software package hMeTiS

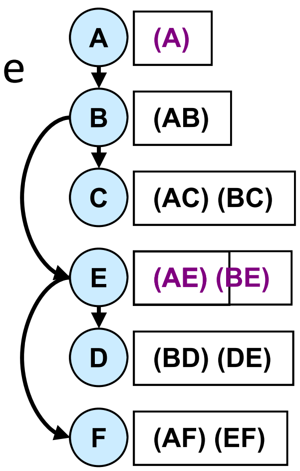
Variable Orderings and Pseudo-Trees

Note: we order from top to bottom here

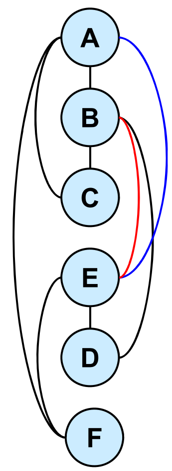
Bucket-tree = pseudo-tree



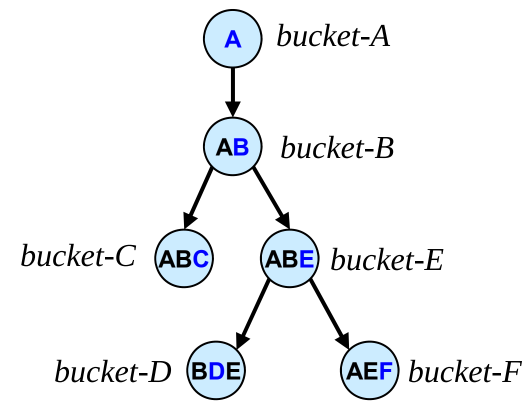
***d*: A B C E D F**



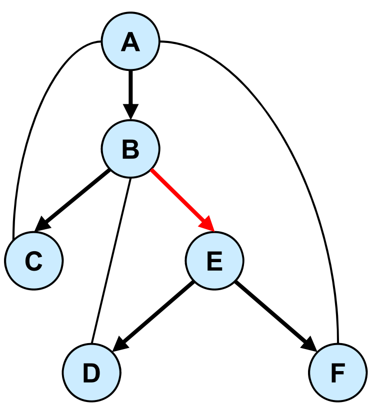
Bucket-tree based on *d*



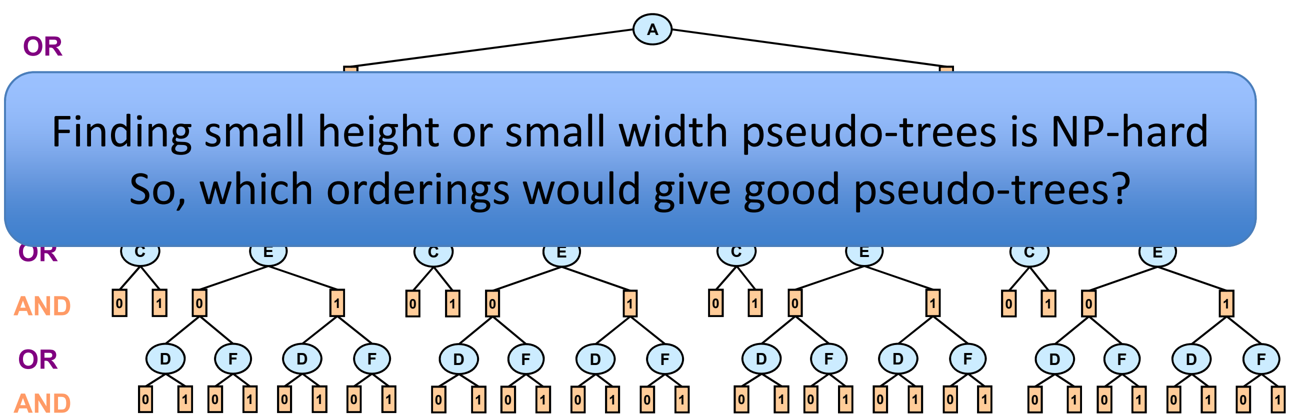
Induced graph



Bucket-tree



Bucket-tree used as pseudo-tree



Finding small height or small width pseudo-trees is NP-hard
So, which orderings would give good pseudo-trees?

AND/OR search tree

Quality of Pseudo-Trees

Network	hypergraph		min-fill	
	width	depth	width	depth
barley	7	**13**	7	23
diabetes	7	**16**	4	77
link	21	**40**	15	53
mildew	5	**9**	4	13
munin1	12	**17**	12	29
munin2	9	**16**	9	32
munin3	9	**15**	9	30
munin4	9	**18**	9	30
water	11	**16**	10	15
pigs	11	**20**	11	26

Bayesian Networks Repository

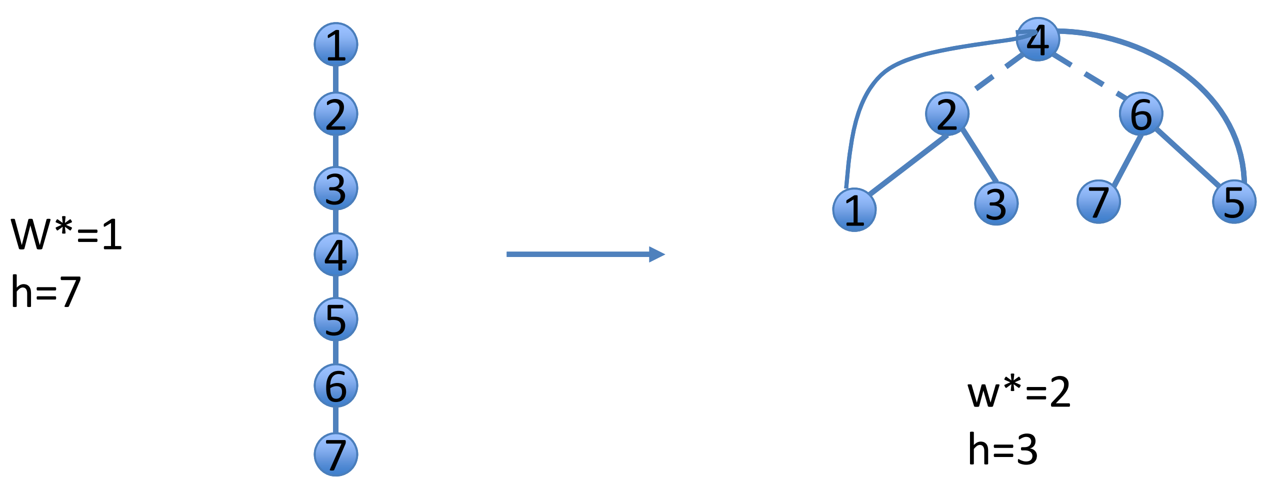
Network	hypergraph		min-fill	
	width	depth	width	depth
spot5	47	152	**39**	204
spot28	108	138	**79**	199
spot29	16	23	**14**	42
spot42	36	48	**33**	87
spot54	12	16	**11**	33
spot404	19	26	**19**	42
spot408	47	52	**35**	97
spot503	11	20	**9**	39
spot505	29	42	**23**	74
spot507	70	122	**59**	160

SPOT5 Benchmarks

For more see [Dechter 2013]

Finding Min-height Pseudo-Trees

- Finding a min height pseudo-tree is NP-complete, but:
- Given a tree-decomposition with treewidth w*, there exists a pseudo-tree whose height satisfies
  - h <= w* log n
- Optimality of h and w* cannot be achieved at once.

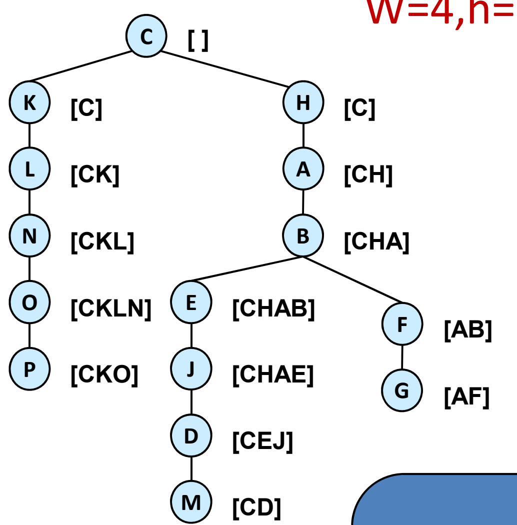


Hypergraphs Decomposition

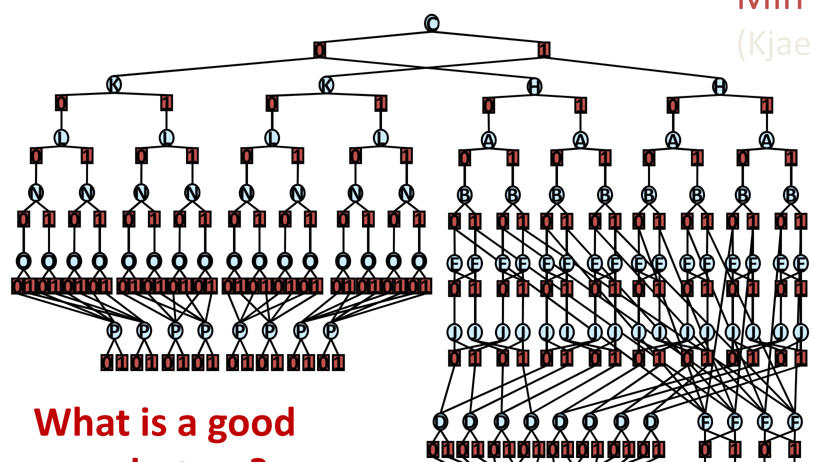
It is well known that the problem of finding the minimal size hypergraph separator is hard. However, heuristic approaches were developed over the years.[1]. Generating a pseudo tree $\mathcal{T}$ (yielding also a tree-decomposition) for $\mathcal{M}$ using hypergraph decomposition is fairly straightforward. The vertices of the hypergraph are partitioned into two balanced (roughly equal-sized) parts, denoted by $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$, respectively, while minimizing the number of hyperedges across. A small number of crossing edges translates into a small number of variables shared between the two sets of functions. $\mathcal{H}_{left}$ and $\mathcal{H}_{right}$ are then each recursively partitioned in the same fashion, until they contain a single vertex. The result of this process is a tree of hypergraph separators which can be shown to also be a pseudo tree of the original model where each separator corresponds to a subset of variables connected by a chain.

The Impact of the Pseudo-Tree

W=4,h=8

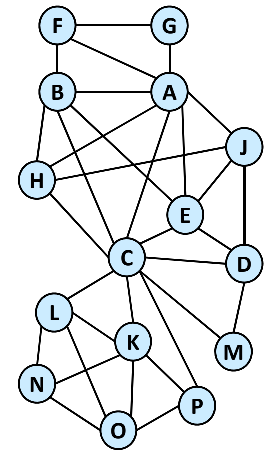


(C K H A B E J L N O



What is a good

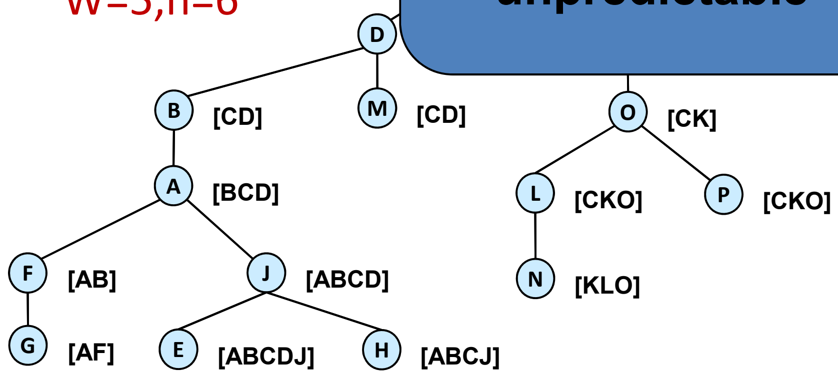
Min-Fill
(Kjaerulff90)

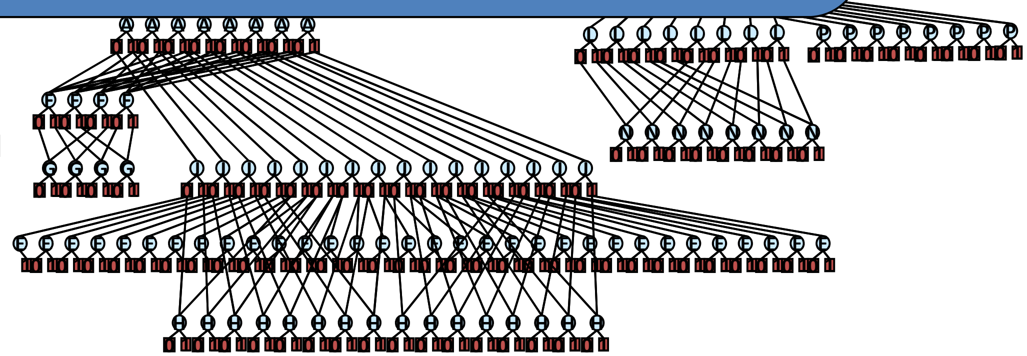


graph
oning
is)

- ***Choose pseudo-tree with a minimal search graph***
- **But determinism and pruning for optimization is unpredictable**

W=5,h=6

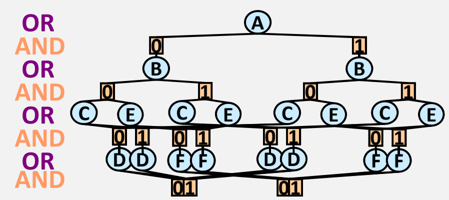




(C D K B A O M L N P J H E F G)

Road Map: Search

- Review Graphical Modes
- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - Generating good pseudo-trees
  - **Brute-force AND/OR**
- Heuristic search (HS) for AND/OR spaces
  - Basic Heuristic search (Depth and Best)
  - AND/OR Depth-first HS (branch and bound)
  - AND/OR Best-first heuristic search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)
- Hybrids of search and Inference
- Summary and Class 2

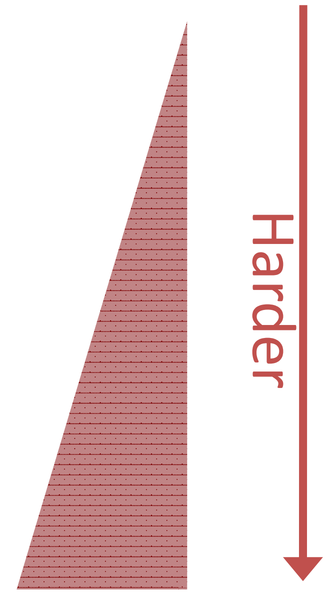


Context minimal AND/OR search graph

18 AND nodes

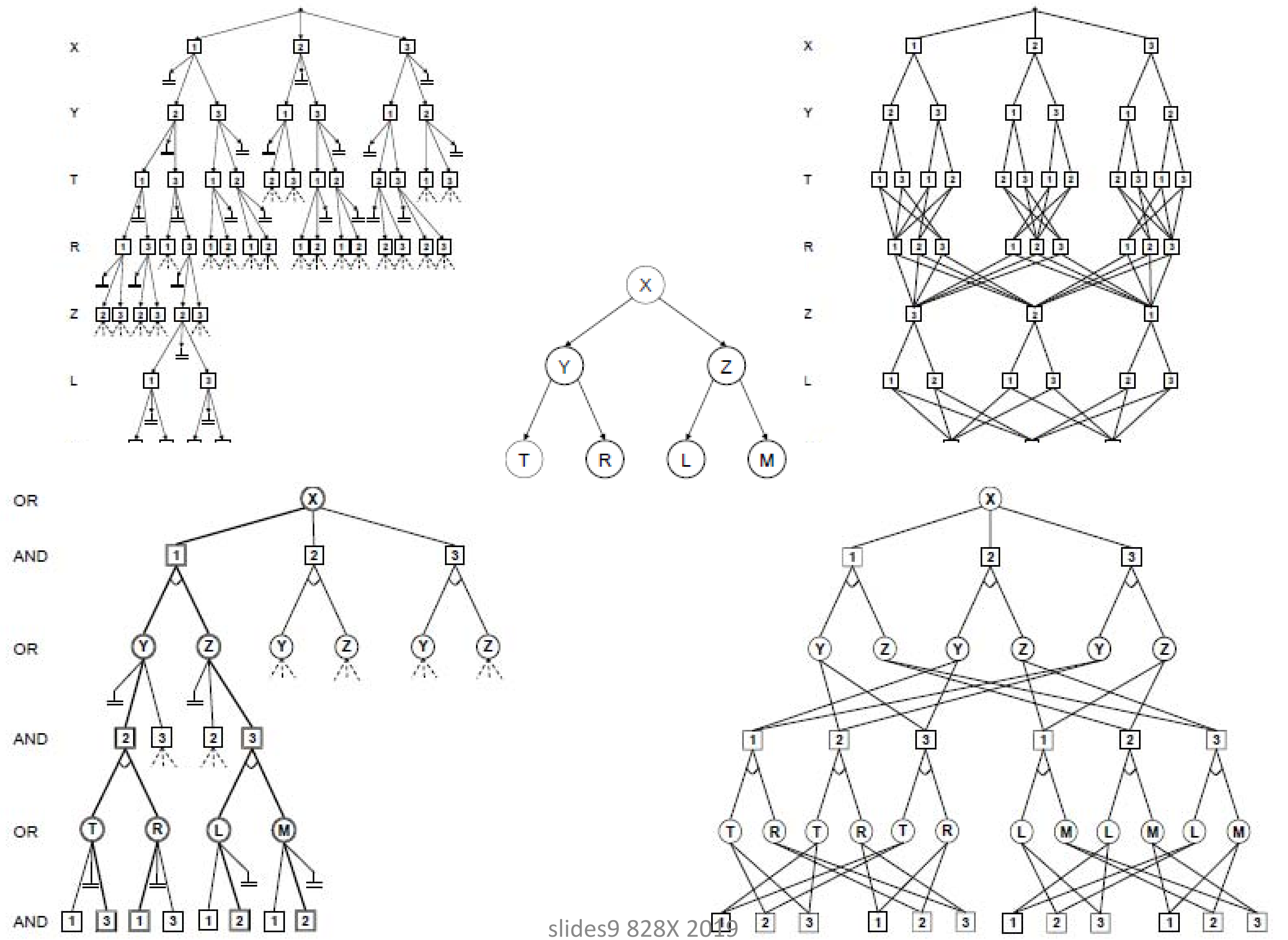
Types of queries

Max-Inference	$f(\mathbf{x}^*) = \max_{\mathbf{x}} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$
Sum-Inference	$Z = \sum_{\mathbf{x}} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$
Mixed-Inference	$f(\mathbf{x}_M^*) = \max_{\mathbf{x}_M} \sum_{\mathbf{x}_S} \prod_{\alpha} f_\alpha(\mathbf{x}_\alpha)$



- **All solved by AND/OR Depth-first search,**
  - **Linear memory, exp(h) time or**
  - **exp(w*) memory and time**
- **But, we can do better by:**
  - **Pruning while searching**
  - **Generating upper and lower bounds anytime**

Search Spaces for a Tree GM



AND/OR Tree DFS Algorithm (Belief Updating)

$P(E \mid A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B \mid A)$

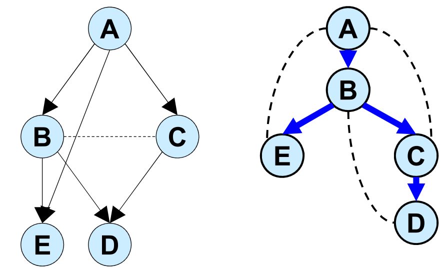
A	B=0	B=1
0	.4	.6
1	.1	.9

$P(C \mid A)$

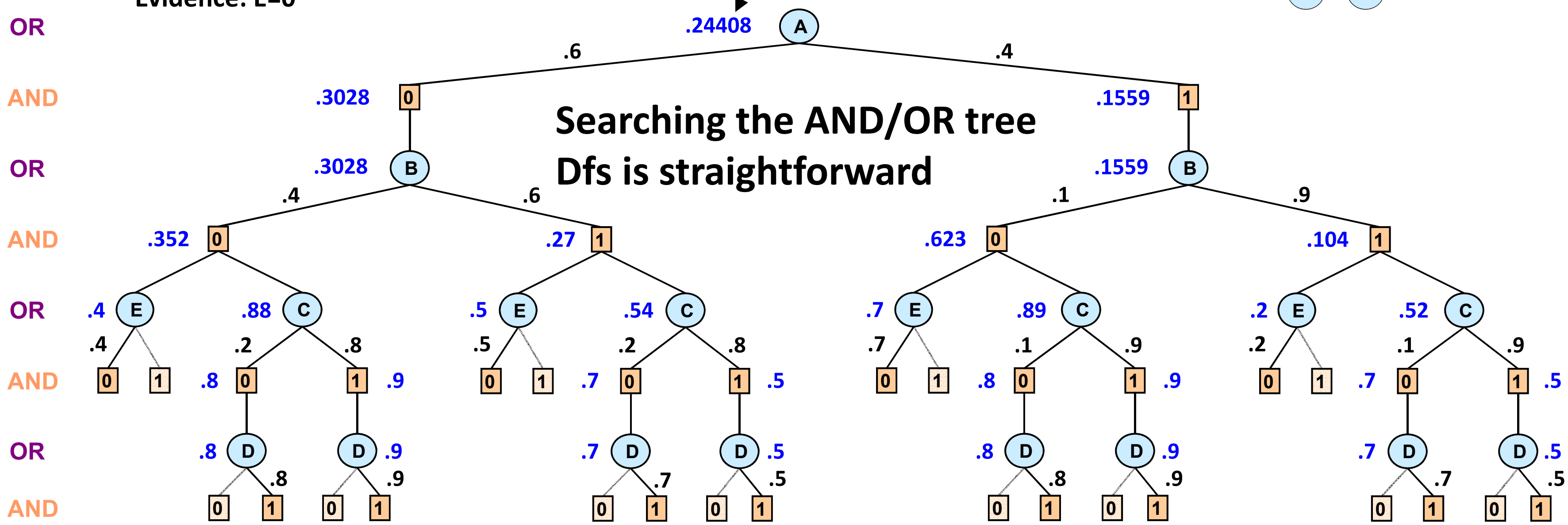
A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4



Result: P(D=1,E=0)



Searching the AND/OR tree
Dfs is straightforward

$P(D \mid B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1

OR node: Marginalization operator (summation)

AND node: Combination operator (product)

Value of node = updated belief for sub-problem below

AND/OR Graph DFS Algorithm (Belief Updating)

$P(E\,|\,A,B)$

A	B	E=0	E=1
0	0	.4	.6
0	1	.5	.5
1	0	.7	.3
1	1	.2	.8

Evidence: E=0

$P(B\,|\,A)$

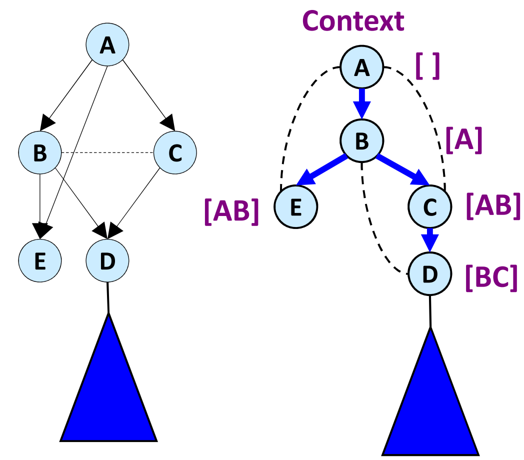
A	B=0	B=1
0	.4	.6
1	.1	.9

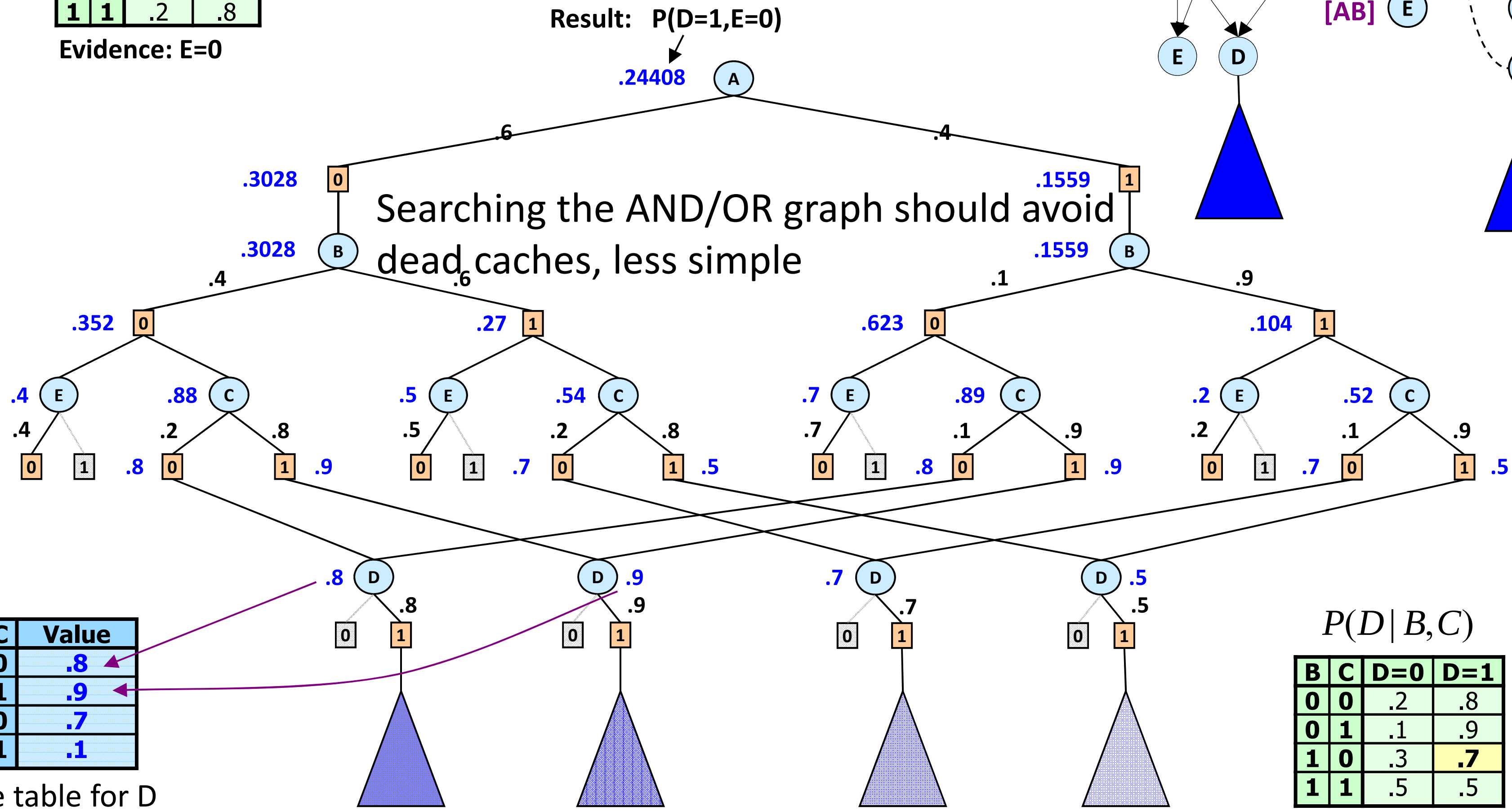
$P(C\,|\,A)$

A	C=0	C=1
0	.2	.8
1	.7	.3

$P(A)$

A	P(A)
0	.6
1	.4





Searching the AND/OR graph should avoid dead caches, less simple

B	C	Value
0	0	.8
0	1	.9
1	0	.7
1	1	.1

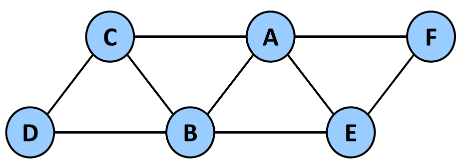
Cache table for D

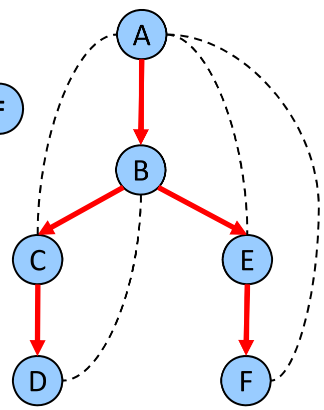
$P(D\,|\,B,C)$

B	C	D=0	D=1
0	0	.2	.8
0	1	.1	.9
1	0	.3	.7
1	1	.5	.5

Evidence: D=1

AND/OR Search Graph (Optimization)





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

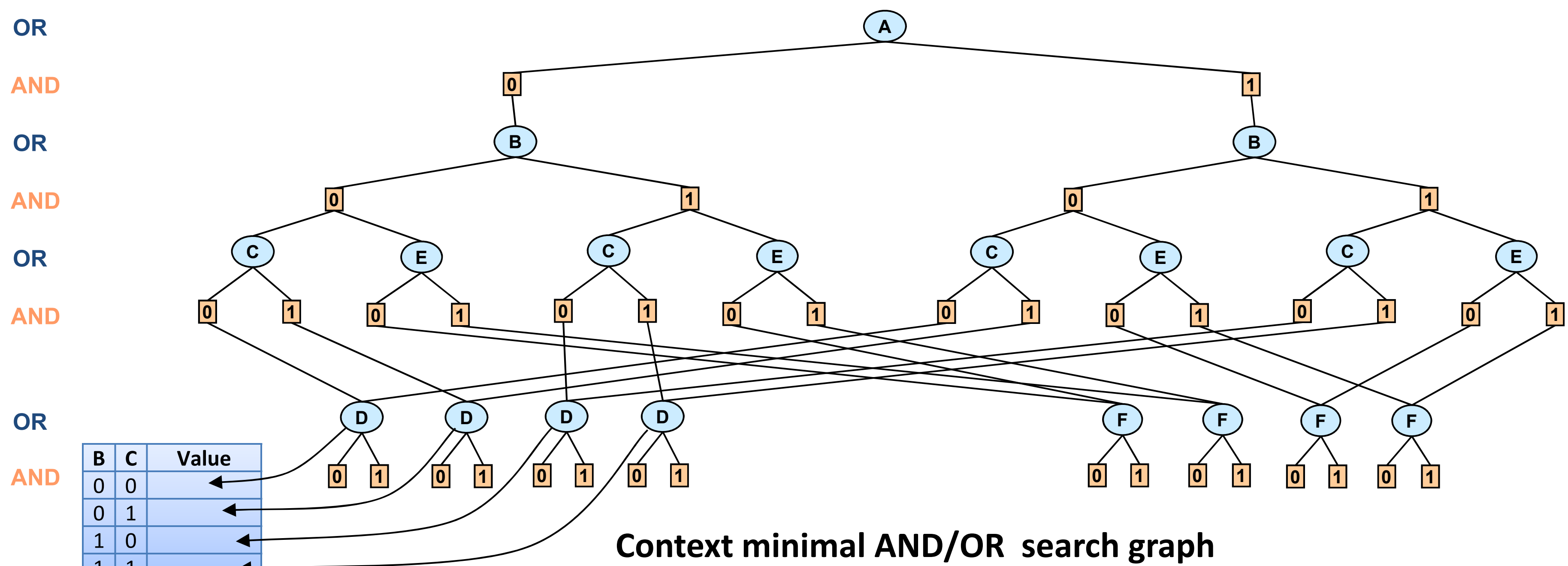
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

Objective function: $F^* = \min_x \sum_\alpha f_\alpha(x_\alpha)$



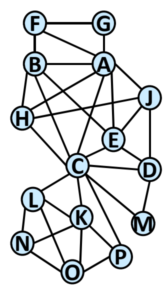
B	C	Value
0	0	
0	1	
1	0	
1	1	

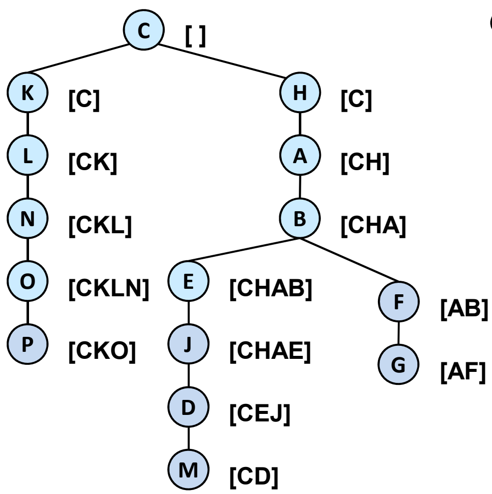
Cache table for **D**

Context minimal AND/OR search graph

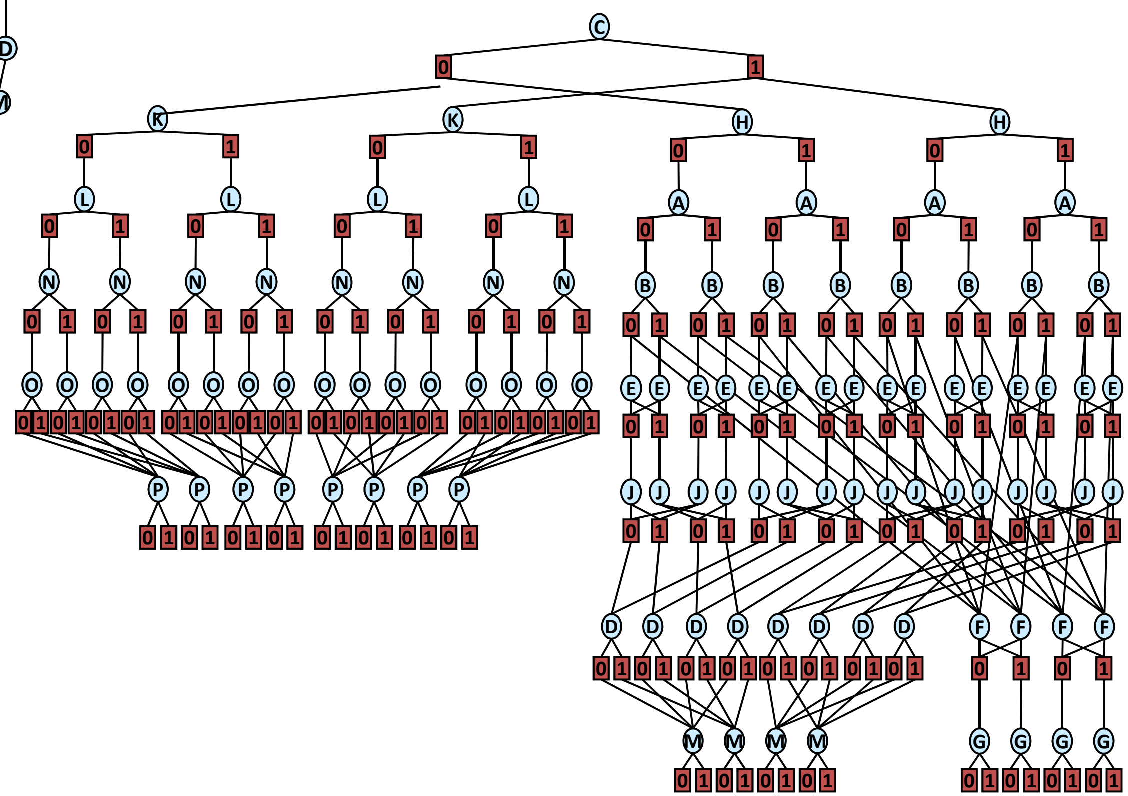
Dead Caches

Definition 8.1.9 (dead cache) *If X is the parent of Y in pseudo-tree $\mathcal{T}$, and $context(X) \subset context(Y)$, then $context(Y)$ represents a* dead cache.





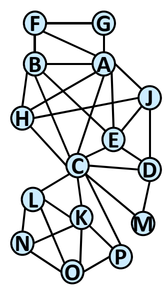
(C K H A B E J L N O D P M F G)

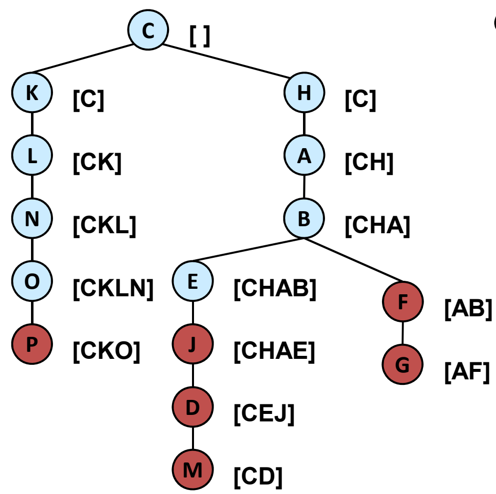


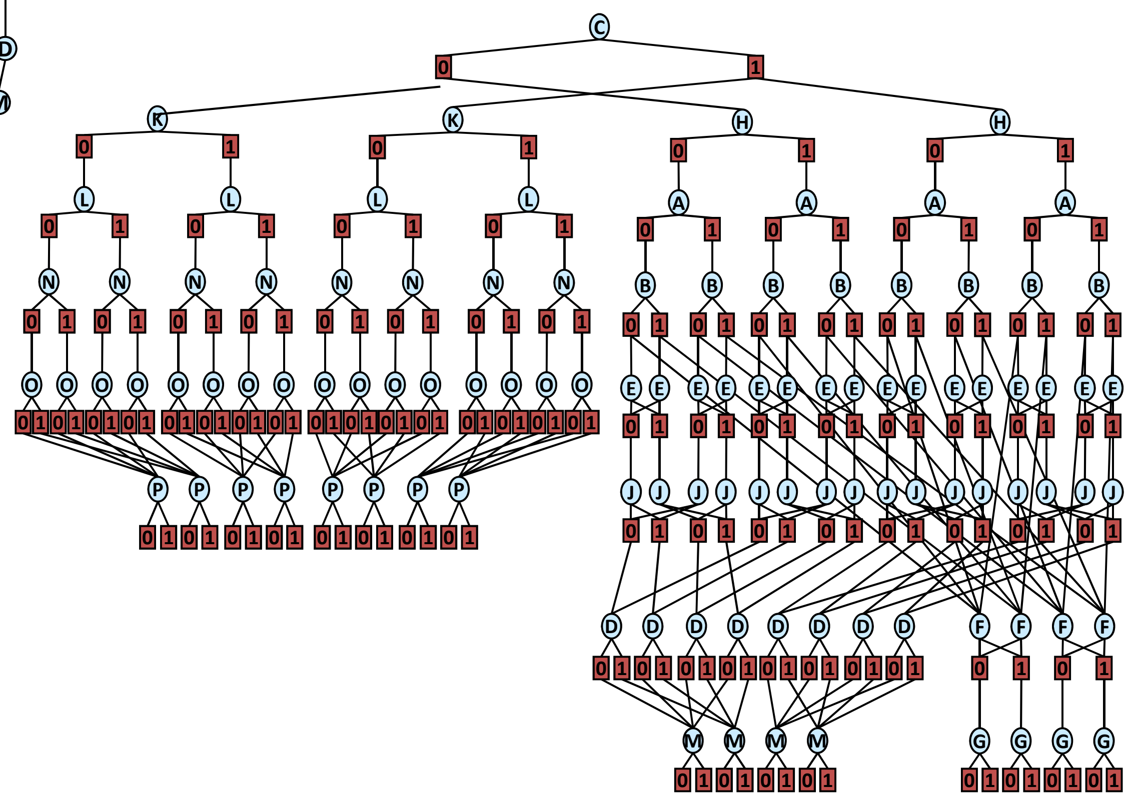
(Darwiche 2000)

Dead Caches

Definition 8.1.9 (dead cache) *If X is the parent of Y in pseudo-tree $\mathcal{T}$, and $context(X) \subset context(Y)$, then $context(Y)$ represents a* dead cache.







(C K H A B E J L N O D P M F G)

(Darwiche 2000)

Algorithm 2: AO-COUNTING / AO-BELIEF-UPDATING

A constraint network $\mathcal{M} = \langle X, D, C \rangle$, or a belief network $\mathcal{P} = \langle X, D, P \rangle$; a pseudo tree $\mathcal{T}$ rooted at X_1; parents pa_i (OR-context) for every variable X_i; caching set to *true* or *false*. The number of solutions, or the updated belief, $v(X_1)$.

if caching $==$ *true* **then** // Initialize cache tables
  Initialize cache tables with entries of "-1"
$v(X_1) \leftarrow 0$; OPEN $\leftarrow \{X_1\}$ // Initialize the stack OPEN
while OPEN $\neq \varphi$ **do**
  n $\leftarrow$ *top*(OPEN); remove n from OPEN
  if caching $==$ *true* **and** n *is OR, labeled* X_i **and** $Cache(asgn(\pi_n)[pa_i]) \neq -1$ **then** // In cache
    $v(\mathrm{n}) \leftarrow Cache(asgn(\pi_n)[pa_i])$ // Retrieve value
    $successors(\mathrm{n}) \leftarrow \varphi$ // No need to expand below
  else // **EXPAND**
    if n *is an OR node labeled* X_i **then** // OR-expand
      $successors(\mathrm{n}) \leftarrow \{\langle X_i, x_i \rangle \mid \langle X_i, x_i \rangle$ is consistent with $\pi_n \}$
      $v(\langle X_i, x_i \rangle) \leftarrow 1$, for all $\langle X_i, x_i \rangle \in successors(\mathrm{n})$
      $v(\langle X_i, x_i \rangle) \leftarrow \prod_{f \in B_{\mathcal{T}}(X_i)} f(asgn(\pi_n)[pa_i])$, for all $\langle X_i, x_i \rangle \in successors(\mathrm{n})$ // AO-BU
    if n *is an AND node labeled* $\langle X_i, x_i \rangle$ **then** // AND-expand
      $successors(\mathrm{n}) \leftarrow children_{\mathcal{T}}(X_i)$
      $v(X_i) \leftarrow 0$ for all $X_i \in successors(\mathrm{n})$
    Add $successors(\mathrm{n})$ to top of OPEN
  while $successors(\mathrm{n}) == \varphi$ **do** // **PROPAGATE**
    if n *is an OR node labeled* X_i **then**
      if $X_i == X_1$ **then** // Search is complete
        return $v(\mathrm{n})$
      if caching $==$ *true* **then**
        $Cache(asgn(\pi_n)[pa_i]) \leftarrow v(\mathrm{n})$ // Save in cache
      $v(\mathrm{p}) \leftarrow v(\mathrm{p}) * v(\mathrm{c})$
      if $v(\mathrm{p}) == 0$ **then** // Check if p is dead-end
        remove $successors(\mathrm{p})$ from OPEN
        $successors(\mathrm{p}) \leftarrow \varphi$
    if n *is an AND node labeled* $\langle X_i, x_i \rangle$ **then**
      let p be the parent of n
      $v(\mathrm{p}) \leftarrow v(\mathrm{p}) + v(\mathrm{n})$;
    remove n from $successors(\mathrm{p})$
    n $\leftarrow$ p

Searching AND/OR Graphs

- AND/OR(i): searches depth-first, cache i-context
  - i = the max size of a cache table (i.e. number of variables in a context)

i=0   **i**   i=w*

Space: O(n)

Time: O(exp(w* log n))

Space: O(exp w*)

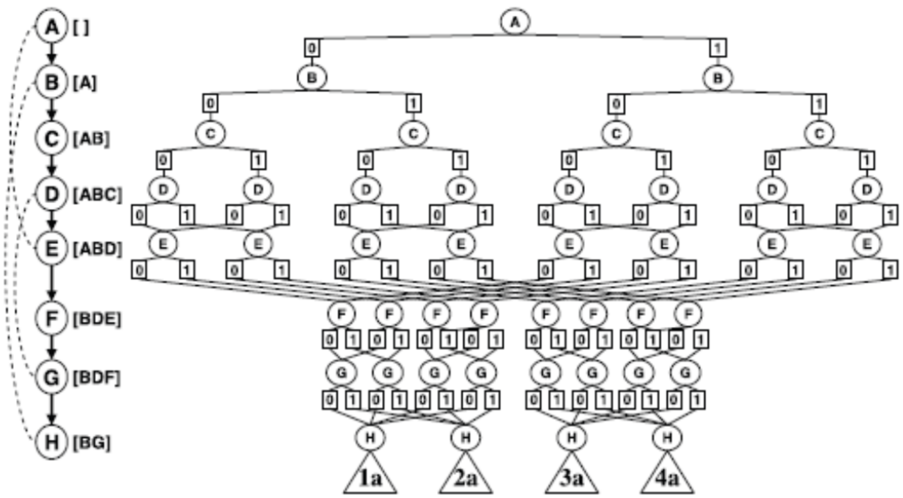
Time: O(exp w*)

Space: O(exp(i))

Time: O(exp(m_i+i)

m_i is related to the size of the i-cutset.

Different Levels of Caching



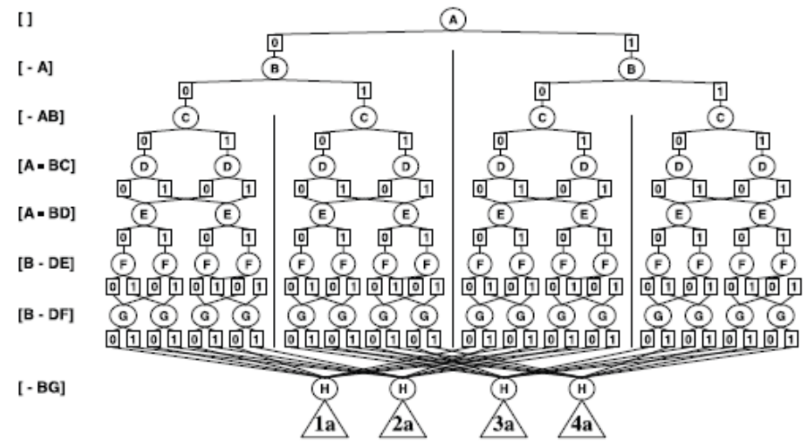


Figure 7.11:
AOC(2) graph (adaptive caching).

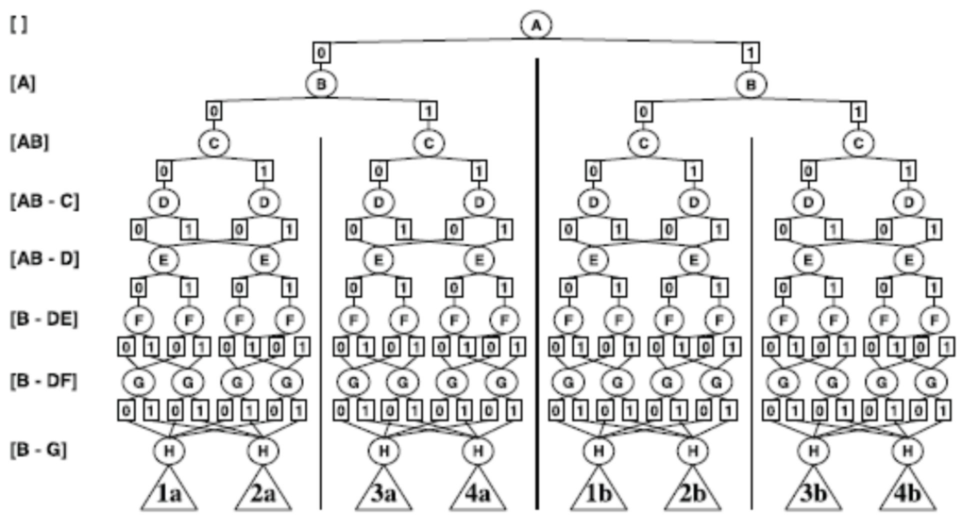


Figure 7.12: AOCutset(2) graph (AND/OR Cutset).

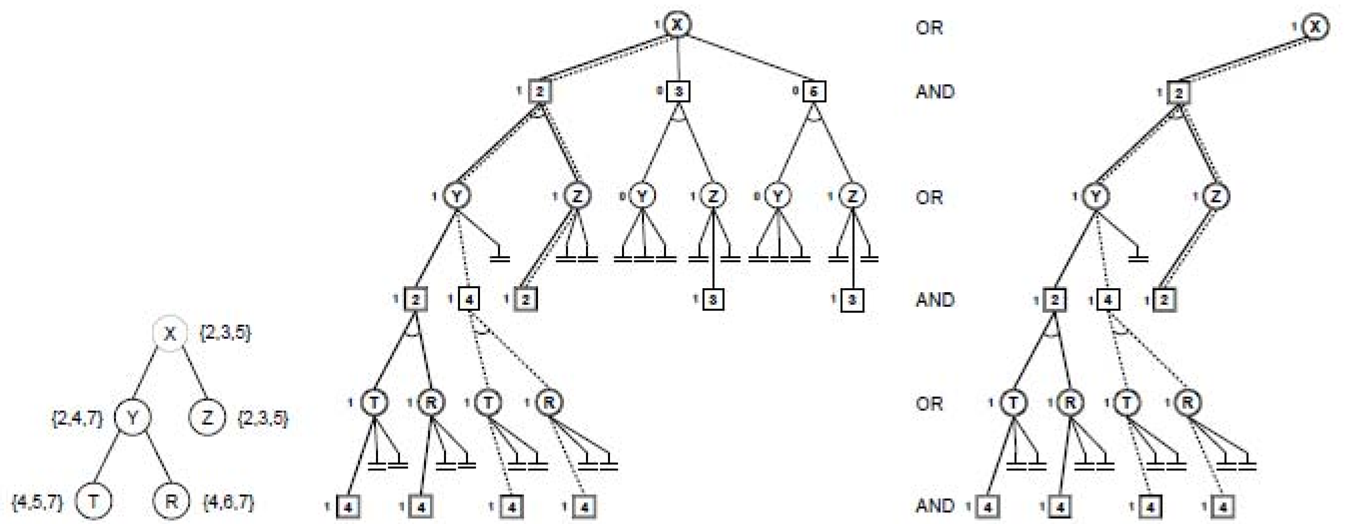
Search for Mixed Deterministic and Probabilistic Graphical Models

AND/OR Search for Mixed Networks

Definition 8.2.1 (backtrack-free AND/OR search tree) *Given graphical model $\mathcal{M}$ and given an AND/OR search tree $S_{\mathcal{T}}(\mathcal{M})$, the* backtrack-free AND/OR search tree *of $\mathcal{M}$ based on $\mathcal{T}$, denoted $BF_{\mathcal{T}}(\mathcal{M})$, is obtained by pruning from $S_{\mathcal{T}}(\mathcal{M})$ all inconsistent subtrees, namely all nodes that root no consistent partial solution.*

- Graph-based No-good and good learning are automatically performed by AND/OR (backjumping) and by caching.

AND/OR Backtrack-Free



(a) A constraint tree (b) Search tree (c) Backtrack-free search tree

Figure 8.1: AND/OR search tree and backtrack-free tree

AND/OR CPE (Constraint Probability Evaluation)

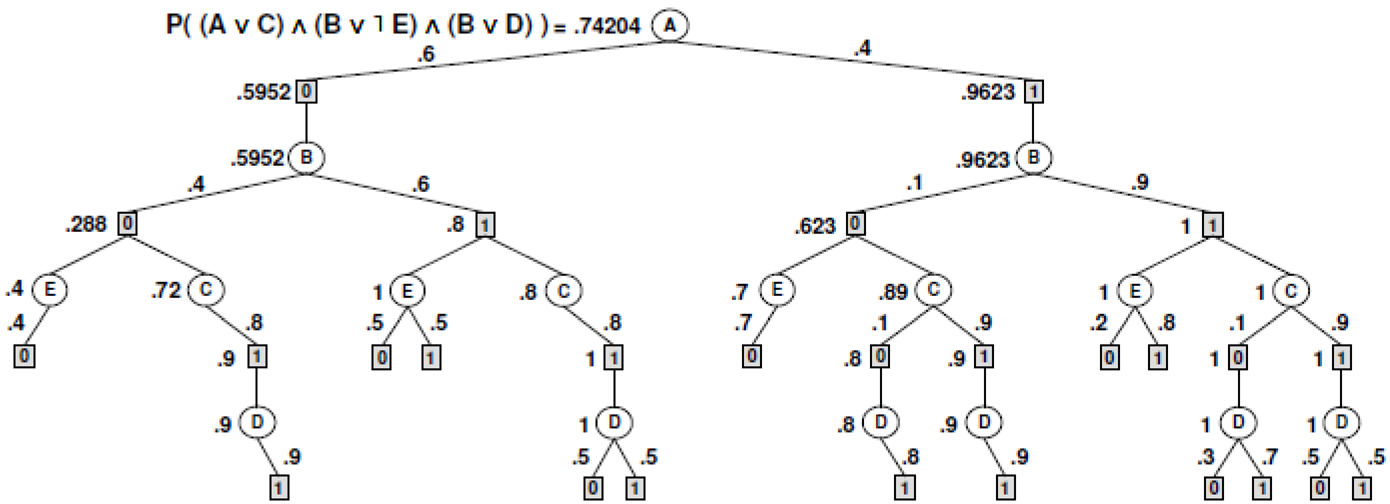
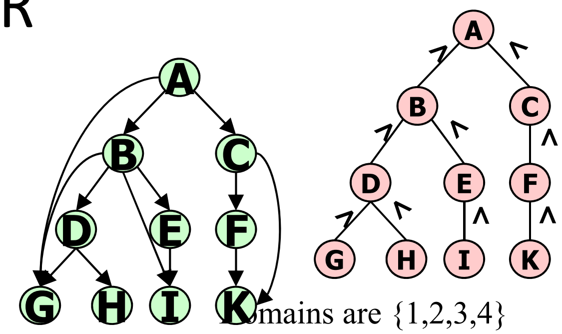
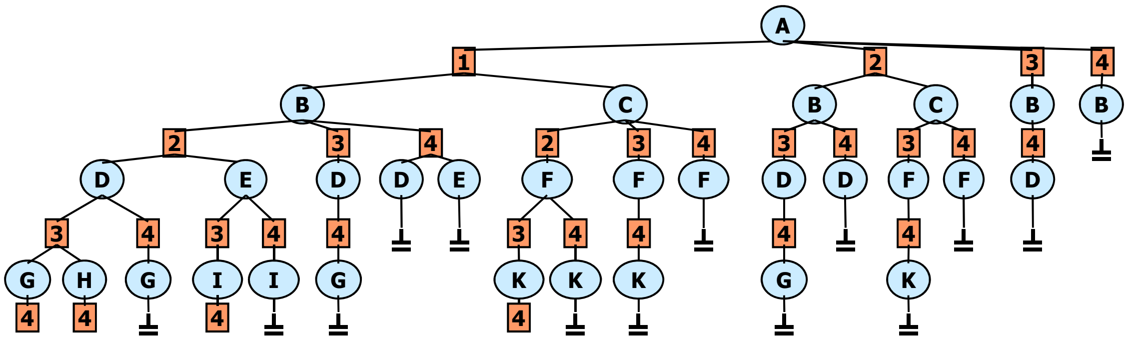


Figure 8.2: Mixed network defined by the query $\varphi = (A \vee C) \wedge (B \vee \neg E) \wedge (B \vee D)$

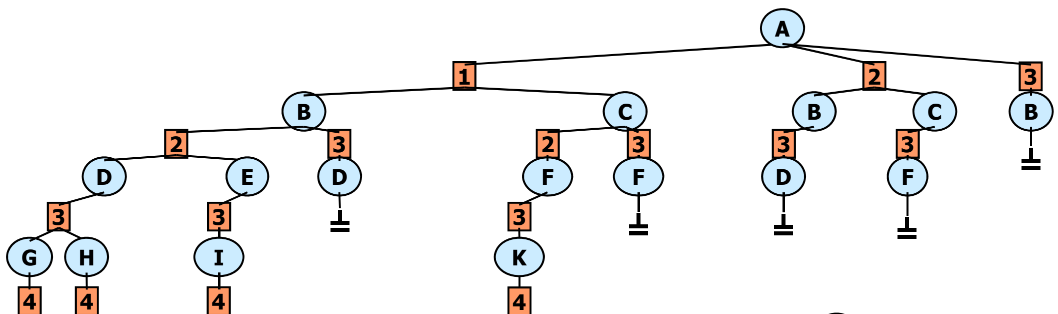
Example 8.2.6 We refer back to the example in Figure 7.4. Consider a constraint network that is defined by the CNF formula $\varphi = (A \vee C) \wedge (B \vee \neg E) \wedge (B \vee D)$. The trace of algorithm AND-OR-CPE without caching is given in Figure 8.2. Notice that the clause $(A \vee C)$ is not satisfied if $A = 0$ and $C = 0$, therefore the paths that contain this assignment cannot be part of a solution of the mixed network. The value of each node is shown to its left (the leaf nodes assume a dummy value of 1, not shown in the figure). The value of the root node is the probability of φ. Figure 8.2 is similar to Figure 7.4. In Figure 7.4 the evidence can be modeled as the CNF formula with unit clauses $D \wedge \neg E$. □

The Effect of Constraint Propagation in AND/OR

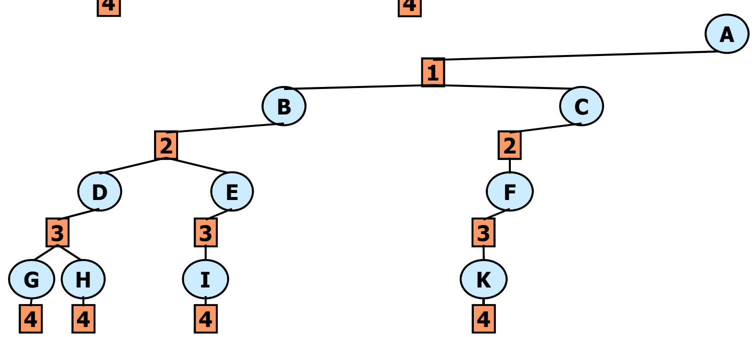




CONSTRAINTS ONLY



FORWARD CHECKING



MAINTAINING ARC CONSISTENCY

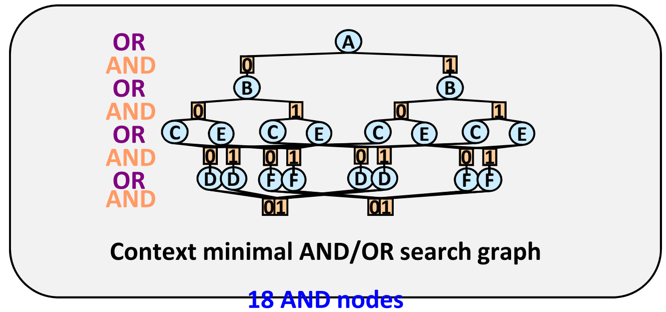
Available code

- http://graphmod.ics.uci.edu/group/Software

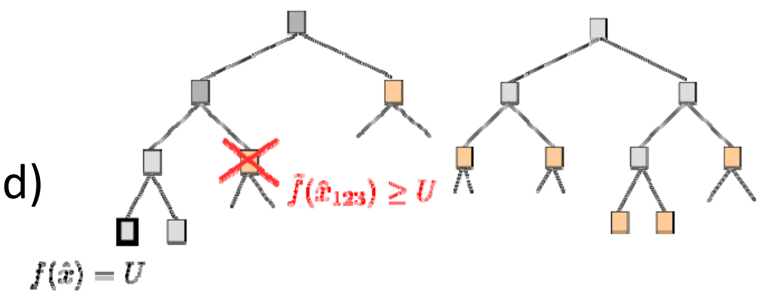
Outline: Search

- Review Graphical Modes

- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - **Generating good pseudo-trees**
  - Brute-force search



- Heuristic search (HS) for AND/OR spaces
  - **Basic Heuristic search (Depth and Best)**
  - AND/OR Depth-first HS (branch and bound)
  - AND/OR Best-first heuristic search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)



- Hybrids of search and Inference
- Summary and Class 2

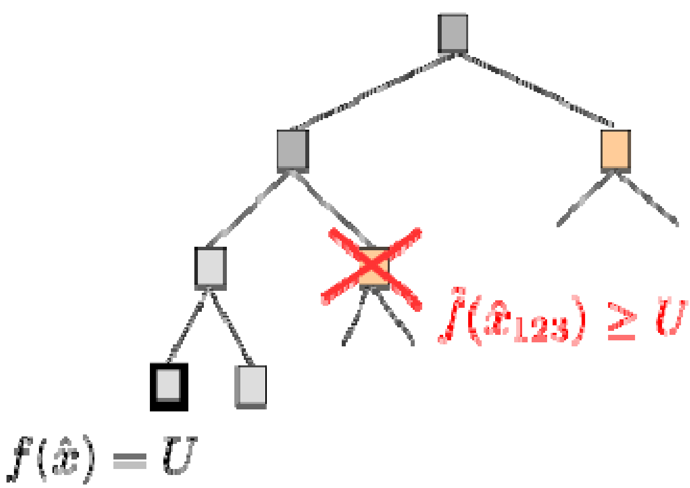
Basic Heuristic Search

We assume min-sum problems in the following

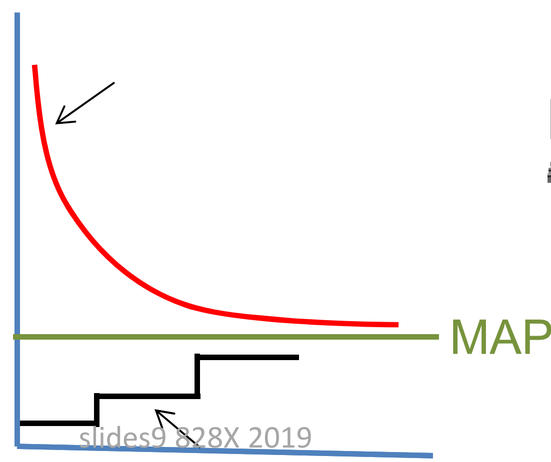
Heuristic function $\tilde{f}(\hat{x}_p)$ computes a lower bound on the best extension of partial configuration $\hat{x}_p$ and can be used to guide heuristic search.
We focus on:

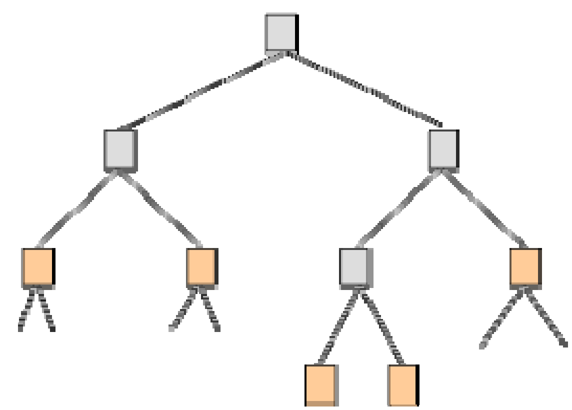
1. Branch-and-Bound
Use heuristic function $\tilde{f}(\hat{x}_p)$ to prune the depth-first search tree
Linear space

2. Best-First Search
Always expand the node with the lowest heuristic value $\tilde{f}(\hat{x}_p)$
Needs lots of memory



BnB is upper-bound anytime

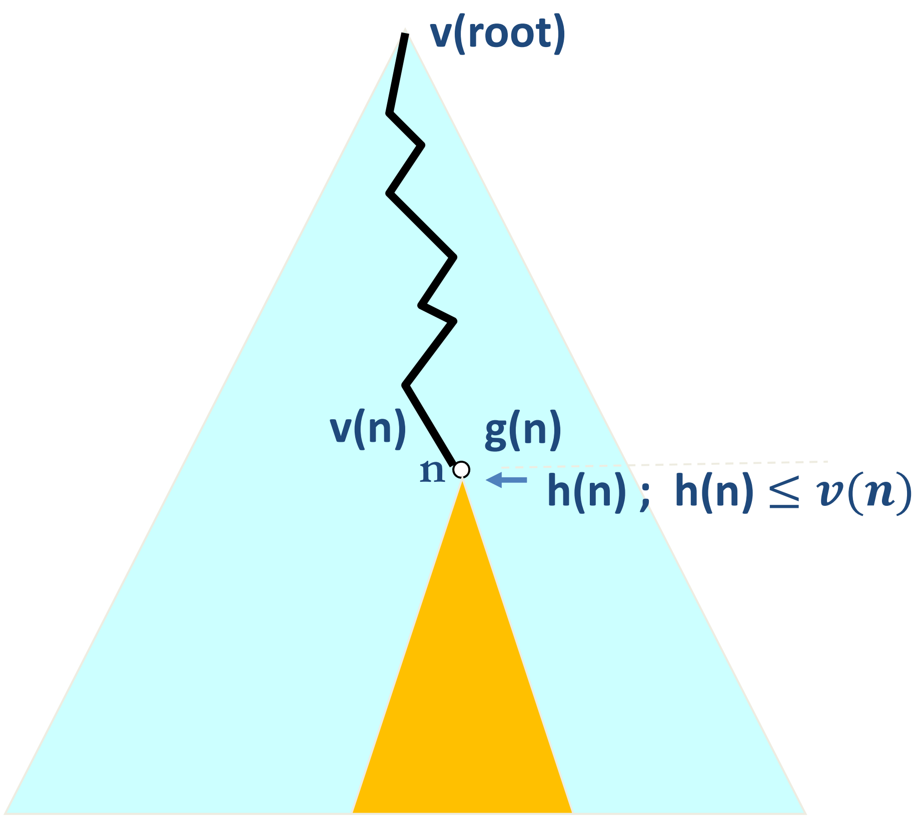




Basic Heuristic Search; **Best-First**

Task: compute v(root): MAP, Marginal ,MMAP

Each node is a sub-problem
(defined by current conditioning)

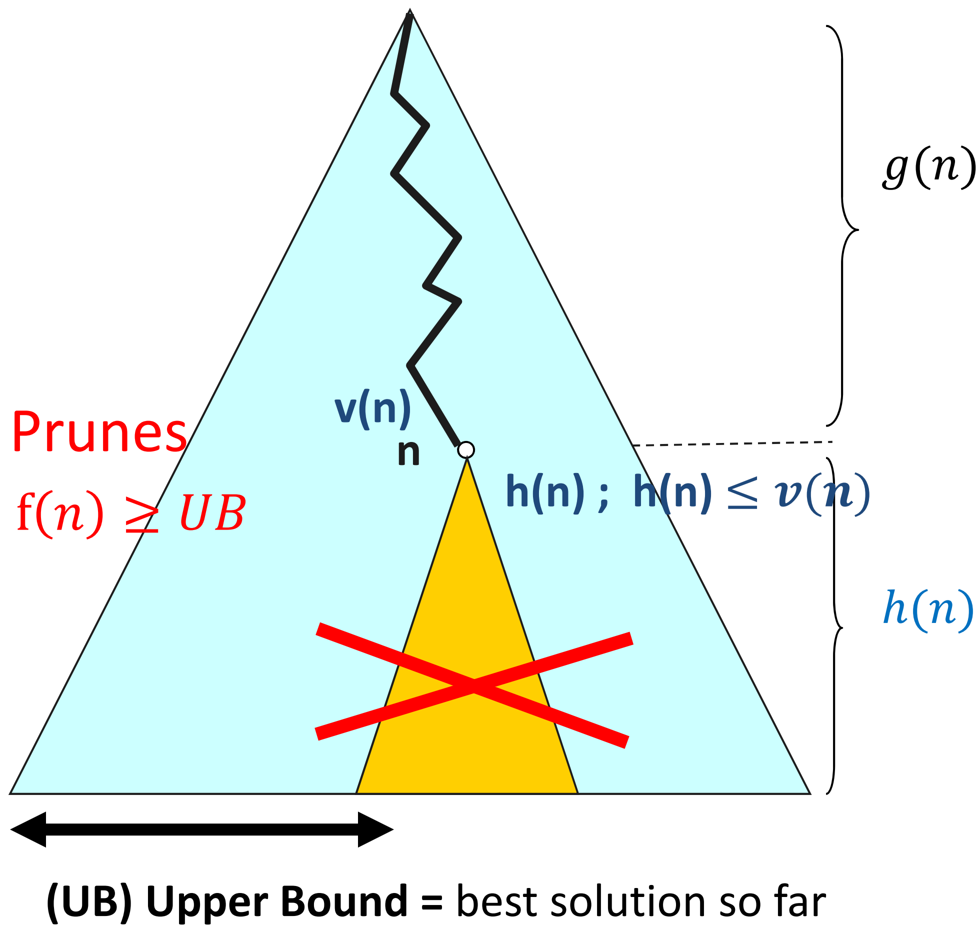


- **Best-First Algorithms, (A*)**
  - Expand nodes in OPEN list in order of min f(n)
  - Terminates with first full solution (for mpe)
- **Properties**
  - Optimal, if **h(n)** $\le v(n)$
  - Expands least set of nodes
  - exponential memory
  - **Not anytime solution for MAP**
  - **Yields lower bounds on value, anytime**

f(n) = g(n) + h(n) $\le g(n) + v(n)$ =f*(n)

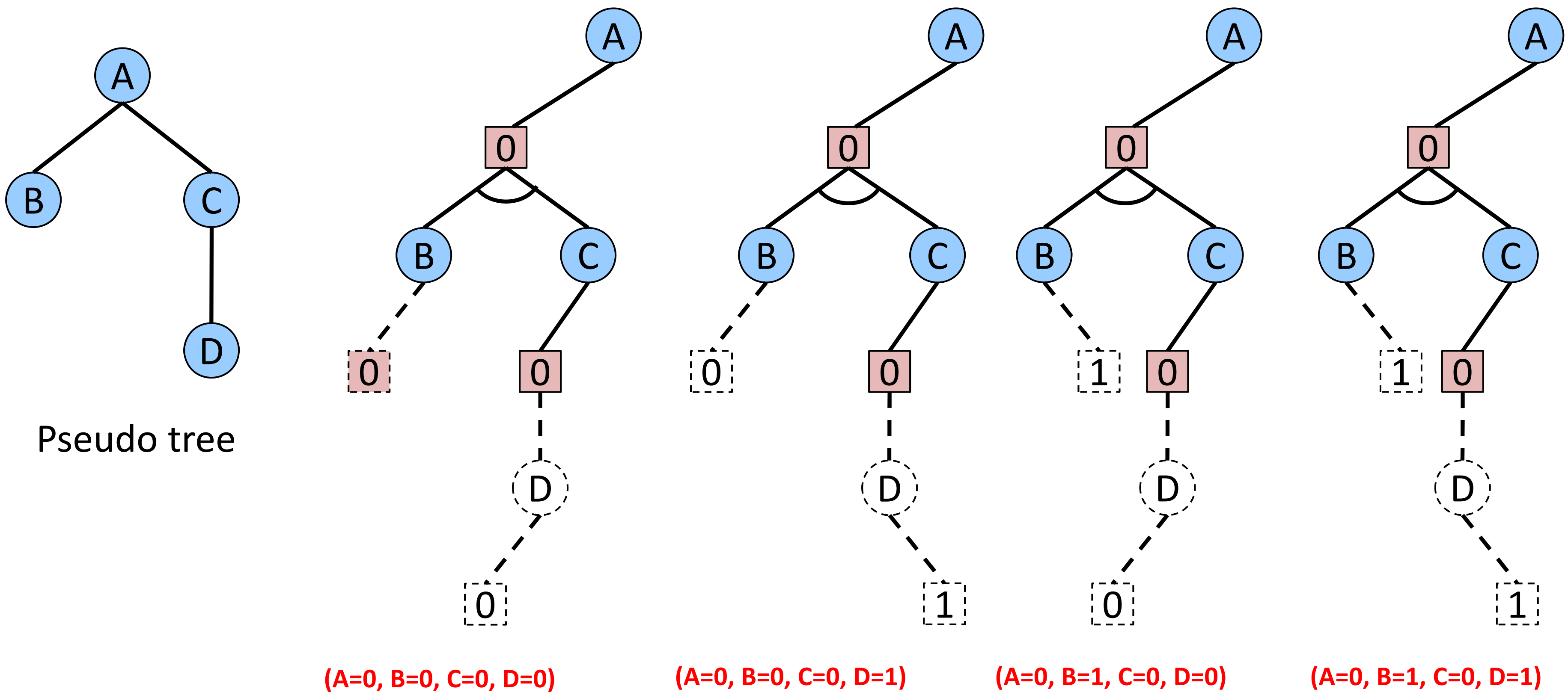
f(n) is a lower bound on best cost through n

Basic Heuristic Search; **Depth-First**



- **Depth-First (B&B for MAP)**
  - Expand in dfs order
  - Update UB with each solution
  - Prunes if f(n) ≥ UB
- **Properties**
  - Can use only linear memory
  - Yields upper bounds anytime

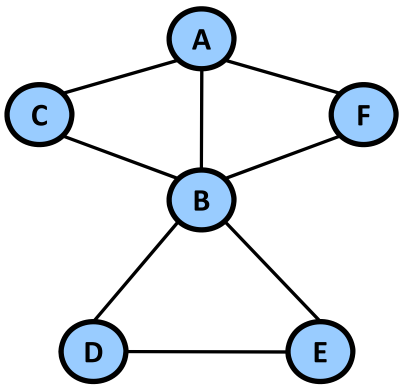
Partial Solution Tree for AND/OR



Extension(T') – solution trees that extend T'
g(T') = conditioned value of a node
V(T') = the combined value below T'
f*(T') = conditioned value through T'

Exact Evaluation Function

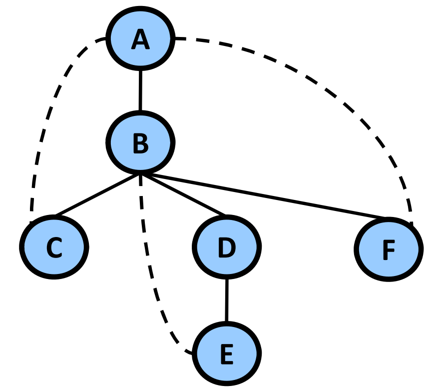
Conditioned value of a node

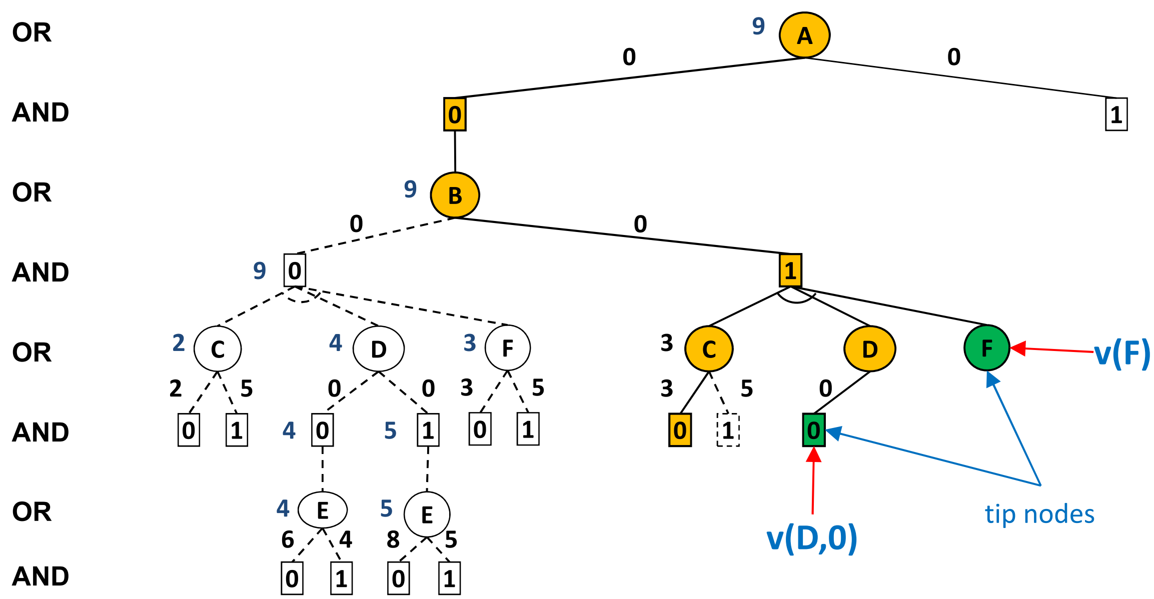


A	B	C	$f_1(ABC)$
0	0	0	2
0	0	1	5
0	1	0	3
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	2

A	B	F	$f_2(ABF)$
0	0	0	3
0	0	1	5
0	1	0	1
0	1	1	4
1	0	0	6
1	0	1	5
1	1	0	6
1	1	1	5

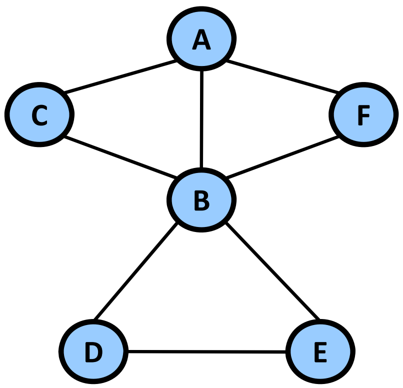
B	D	E	$f_3(BDE)$
0	0	0	6
0	0	1	4
0	1	0	8
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	4





f*(T') = w(A,0) + w(B,1) + w(C,0) + w(D,0) + v(D,0) + v(F)

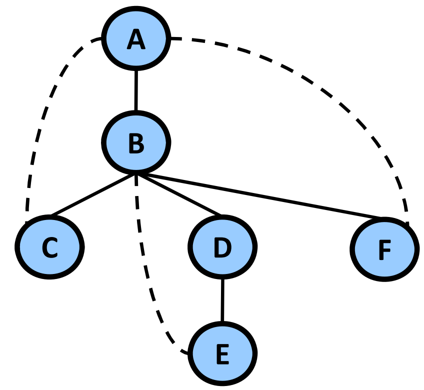
Heuristic Evaluation Function

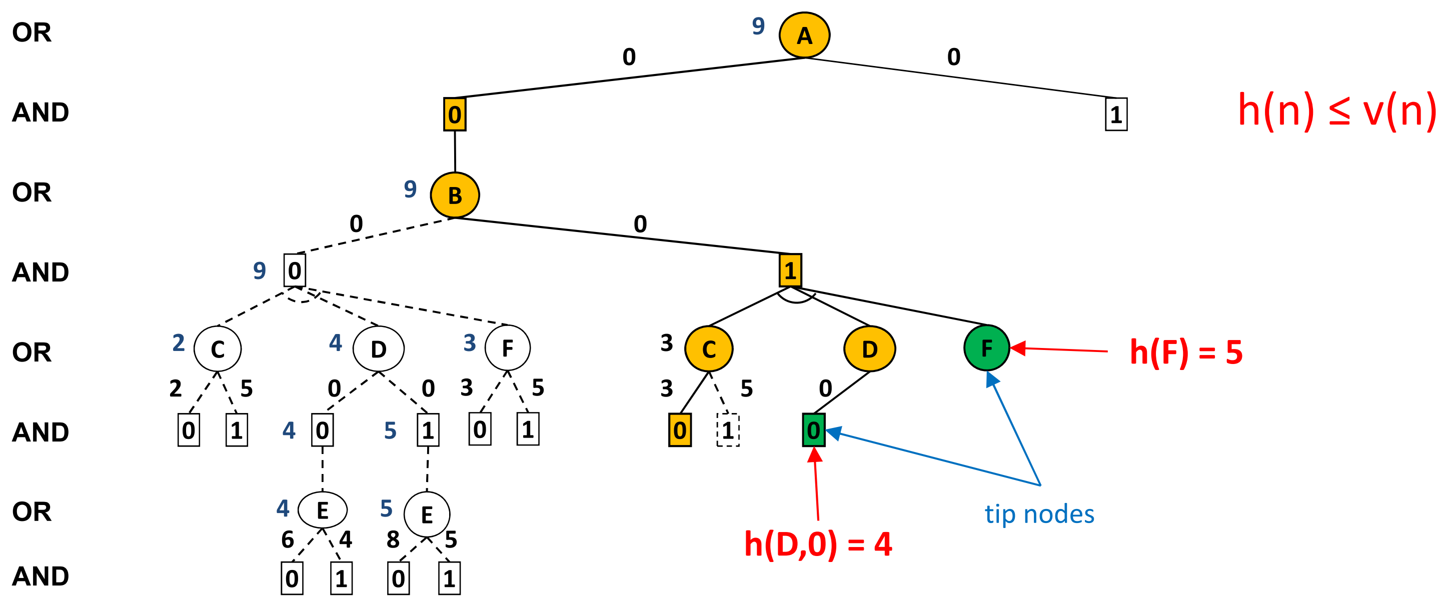


A	B	C	$f_1(ABC)$
0	0	0	**2**
0	0	1	**5**
0	1	0	**3**
0	1	1	**5**
1	0	0	**9**
1	0	1	**3**
1	1	0	**7**
1	1	1	**2**

A	B	F	$f_2(ABF)$
0	0	0	**3**
0	0	1	**5**
0	1	0	**1**
0	1	1	**4**
1	0	0	**6**
1	0	1	**5**
1	1	0	**6**
1	1	1	**5**

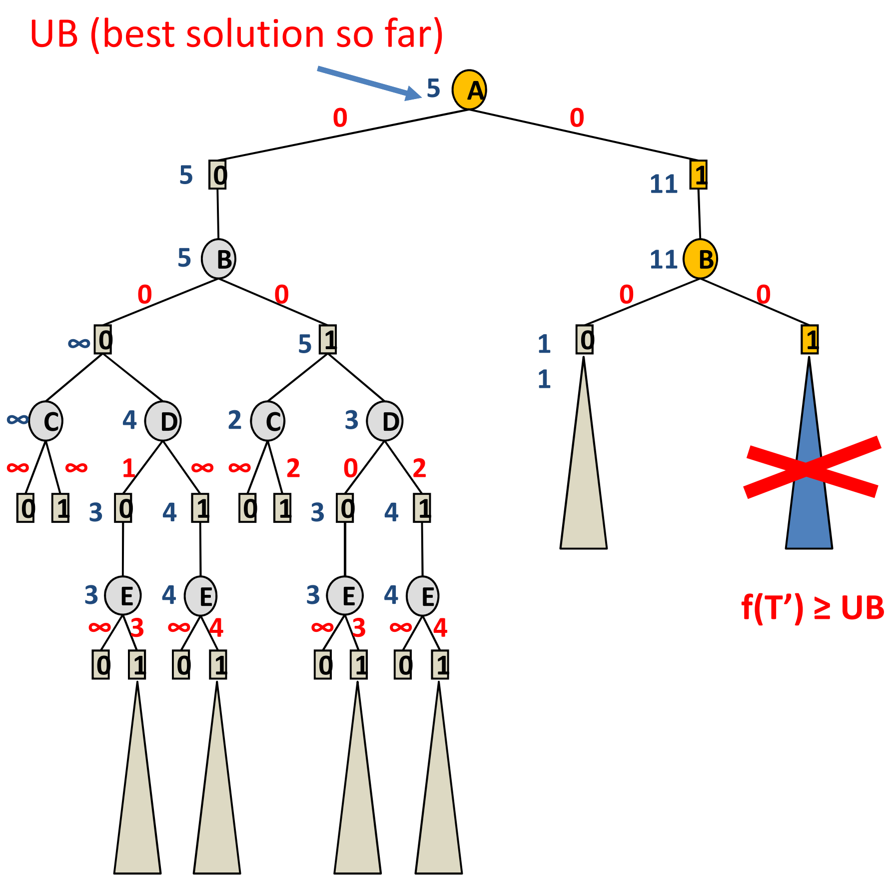
B	D	E	$f_3(BDE)$
0	0	0	**6**
0	0	1	**4**
0	1	0	**8**
0	1	1	**5**
1	0	0	**9**
1	0	1	**3**
1	1	0	**7**
1	1	1	**4**





$$f(T') = w(A,0) + w(B,1) + w(C,0) + w(D,0) + h(D,0) + h(F) = 12 \leq f^*(T')$$

Depth-First AND/OR Branch-and-Bound



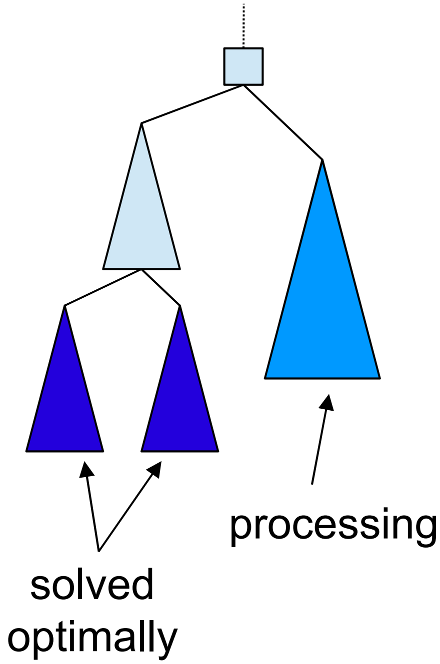
- Associate each node n with a heuristic lower bound h(n) on v(n)

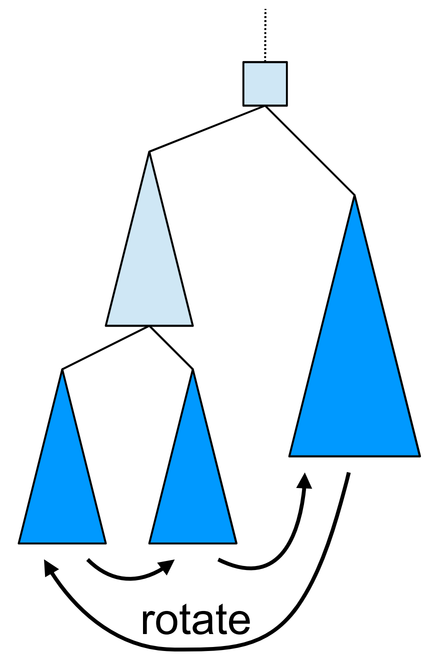
Algorithm AOBB:

- **EXPAND** (top-down)
  - Evaluate f(T') and prune search if f(T') ≥ UB
  - If not in cache, generate successors of the tip node n
- **PROPAGATE** (bottom-up)
  - Update value of the parent p of n
    - OR nodes: minimization
    - AND nodes: summation
  - Cache value of n based on context **only if fully explored**

Anytime Performance

- OR Branch-and-Bound is anytime
- But AND/OR breaks anytime behavior of depth-first scheme:
  - First anytime solution delayed until last sub-problem starts processing
- **Breadth-Rotating AOBB:**
  - Take turns processing sub-problems
    - Limit number of expansions per visit
  - Solve each sub-problem depth-first
    - Maintain favorable complexity bounds





[Otten and Dechter, 2012]

Summary: AND/OR Branch-and-Bound Search (AOBB)

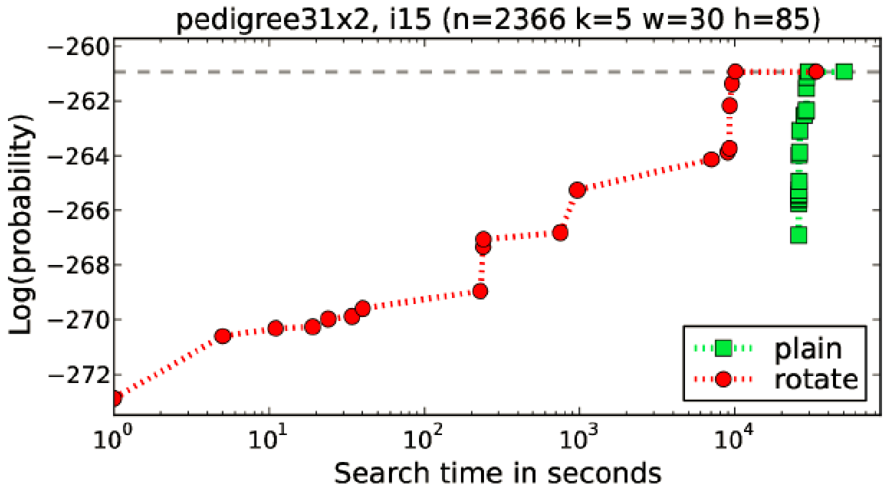
- Associate each node n with a heuristic lower bound h(n) on v(n)

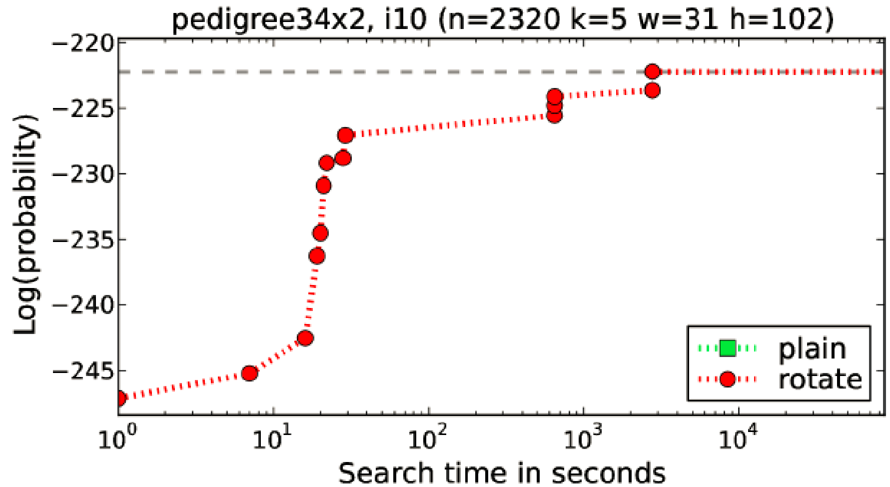
- **EXPAND** (top-down)
  - Evaluate f(T') and prune search if f(T') ≥ UB
  - Generate successors of the tip node n

- **PROPAGATE** (bottom-up)
  - Update value of the parent p of n
    - OR nodes: minimization
    - AND nodes: summation

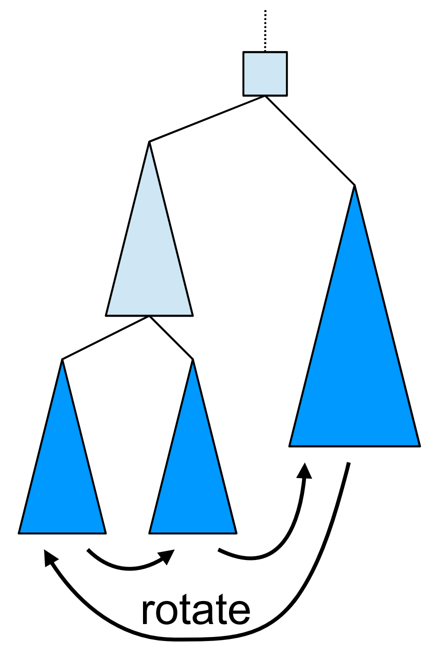
[Marinescu and Dechter, 2005; 2009]

Anytime Performance



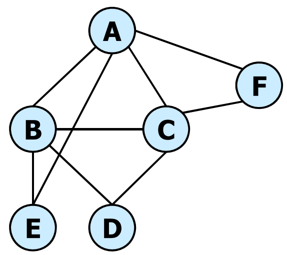


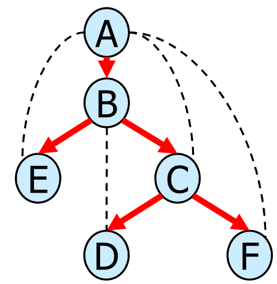
- **Breadth-Rotating AOBB:**
  - Take turns processing sub-problems
    - Limit number of expansions per visit
  - Solve each sub-problem depth-first
    - Maintain favorable complexity bounds

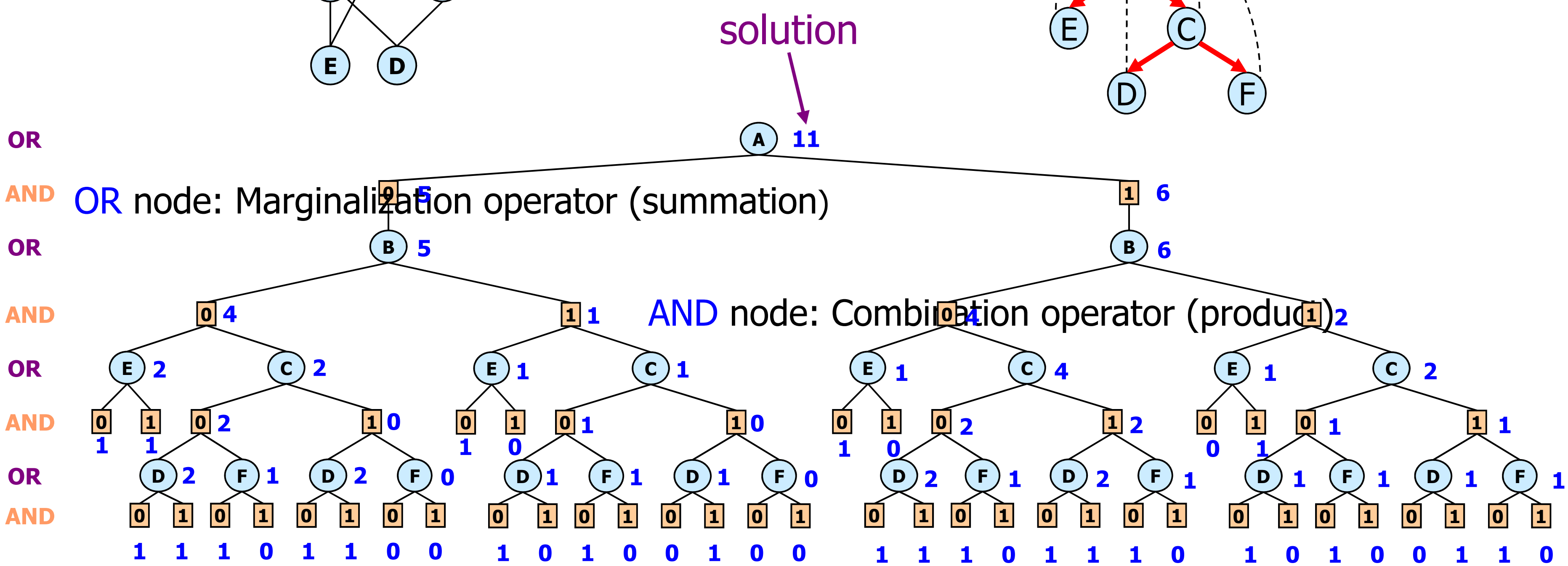


[Otten and Dechter, 2012]

DFS Algorithm (#CSP Example)





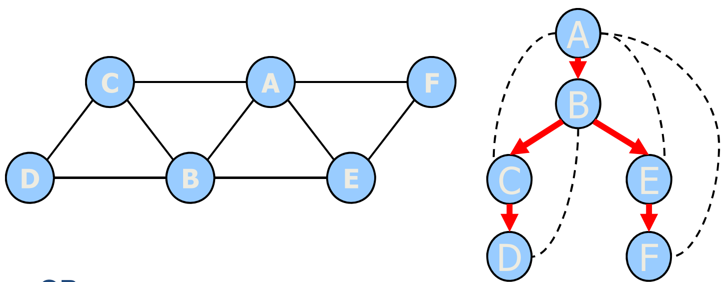


OR node: Marginalization operator (summation)

AND node: Combination operator (product)

Value of node = number of solutions below it

AND/OR Tree Search for Optimization



A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

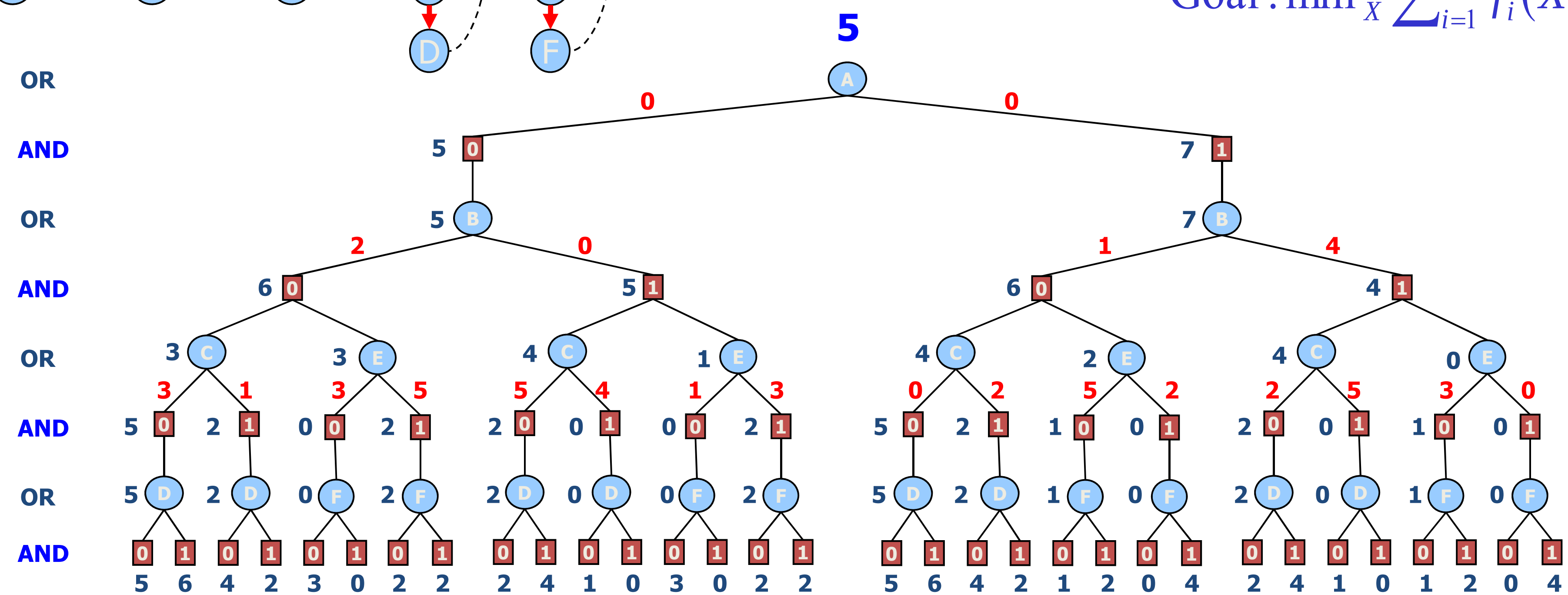
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

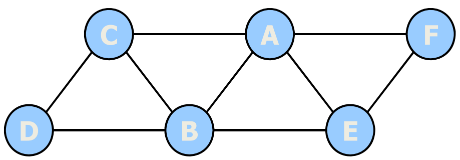
$$\text{Goal} : \min_X \sum_{i=1}^{9} f_i(X)$$

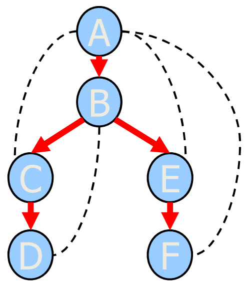


AND node = Combination operator (summation)

OR node = Marginalization operator (minimization)

AND/OR Tree Search for Optimization





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

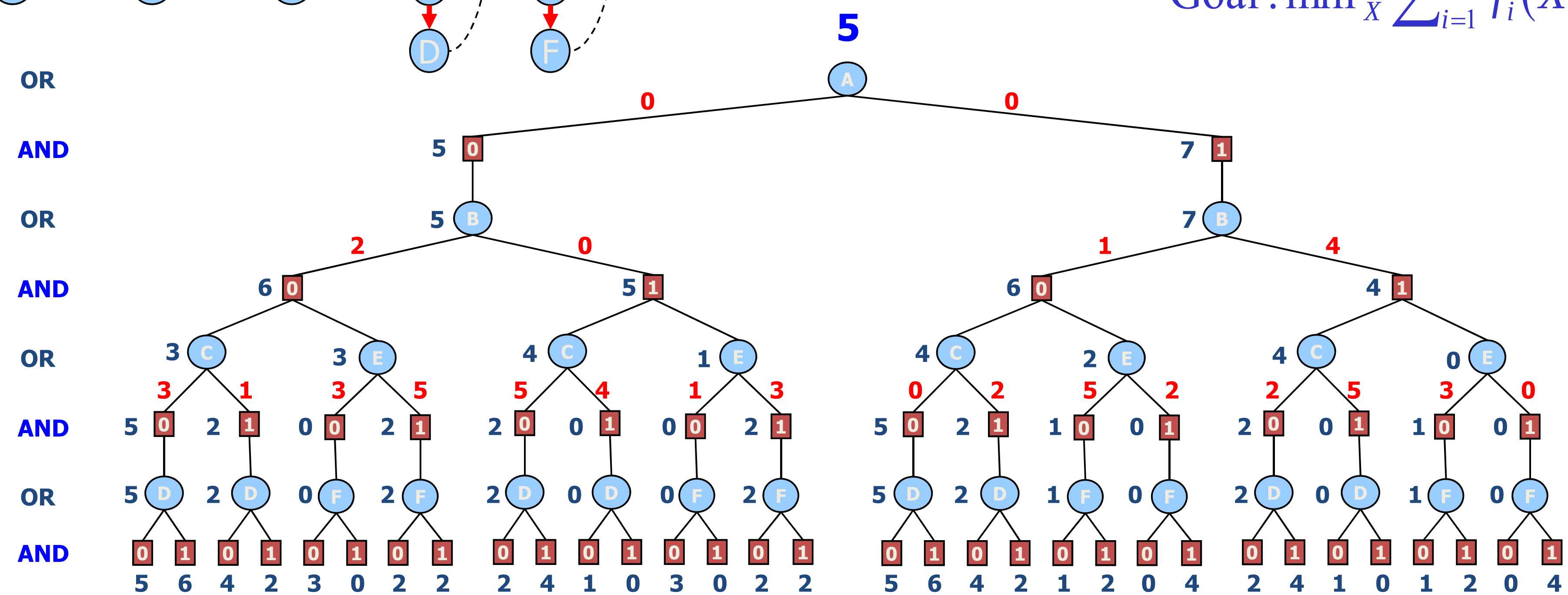
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

$$\text{Goal} : \min_X \sum_{i=1}^{9} f_i(X)$$

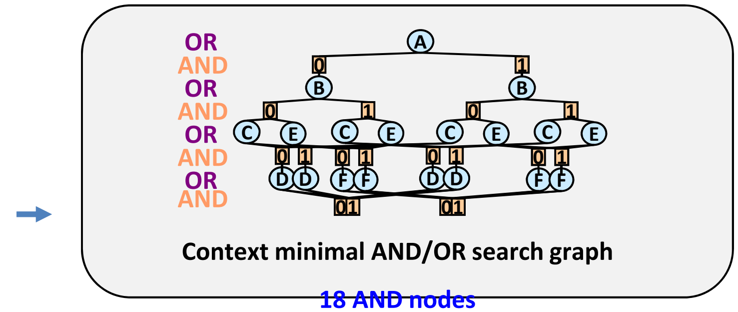


AND node = Combination operator (summation)

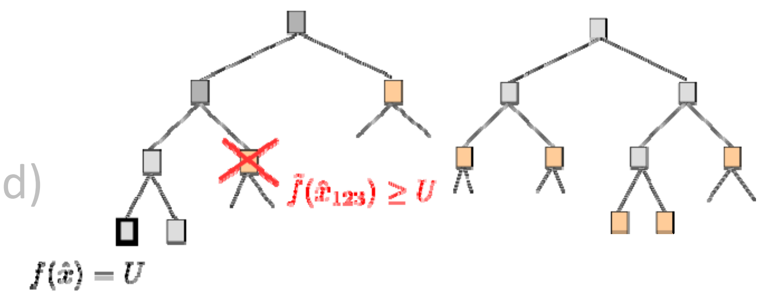
OR node = Marginalization operator (minimization)

Outline: Search

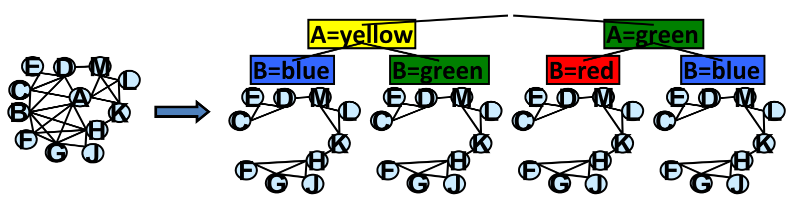
- Review Graphical Modes
- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - Generating good pseudo-trees
  - Brute-force search



- Heuristic search (HS) for AND/OR spaces
  - Basic Heuristic search (Depth and Best)
  - AND/OR Depth-first HS (branch and bound)
  - AND/OR Best-first heuristic search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)

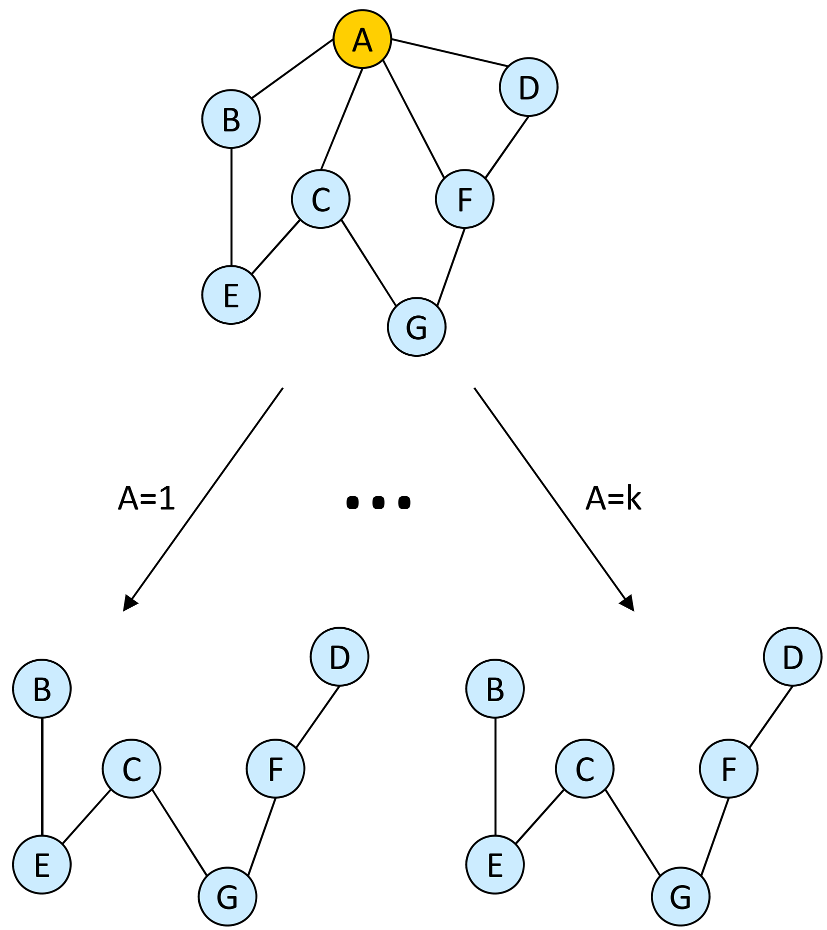


- **Hybrids of search and Inference**
- Summary and Class 2



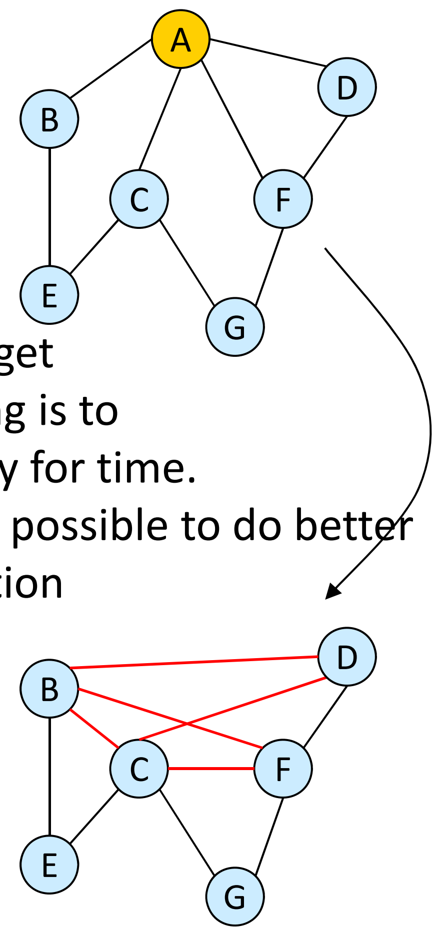
Conditioning versus Elimination

Conditioning (search)



k “sparser” problems

Elimination (inference)



The main target
In conditioning is to
Trade memory for time.
It is not really possible to do better
Then elimination

1 “denser” problem

The Idea of Cutset-Conditioning

We observed that when variables are assigned, connectivity reduces.
The magnitude of saving is reflected through the “conditioned-induced graph”

- Cutset-conditioning exploit this in a systematic way:
- Select a subset of variables, assign them values, and
- Solve the conditioned problem by bucket-elimination.
- Repeat for all assignments to the cutset.

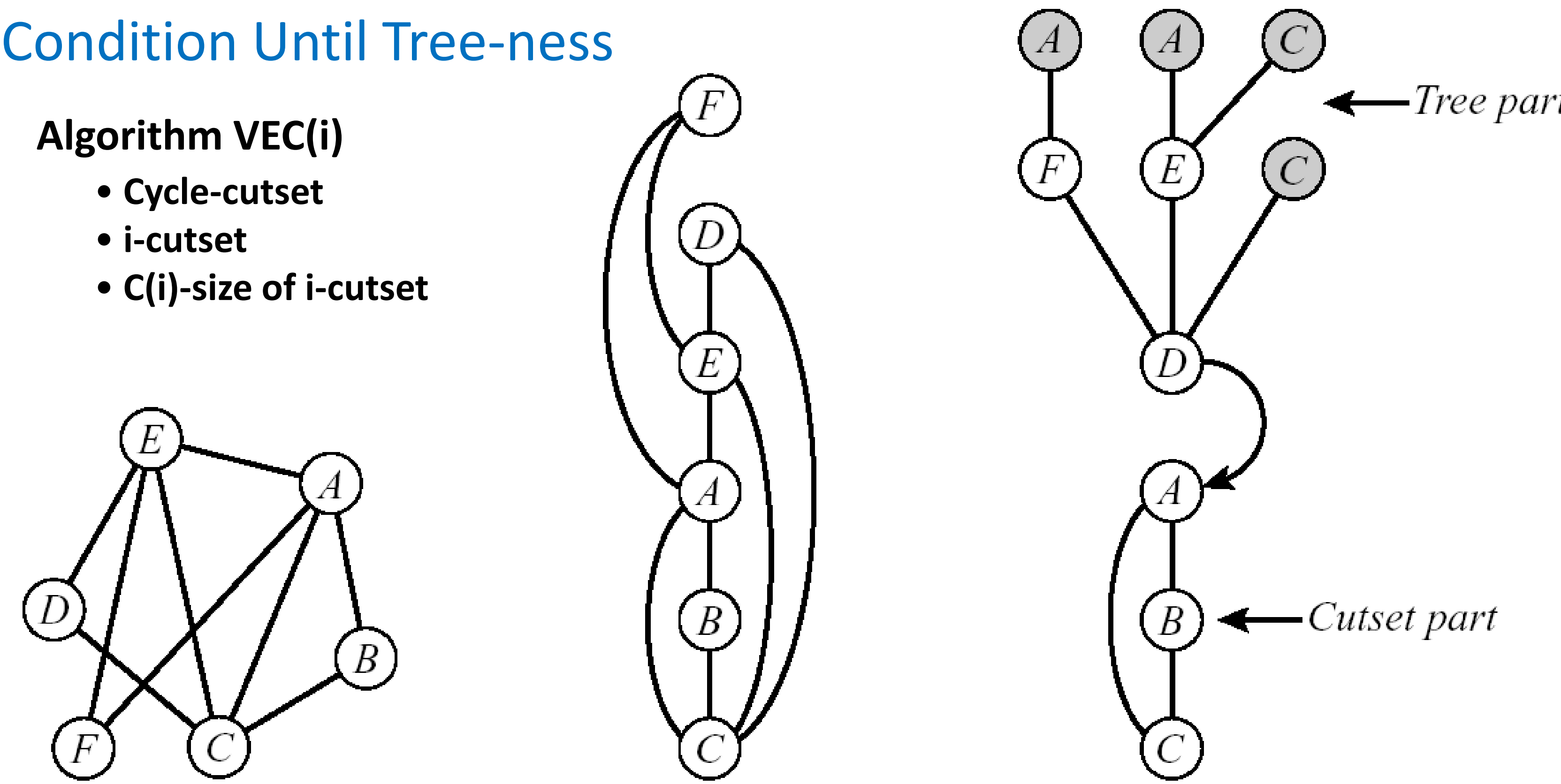
Algorithm VEC

Hybrid: (i)-Cutset-Conditioning

Condition Until Tree-ness

Algorithm VEC(i)

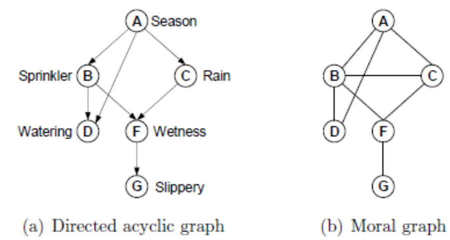
- **Cycle-cutset**
- **i-cutset**
- **C(i)-size of i-cutset**



(a) (b) (c)

Space: exp(i), Time: O(exp(i+c(i)))

The impact of observations



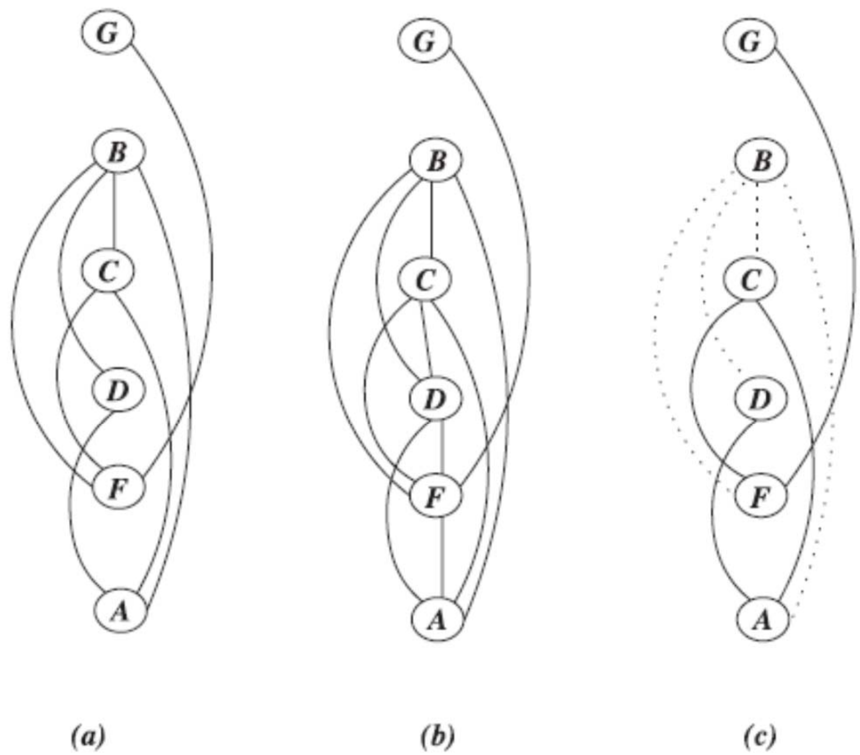
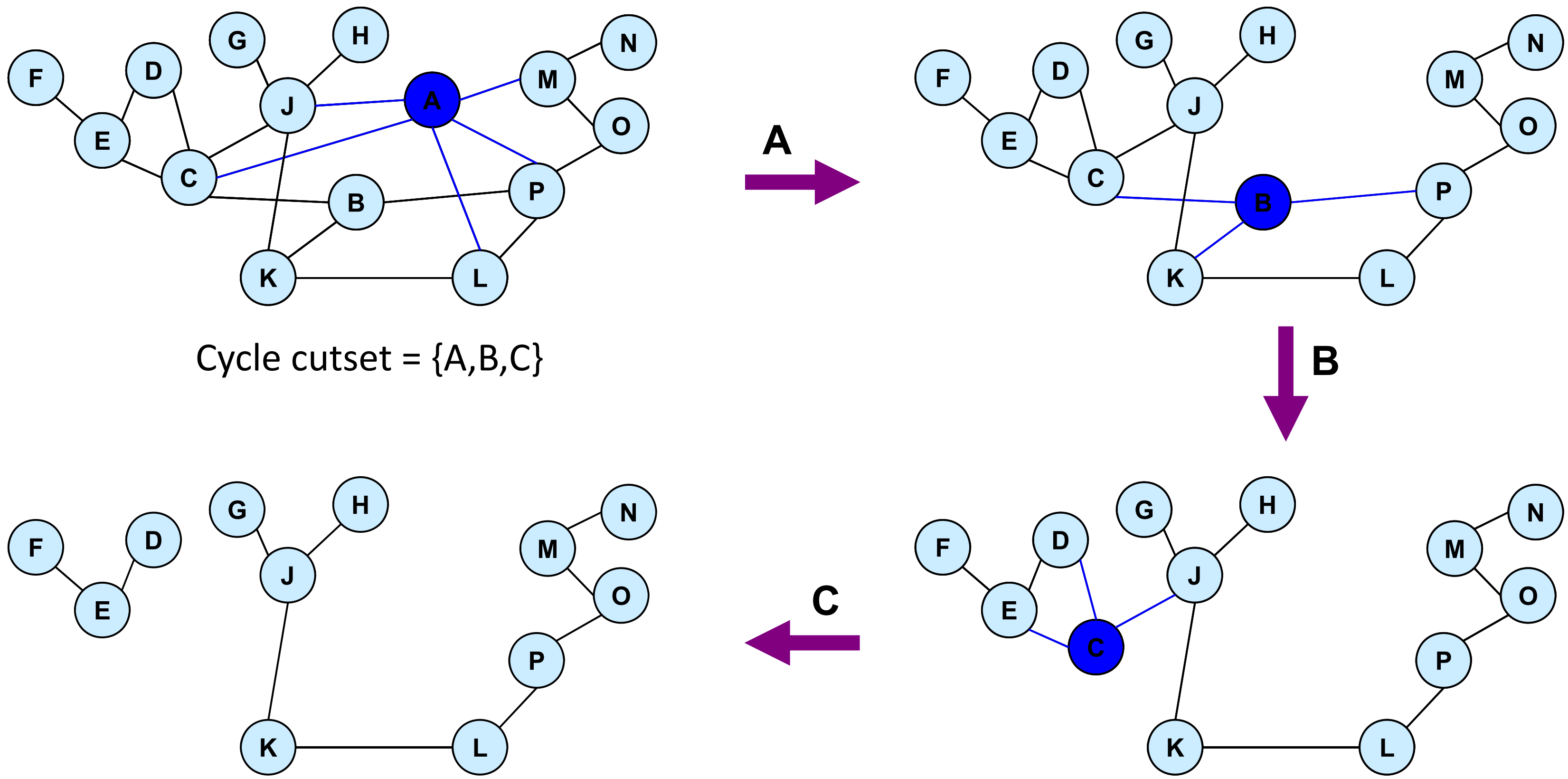


Figure 4.9: Adjusted induced graph relative to observing B.

Ordered graph Induced graph Ordered conditioned graph

A Cycle-Cutset



1-cutset = {A,B,C}, size 3

Loop-Cutset, q-Cutset, Cycle-Cutset

- A loop-cutset is a subset of nodes of a directed graph that when removed the remaining graph is a poly-tree
- A q-cutset is a subset of nodes of an undirected graph that when removed the remaining graph is has an induced-width of q or less.
- A cycle-cutset is a q-cutset such that q=1.

q-Cutset, minimal

Definition 7.3 ***q*-cutset, minimal.** Given a graph G, a subset of nodes is called a *q-cutset* for an integer q iff when removed, the resulting graph has an induced-width less than or equal to q. A minimal *q-cutset* of a graph has a smallest size among all q-cutsets of the graph. A cycle-cutset is a 1-cutset of a graph.

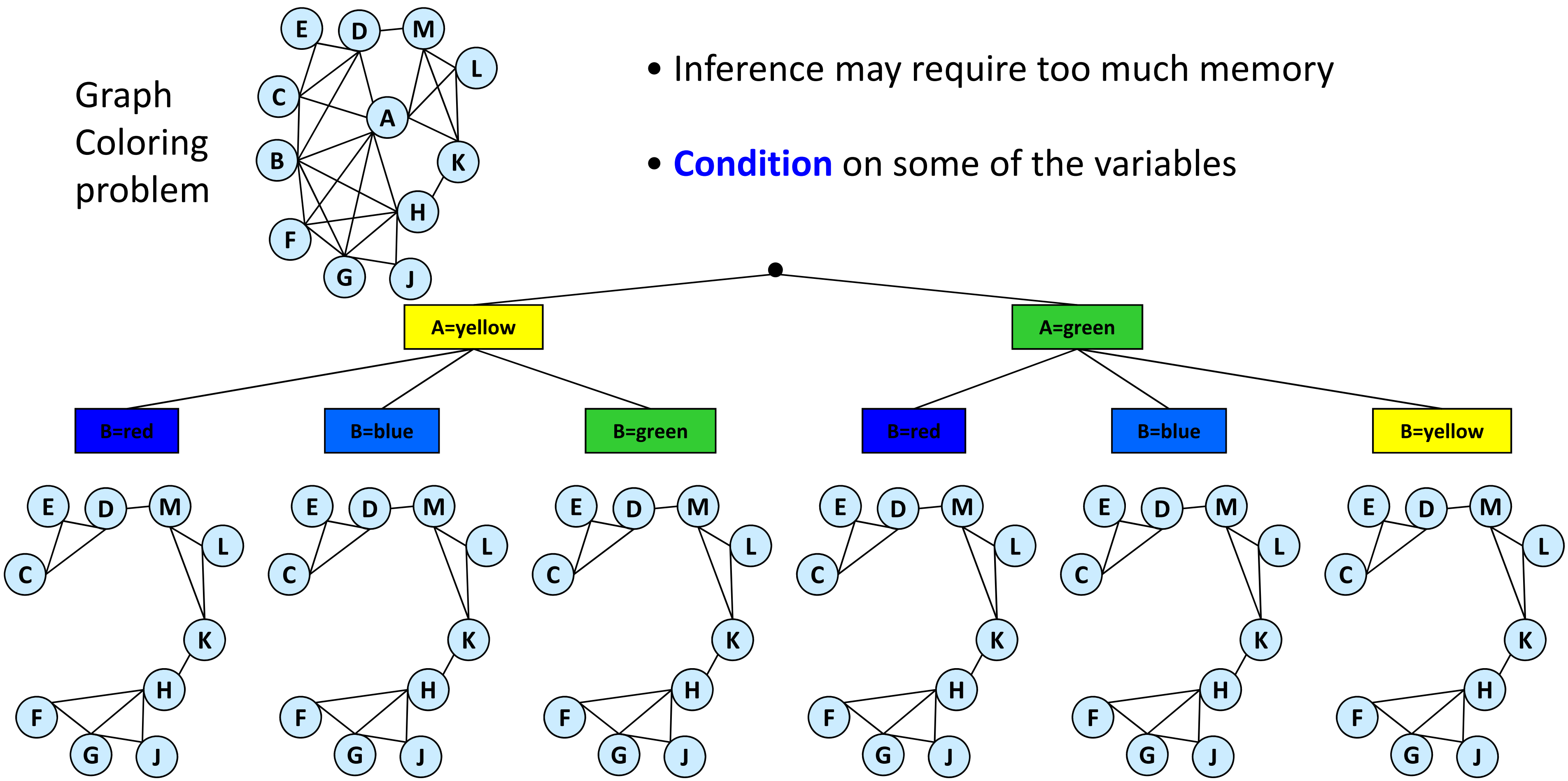
Finding a minimal q-cutset is clearly a hard task [A. Becker and Geiger, 1999; Bar-Yehuda *et al.*, 1998; Becker *et al.*, 2000; Bidyuk and Dechter, 2004]. However, like in the special case of a cycle-cutset we can settle for a non-minimal q-cutset relative to a given variable ordering. Namely,

Example 7.4 Consider as another example the contsaint graph of a graph coloring problem given in Figure 7.3a. The search space over a 2-cutset, and the induced-graph of the conditioned instances are depicted in 7.3b.

Example: 2-cutset conditioning (VEC(2))

Graph Coloring problem

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



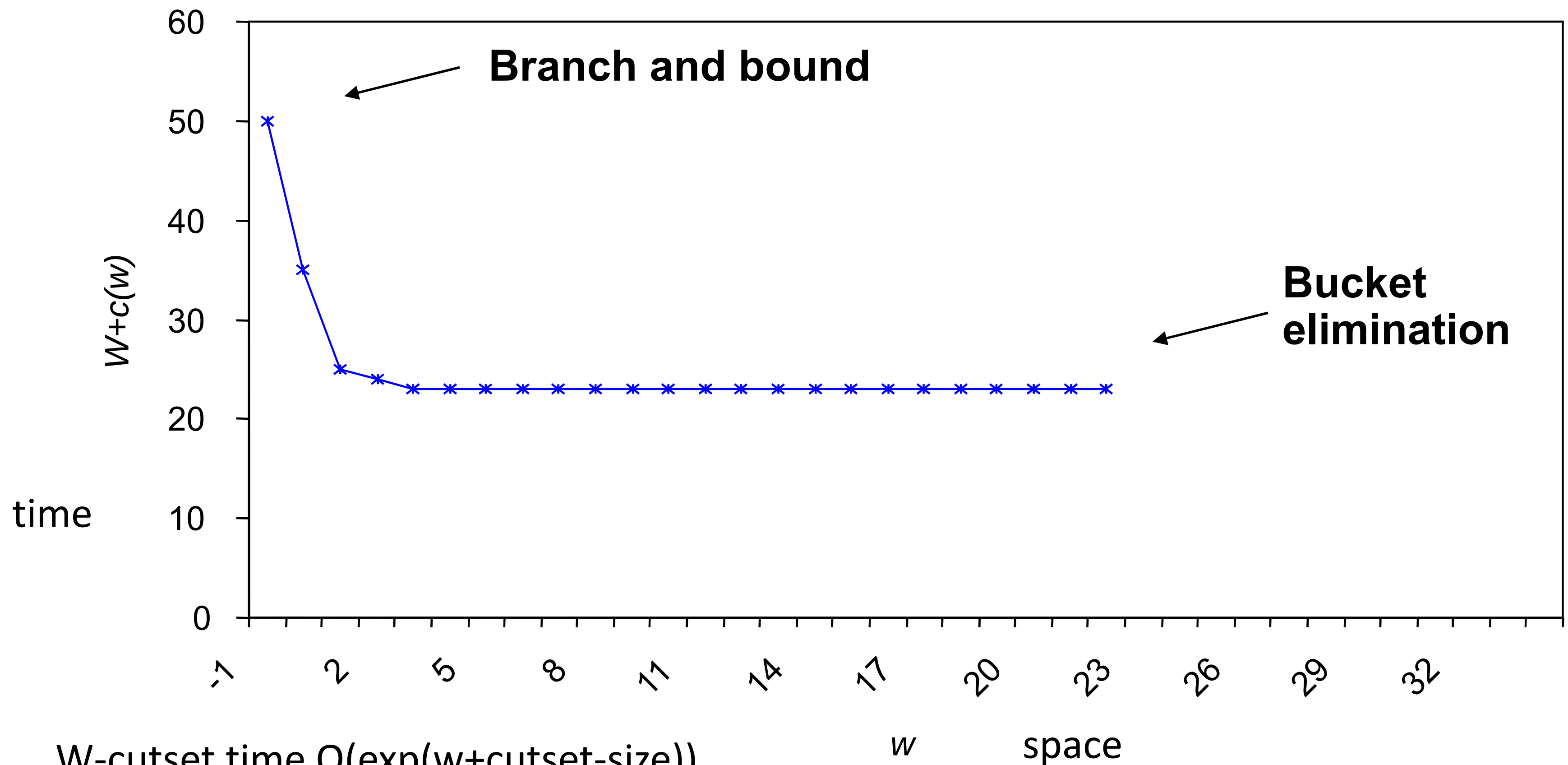
VEC(q) : Variable Elimination with Conditioning;

- VEC for Probability of evidence:
- Identify a q-cutset, C, of size |C| of the network
- For each assignment to C=c solve by CTE or BE the conditioned sub-problem.
- Accumulate probability.
- Time complexity: $nk^{|c|+q+1}$
- Space complexity: nk^{q}

Time vs Space for w-cutset

(Dechter and El-Fatah, 2000)
(Larrosa and Dechter, 2001)
(Rish and Dechter 2000)

- **Random Graphs (50 nodes, 200 edges, average degree 8, *w**≈23)**



W-cutset time O(exp(w+cutset-size))
Space O(exp(w))

VEC(q) : Variable Elimination with Conditioning;

- VEC for Probability of evidence:
- Identify a q-cutset, C, of size |C| of the network
- For each assignment to C=c solve by CTE or BE the conditioned sub-problem.
- Accumulate probability.
- Time complexity: $nk^{|c|+q+1}$
- Space complexity: nk^{q}

What w should we use?

W=1? W=0? W=w*

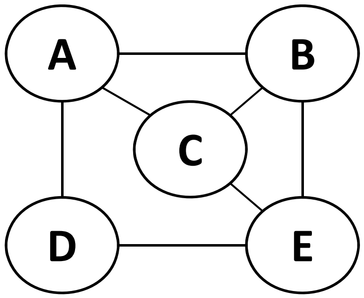
Depends on the graph

Practice: use the largest w allowed by space

Alternate conditioning and elimination?

Hybrid: Cutset-Conditioning

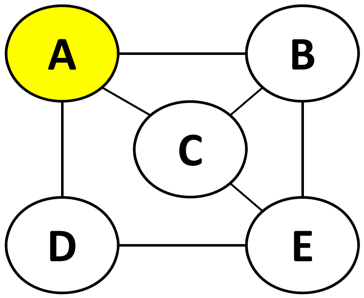
Variable Branching by Conditioning



Hybrid: Cutset-Conditioning

Variable Branching by Conditioning

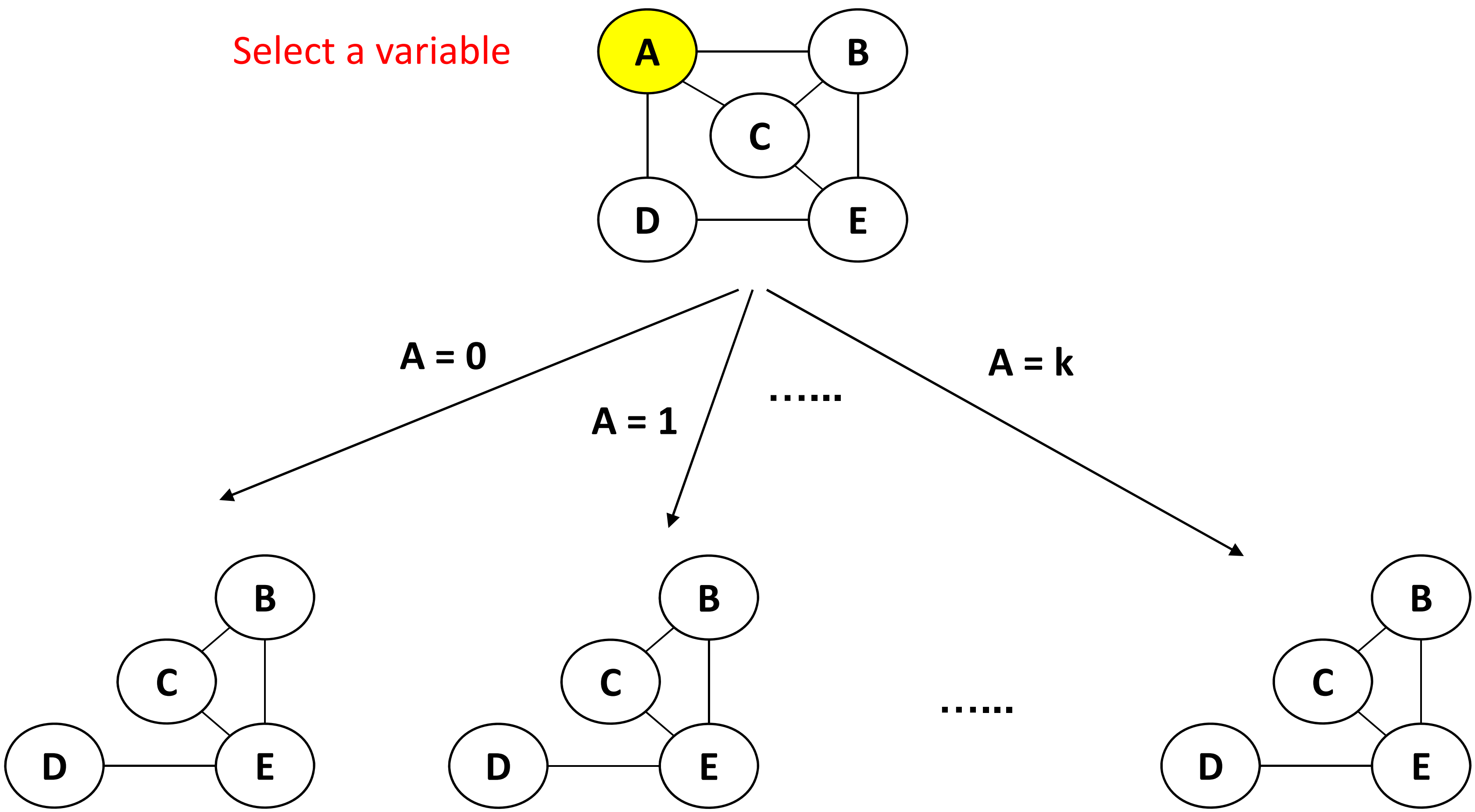
Select a variable



Hybrid: Cutset-Conditioning

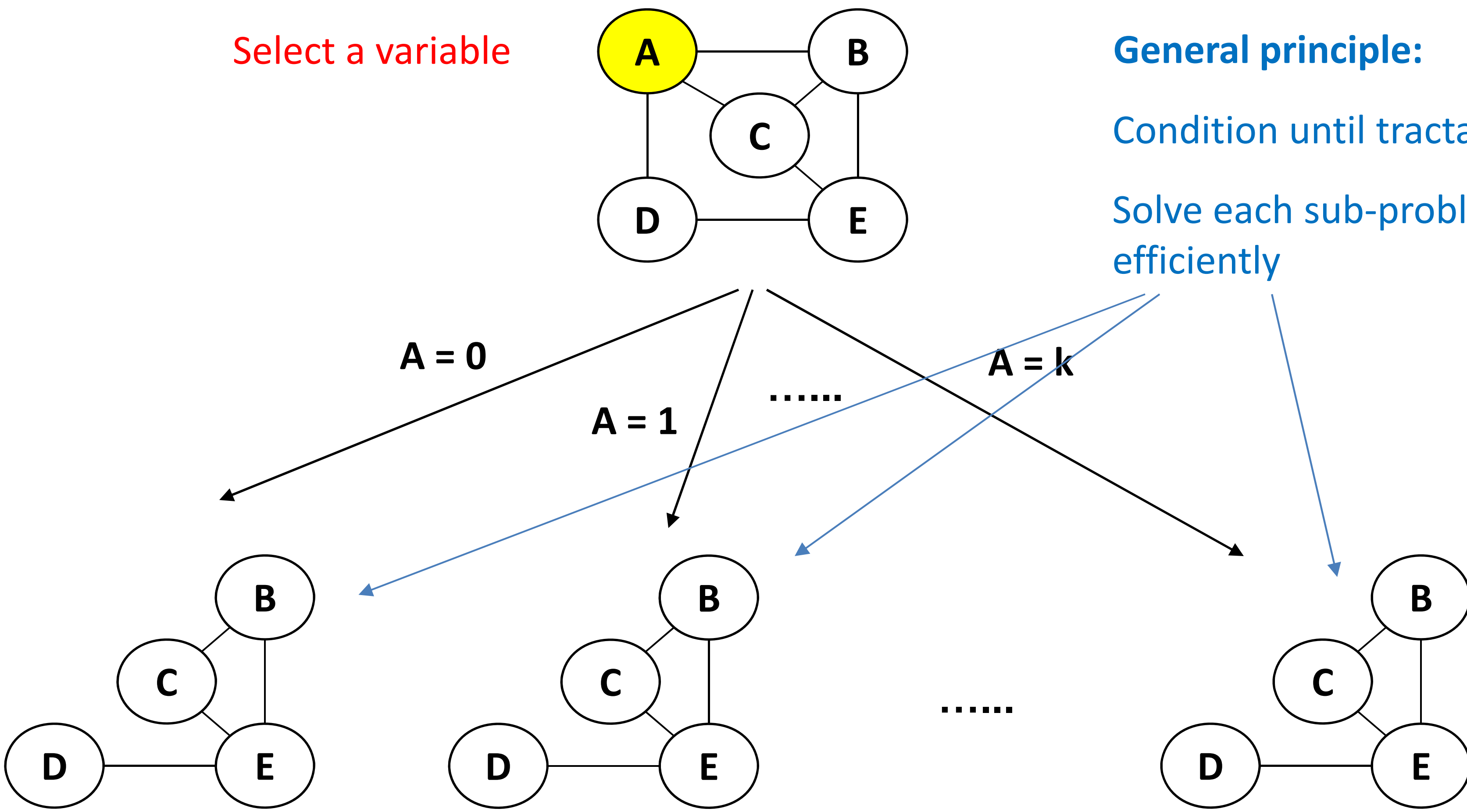
Variable Branching by Conditioning

Select a variable



Hybrid: Cutset-Conditioning

Variable Branching by Conditioning



General principle:

Condition until tractable

Solve each sub-problem efficiently

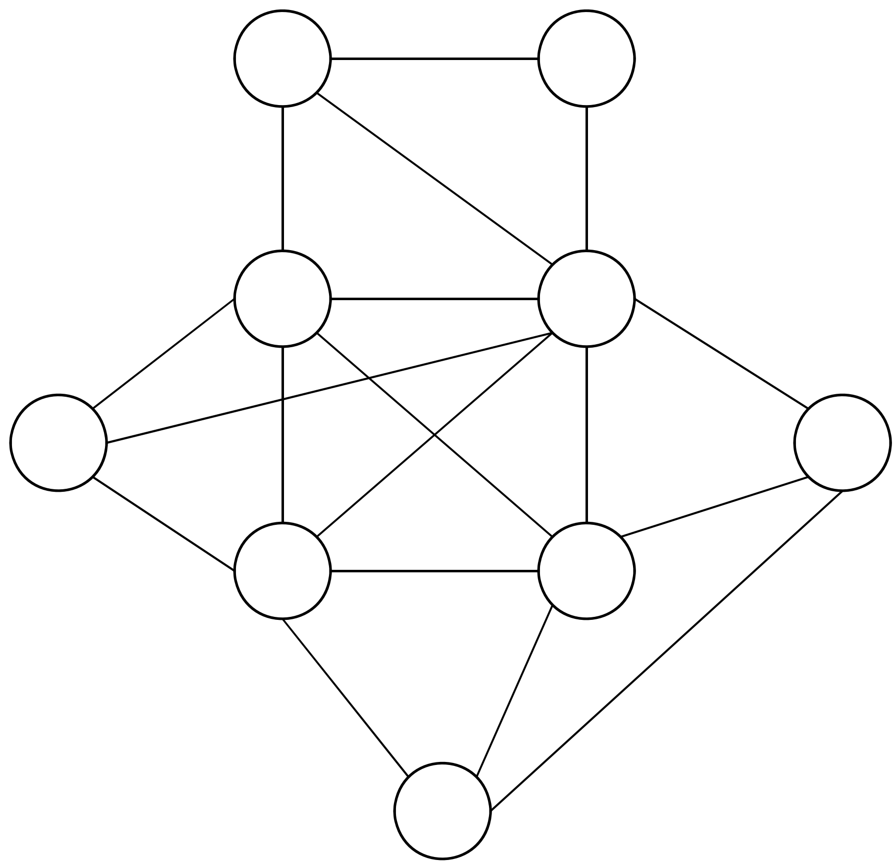
Hybrids Variants

- **Condition, condition, condition,** … and then only eliminate (w-cutset, cycle-cutset VEC(i))

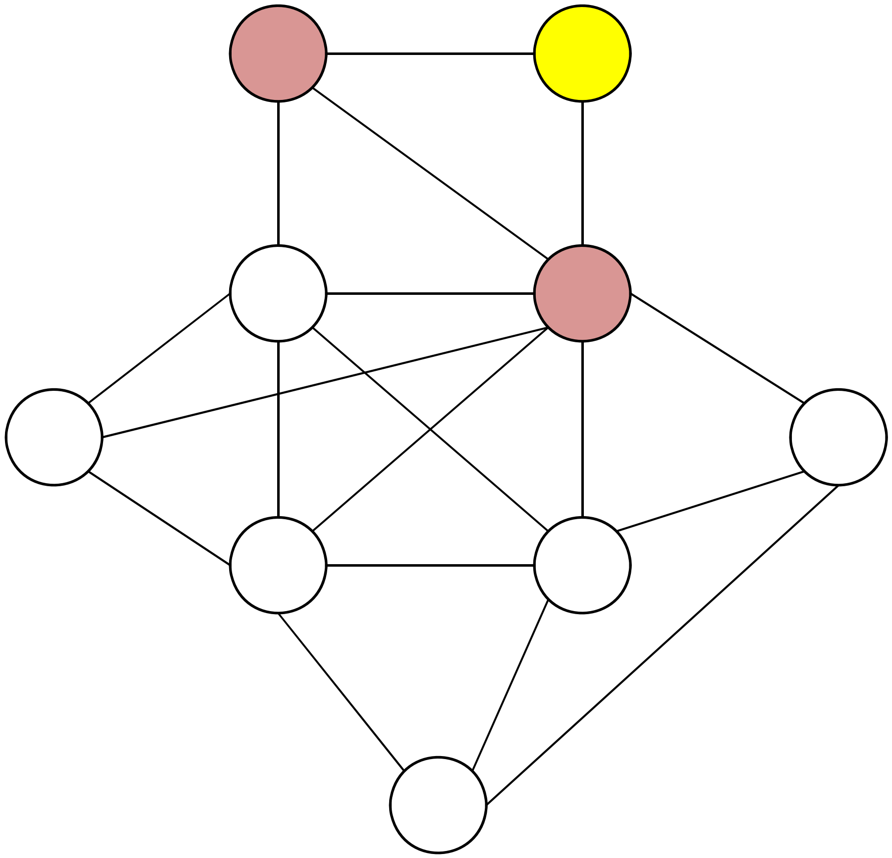
- **Eliminate, eliminate, eliminate,** … and then only search

- **Alternate** conditioning and elimination steps (elim-cond(i), ALT-VEC(i))

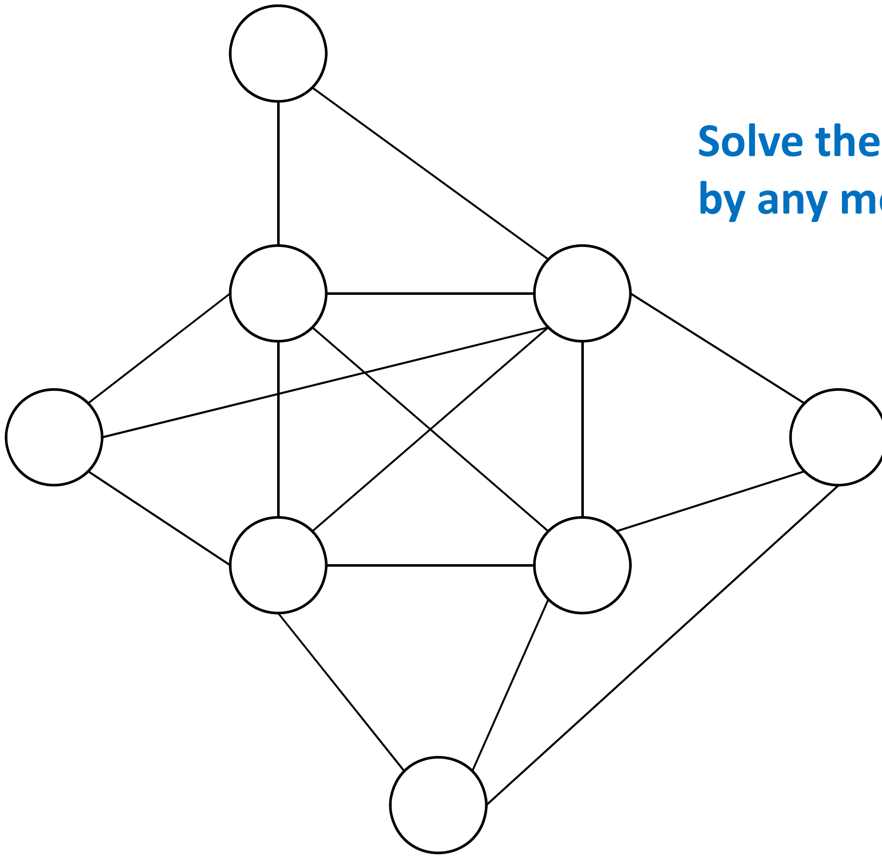
Eliminate First



Eliminate First

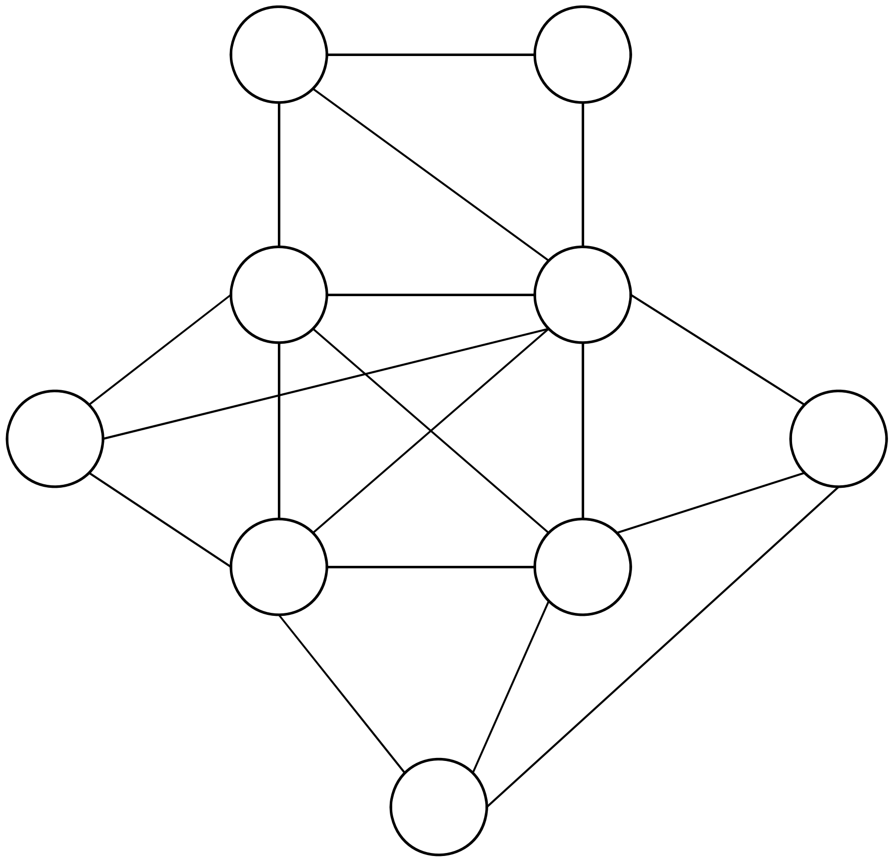


Eliminate First



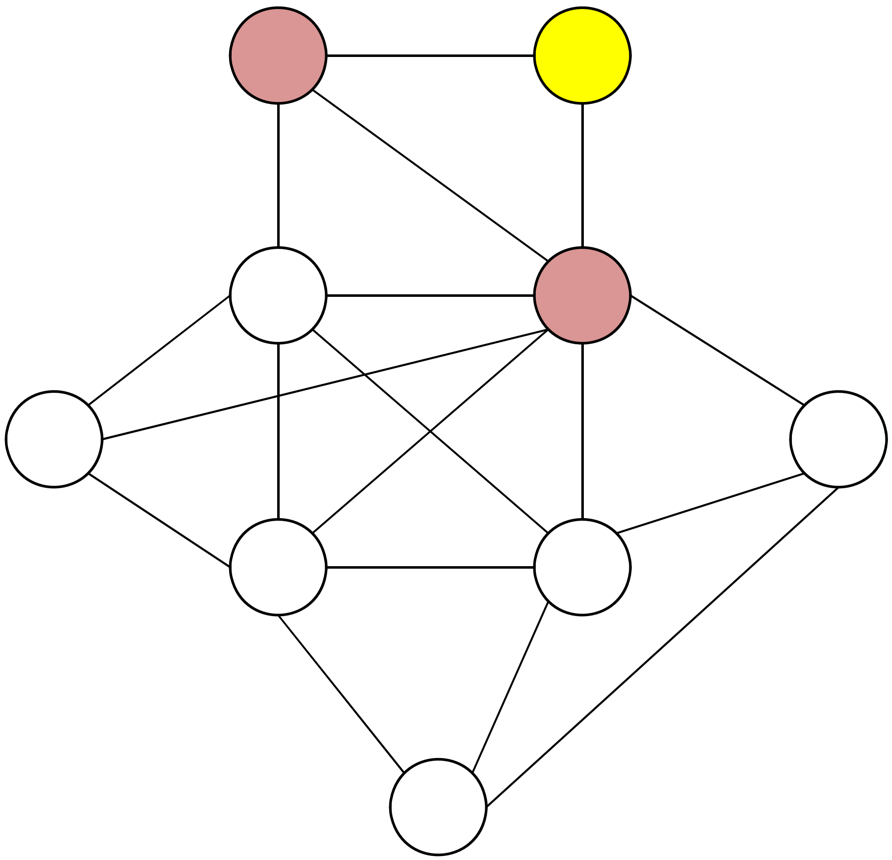
Solve the rest of the problem by any means

Alternate Conditioning and Elimination



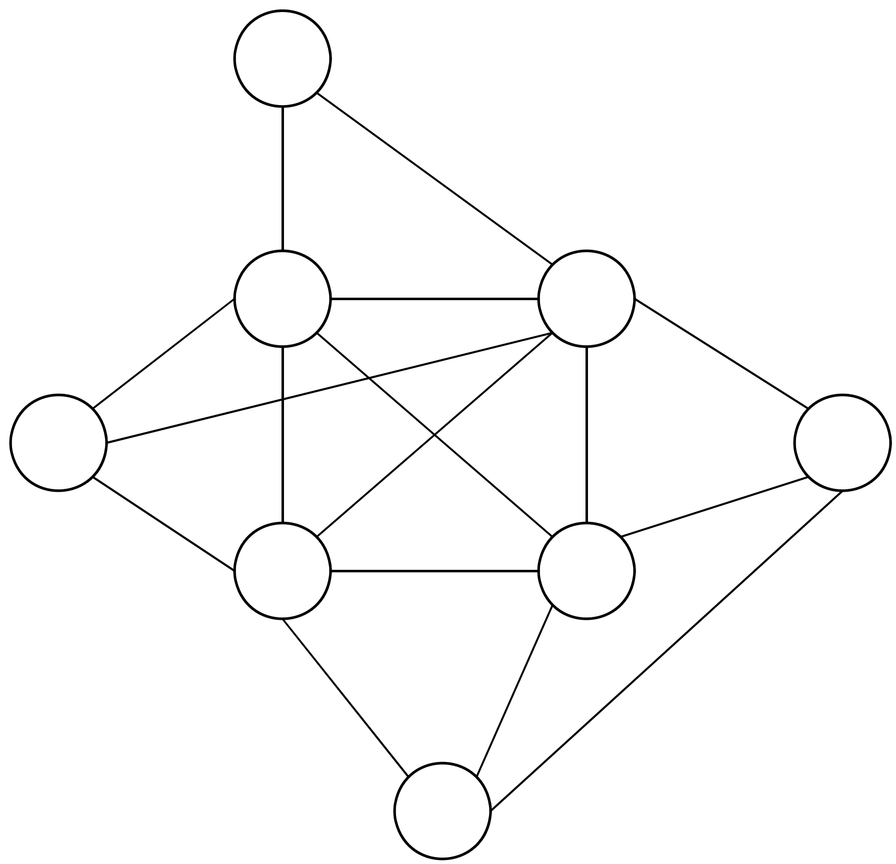
[Larrosa and Dechter, 2002]

Alternate Conditioning and Elimination



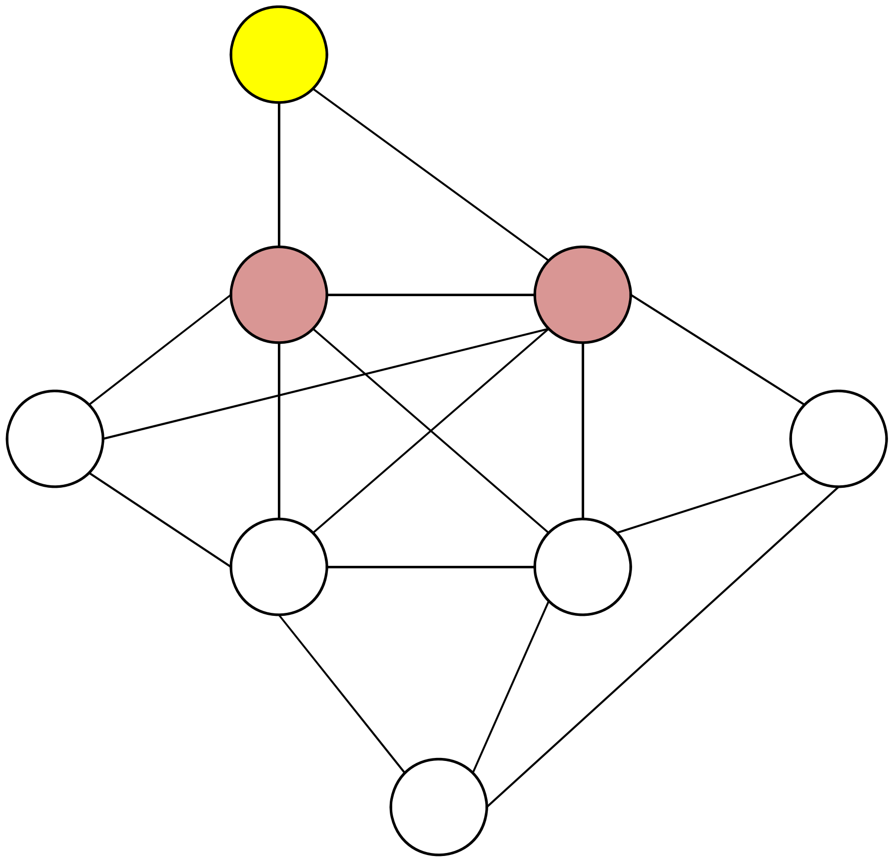
[Larrosa and Dechter, 2002]

Alternate Conditioning and Elimination



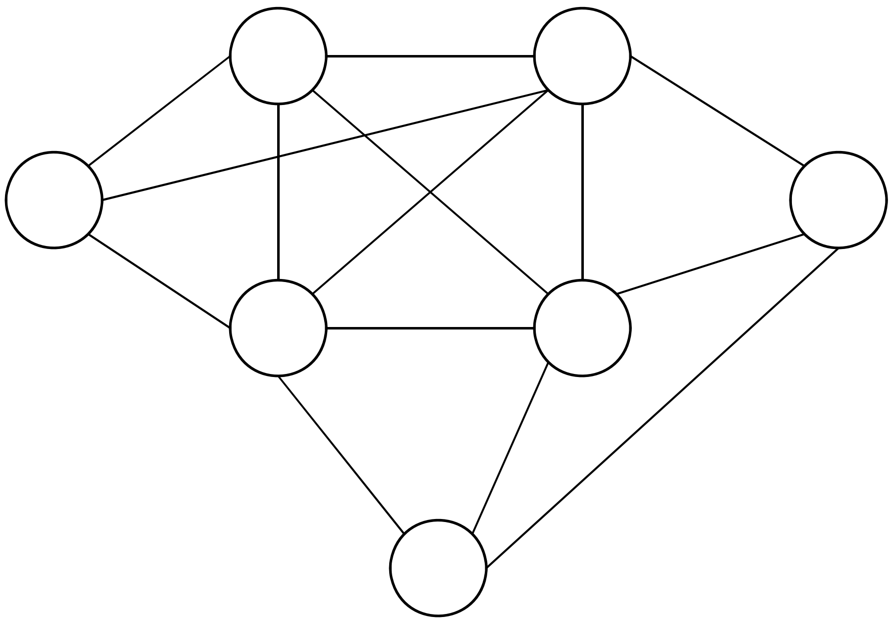
[Larrosa and Dechter, 2002]

Alternate Conditioning and Elimination



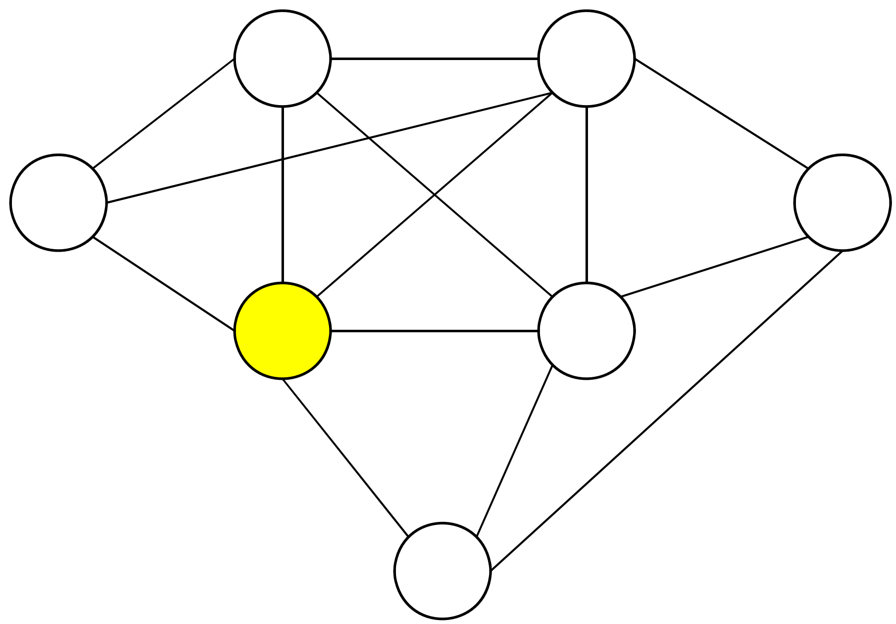
[Larrosa and Dechter, 2002]

Alternate Conditioning and Elimination

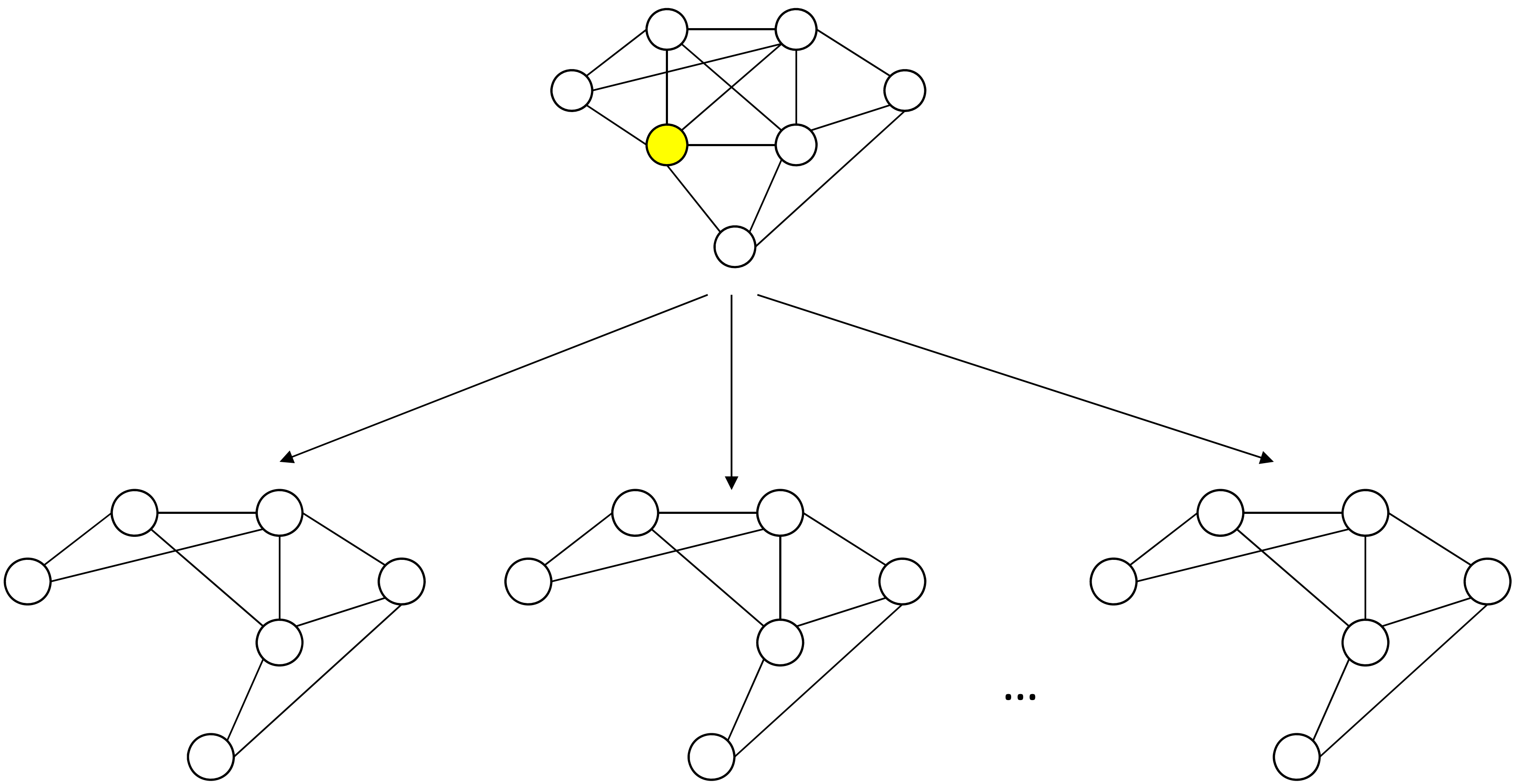


[Larrosa and Dechter, 2002]

Alternate Conditioning and Elimination



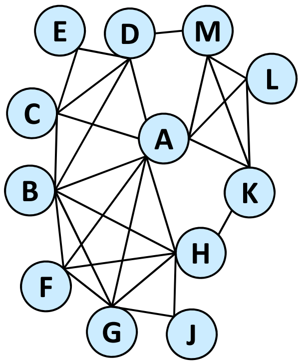
Alternate Conditioning and Elimination



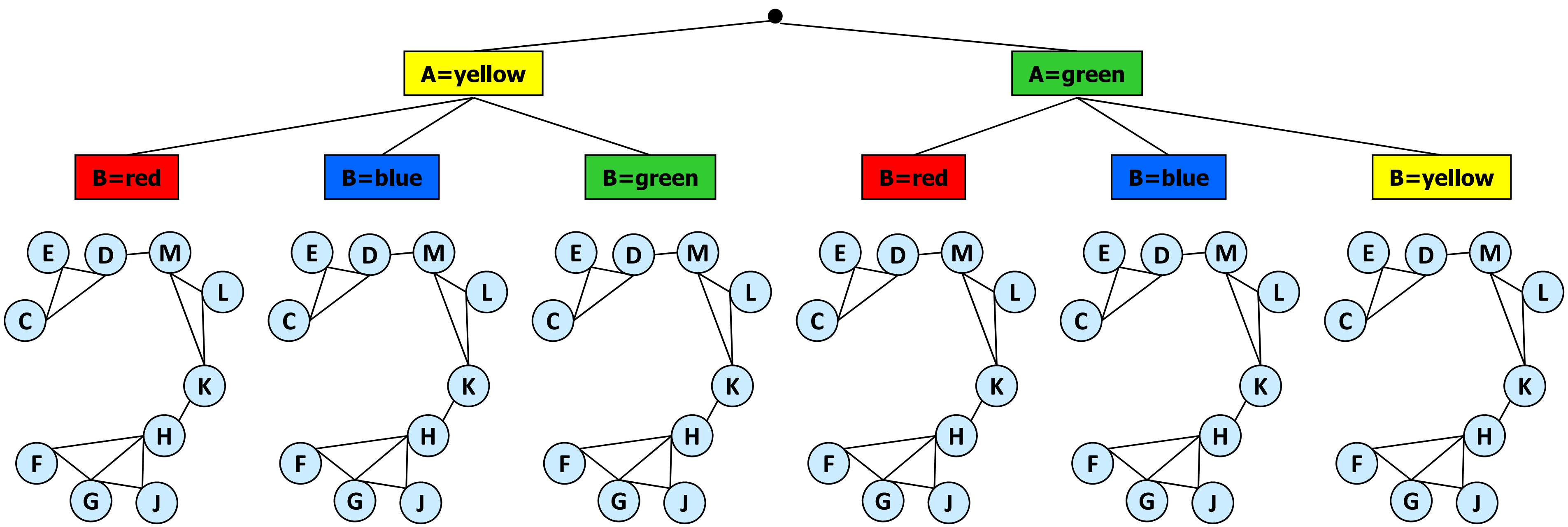
And/or W-CUTSET

OR w-Cutset

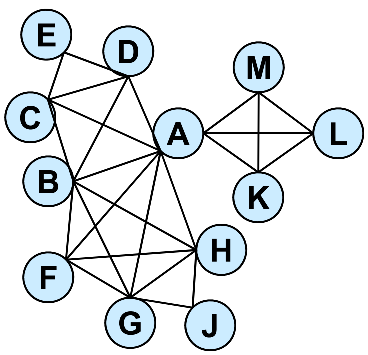
Graph Coloring problem



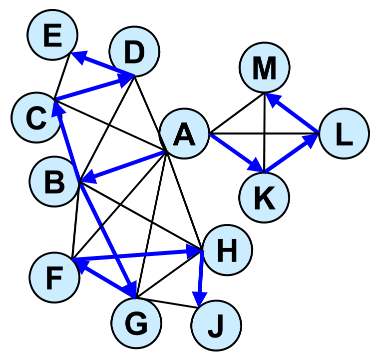
- Inference may require too much memory
- **Condition** on some of the variables



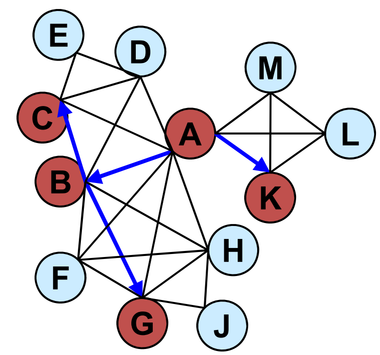
AND/OR w-cutset



grahpical model

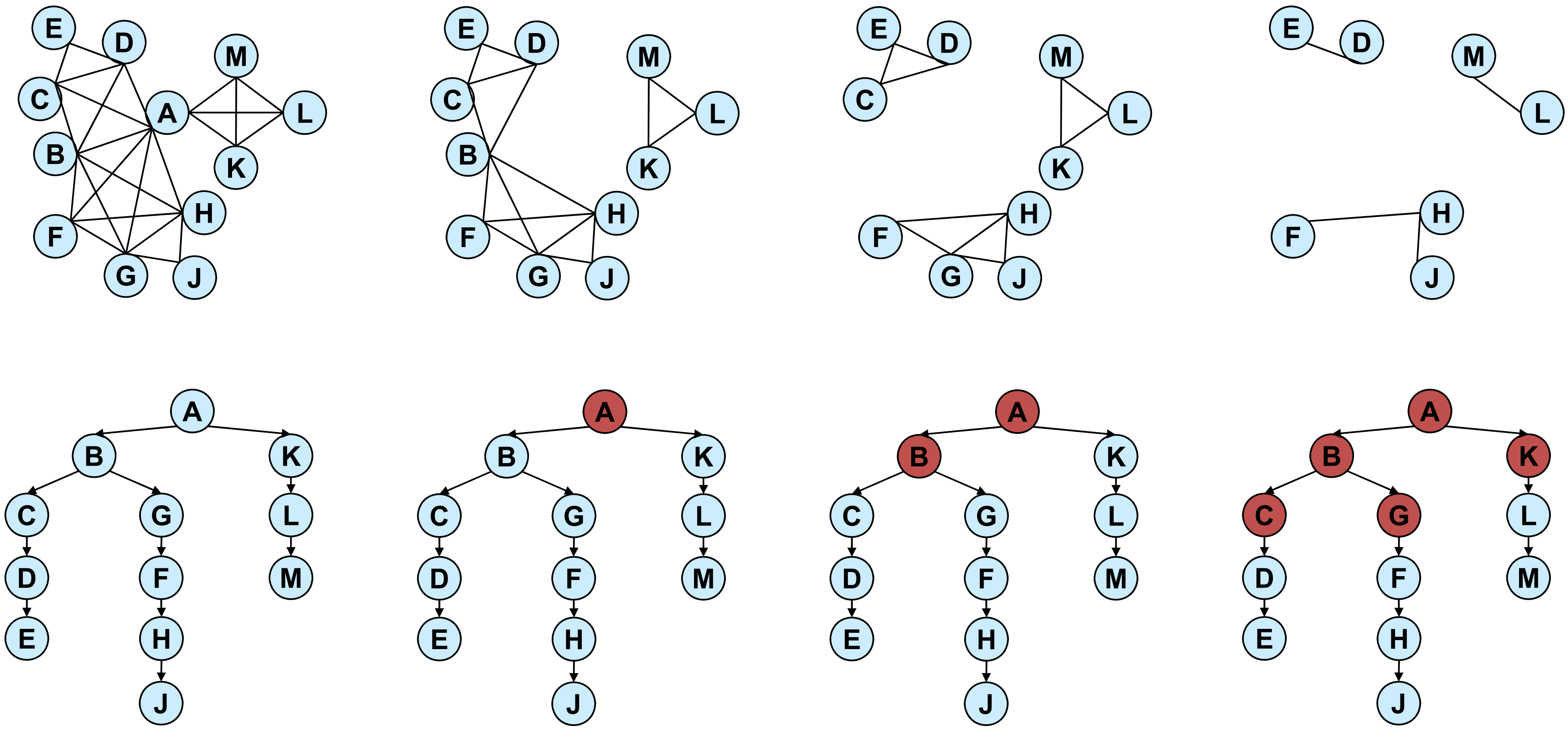


pseudo tree



1-cutset tree

AND/OR w-cutset



3-cutset 2-cutset 1-cutset

w-Cutset Trees Over AND/OR Space

- Definition:
  - T_w is a w-cutset tree relative to pseudo-tree T, iff T_w roots T and when removed, yields tree-width w.

- Theorem:
  - AO(i) time complexity for pseudo-tree T is time $O(\exp(i+m_i))$ and space O(i), m_i is the depth of the T_i tree.

- Better than w-cutset: $O(\exp(i+c_i))$ when c_i is the number of nodes in T_i

Summary: AND/OR Cutset-Conditioning

- Trade memory for time.
- We never improve time: cycle-cutset size is larger of equal to treewidth+1
- Sometime we do not worsen the time and memory can be much better (e.g., when the induced-width is high)

Software

- **aolib**
  - http://graphmod.ics.uci.edu/group/Software

  (standalone AOBB, AOBF solvers)
- **daoopt**
  - https://github.com/lotten/daoopt

  (distributed and standalone AOBB solver)
- **merlin**
  - https://developer.ibm.com/open/merlin

  (standalone WMB, AOBB, AOBF, RBFAOO solvers)

  open source, BSD license

UAI Probabilistic Inference Competitions

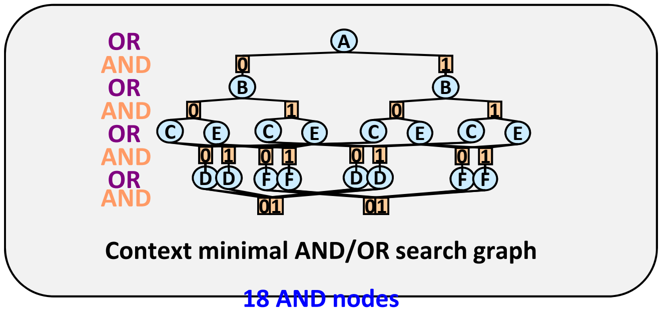
- **2006**  (aolib)
- **2008**  (aolib)
- **2012**  (daoopt)
- **2014**  (daoopt)

 (daoopt)  (merlin)

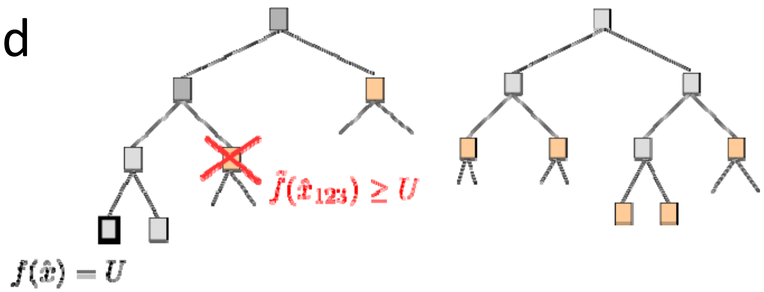
Marginal Map

Summary of Search

- AND/OR search spaces, pseudo-trees
  - AND/OR search trees
  - AND/OR search graphs
  - Generating good pseudo-trees
  - Brute-force search



- Heuristic search for AND/OR spaces
  - Depth-first AND/OR branch and bound
  - Best-first AND/OR search
  - The Guiding MBE heuristic
  - Marginal Map (max-sum-product)



- Hybrids of search and Inference
- **Summary and Class 2**