Outline

- Introduction
- Inference
- Bounds and heuristics
- **AND/OR search**
  - AND/OR search spaces
  - Depth-first AND/OR branch and bound
  - Best-first AND/OR search
  - Advanced searches and tasks
- **Exploiting parallelism**
- **Software**

Outline

- Introduction
- Inference
- Bounds and heuristics
- **AND/OR search**
  - AND/OR search spaces
  - Depth-first AND/OR branch and bound
  - Best-first AND/OR search
  - Advanced searches and tasks
- Exploiting parallelism
- Software

Solution Techniques

AND/OR search

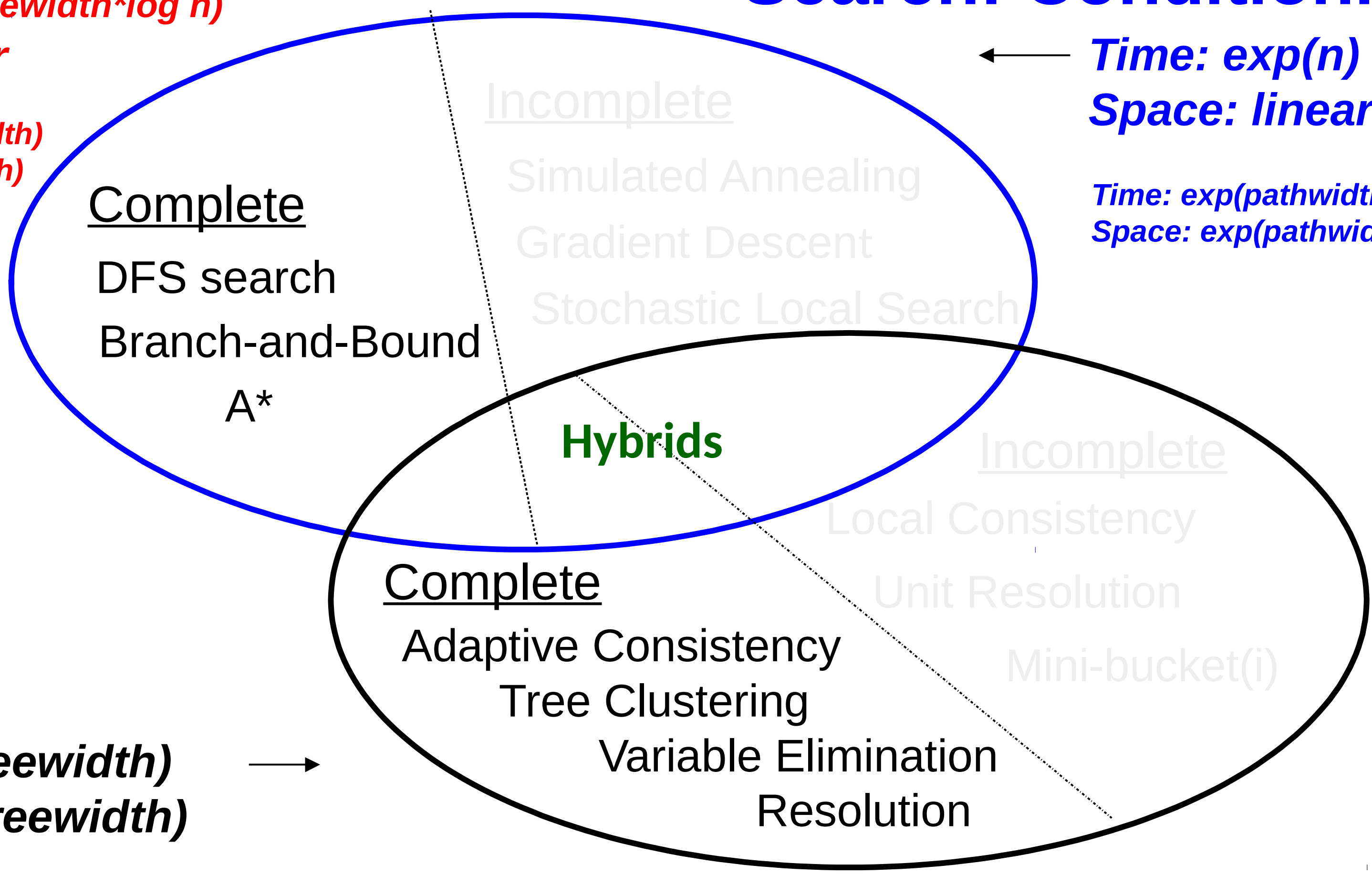
Time: exp(treewidth*log n)
Space: linear

Space: exp(treewidth)
Time: exp(treewidth)

Search: Conditioning

Time: exp(n)
Space: linear

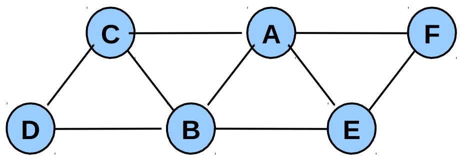
Time: exp(pathwidth)
Space: exp(pathwidth)



Time: exp(treewidth)
Space:exp(treewidth)

Inference: Elimination

Classic OR Search Space



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**

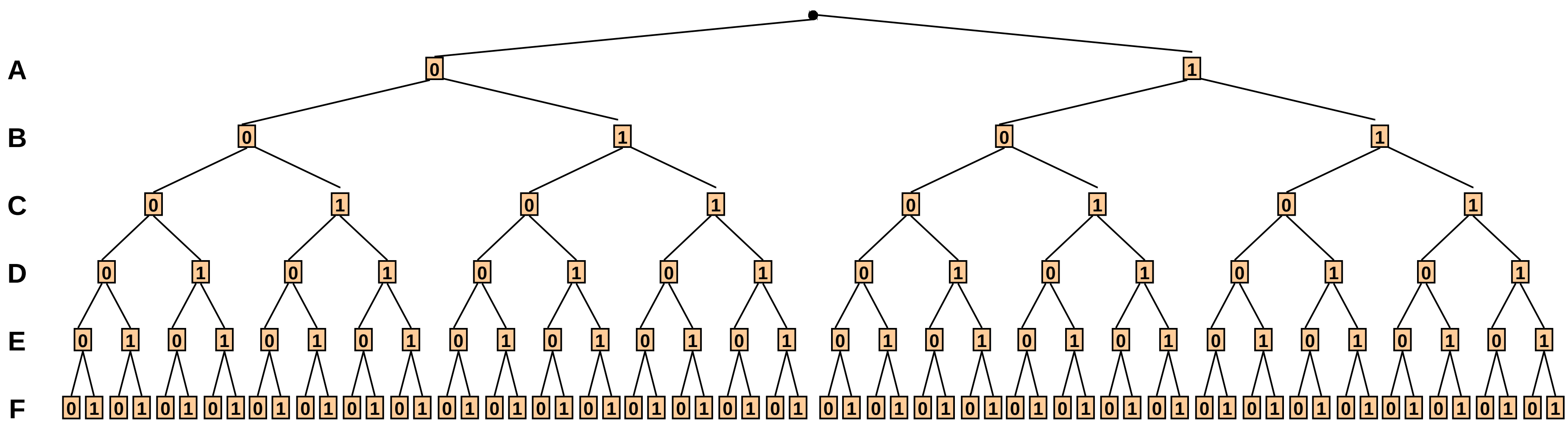
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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

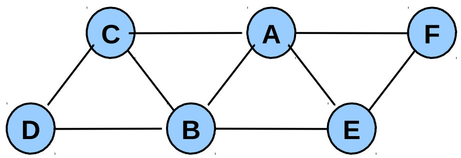
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

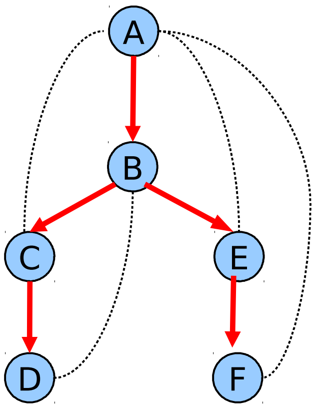
E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

Objective function: $F^* = \min_X \sum_i f_i(X)$



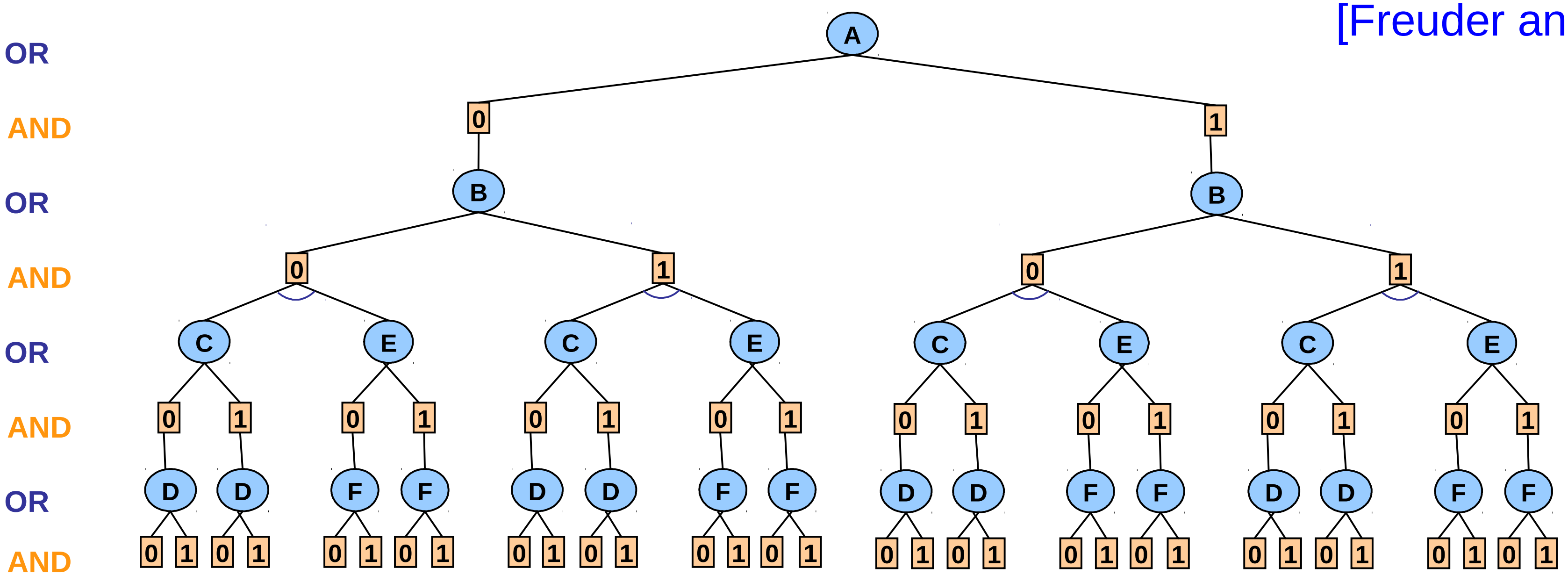
The AND/OR Search Tree





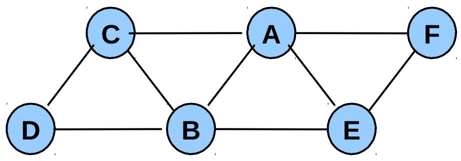
Pseudo tree

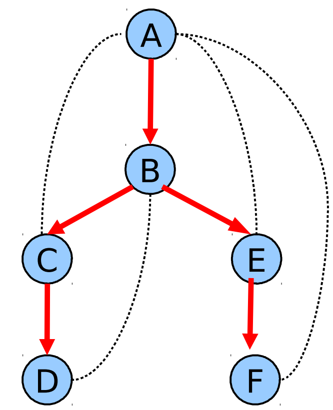
[Freuder and Quinn, 1985]



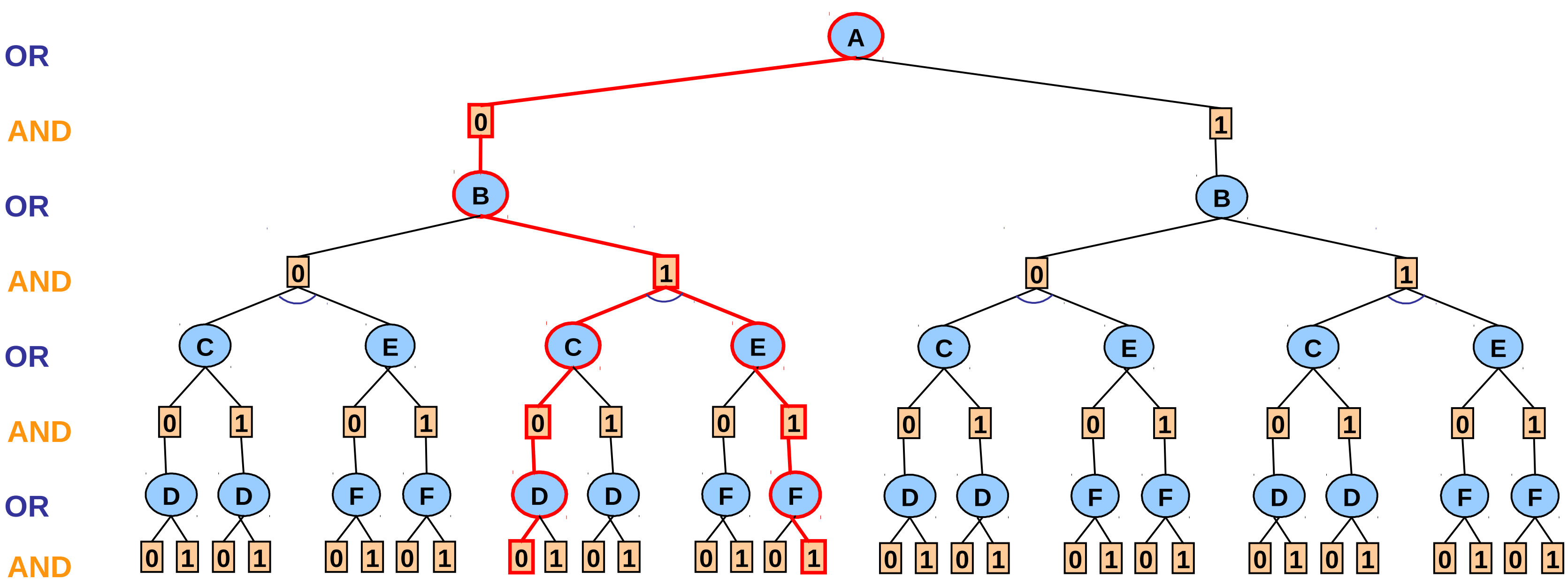
[Dechter and Mateescu, 2007]

The AND/OR Search Tree



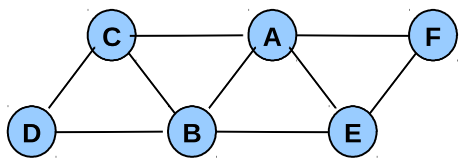


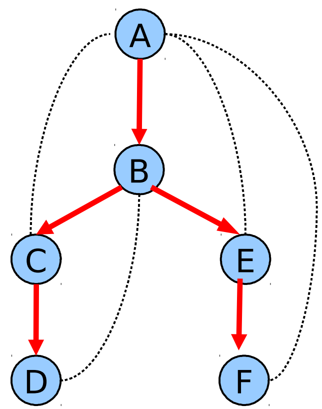
Pseudo tree



A solution subtree is (A=0, B=1, C=0, D=0, E=1, F=1)

Weighted AND/OR Search Tree





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**

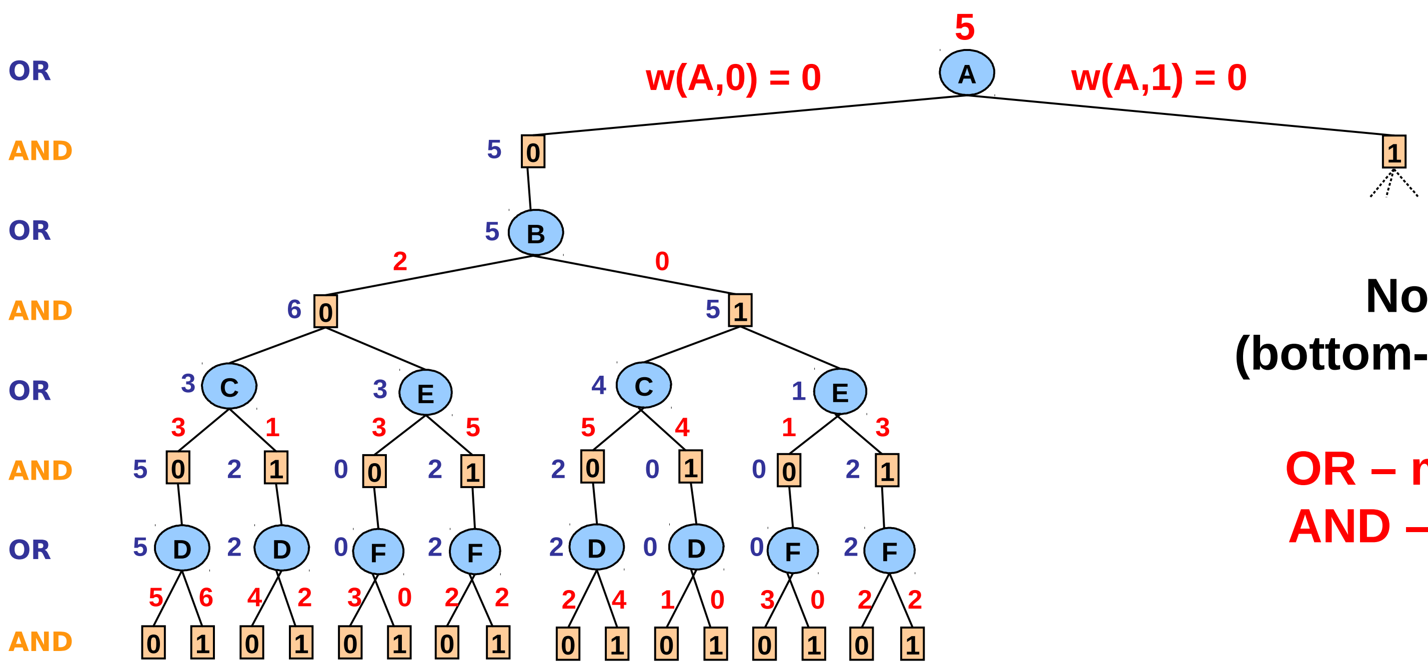
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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**

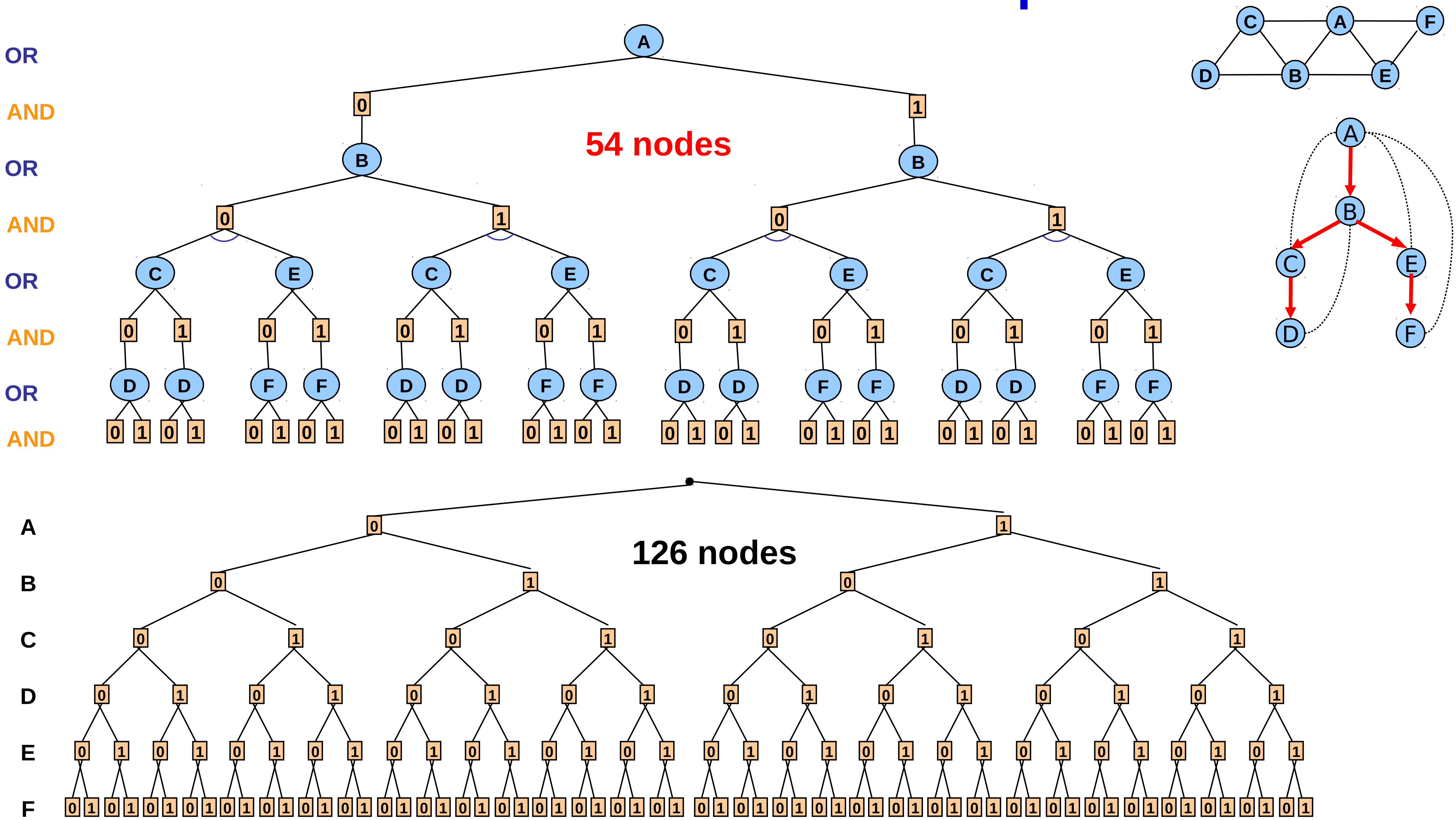
Objective function: $F^* = \min_X \sum_i f_i(X)$



Node Value
(bottom-up evaluation)

OR – minimization
AND – summation

AND/OR versus OR Spaces



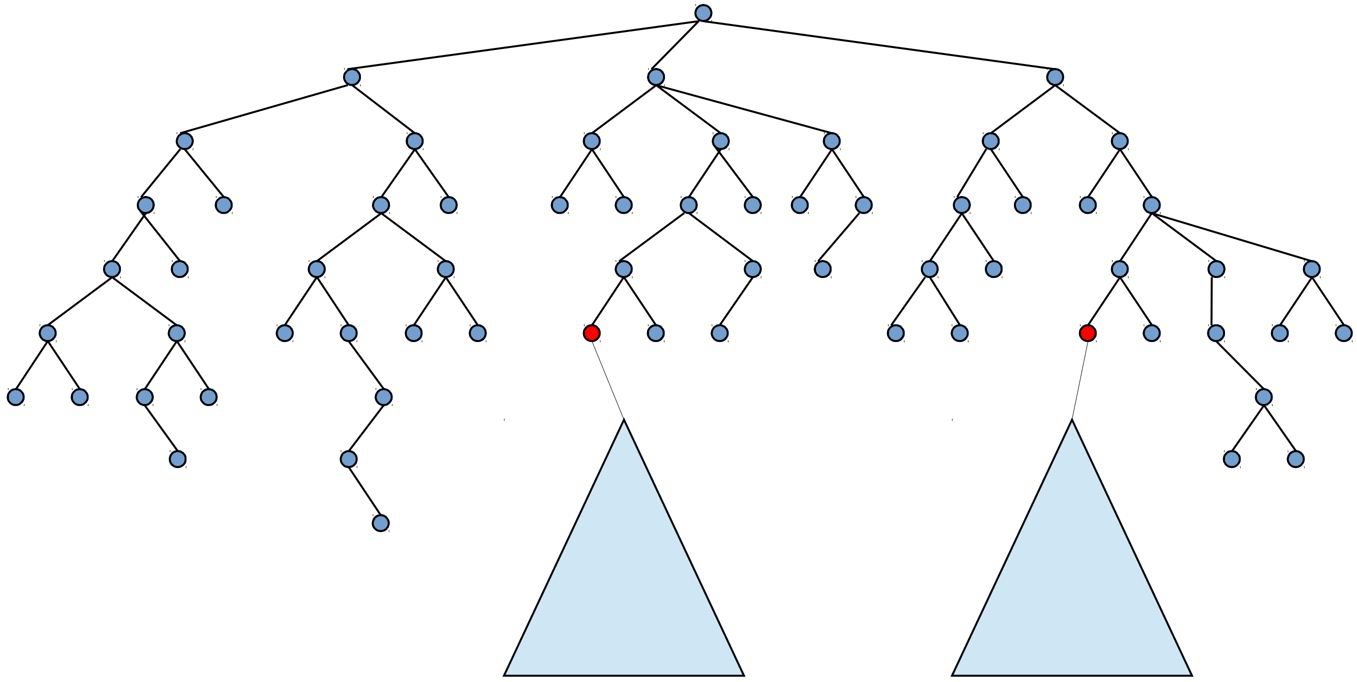
Complexity of AND/OR Tree Search

	AND/OR tree	OR tree
Space	$O(n)$	$O(n)$
Time	$O(n\,d^t)$ $O(n\,d^{(w^* \log n)})$ [Freuder & Quinn85], [Collin, Dechter & Katz91], [Bayardo & Miranker95], [Darwiche01]	$O(d^n)$

d = domain size
t = depth of pseudo tree
n = number of variables
w* = induced width

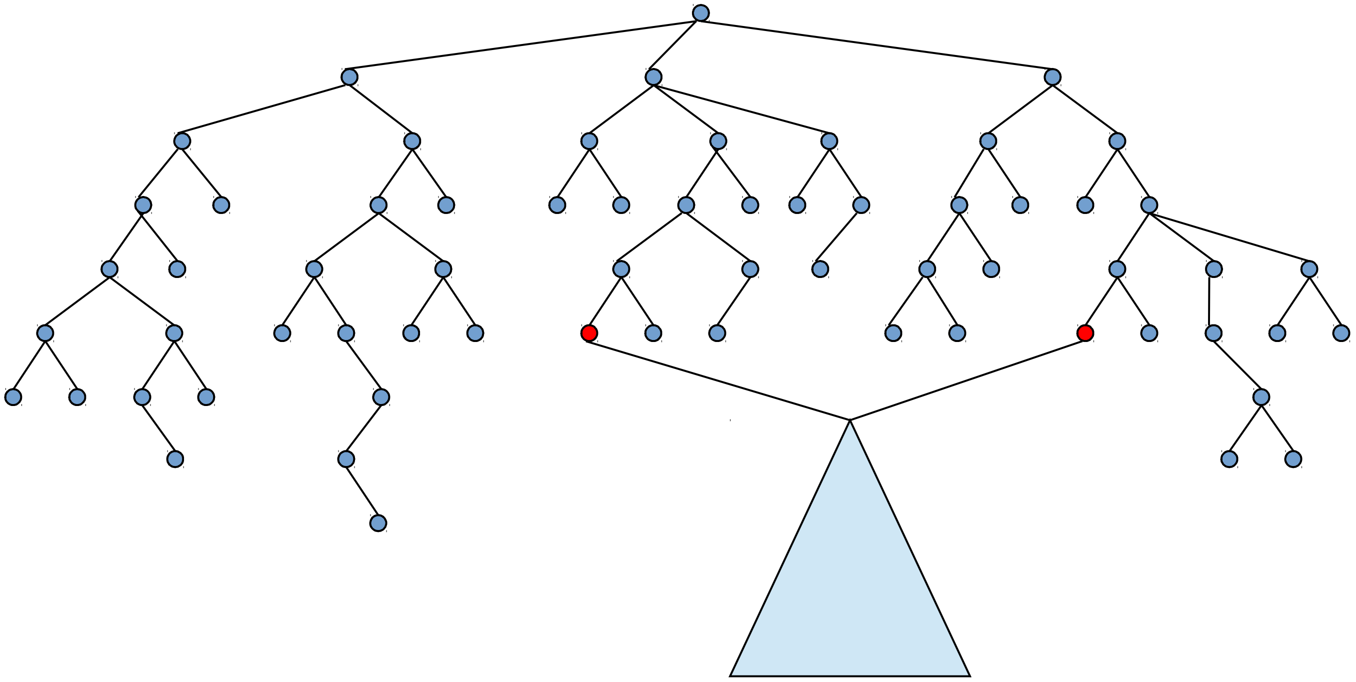
From Search Trees to Search Graphs

- Any two nodes that root identical sub-trees or sub-graphs can be merged



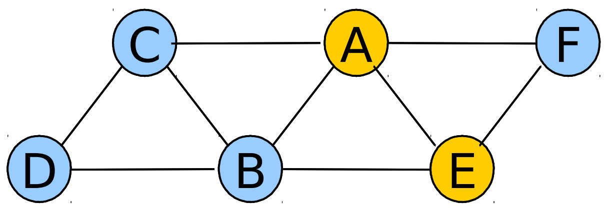
From Search Trees to Search Graphs

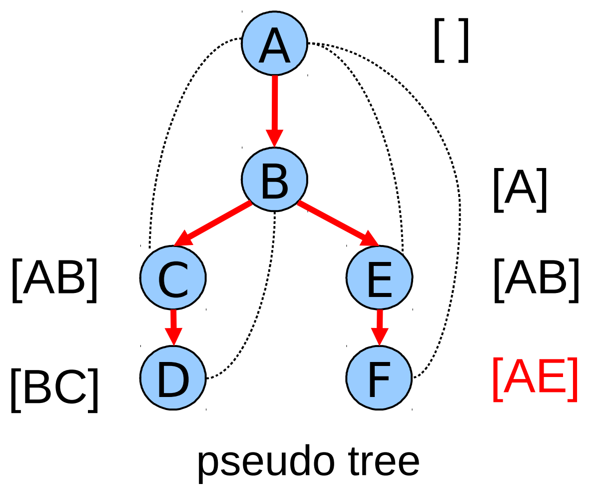
- Any two nodes that root identical subtrees or subgraphs can be merged



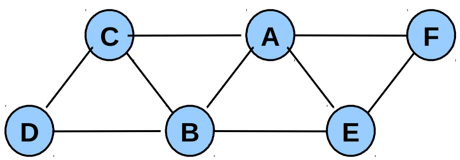
Merging Based on Contexts

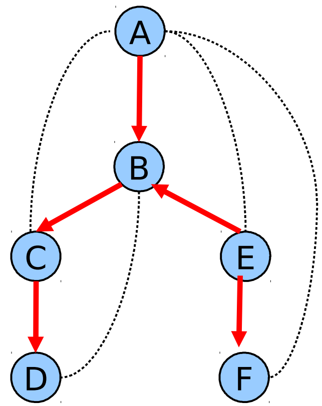
- One way of recognizing nodes that can be merged (based on the graph structure)
  - **context**(X) = ancestors of X in the pseudo tree that are connected to X or to descendants of X





AND/OR Search Graph





A	B	f_{ab}
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_{ac}
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_{ae}
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_{af}
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

B	C	f_{bc}
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

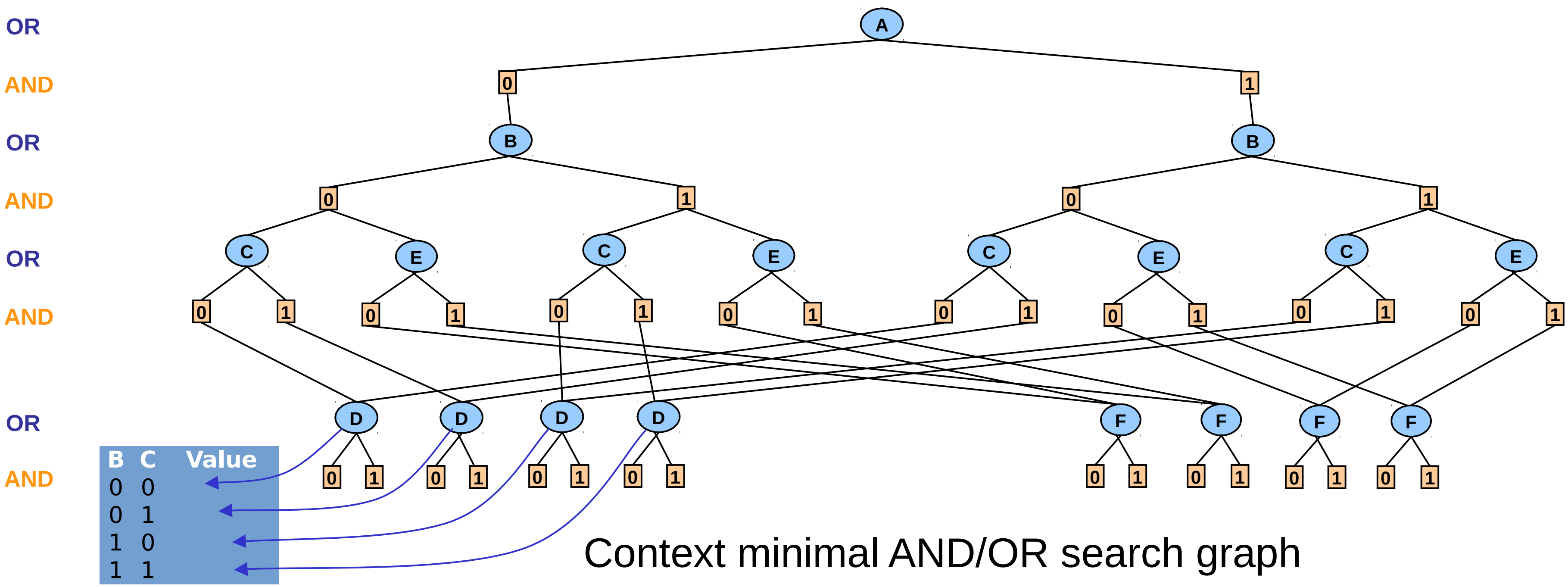
B	D	f_{bd}
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_{be}
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

C	D	f_{cd}
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_{ef}
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

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



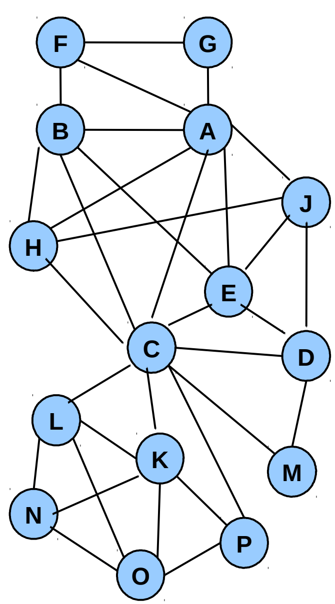
B	C	Value
0	0	
0	1	
1	0	
1	1	

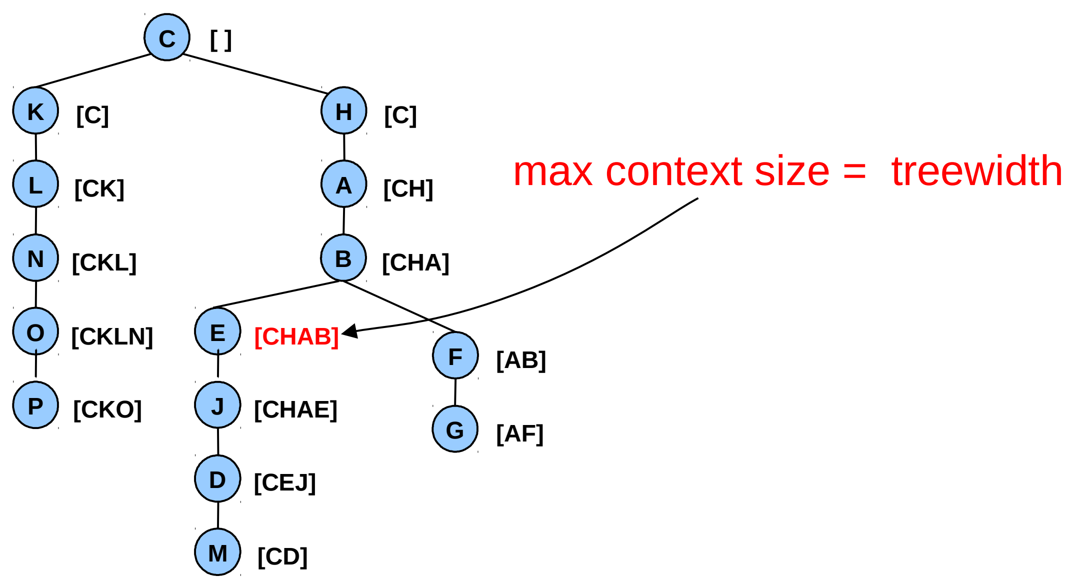
Context minimal AND/OR search graph

Cache table for **D**

How Big Is The Context?

- **Theorem**: The maximum context size for a pseudo tree is equal to the treewidth of the graph along the pseudo tree.





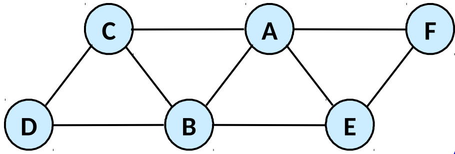
(C K H A B E J L N O D P M F G)

Complexity of AND/OR Graph Search

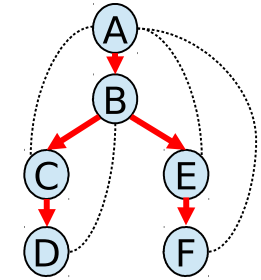
	AND/OR graph	OR graph
Space	$O(n\, d^{w^*})$	$O(n\, d^{pw^*})$
Time	$O(n\, d^{w^*})$	$O(n\, d^{pw^*})$

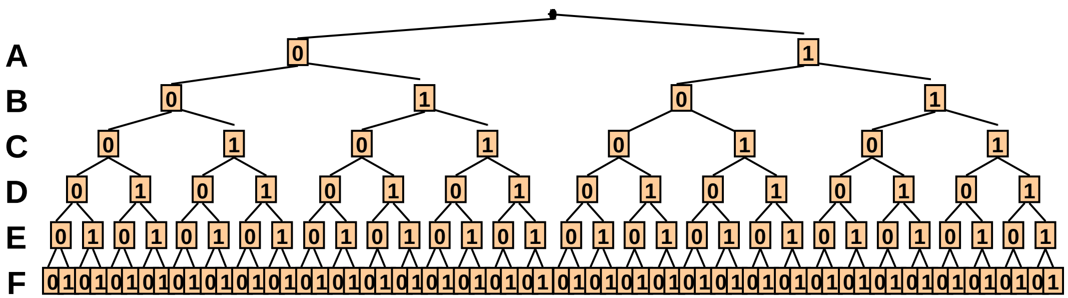
d = domain size
w* = induced width
n = number of variables
pw* = pathwidth

w* ≤ pw* ≤ w* log n



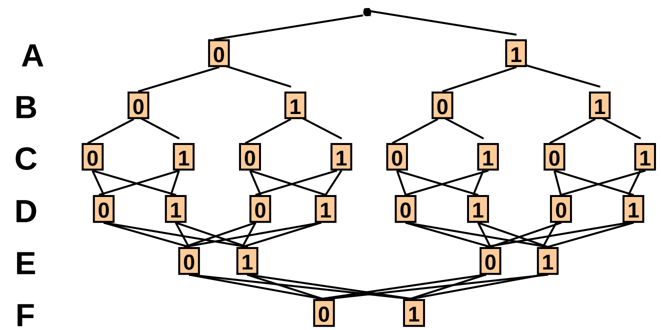
All Four Search Spaces





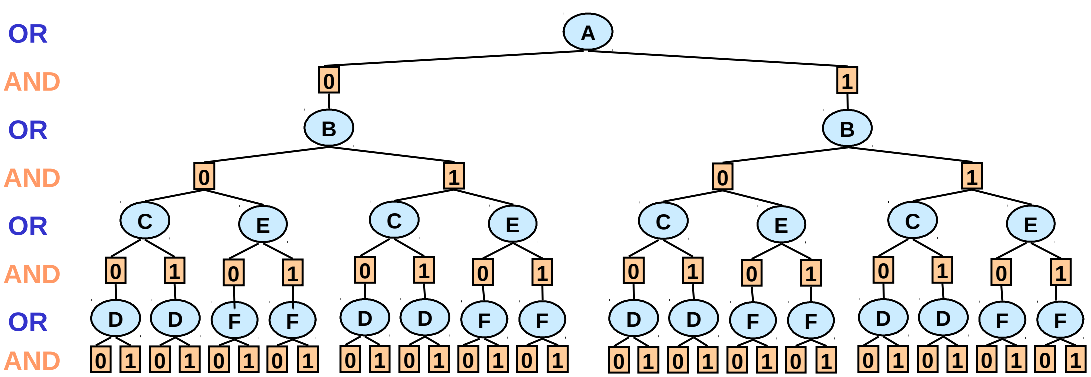
Full OR search tree

126 nodes



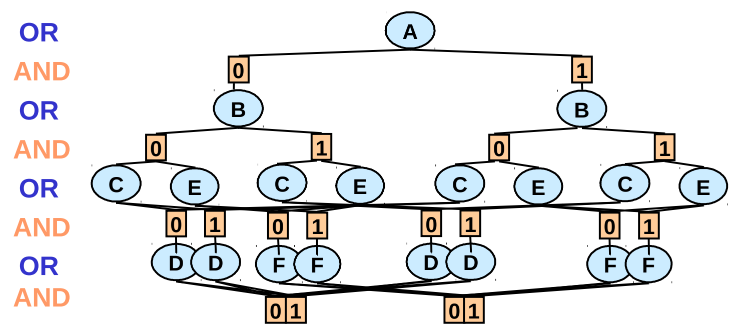
Context minimal OR search graph

28 nodes



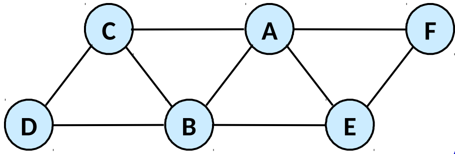
Full AND/OR search tree

54 AND nodes

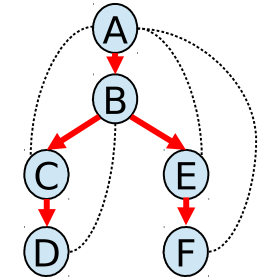


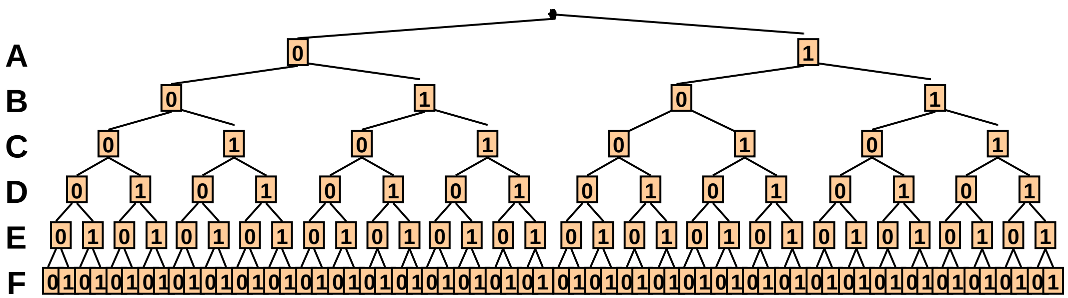
Context minimal AND/OR search graph

18 AND nodes



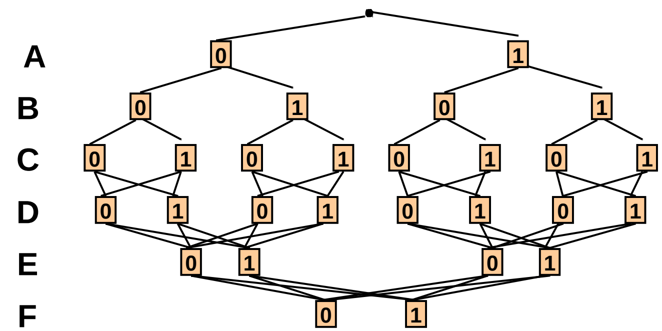
All Four Search Spaces





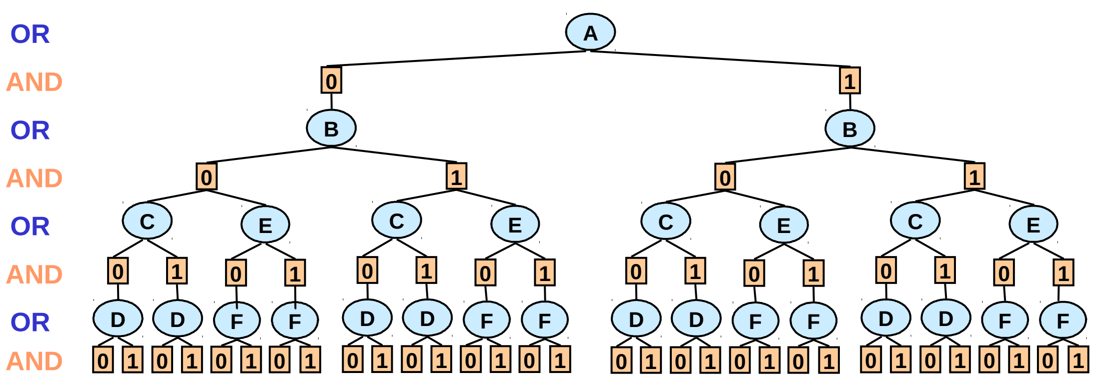
Full OR search tree

126 nodes



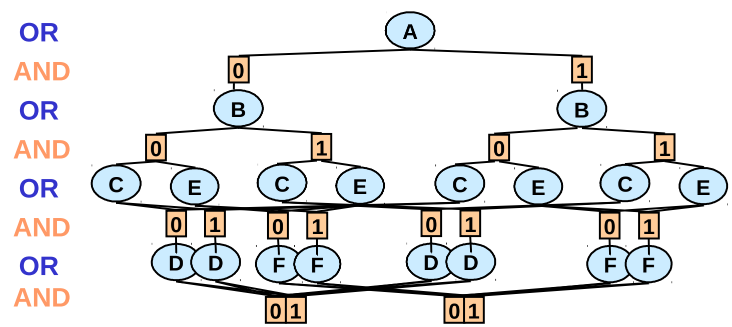
Context minimal OR search graph

28 nodes



Full AND/OR search tree

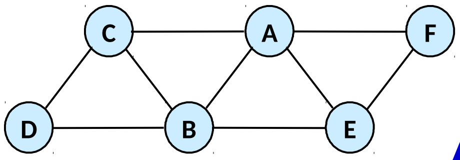
54 AND nodes

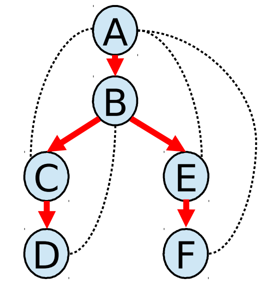


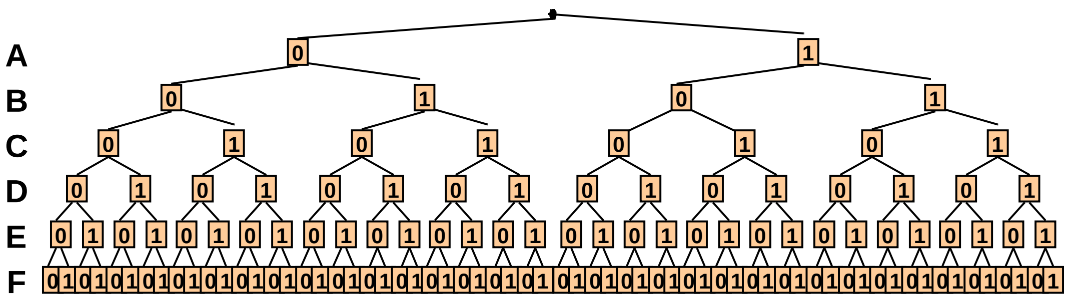
Context minimal AND/OR search graph

18 AND nodes

All Four Search Spaces

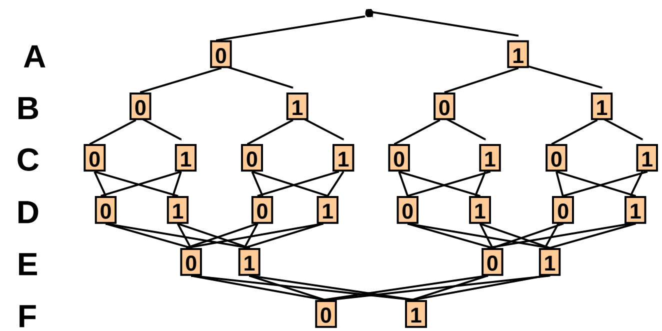






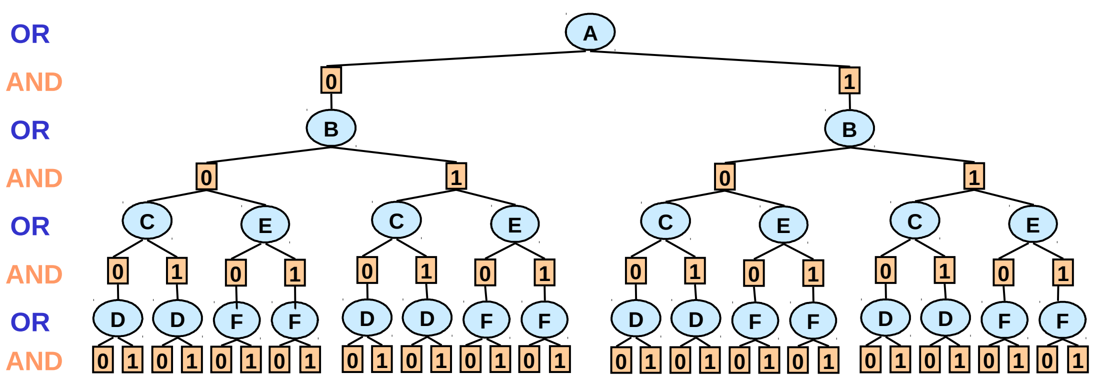
Full OR search tree

126 nodes



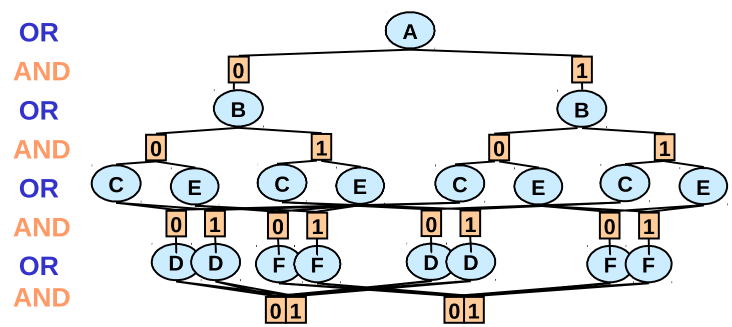
Context minimal OR search graph

28 nodes



Full AND/OR search tree

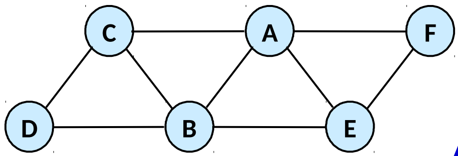
54 AND nodes



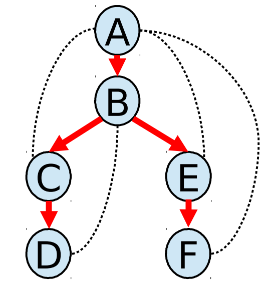
Context minimal AND/OR search graph

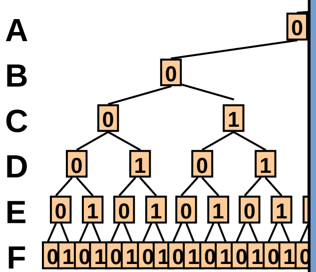
18 AND nodes

Any query is best computed over the c-minimal AO space

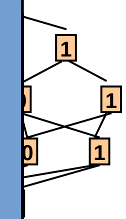


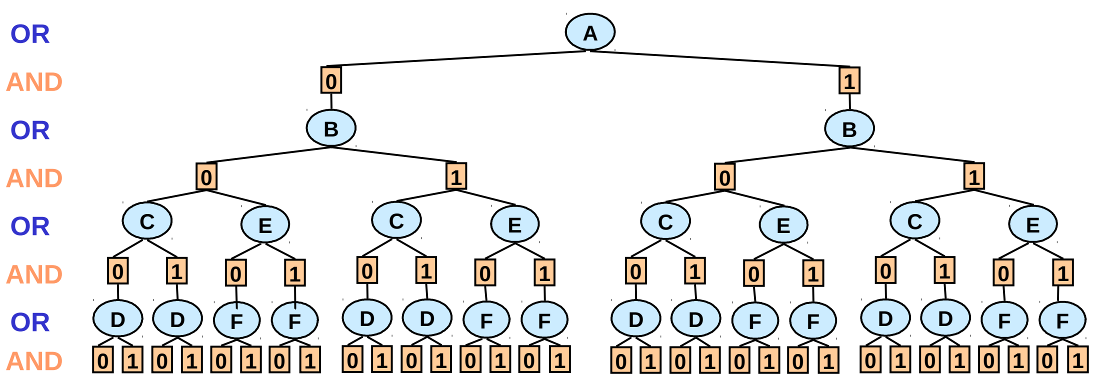
All Four Search Spaces





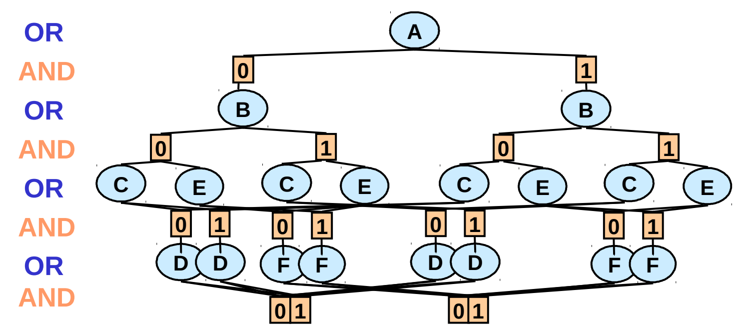
	AND/OR graph	OR graph
Space	$O(\,n\,d^{w^*}\,)$	$O(\,n\,d^{pw^*}\,)$
Time	$O(\,n\,d^{w^*}\,)$	$O(\,n\,d^{pw^*}\,)$





Full AND/OR search tree

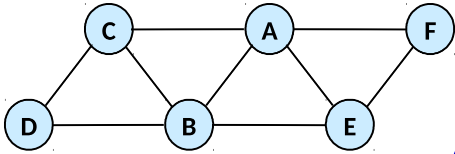
54 AND nodes



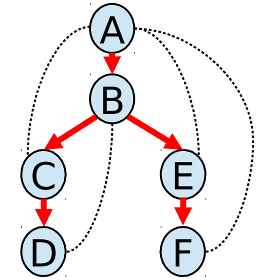
Context minimal AND/OR search graph

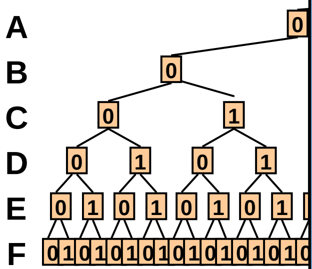
18 AND nodes

Any query is best computed over the c-minimal AO space

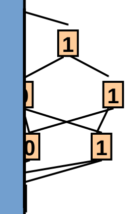


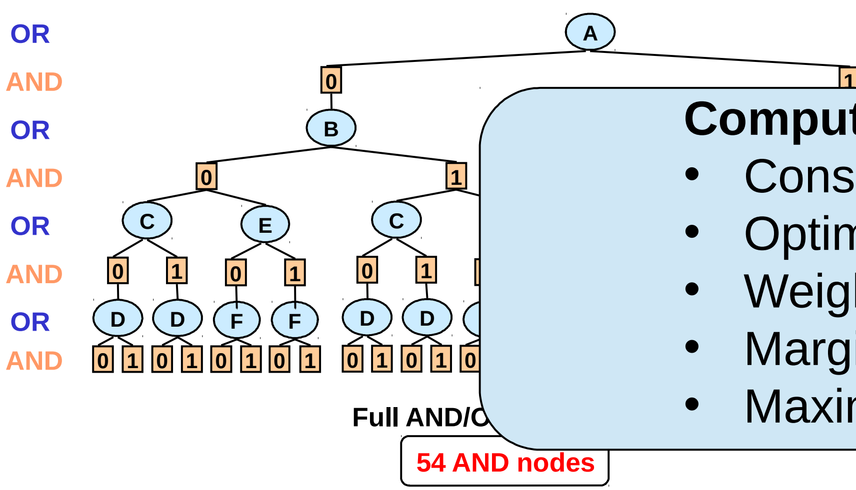
All Four Search Spaces





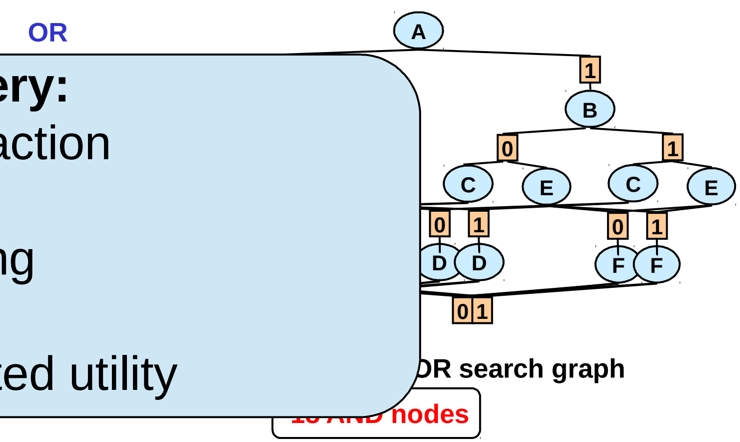
	AND/OR graph	OR graph
Space	$O(n\, d^{w^*})$	$O(n\, d^{pw^*})$
Time	$O(n\, d^{w^*})$	$O(n\, d^{pw^*})$





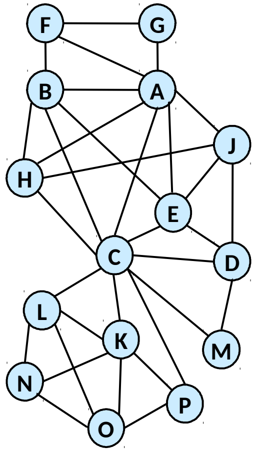
Computes any query:

- Constraint satisfaction
- Optimization
- Weighted counting
- Marginal MAP
- Maximum expected utility

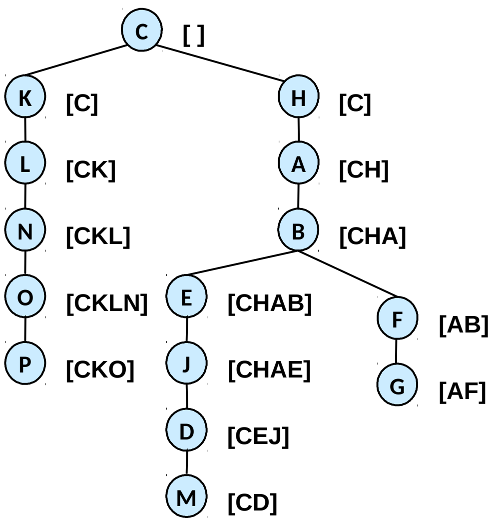


Any query is best computed over the c-minimal AO space

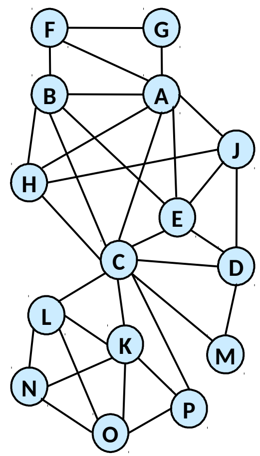
The Impact of the Pseudo Tree

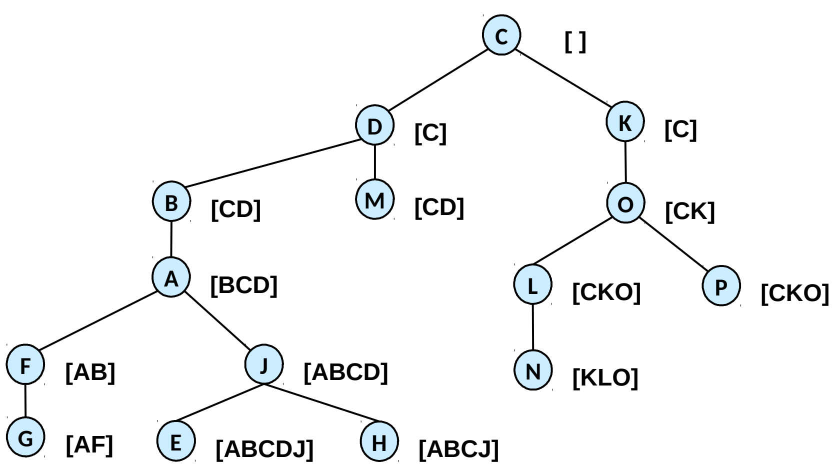


The Impact of the Pseudo Tree



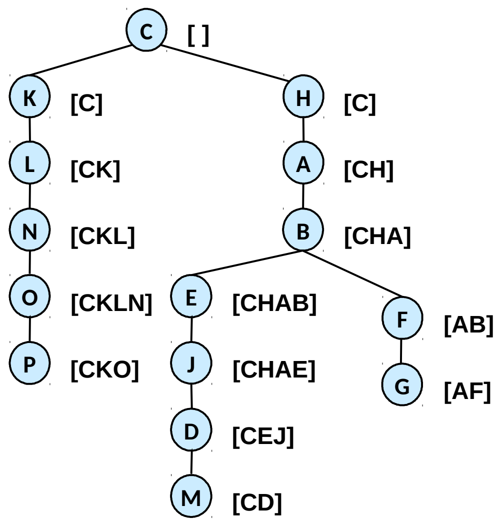
(C K H A B E J L N O D P M F G)

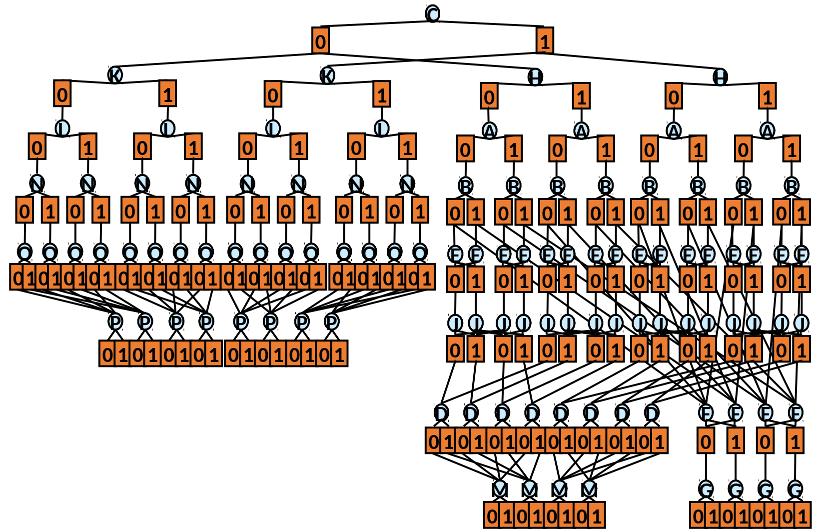


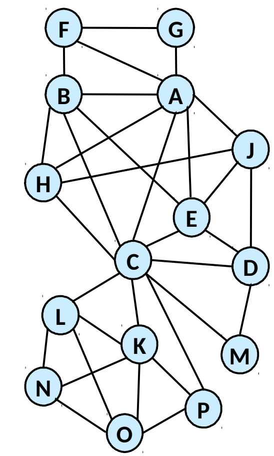


(C D K B A O M L N P J H E F G)

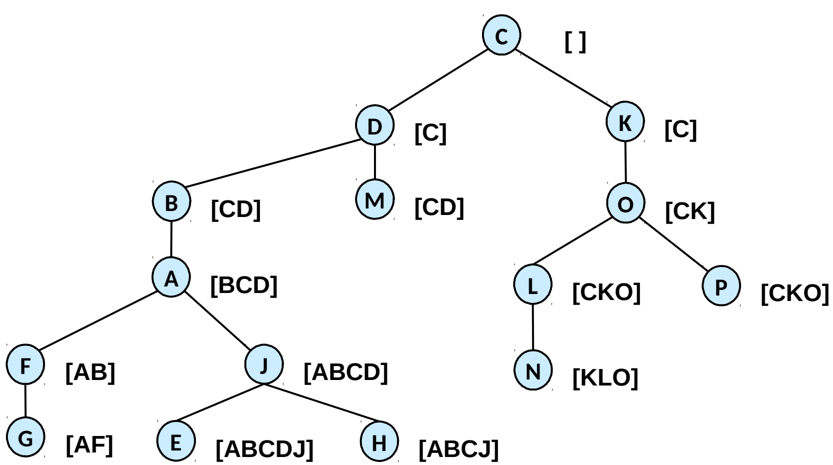
The Impact of the Pseudo Tree

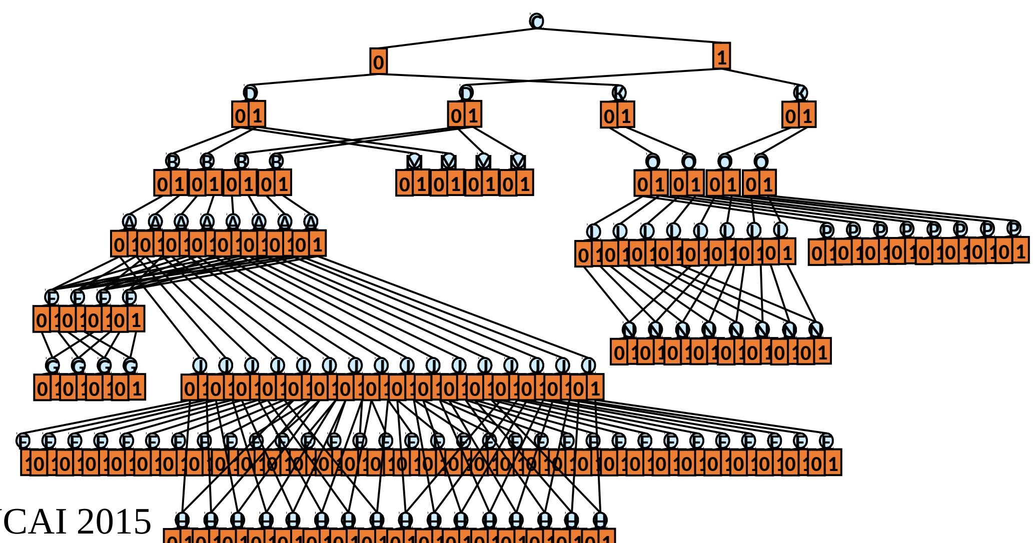






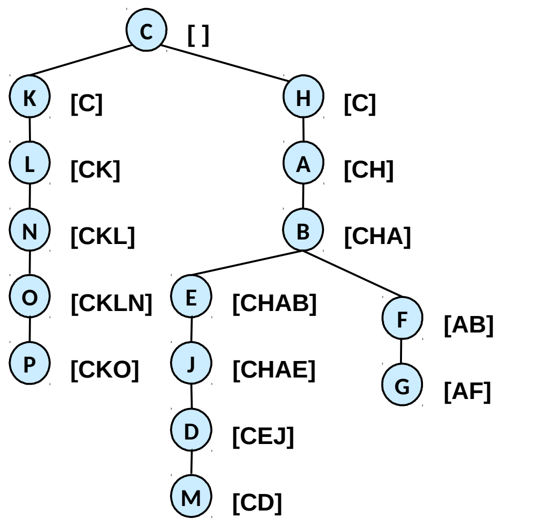
(C K H A B E J L N O D P M F G)



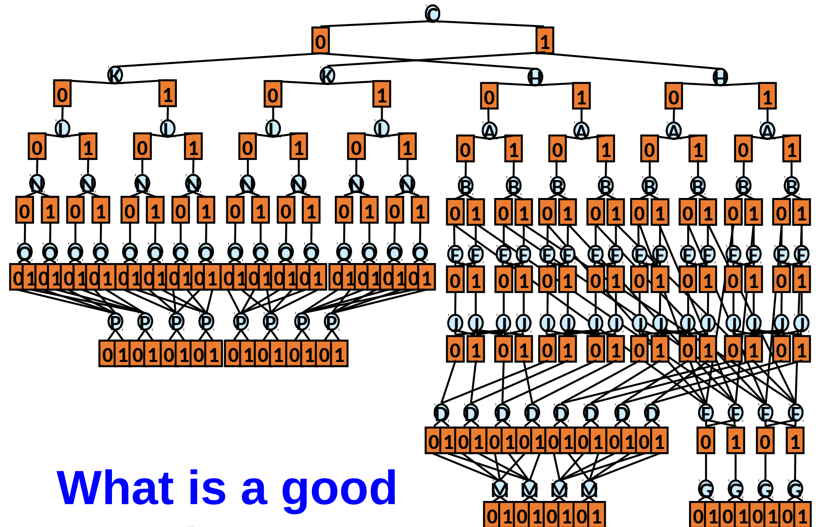


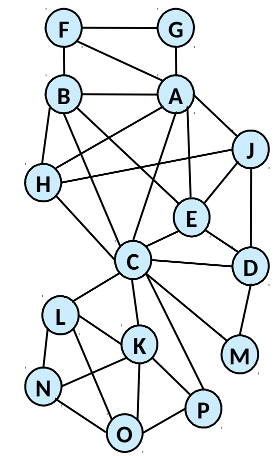
(C D K B A O M L N P J H E F G)

The Impact of the Pseudo Tree



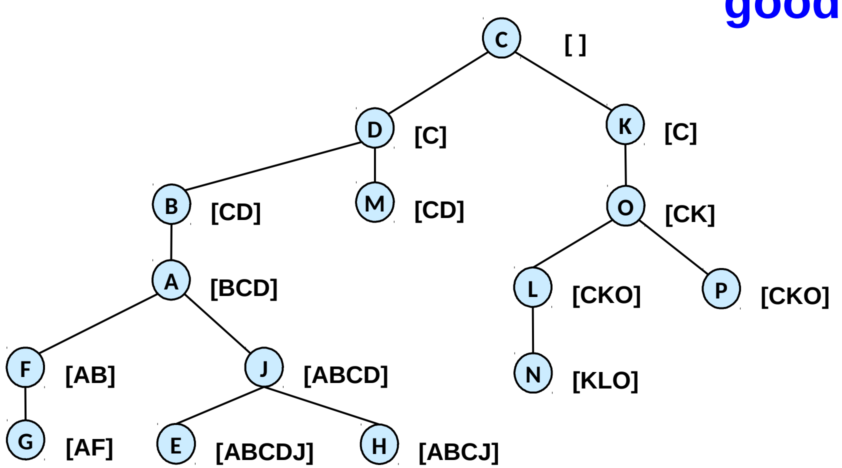
(C K H A B E J L N O D P M F G)



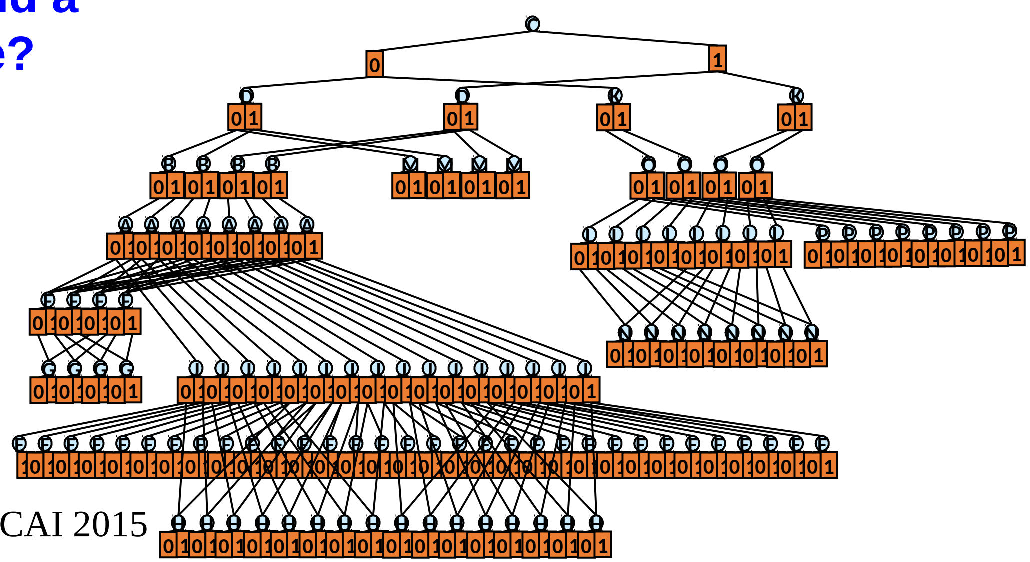


What is a good pseudo-tree?

How to find a good one?

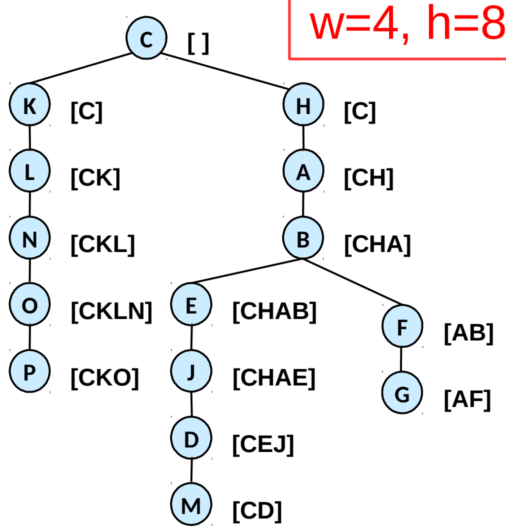


(C D K B A O M L N P J H E F G)

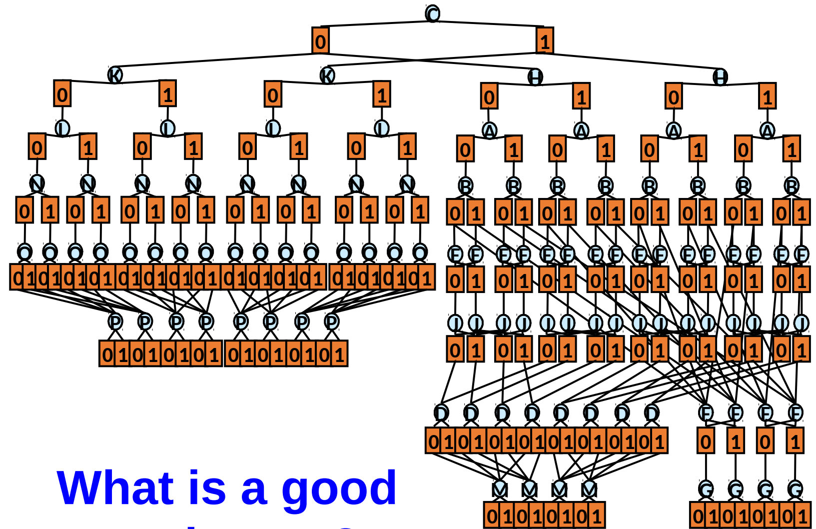


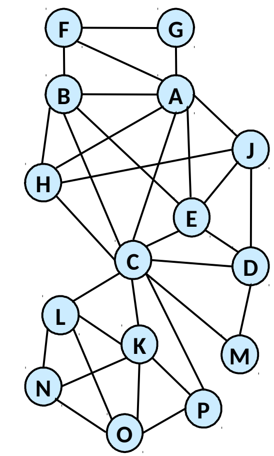
The Impact of the Pseudo Tree

w=4, h=8



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

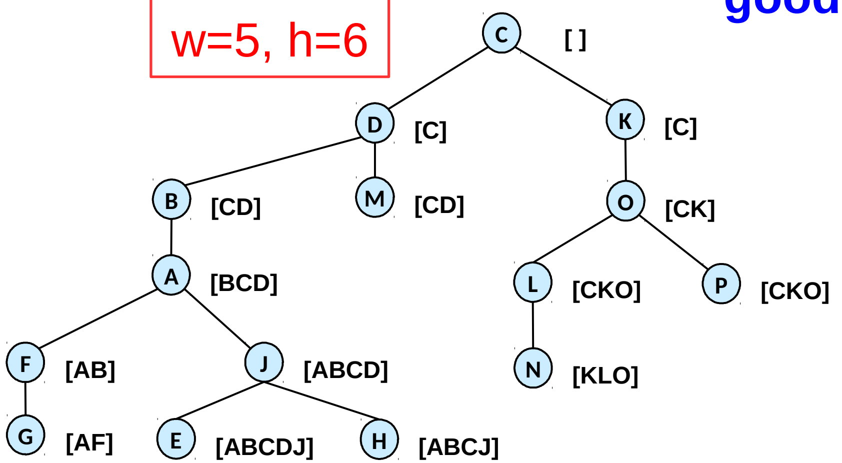




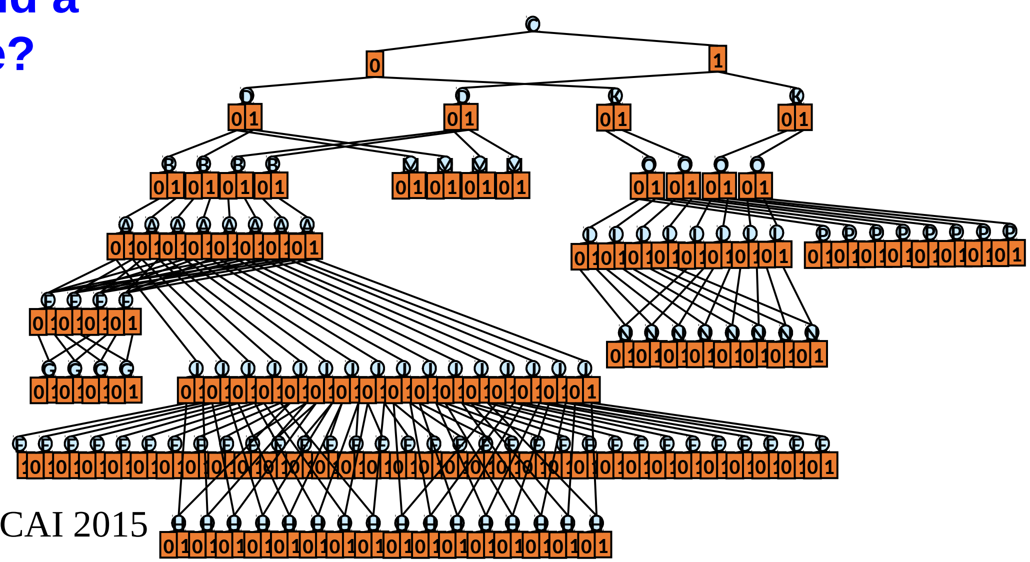
What is a good pseudo-tree?

How to find a good one?

w=5, h=6

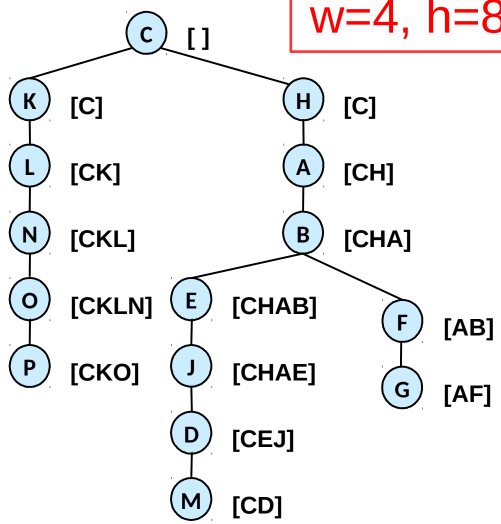


(C D K B A O M L N P J H E F G)

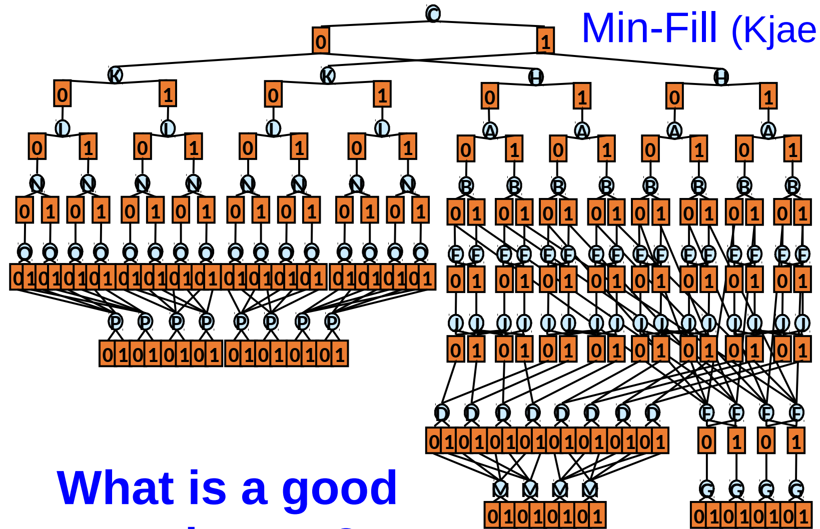


The Impact of the Pseudo Tree

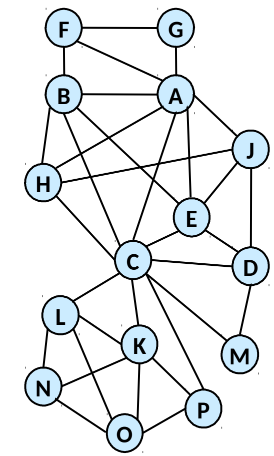
w=4, h=8



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



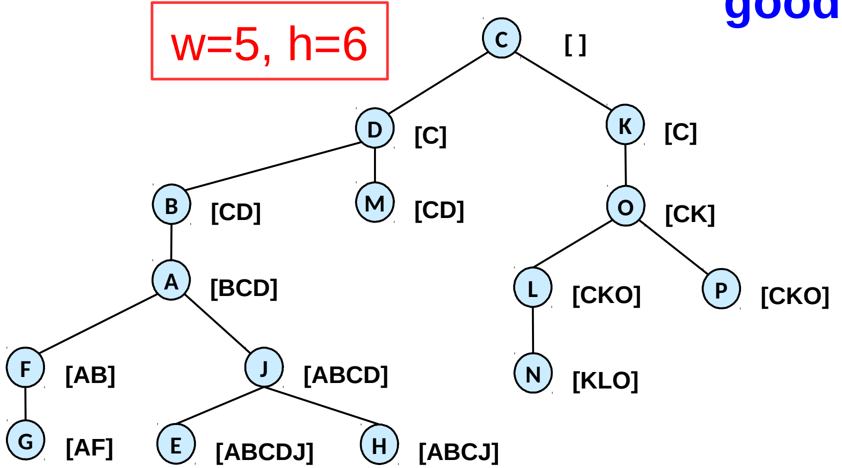
Min-Fill (Kjaerulff90)



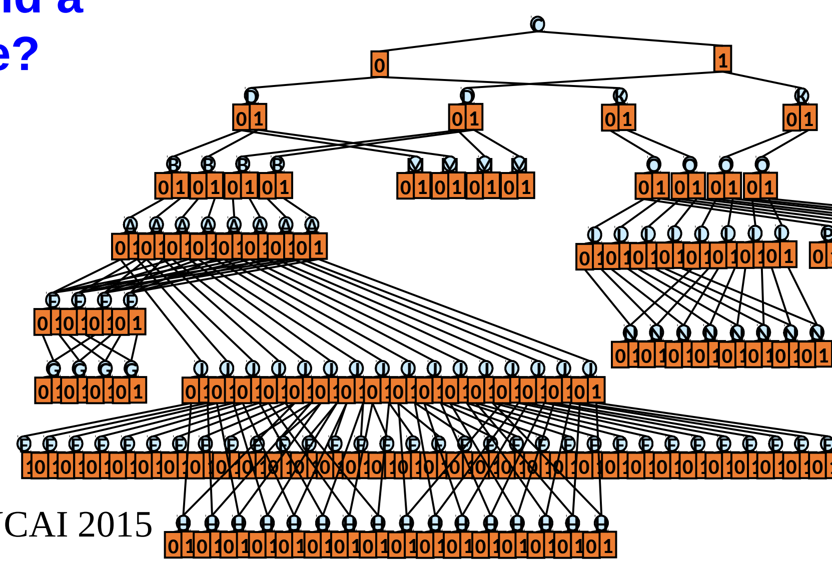
What is a good pseudo-tree?

How to find a good one?

w=5, h=6



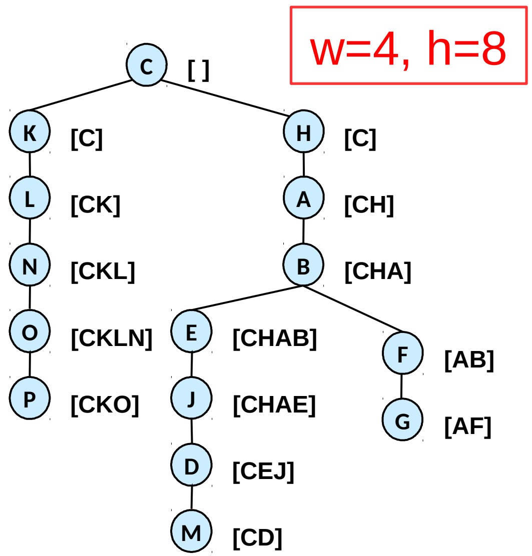
(C D K B A O M L N P J H E F G)



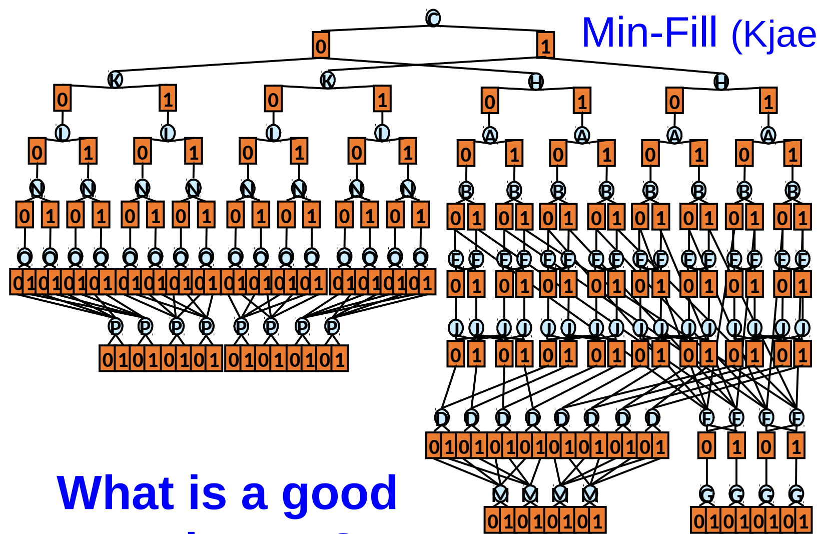
Hypergraph Partitioning (h-Metis)

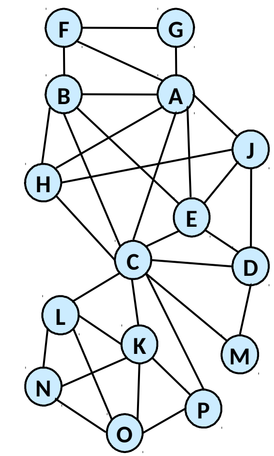
IJCAI 2015

The Impact of the Pseudo Tree



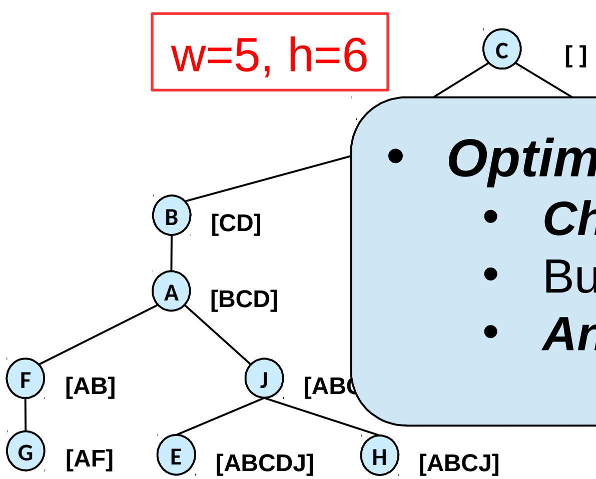
(C K H A B E J L N O D P M F G)



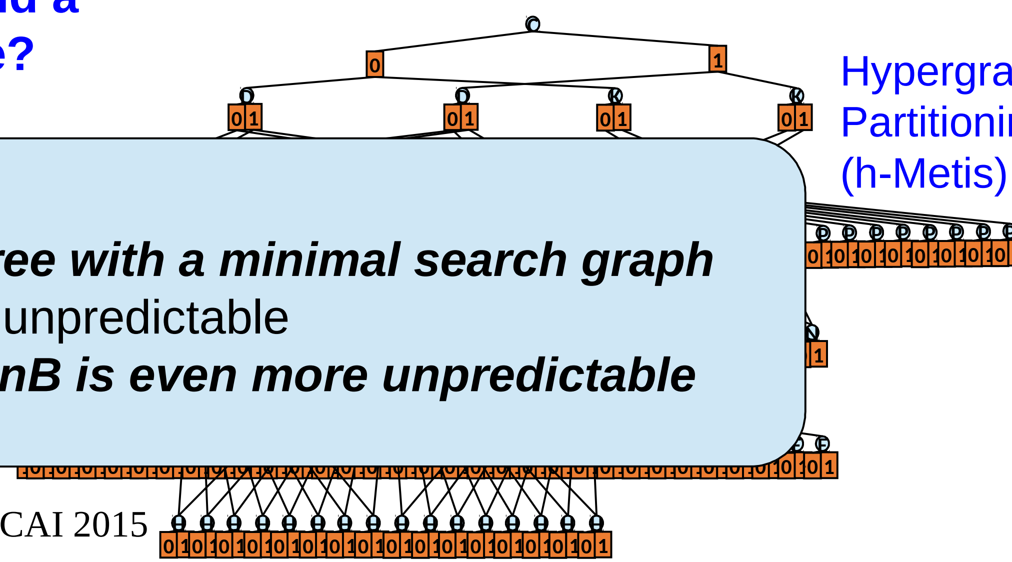


What is a good pseudo tree?

How to find a good one?



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

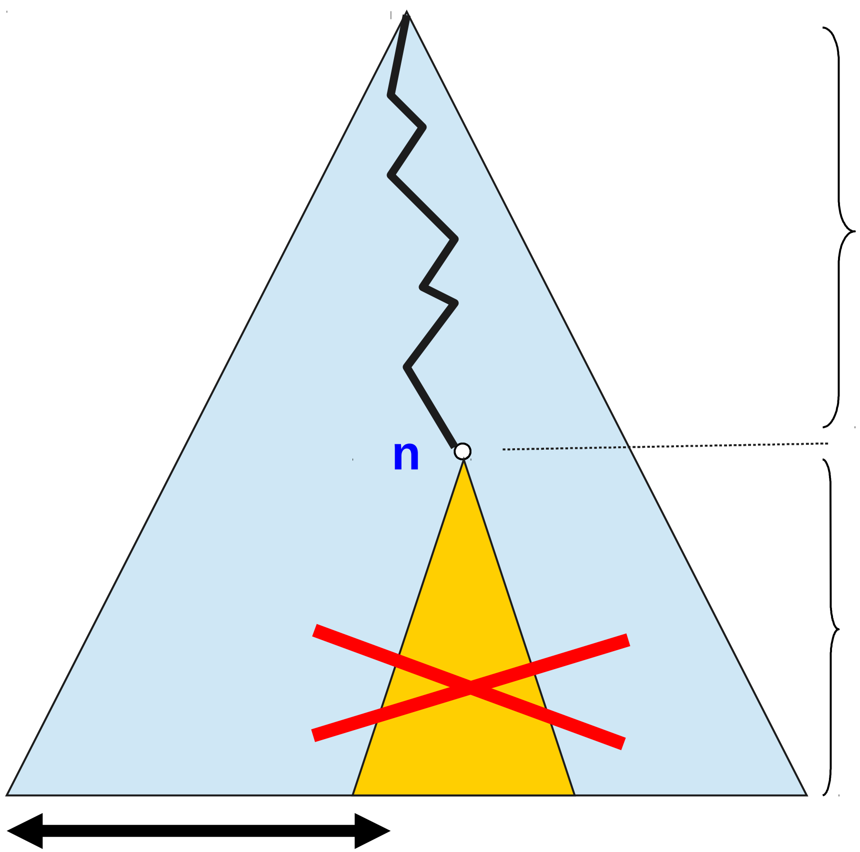


- ***Optimization***
  - ***Choose pseudo tree with a minimal search graph***
  - But determinism is unpredictable
  - ***And pruning by BnB is even more unpredictable***

Outline

- **Introduction**
- **Inference**
- **Bounds and heuristics**
- **AND/OR search**
  - AND/OR search spaces
  - Depth-first AND/OR branch and bound
  - Best-first AND/OR search
  - Advanced searches and tasks
- **Exploiting parallelism**
- **Software**

Classic Depth-First Branch and Bound



(UB) Upper Bound = best solution so far

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

g(n) : cost of the path from root to n

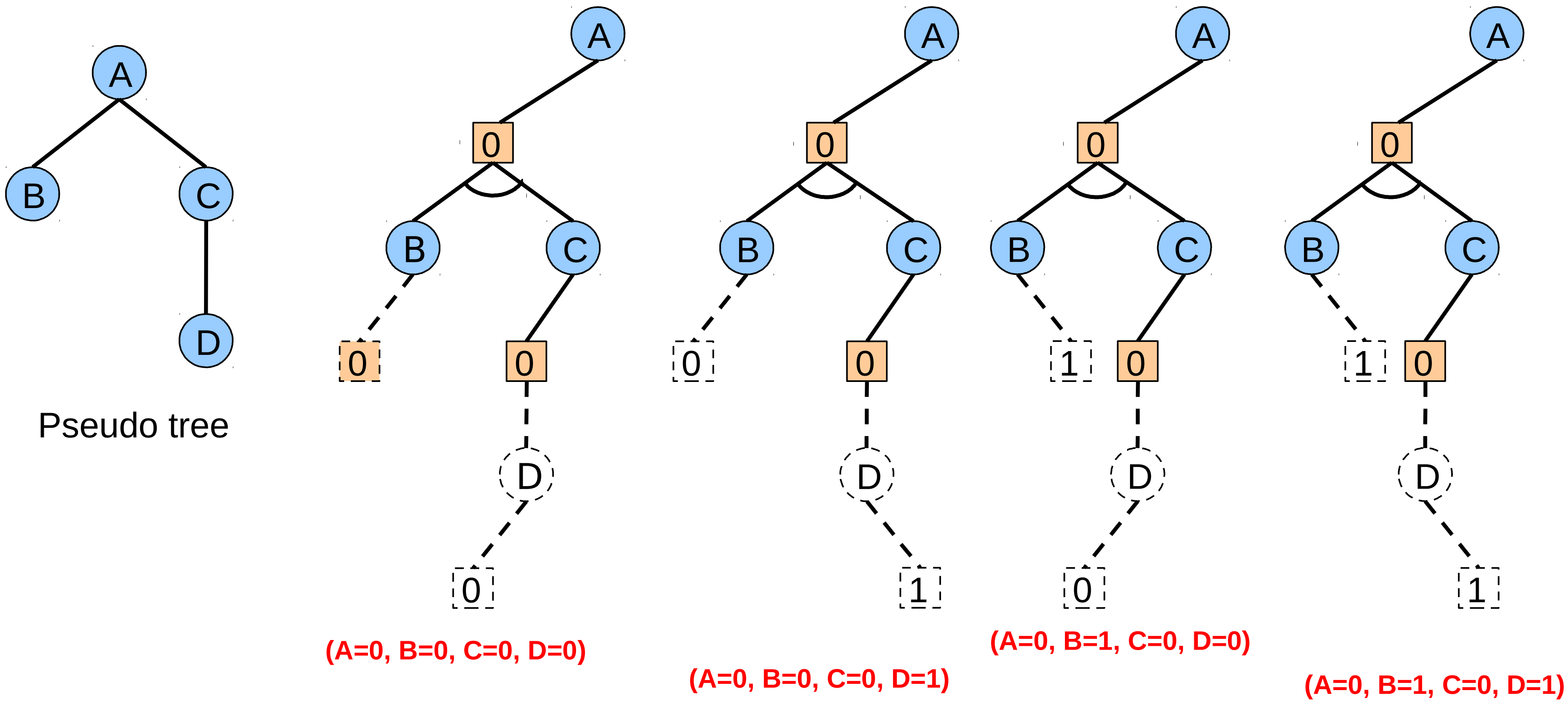
$$\tilde{f}(n) = g(n) + \tilde{h}(n)$$

(lower bound)

Prune if $\tilde{f}(n) \geq UB$

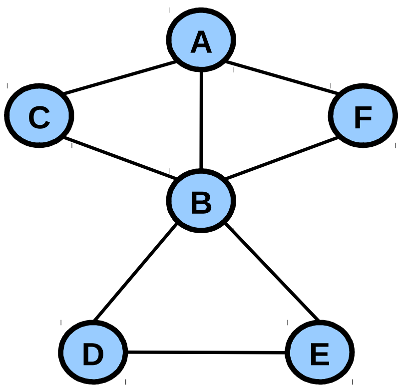
$\tilde{h}(n)$: under-estimates optimal cost below n

Partial Solution Tree



Extension(T') – solution trees that extend T'

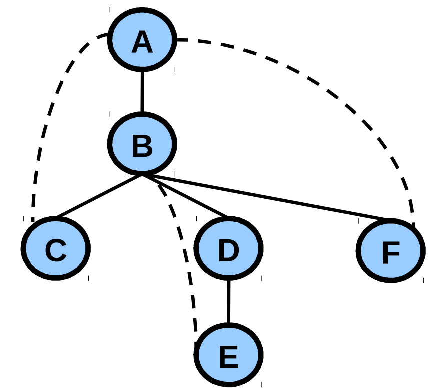
Exact Evaluation Function

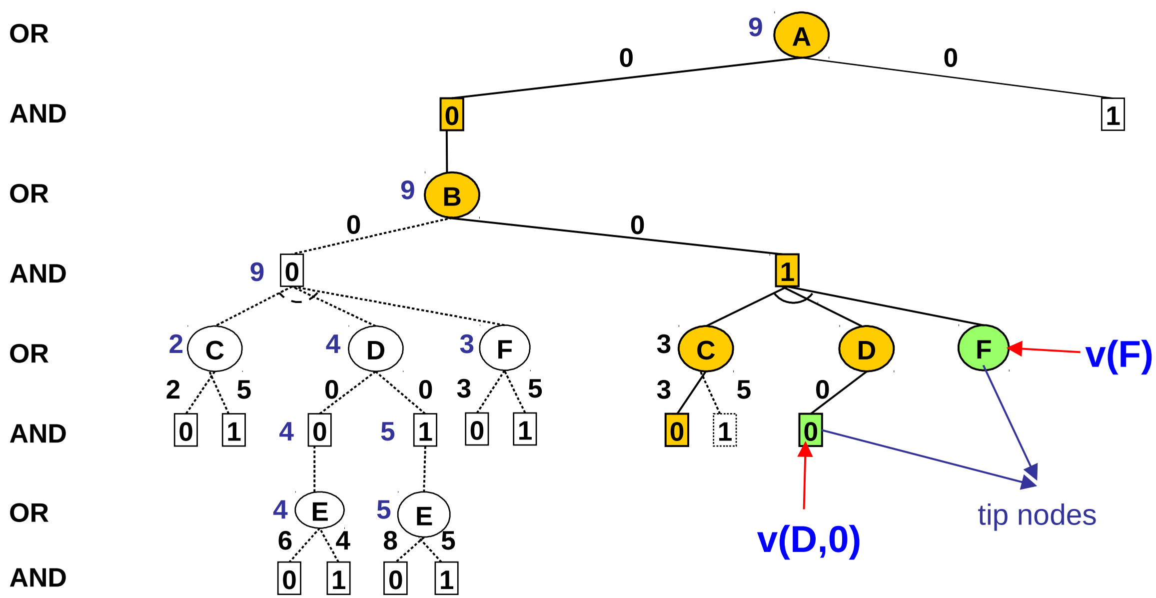


A	B	C	$f_1(ABC)$
0	0	0	2
0	0	1	5
0	1	0	3
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	2

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

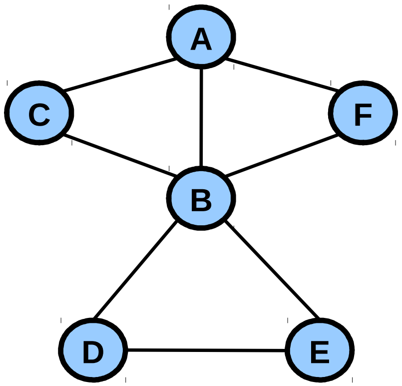
B	D	E	$f_3(BDE)$
0	0	0	6
0	0	1	4
0	1	0	8
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	4





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

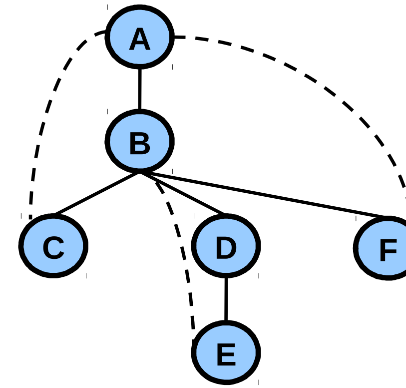
Heuristic Evaluation Function

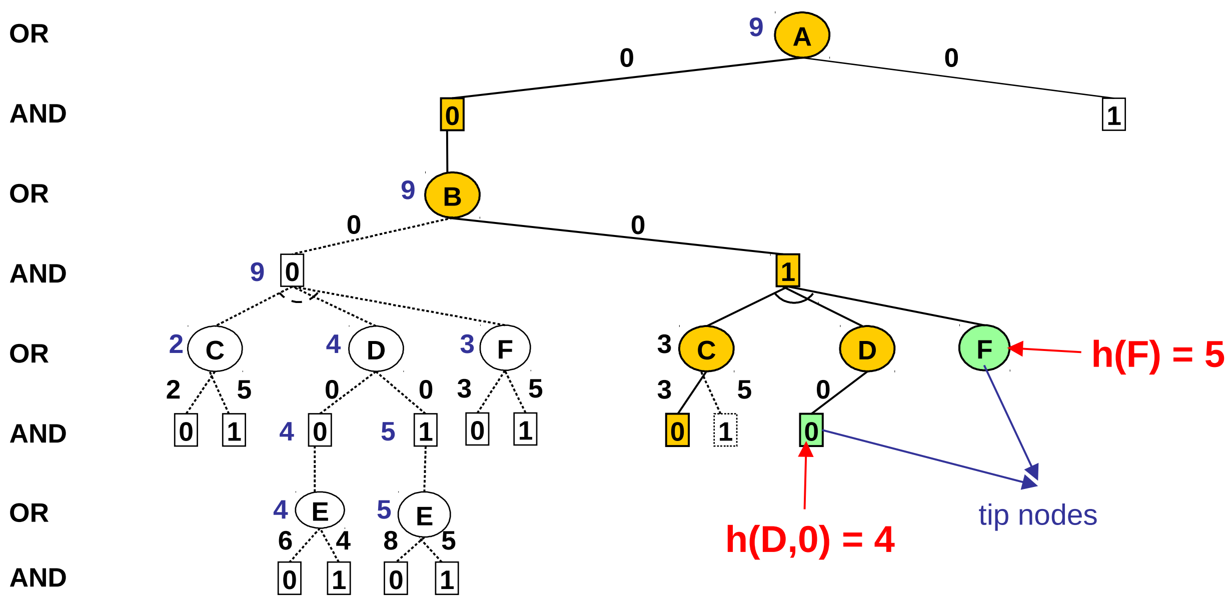


A	B	C	$f_1(ABC)$
0	0	0	2
0	0	1	5
0	1	0	3
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	2

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

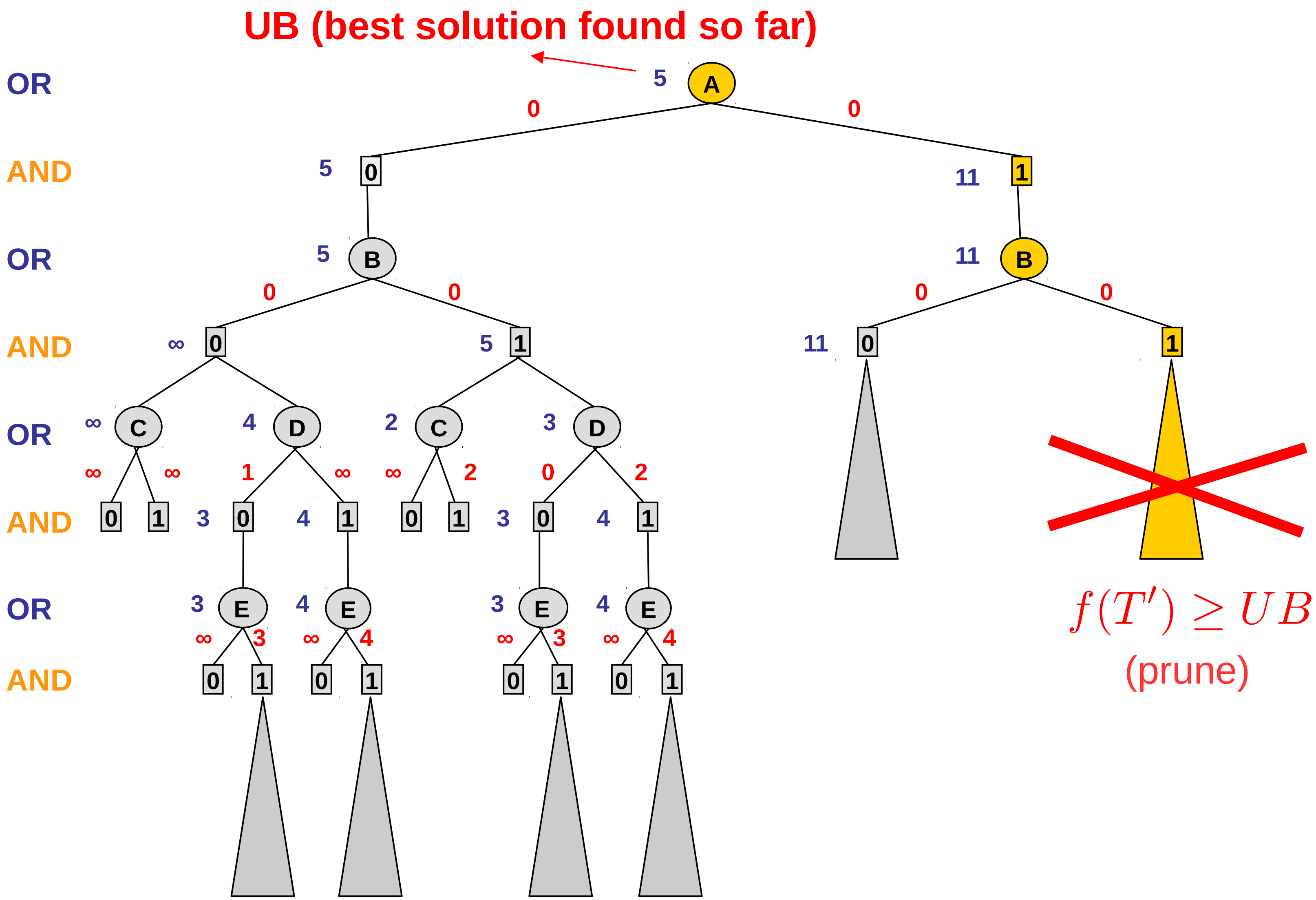
B	D	E	$f_3(BDE)$
0	0	0	6
0	0	1	4
0	1	0	8
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	4





f(T’) = w(A,0) + w(B,1) + w(C,0) + w(D,0) + h(D,0) + h(F) = 12 ≤ f*(T')

AND/OR Branch and Bound Search



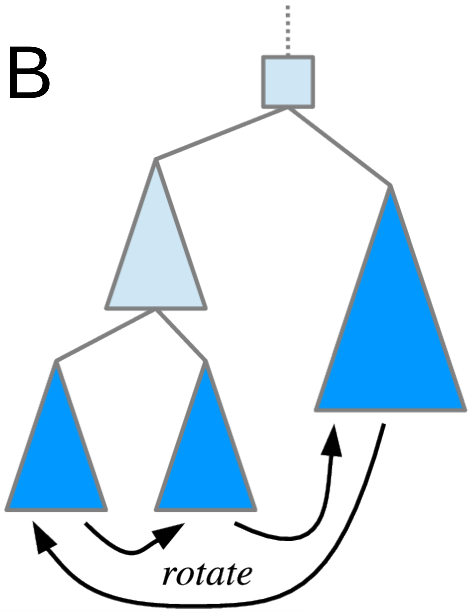
AND/OR Branch and Bound (AOBB)

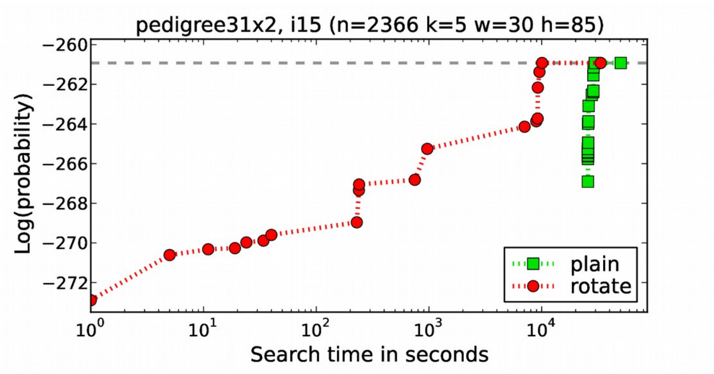
- Each node n: heuristic lower bound *h(n)* on *v(n)*
- **EXPAND** (top-down)
  - Evaluate f(T') and prune search if f(T') ≥ UB
  - If not in cache, generate successors of the tip node n
- **UPDATE** (bottom-up)
  - Update value of the parent p of n
    - OR nodes: minimization
    - AND nodes: summation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

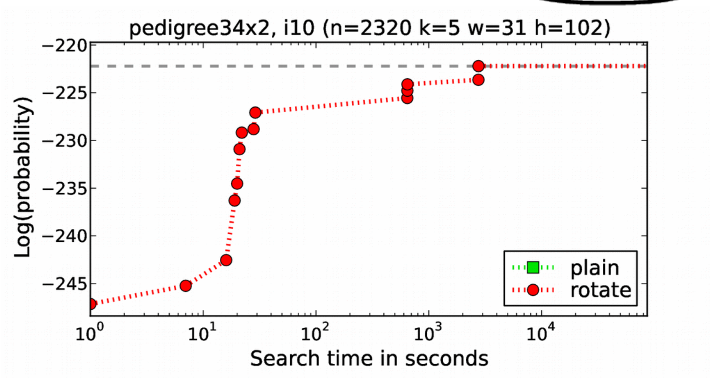
[Marinescu and Dechter, 2006; 2009]

Breadth-Rotating AOBB

- AND/OR decomposition *vs.* depth-first search:
  - Compromises anytime property of AOBB
- **Breadth-Rotating AOBB**:
  - Combined breadth/depth-first schedule
  - Maintains depth-first complexity
  - Superior experimental results







[Otten and Dechter, 2012]

Mini-Bucket Heuristics for AND/OR Search

- The depth-first and best-first AND/OR search algorithms use h(n) that can be computed:
  - Static Mini-Bucket Heuristics
    - Pre-compiled
    - Reduced computational overhead
    - Less accurate
    - Static variable ordering
  - Dynamic Mini-Bucket Heuristics
    - Computed dynamically, during search
    - Higher computational overhead
    - High accuracy
    - Dynamic variable ordering

Outline

- **Introduction**
- **Inference**
- **Bounds and heuristics**
- **AND/OR search**
  - AND/OR search spaces
  - Depth-first AND/OR branch and bound
  - Best-first AND/OR search
  - Advanced searches and tasks
- **Exploiting parallelism**
- **Software**

Basic Heuristic Search Schemes

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

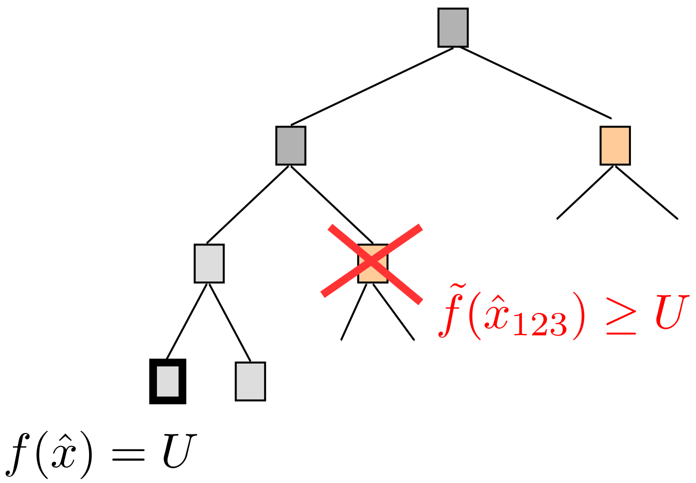
- *improve U from "above"*

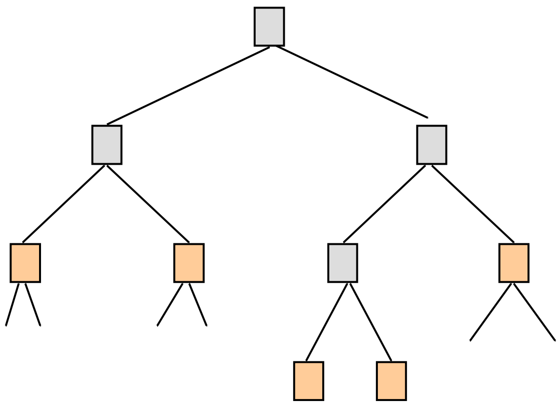
2. Best-First Search

Always expand the node with the lowest heuristic value $\tilde{f}(\hat{x}_p)$

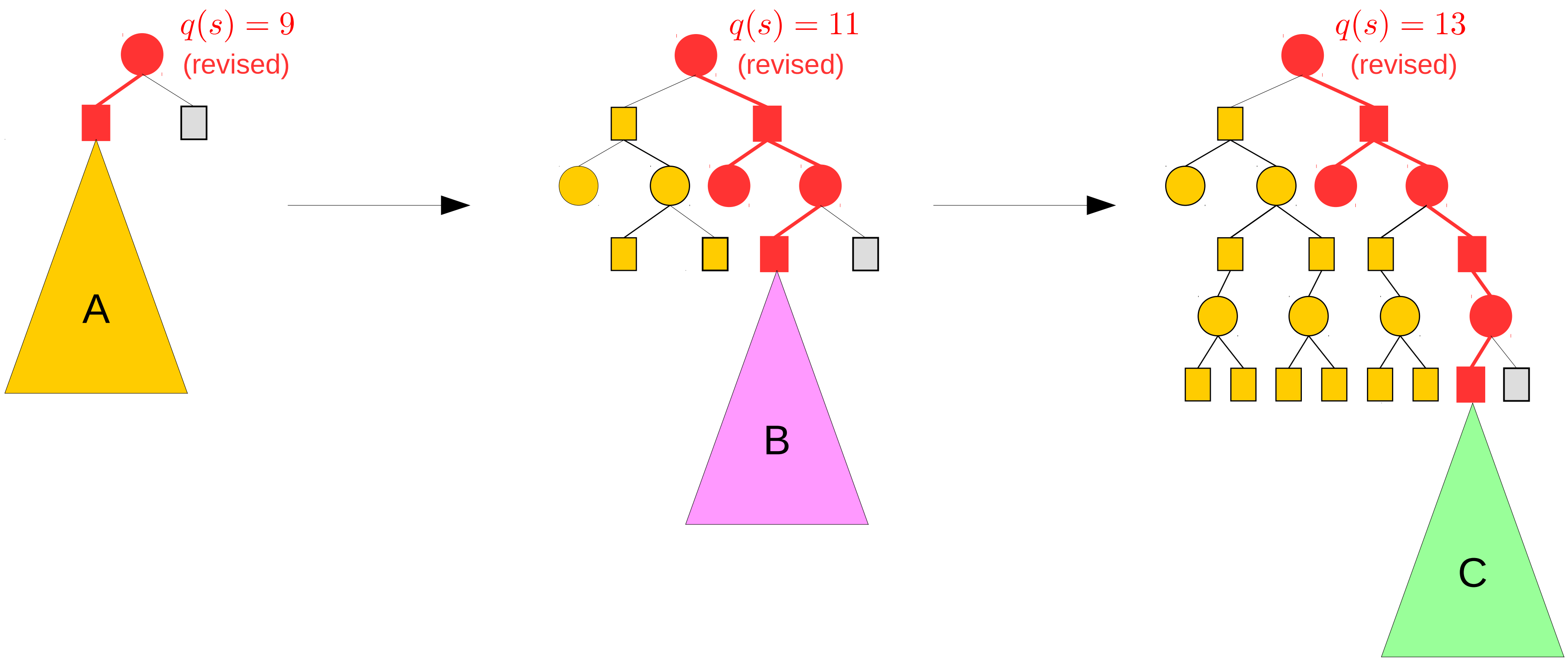
Needs lots of memory

- *improve L from "below"*





AOBF: Best-First AND/OR Search



– Each node maintains a q-value $q(n)$ initially $q(n) = h(n)$
– Node q-values revised bottom-up after each node expansion
– Update current best partial solution subtree (a tip node expanded next)
– All expanded nodes are stored in memory
– Search terminates with optimal solution

[Marinescu and Dechter, 2006; 2009]

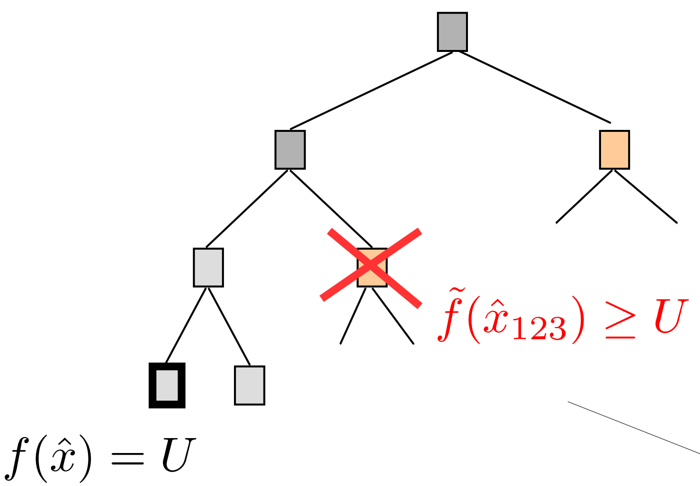
AOBF: Best-First AND/OR Search

- AO*-like traversal of the context minimal AND/OR graph
  - All nodes expanded are stored in memory
  - Each node maintains a q-value: $q(n)$
    - best lower bound on optimal cost below n
- Node q-values are revised bottom-up after each expansion
  - OR: minimization: $q(n) = \min_{n' \in succ(n)} (w(n,n') + q(n'))$
  - AND: summation: $q(n) = \Sigma_{n' \in succ(n)} q(n')$
    - (initially, q(n) = h(n) – heuristic lower bound on cost below n)
- Current best partial solution tree updated using efficient arc-marking mechanism
  - OR nodes mark best AND successor, following cost revision
  - Any of its tip nodes will be expanded at the next iteration

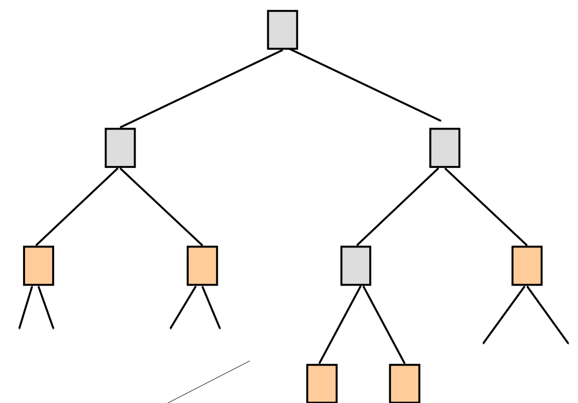
[Marinescu and Dechter, 2006; 2009]

Recursive Best-First Search

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

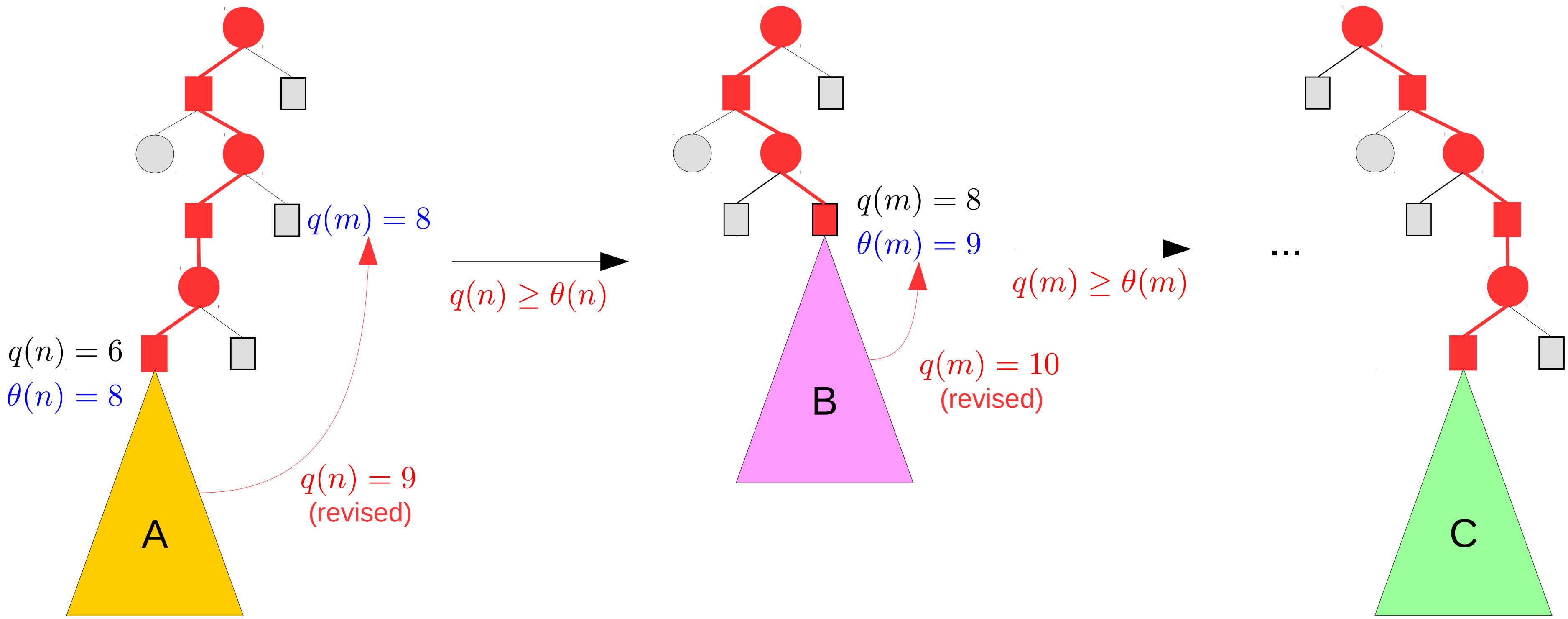


2. Best-First Search (AOBF)
Always expand the node with the lowest heuristic value $\tilde{f}(\hat{x}_p)$
Needs lots of memory



3. Recursive Best-First Search
Expand nodes best-first with the lowest heuristic value $\tilde{f}(\hat{x}_p)$
Linear or Bounded memory

RBFAOO: Recursive Best-First AND/OR Search



– Threshold $\theta(n)$ of node n is the second best cost to $q(n)$
– Search sub-problem A (blow n) until revised $q(n)$ exceeds threshold
– Backtrack to node m and discard (or cache) nodes just expanded
– Backup revised q(n) as new threshold for m and search sub-problem B
– ...

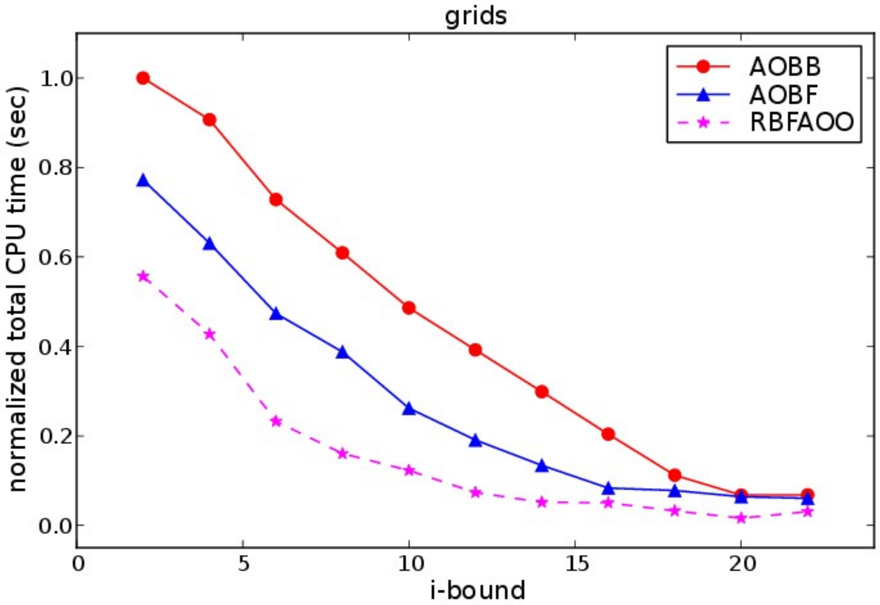
[Kishimoto and Marinescu, 2014]

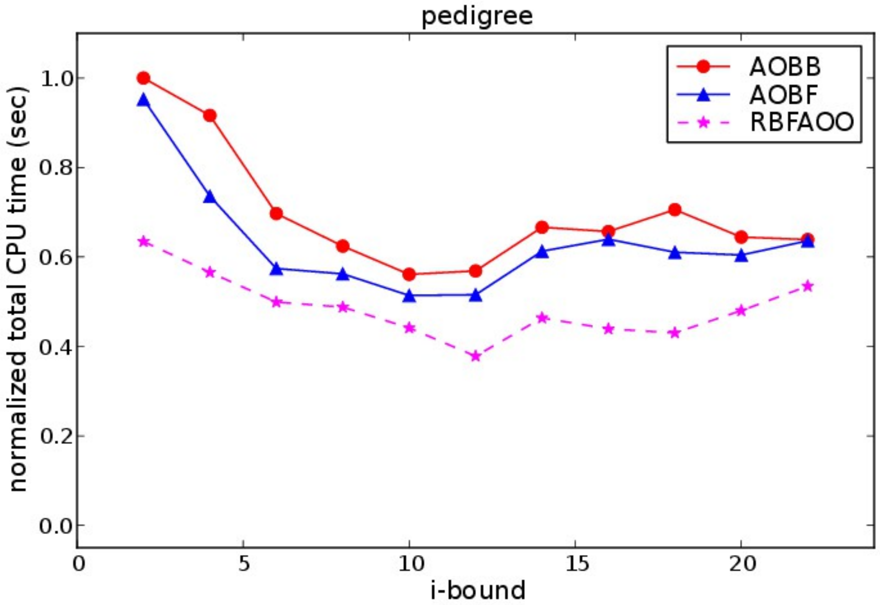
RBFAOO: Recursive Best-First AND/OR Search

- AO*-like best-first search is transformed into depth-first search with a threshold
  - Backtrack whenever: $q(n) \geq \theta(n)$
- Node q-values are updated in the usual manner
  - OR: minimization: $q(n) = \min_{n' \in succ(n)} (w(n,n') + q(n'))$
  - AND: summation: $q(n) = \Sigma_{n' \in succ(n)} q(n')$
  - (initially, q(n) = h(n) – heuristic lower bound on cost below n)
- Context-based caching is used for efficiency
- Overestimation is used to avoid frequent node re-expansions

[Kishimoto and Marinescu, 2014]

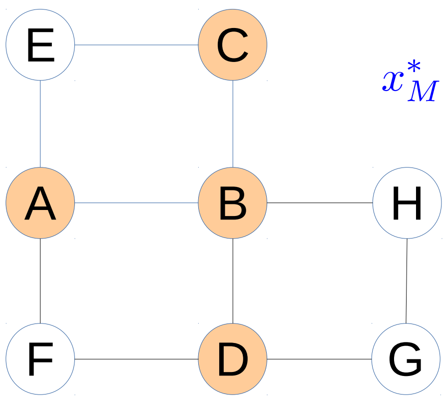
Empirical Evaluation





Exact MAP inference. Grid and Pedigree benchmarks. Time limit 1 hour.

Marginal MAP

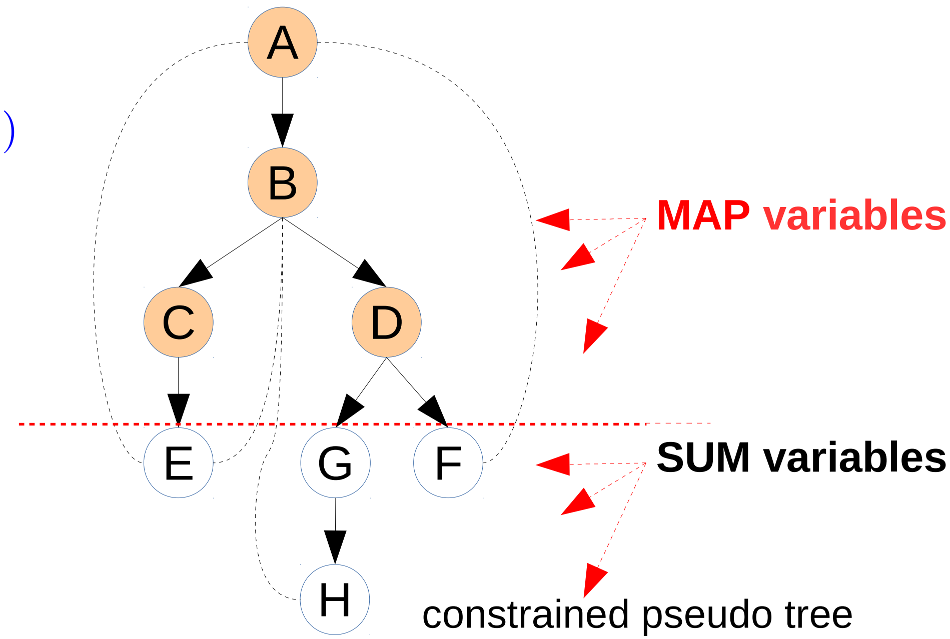


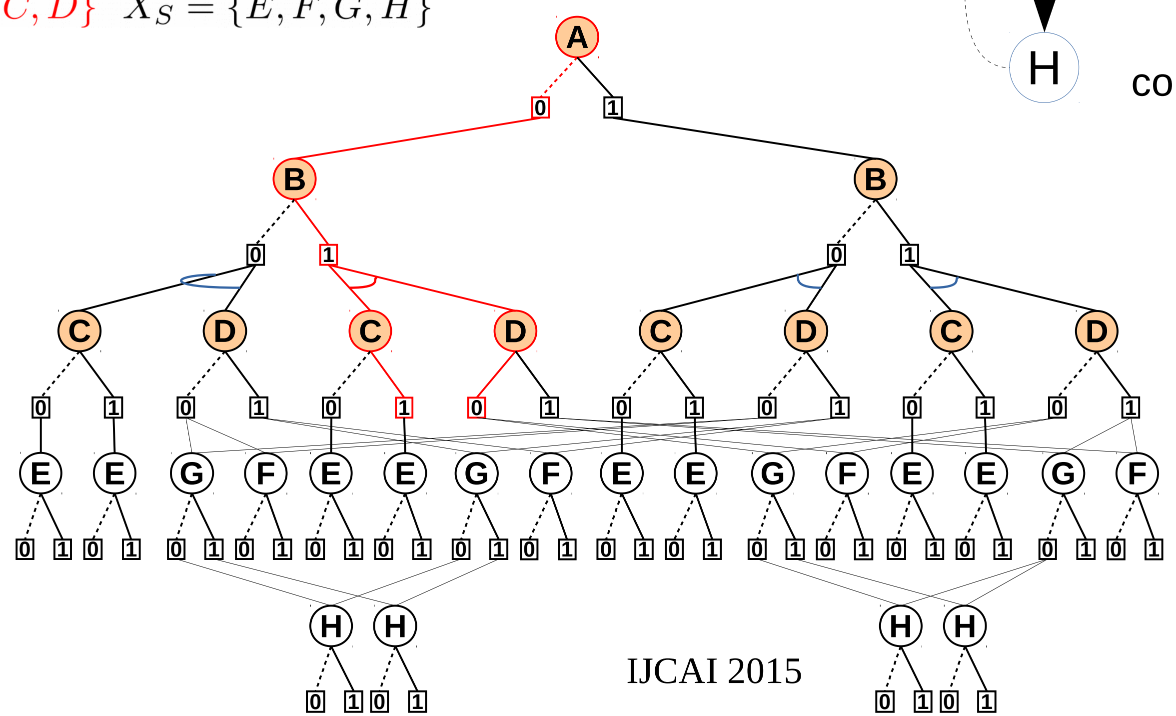
primal graph

$$x^*_M = argmax_{x_M} \sum_{x_S} \prod_{\alpha} f_\alpha(x)$$

(NP^{PP}-complete)

$X_M = \{A, B, C, D\}$ $X_S = \{E, F, G, H\}$





- **Node types**
  - OR (MAP): max
  - OR (SUM): sum
  - AND: multiplication
- **Arc weights**
  - derived from input **F**
- Problem decomposition over MAP variables

AND/OR Search for Marginal MAP

- **New advances**
  - AND/OR Branch and Bound
  - Best-First and recursive best-first AND/OR Search
  - Anytime depth-first and best-first search

    [Marinescu, Dechter, Ihler 2014,2015]; [Lee, Marinescu, Dechter, Ihler, 2016]

    - Best-performing exact and anytime Marginal MAP solvers
    - Heuristics based on Weighted Mini-Buckets (WMB)
      - WMB-MM: single pass with cost-shifting by moment matching
      - WMB-JG: iterative updates by message passing along the join-graph

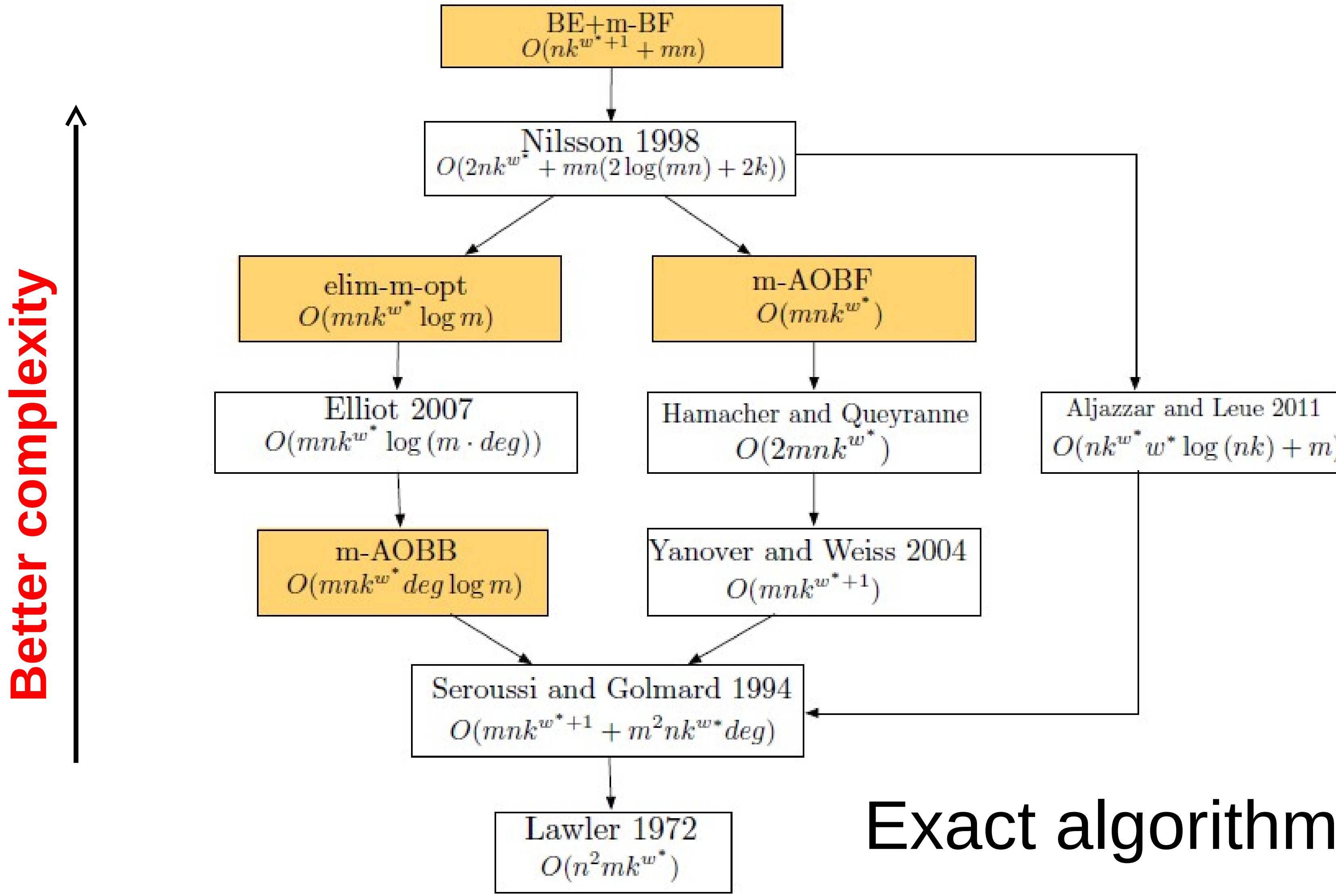
Searching for M Best Solutions

- New inference and search based algorithms for the task of finding the m best solutions
  - Search: m-A*, m-BB
  - Inference: elim-m-opt, BE+m-BF
- Extended m-A* and m-BB to AND/OR search spaces for graphical models
  - Algorithms: m-AOBB and m-AOBF
- Competitive and often superior to alternative (approximate) approaches based on LP relaxations

[Fromer and Globerson, 2009], [Batra, 2012]

[Dechter, Flerova and Marinescu; 2012]

Searching for M Best Solutions



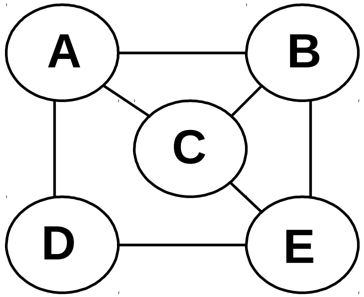
Exact algorithms!

Hybrid of Variable Elimination and Search

- Tradeoff space and time

Search Basic Step: Conditioning

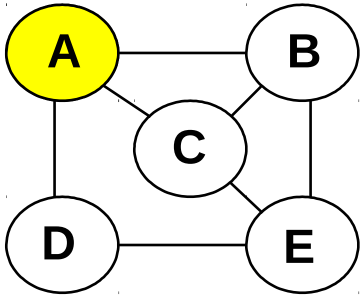
Variable Branching by Conditioning



Search Basic Step: Conditioning

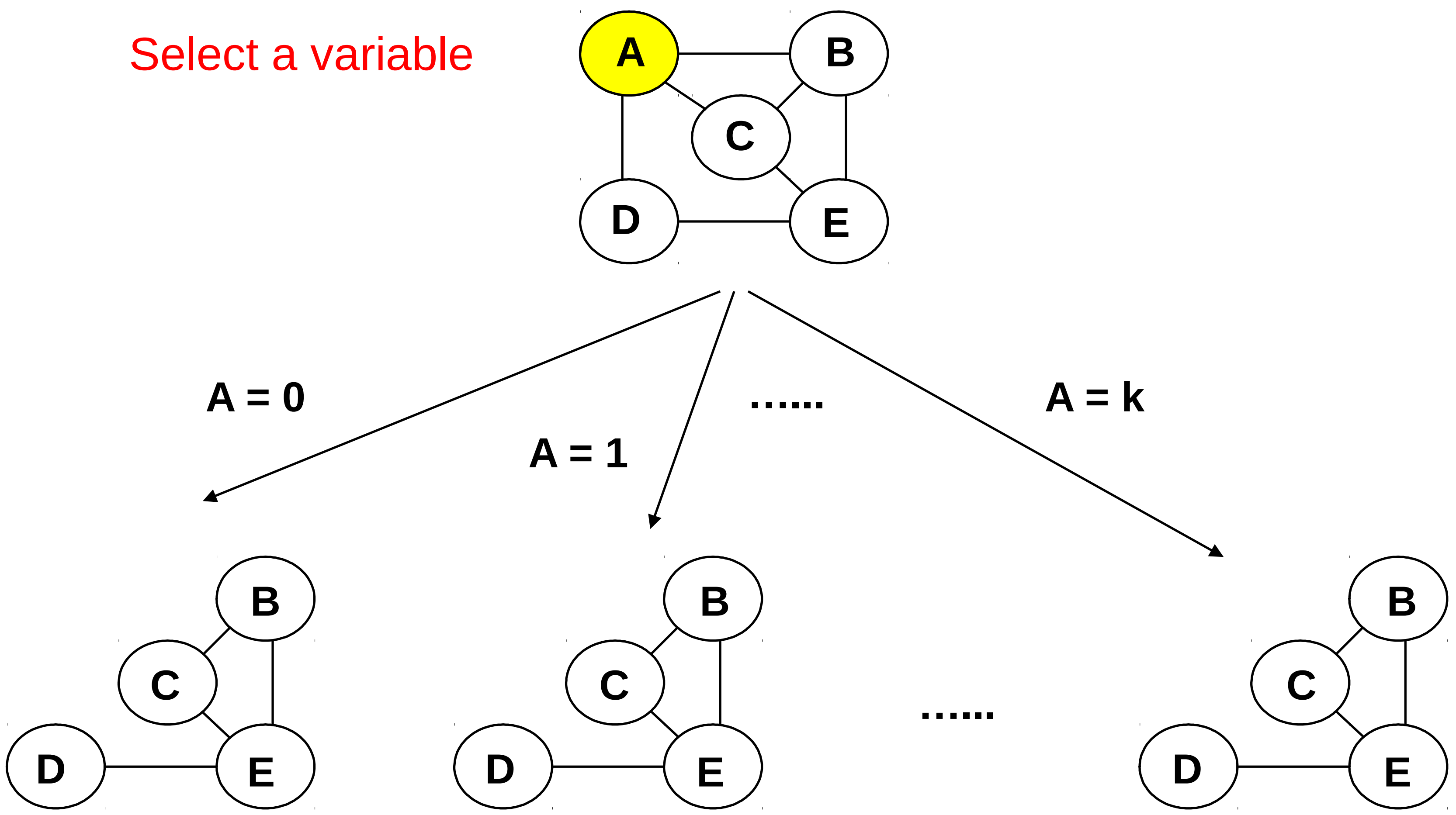
Variable Branching by Conditioning

Select a variable



Search Basic Step: Conditioning

Variable Branching by Conditioning



Search Basic Step: Conditioning

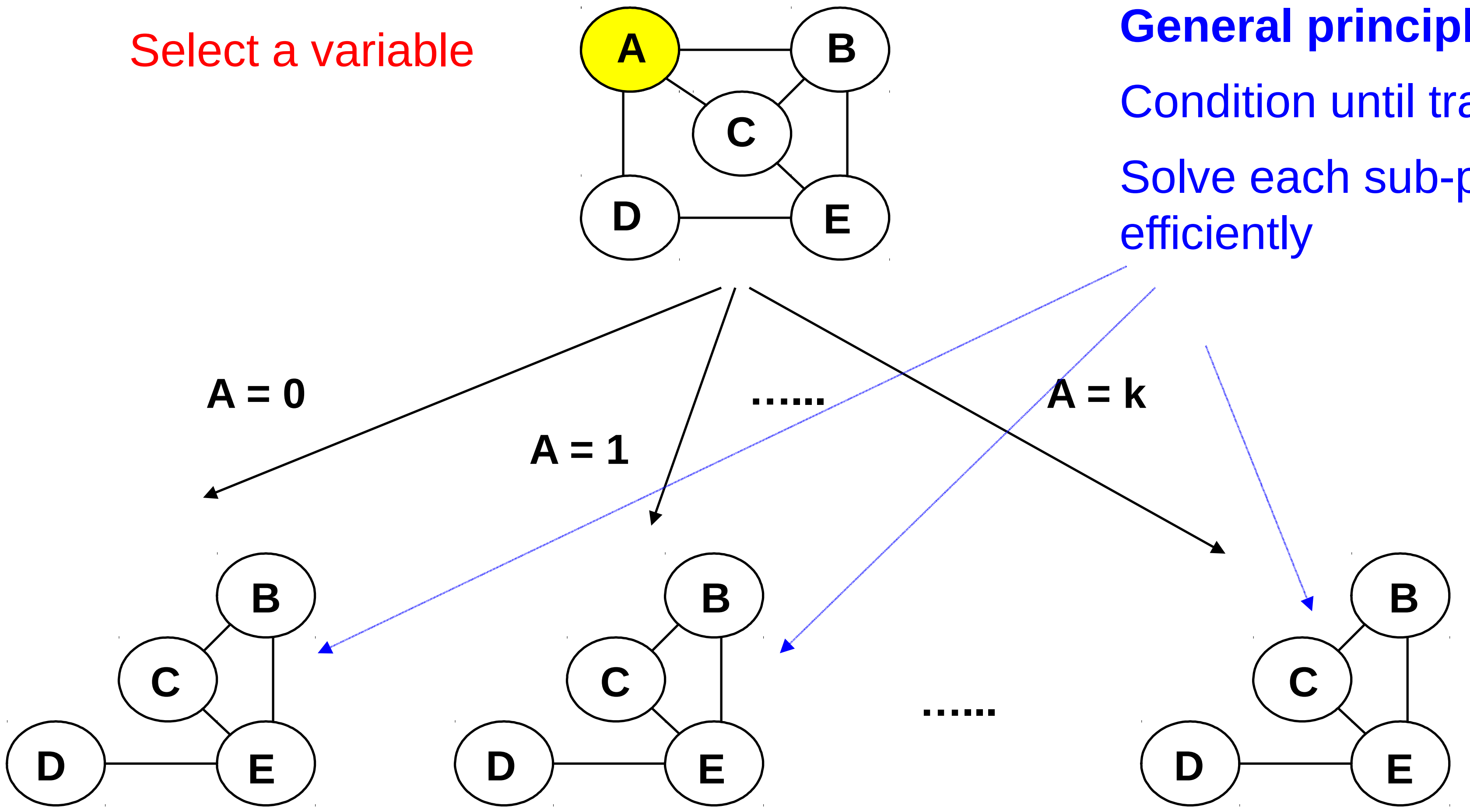
Variable Branching by Conditioning

Select a variable

General principle:

Condition until tractable

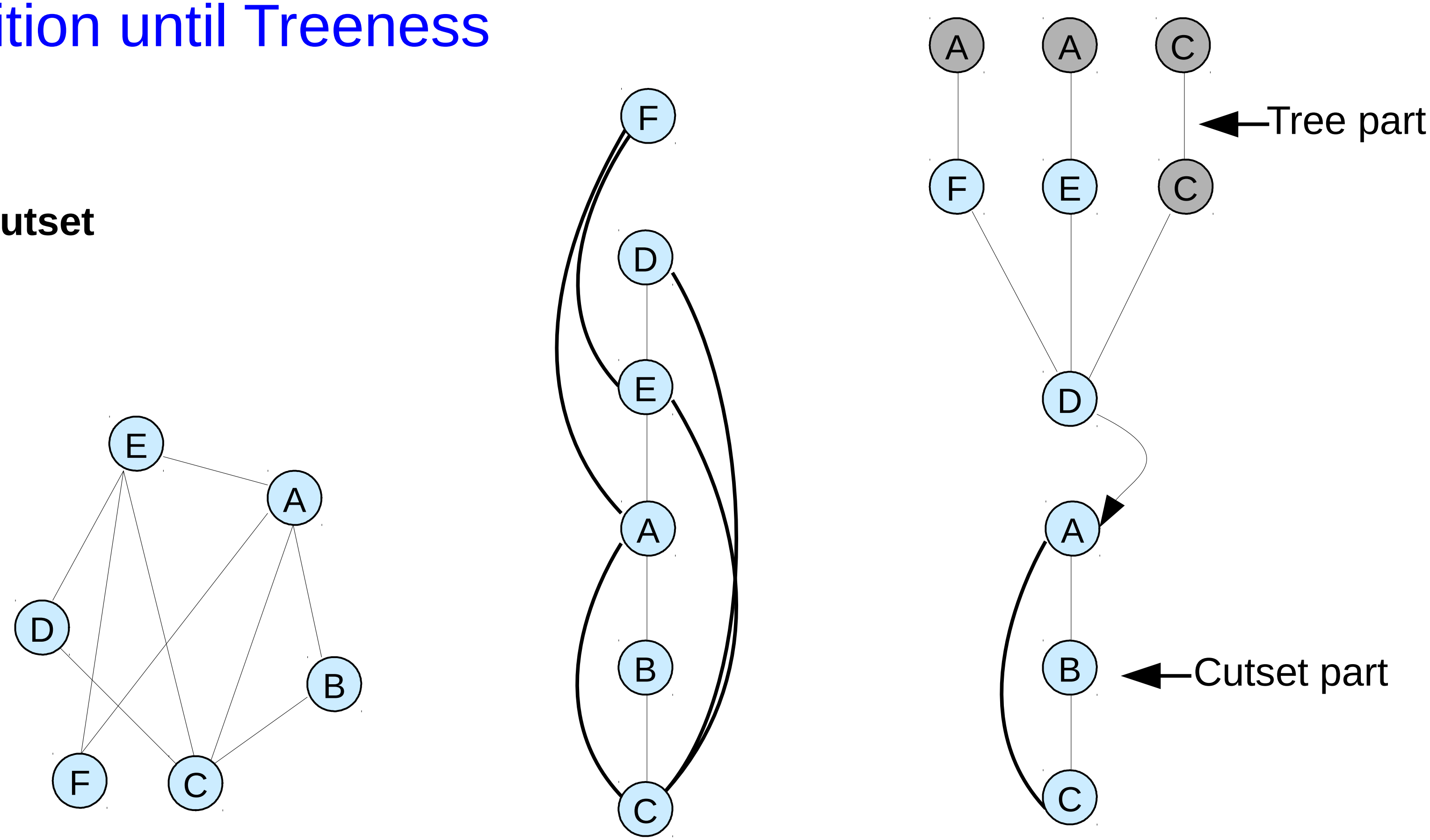
Solve each sub-problem efficiently



The Cycle-Cutset Scheme

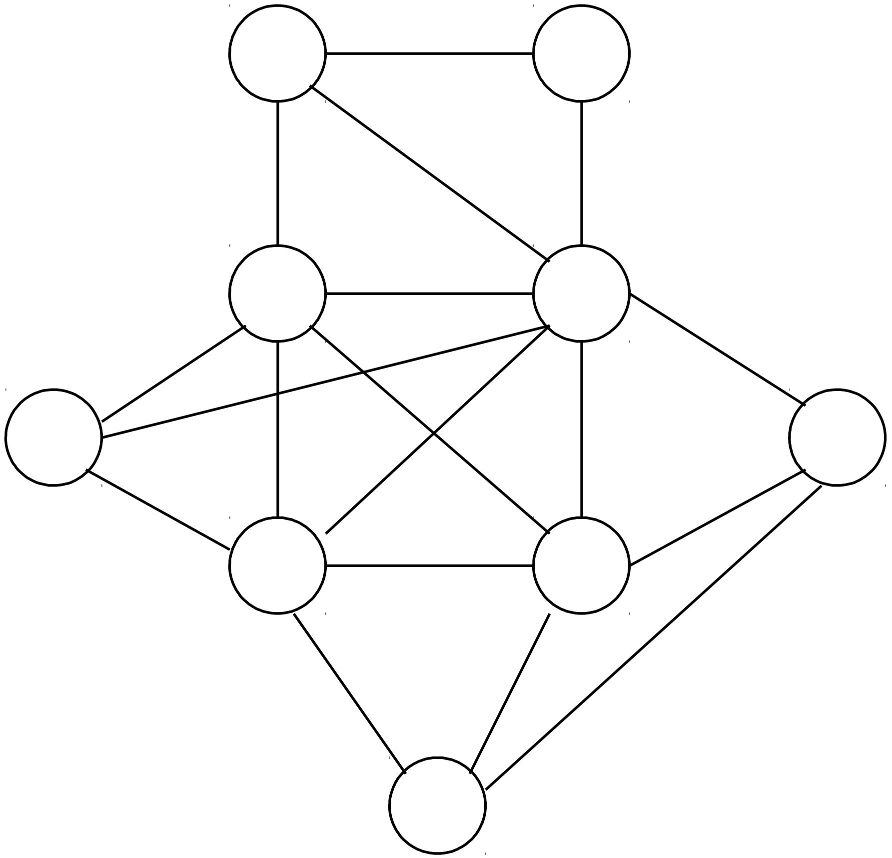
Condition until Treeness

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

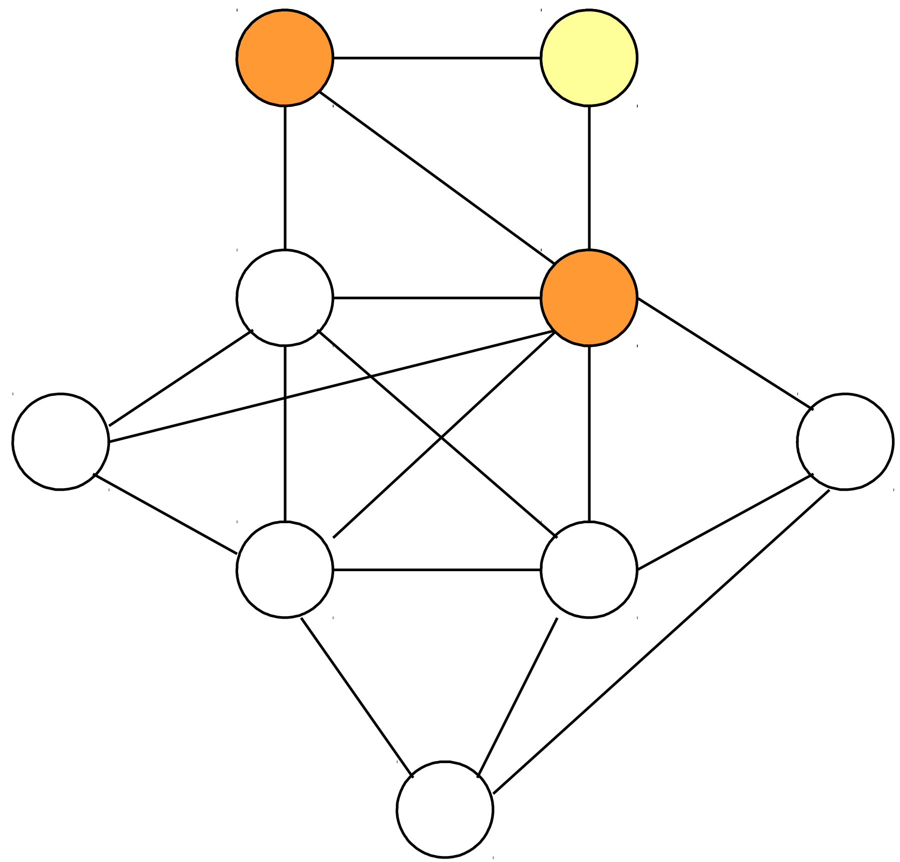


Space: exp(i), Time: O(exp(i+c(i))

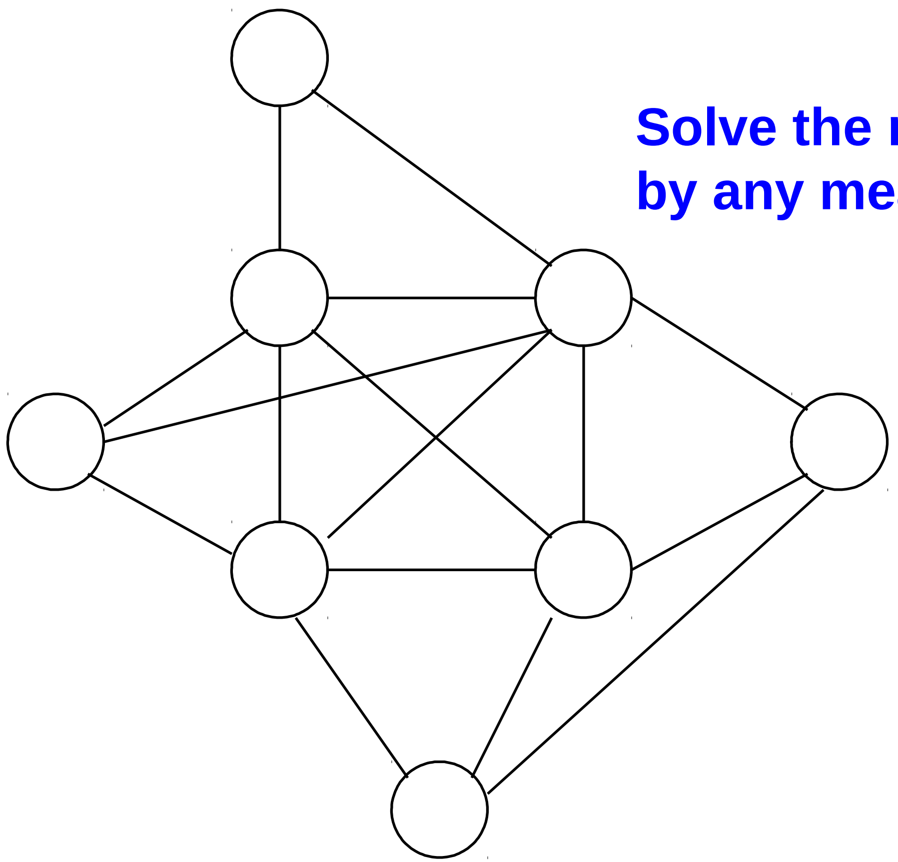
Eliminate First



Eliminate First



Eliminate First

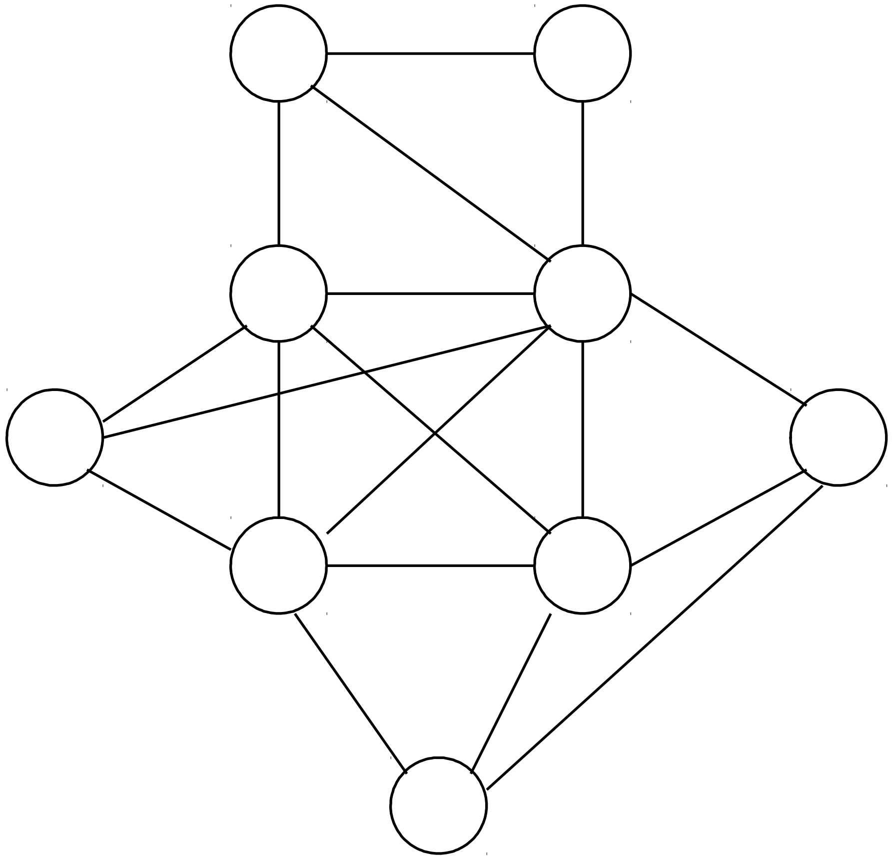


Solve the rest of the problem by any means

Hybrid Variants

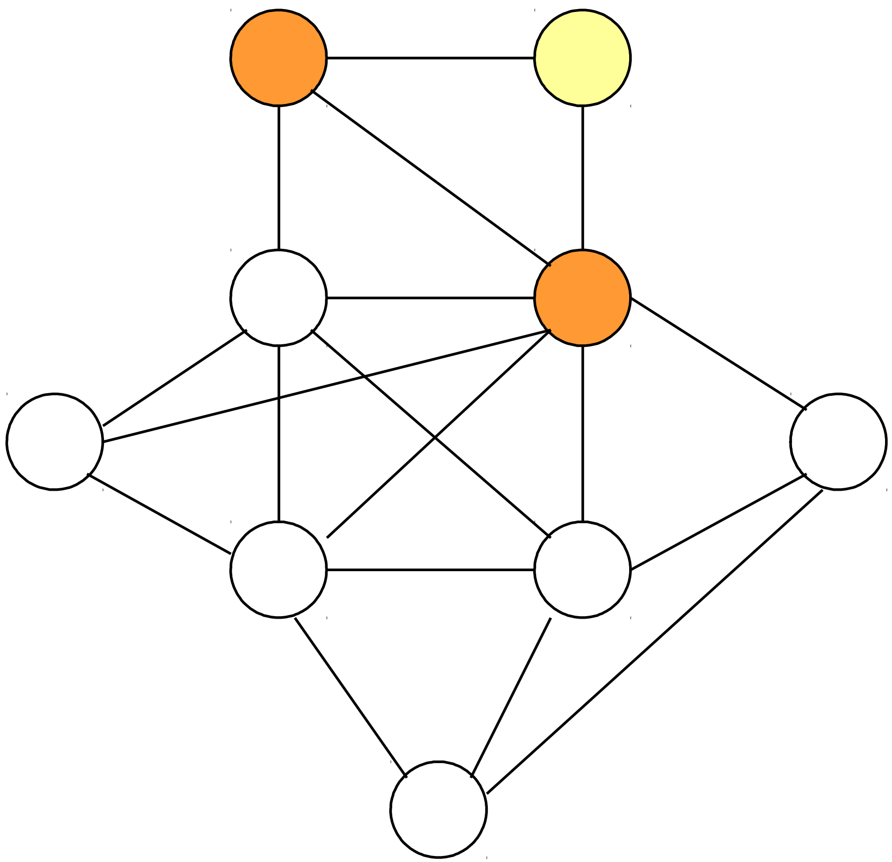
- **Condition**, **condition**, **condition**, ... and then only eliminate (w-cutset, cycle-cutset)
- **Eliminate**, **eliminate**, **eliminate**, ... and then only search
- **Interleave** conditioning and elimination steps (elim-cond(i), VE+C)

Interleaving Conditioning and Elimination

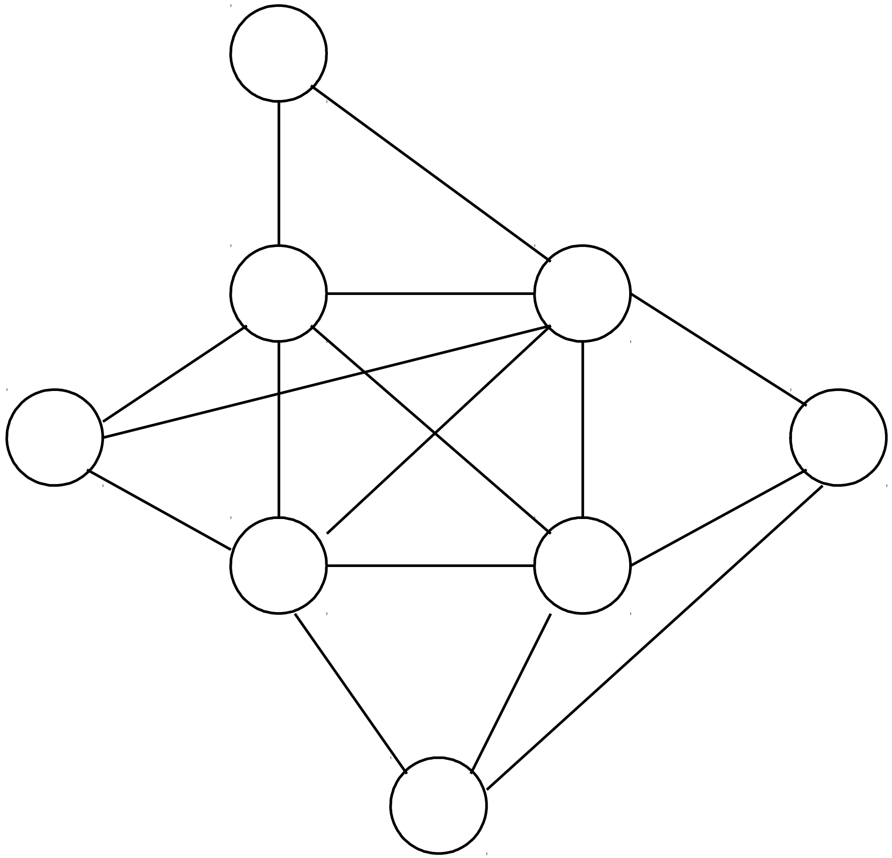


[Larrosa and Dechter, 2002]

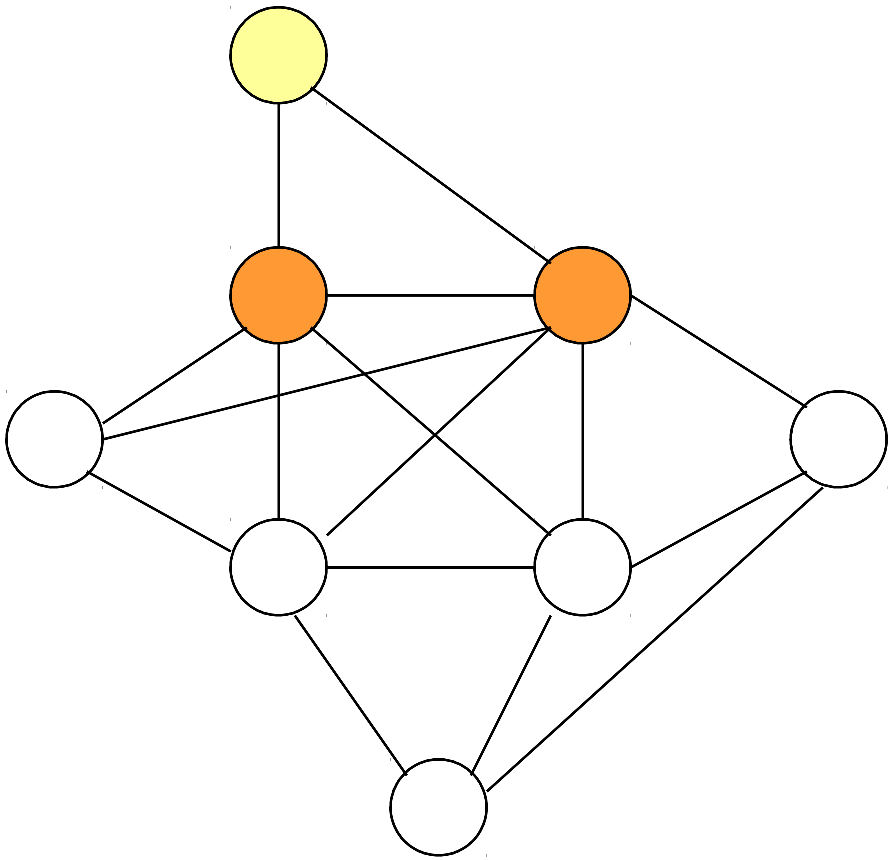
Interleaving Conditioning and Elimination



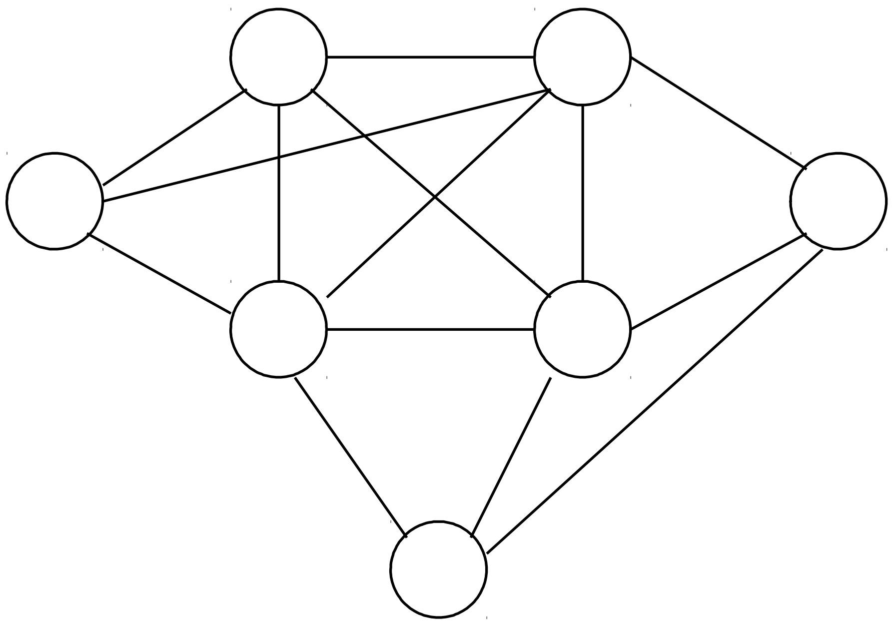
Interleaving Conditioning and Elimination



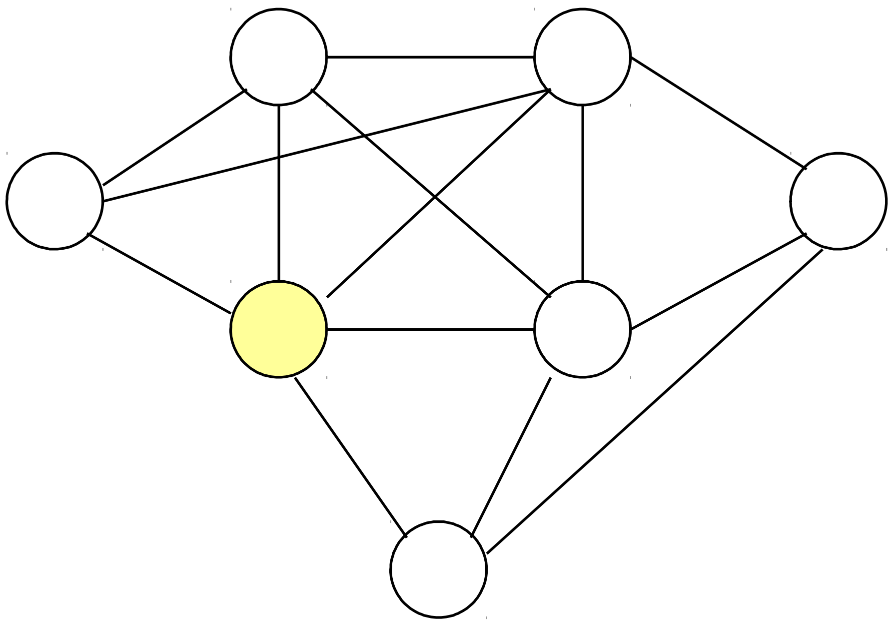
Interleaving Conditioning and Elimination



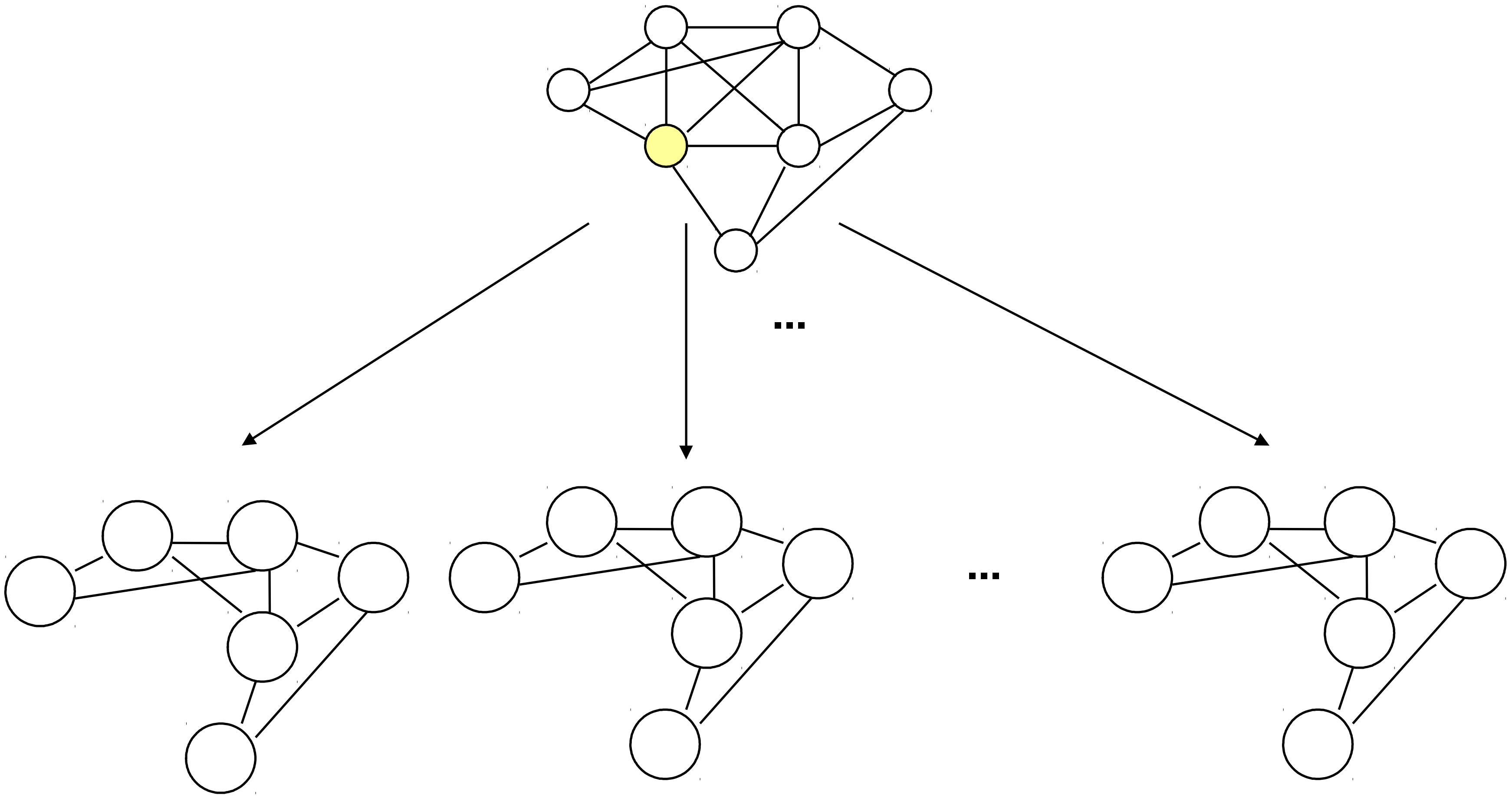
Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



Outline

- Introduction
- Inference
- Bounds and heuristics
- AND/OR search
- **Exploiting parallelism**
  - Distributed and parallel search
- Software

New Advances

- Parallel AOBB, first of its kind
  - Runs on computational grid
  - Extends parallel tree search paradigm
  - Two variants with different parallelization logic

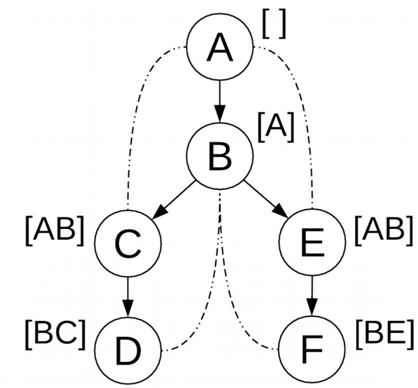
    [Otten and Dechter, 2012]
- Parallel shared-memory RBFAOO
  - Parallelization of the sequential RBFAOO

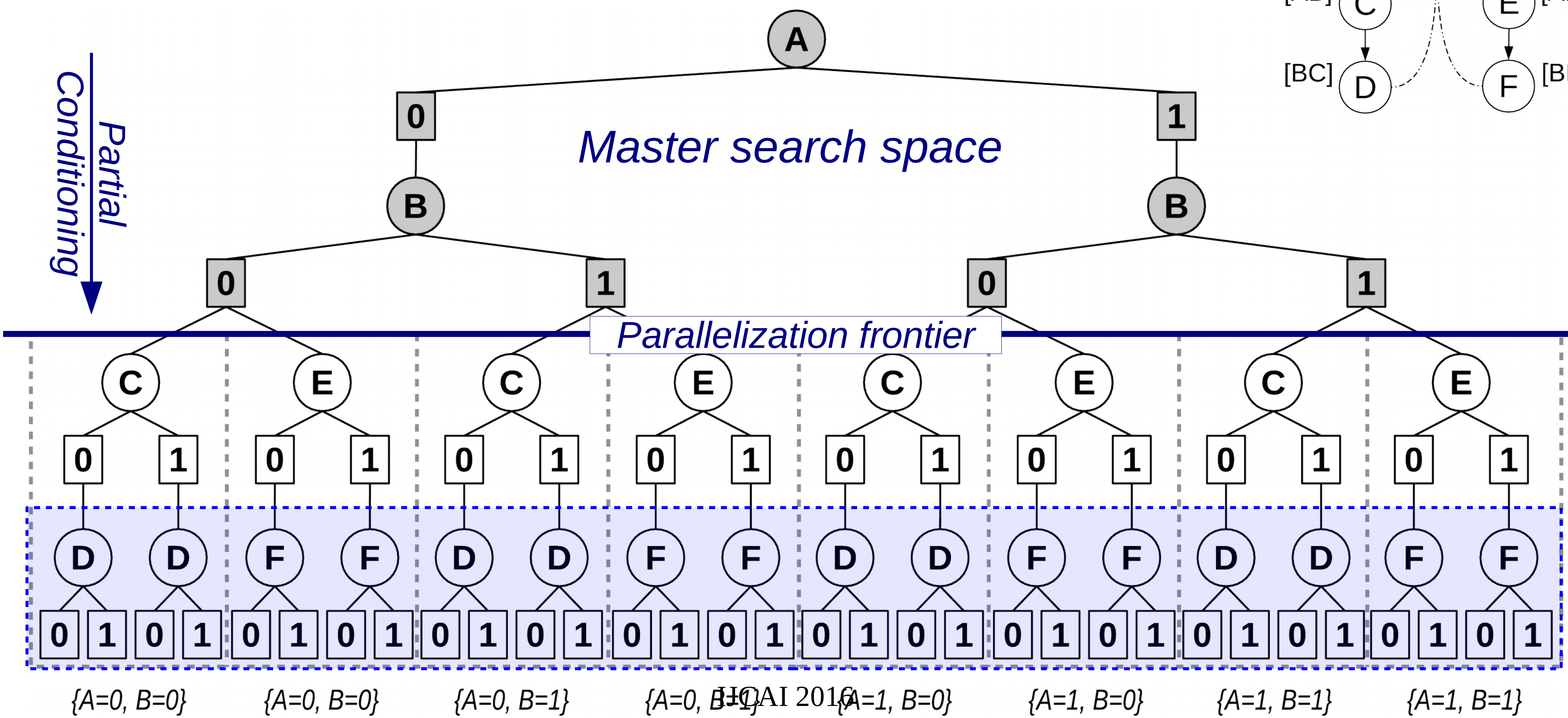
    [Kishimoto, Marinescu, Botea, 2015]
- Parallel dovetailing for AOBB, RBFAOO, SPRBFAOO
  - Towards large-scale MAP/MMAP inference

    [Kishimoto, Marinescu, Botea, 2016]

Parallel AOBB Illustrated

- *Master process* applies partial conditioning to obtain parallel subproblems.

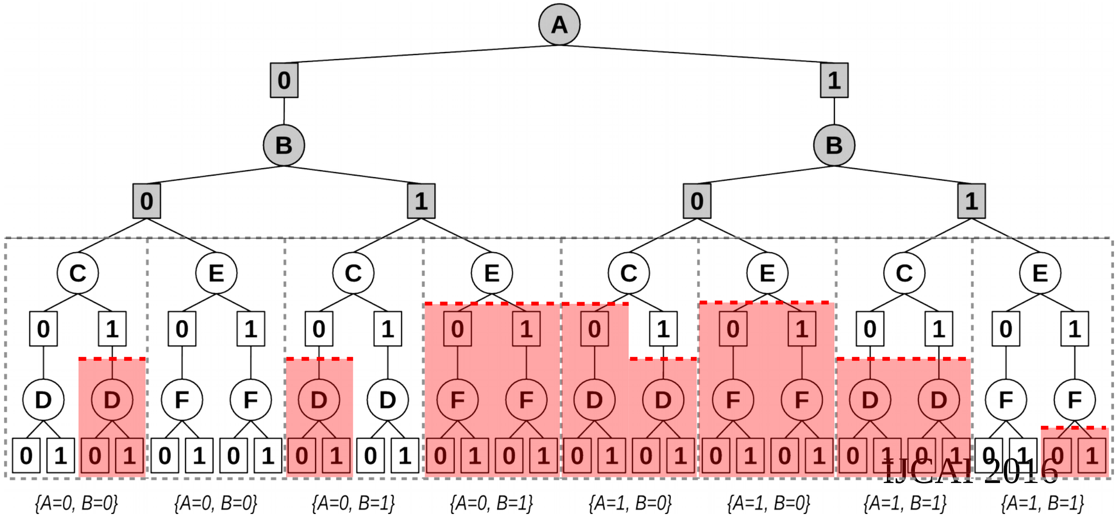


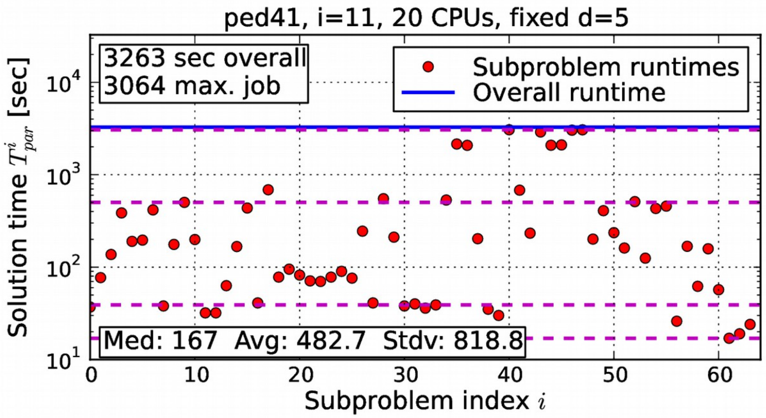


8 independent sub-problem search spaces

Fixed-depth Parallel AOBB

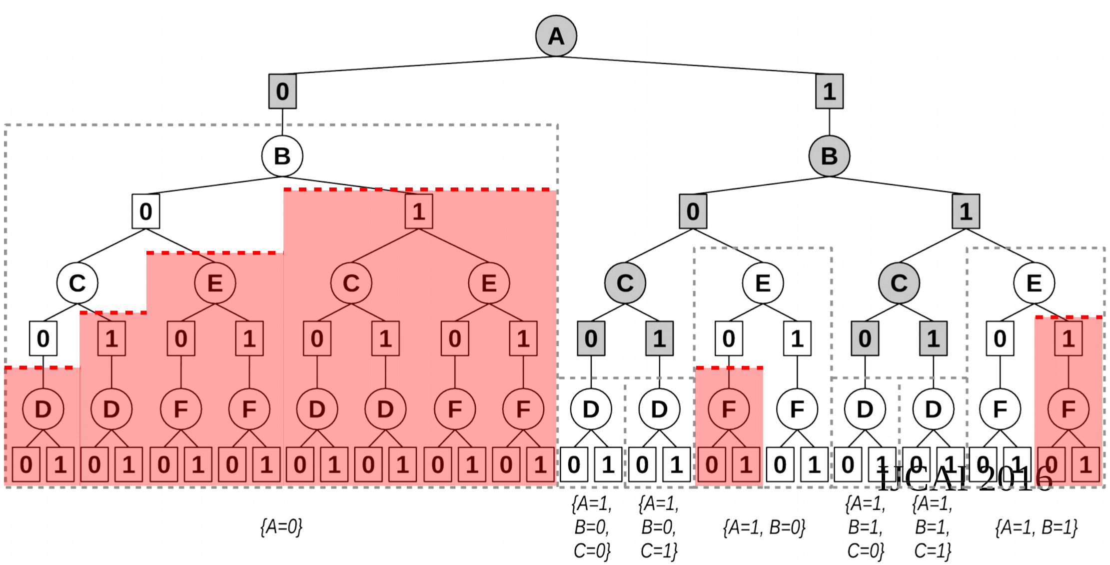
- Algorithm receives cutoff depth *d* as input:
  - Expand nodes centrally until depth *d*.
  - At depth *d*, submit to grid job queue.
- Explored sub-problem search spaces potentially very unbalanced.

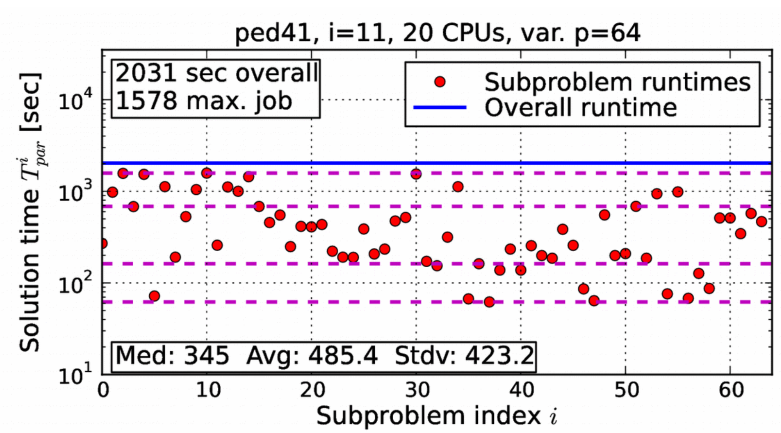




Variable-depth Parallel AOBB

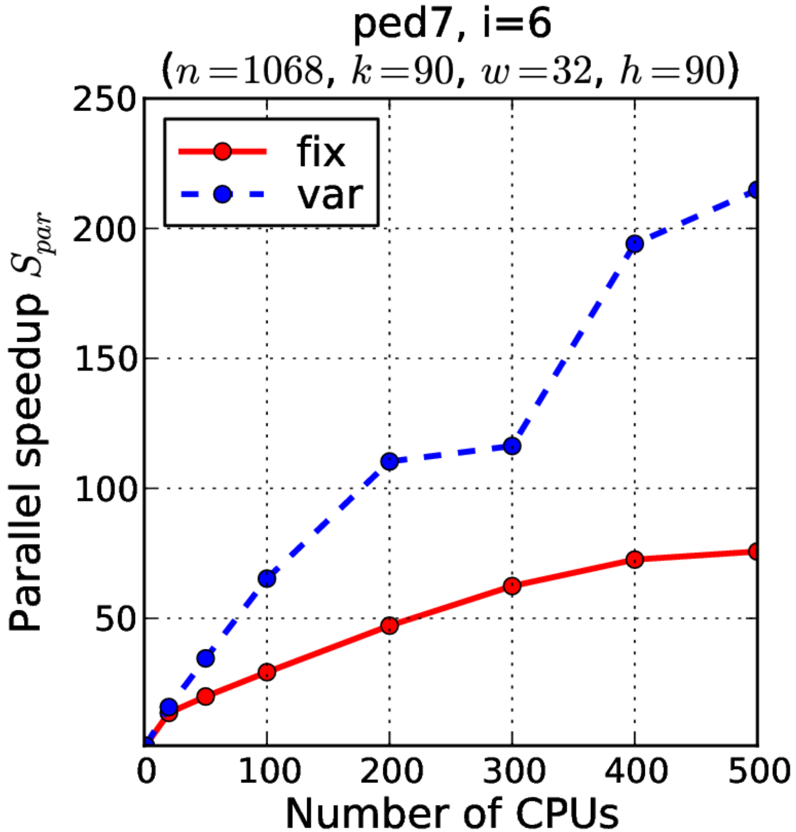
- Given sub-problem count *p* and estimator *N* :
  - Iteratively deepen frontier until size *p* reached:
    - Pick sub-problem *n* with largest estimate *N(n)* and split.
  - Submit sub-problems into job queue by descending complexity estimates.
- Hope to achieve better sub-problem balance.





Parallel Scaling Summary

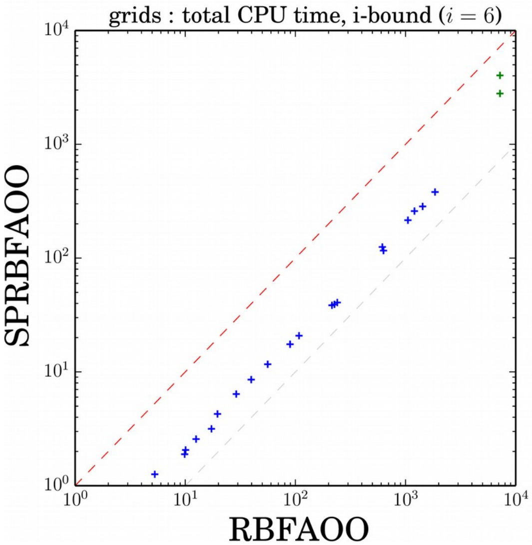
- Plot speedup against CPU count.
  - Trade off load balancing vs. overhead:
    - *#subproblems* ≈ 10 × *#CPUs*

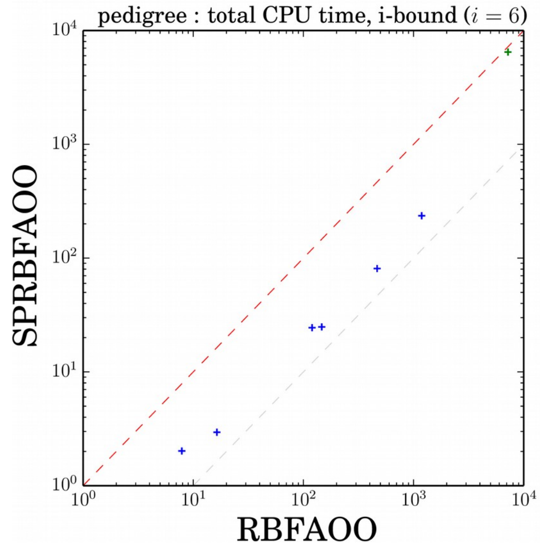


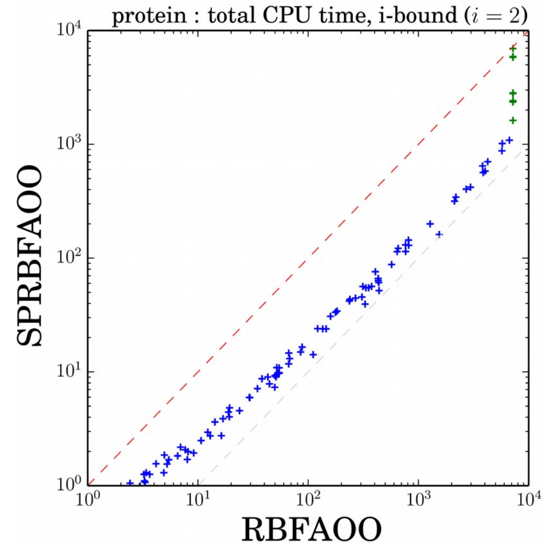
SPRBFAOO: Parallel Shared-Memory RBFAOO

- All threads start from the root with the same search strategy and with one shared cache table
- The *virtual q-value vq(n)* for node n is used to control parallel search
  - Initially, vq(n) is set to q(n)
  - When a thread examines n, vq(n) is incremented by a small value ζ
  - When all threads finish examining n, vq(n) is set to q(n)
- An effective load balancing is obtained without any sophisticated schemes, while promising portions of the search space are examined
- The algorithm guarantees solution optimality

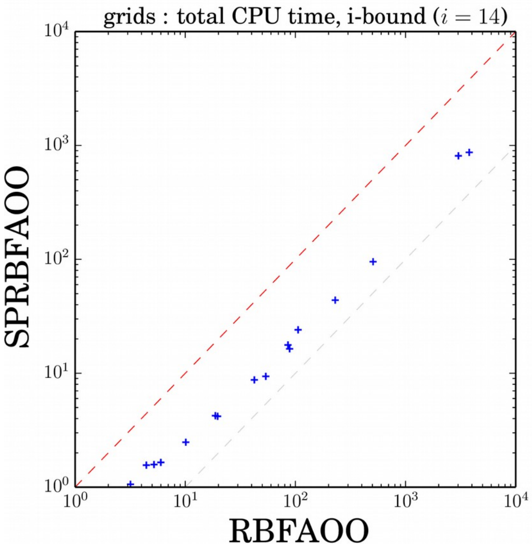
Empirical Evaluation

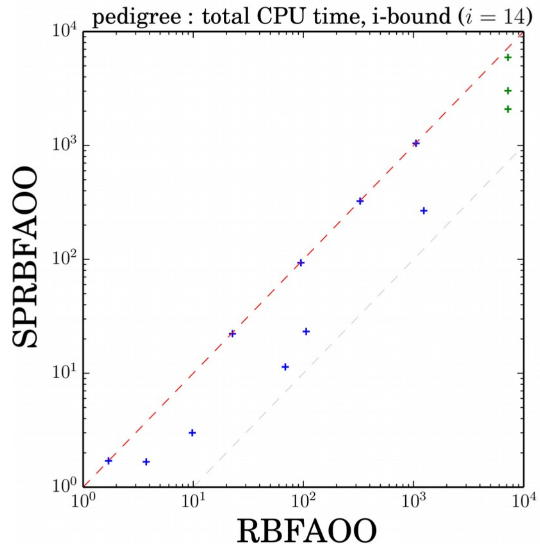


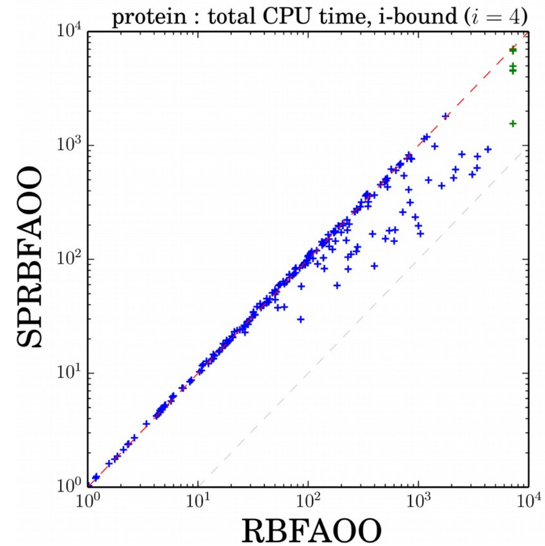




(up to 7-fold speedup with 12 threads)

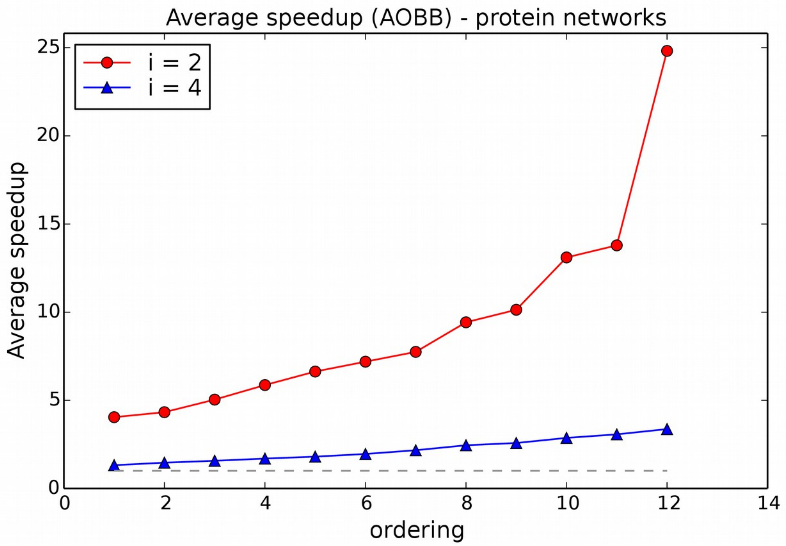




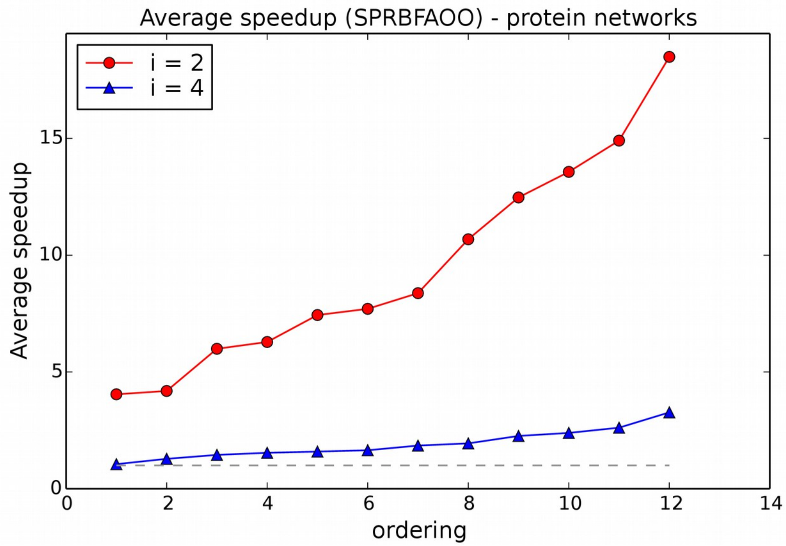


Parallel Dovetailing

- Simple distributed scheme
  - Launch in parallel m instances of the inference algorithm (AOBB, AOBF, RBFAOO, SPRBFAOO) each one solving the same problem instance but with a different input parameter configuration
    - e.g., parameter configuration = pseudo tree (ordering)



12 cores



12 nodes x 12 cores = 144 cores

Outline

- Introduction
- Inference
- Bounds and heuristics
- AND/OR search
- Exploiting parallelism
- **Software**
  - UAI probabilistic inference competitions

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

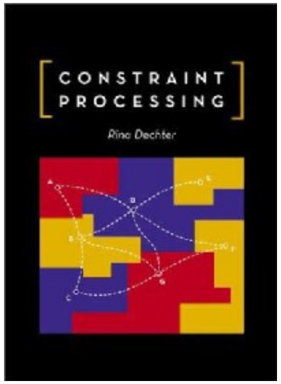
UAI Probabilistic Inference Competitions

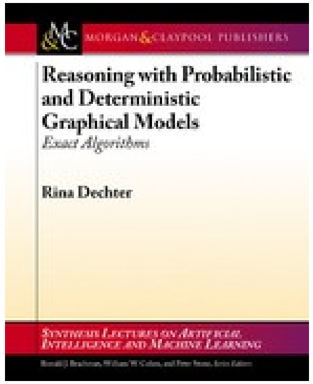
- **2006**  (aolib)
- **2008**  (aolib)
- **2011**  (daoopt)
- **2014**  (daoopt) |  (daoopt)  (merlin)

MPE/MAP | **MMAP**

Summary

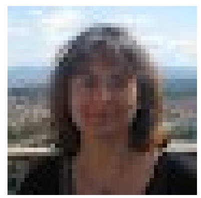
- Only a few principles
  - Inference and search should be combined
    - Time-space tradeoff
  - AND/OR search should be used
  - Caching in search should be used
  - Parallel search should be used if distributed and/or shared-memory environments are available





For publication see:
http://www.ics.uci.edu/~dechter/publications.html



Thank You

Kalev Kask
Irina Rish
Bozhena Bidyuk
Robert Mateescu
Radu Marinescu
Vibhav Gogate
Emma Rollon
Lars Otten
Natalia Flerova