Advances in Combinatorial Optimization for Graphical Models

Rina Dechter
University of California, Irvine

Radu Marinescu
IBM Research - Ireland

Alexander Ihler
University of California, Irvine

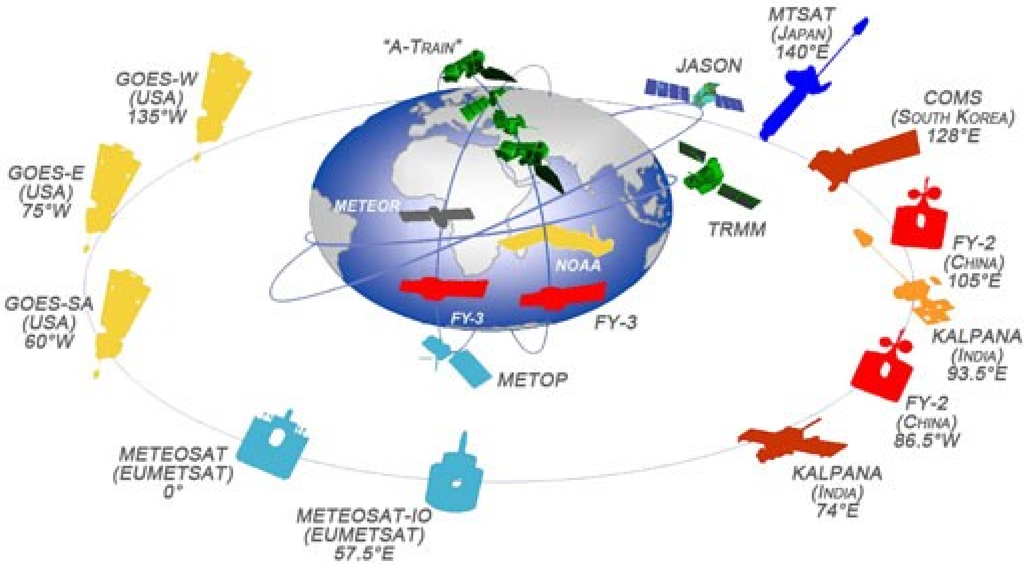
Lars Otten
University of California, Irvine
(now Google Inc.)

Outline

- **Introduction**
  - Graphical models
  - Optimization tasks for graphical models
- **Inference**
  - Variable Elimination, Bucket Elimination
- **Bounds and heuristics**
  - Basics of search
  - Bounded variable elimination and iterative cost shifting
- **AND/OR Search**
  - AND/OR search spaces
  - Depth-First Branch and Bound, Best-First search
- **Exploiting parallelism**
  - Distributed and parallel search
- **Software**

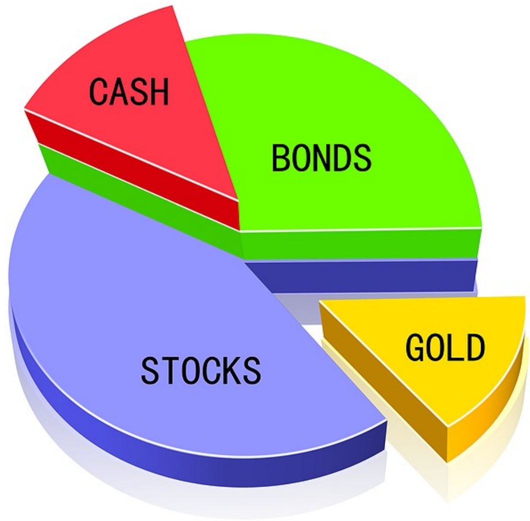
Combinatorial Optimization

Earth observing satellites



Find an optimal schedule for the satellite that maximizes the number of photographs taken, subject to on-board recording capacity

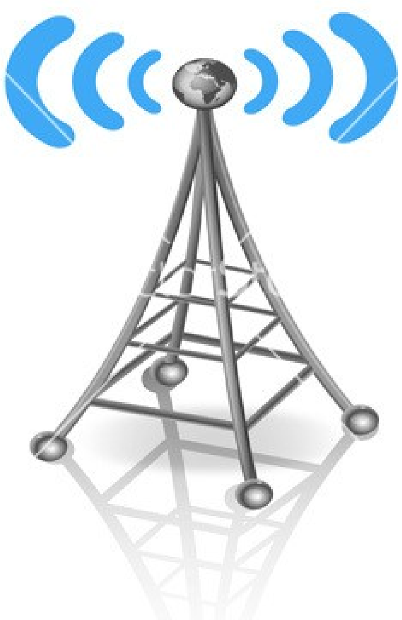
Investments



How much to invest in each asset to earn 8 cents per Invested dollar and the investment risk is minimized

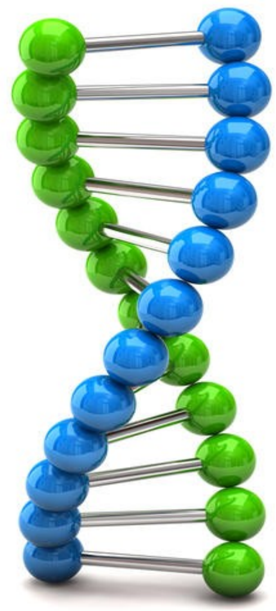
Combinatorial Optimization

Communications



Assign frequencies to a set of radio links such that interferencies are minimized

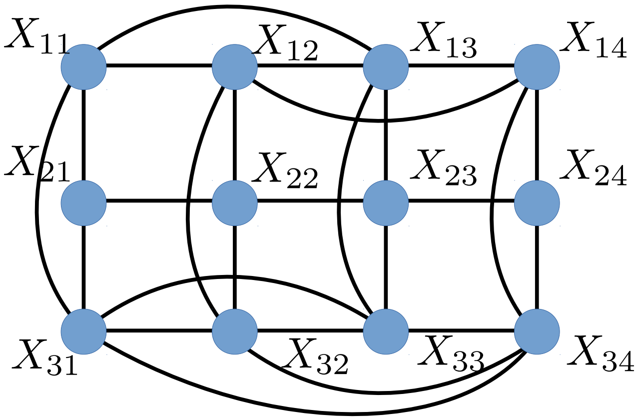
Bioinformatics



Find a joint haplotype configuration for all members of the pedigree which maximizes the probability of data

Constrained Optimization

Power plant scheduling



Unit #	Min Up Time	Min Down Time
1	3	2
2	2	1
3	4	1

Variables: $X_1, X_2, \ldots, X_n$ Domains: $\mathrm{ON},\ \mathrm{OFF}$
Constraints: $X_1 \vee X_2,\quad \neg X_3 \vee X_4$, min-uptime, min-downtime
Power demand: $\mathrm{Power}(X_i) \geq \mathrm{Demand}$

Objective: minimize $\mathrm{TotalFuelConsumption}(X_1, \ldots, X_n)$

Constraint Optimization Problems

A *finite COP* is a triple $R = \langle X, D, F \rangle$ where:

$X = \{x_1, \ldots, x_n\}$ -- variables

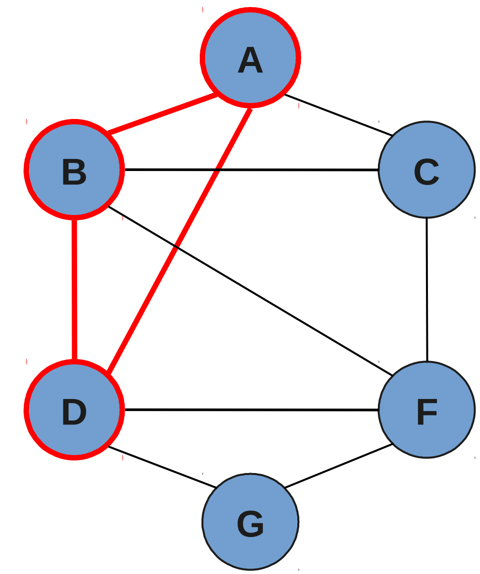
$D = \{D_1, \ldots, D_n\}$ -- domains

$F = \{f_{\alpha_1}, \ldots, f_{\alpha_m}\}$ -- cost functions

f(A,B,D) has scope {A,B,D}

A	B	D	Cost
1	2	3	3
1	3	2	2
2	1	3	∞
2	3	1	0
3	1	2	5
3	2	1	0

Primal graph:
- Variables - nodes
- Functions - arcs / cliques



$$F(a,b,c,d,f,g) = f_1(a,b,d) + f_2(d,f,g) + f_3(b,c,f) + f_4(a,c)$$

Global Cost Function

$$F(X) = \sum_{\alpha} f_\alpha(x_\alpha)$$

Constraint Networks

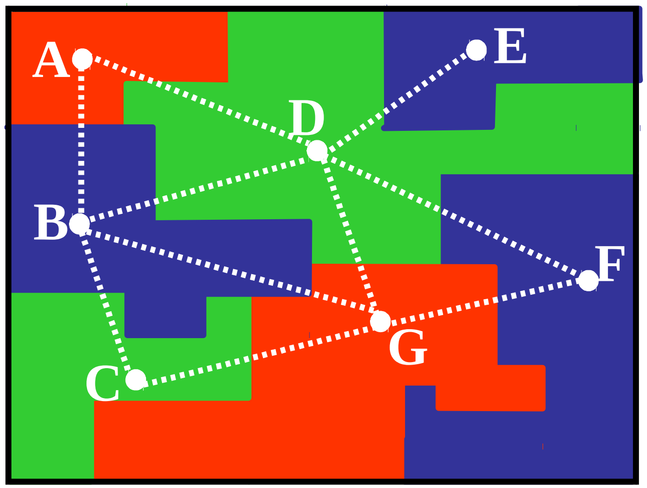
Map coloring

Variables: countries (A, B, C, etc.)

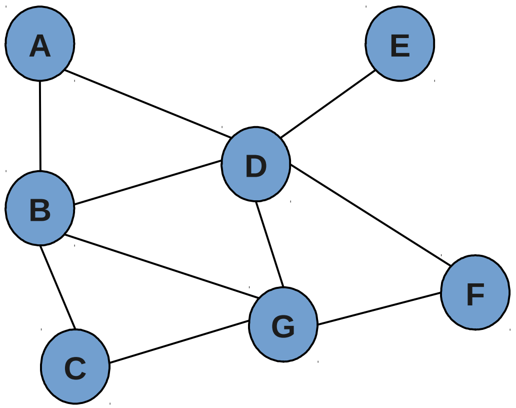
Values: colors (red, green, blue)

Constraints: A ≠ B, B ≠ D, A ≠ D, etc.

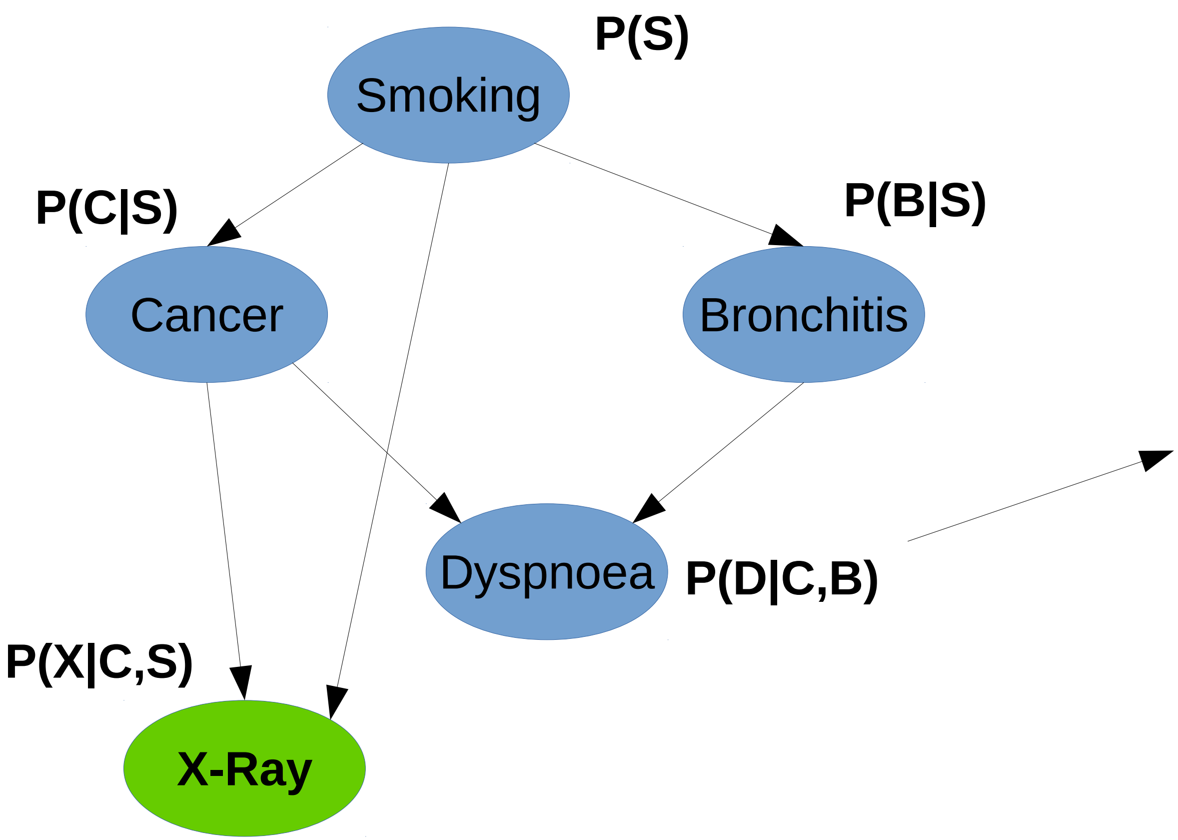
A	B
red	green
red	blue
green	red
green	blue
blue	red
blue	green



Constraint graph



Probabilistic Networks



P(D|C,B)

C	B	D=0	D=1
0	0	0.1	0.9
0	1	0.7	0.3
1	0	0.8	0.2
1	1	0.9	0.1

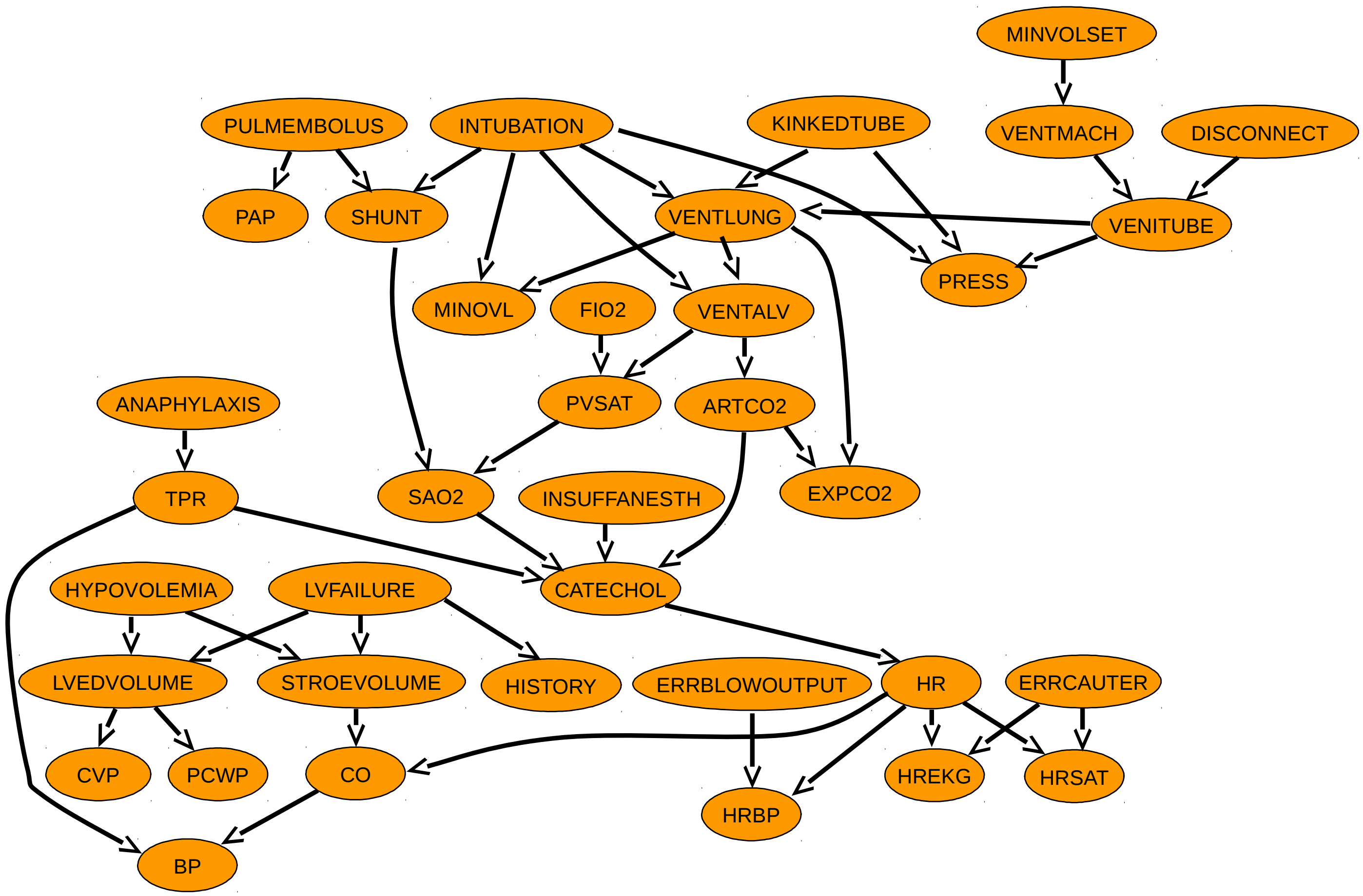
$$P(S,C,B,X,D) = P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C,S) \cdot P(D|C,B)$$

MPE: Find a maximum probability assignment, given evidence

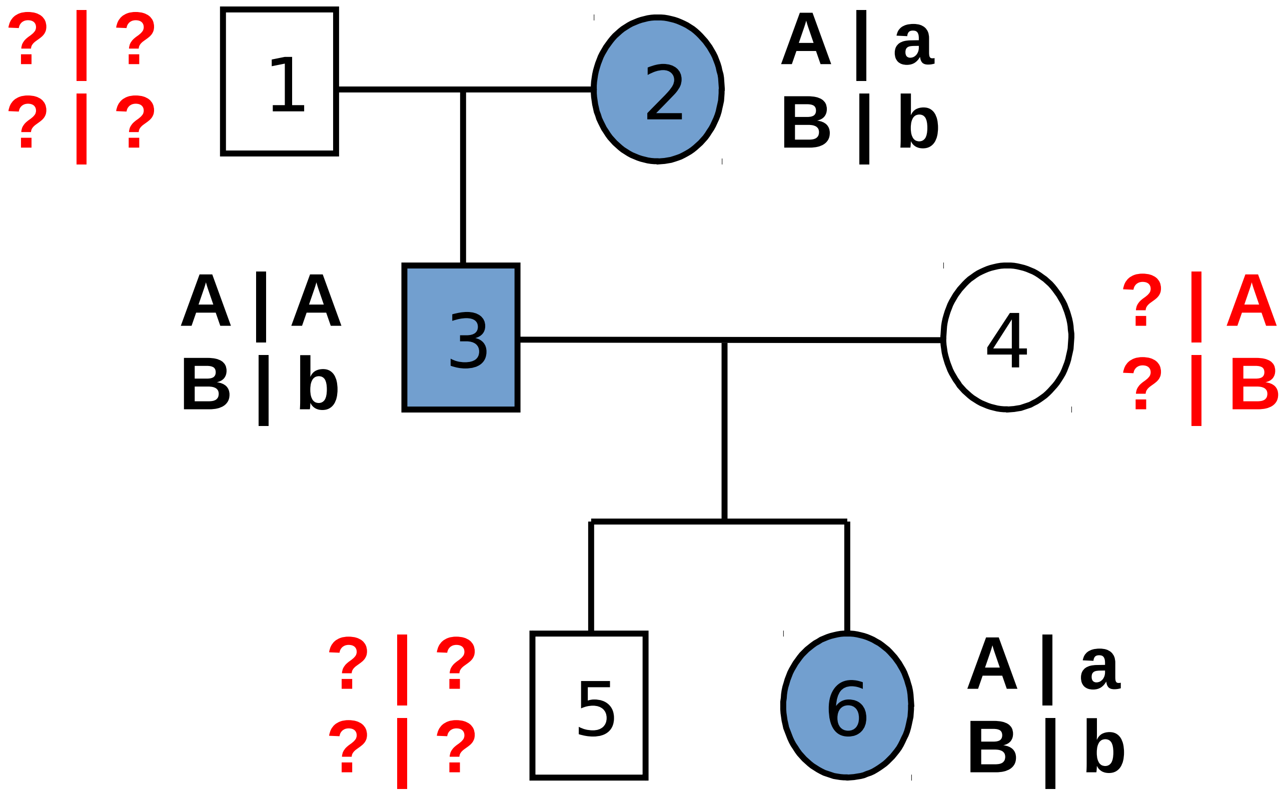
= Find $\arg\max P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C,S) \cdot P(D|C,B)$

Monitoring Intensive-Care Patients

The “alarm” network – 37 variables, 509 parameters (instead of 2^{37})

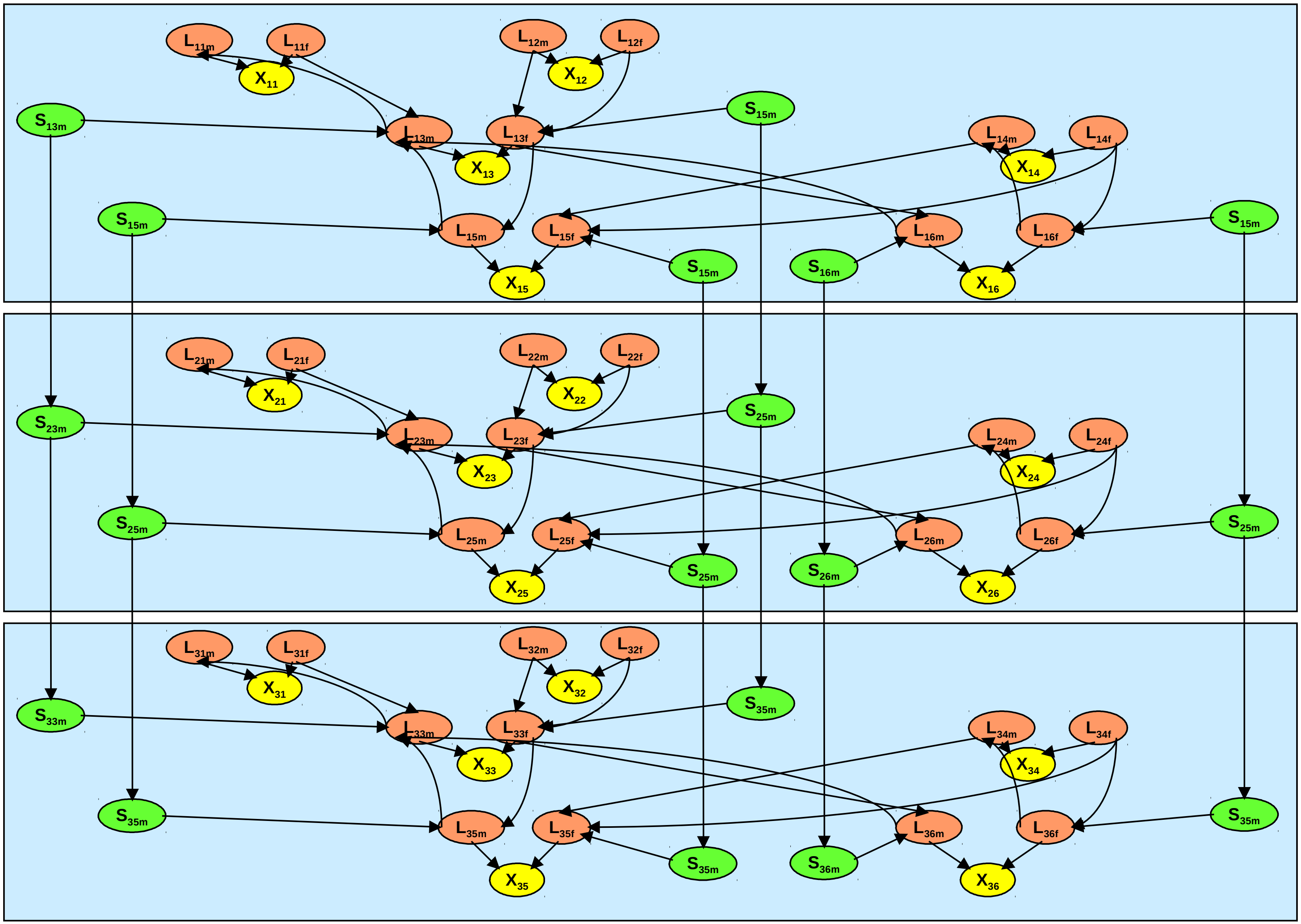


Genetic Linkage Analysis

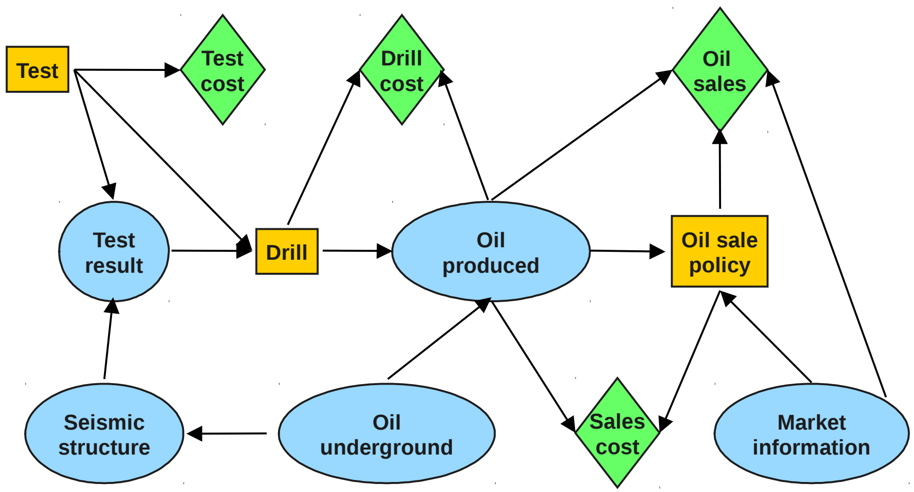


- 6 individuals
- Haplotype: {2, 3}
- Genotype: {6}
- Unknown

Pedigree: 6 people, 3 markers



Influence Diagrams



Task: find optimal policy

$$E = \max_{\Delta=(\delta_1,\ldots,\delta_m)} \sum_{x=(x_1,\ldots,x_n)} \prod_i P_i(x)\, u(x)$$

Chance variables: $X = x_1, \ldots, x_n$

Decision variables: $D = d_1, \ldots d_m$

CPDs for chance variables: $P_i = P(x_i | x_{\mathrm{pa}_i}),\ i = 1, \ldots, n$

Reward components: $r = \{r_1, \ldots, r_j\}$

Utility function: $u(X) = \sum_i r_i(X)$

Graphical Models

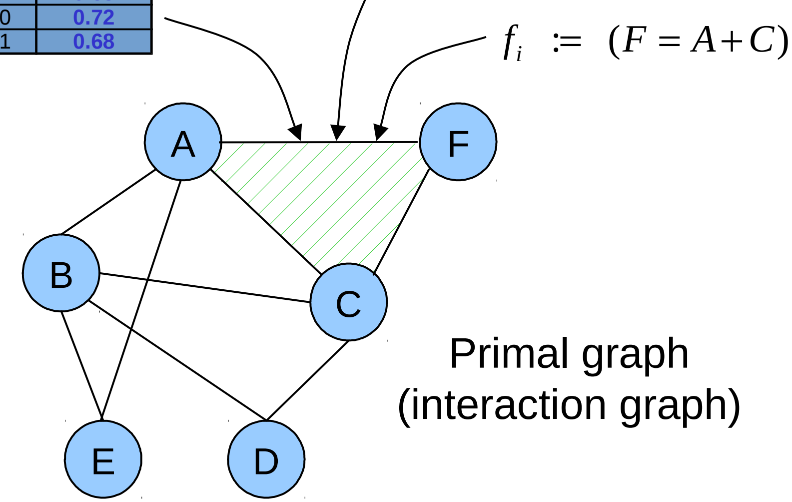
- A graphical model **(X, D, F)**:
  - $X = \{x_1, \ldots, x_n\}$ variables
  - $D = \{D_1, \ldots, D_n\}$ domains
  - $F = \{f_{\alpha_1}, \ldots, f_{\alpha_m}\}$ functions
    - (constraints, CPTs, CNFs, ...)
- Operators
  - Combination
  - Elimination (projection)
- Tasks
  - **Belief updating**: $\sum_{X \setminus Y} \prod_j P_j$
  - **MPE/MAP**: $\max_X \prod_j P_j$
  - **Marginal MAP**: $\max_Y \sum_{X \setminus Y} \prod_j P_j$
  - **CSP**: $\prod_j C_j(x)$
  - **WCSP**: $\min_X \sum_j f_j$
  - **MEU**: $\max_\Delta \sum_x P(x)u(x)$

CPT

A	C	F	P(F|A,C)
0	0	0	0.14
0	0	1	0.96
0	1	0	0.40
0	1	1	0.60
1	0	0	0.35
1	0	1	0.65
1	1	0	0.72
1	1	1	0.68

Relation

A	C	F
red	green	blue
blue	red	red
blue	blue	green
green	red	blue



Primal graph
(interaction graph)

- All these tasks are NP-hard
  - Exploit problem structure
  - Identify special cases
  - Approximate

Example Domains for Graphical Models

- Web Pages and Link Analysis
- Communication Networks (Cell phone fraud detection
- Natual Language Processing (e.g., information extraction and semantic parsing)
- Battlespace Awarness
- Epidemiological Studies
- Citation Networks
- Intelligence Analysis (terrorist networks)
- Financial Transactions (money laundering)
- Computational Biology
- Object Recognition and Scene Analysis
- ...

Combinatorial Optimization Tasks

- Most Probable Explanation (MPE),
  or Maximum A Posteriori (MAP)
- M Best MPE/MAP
- Marginal MAP (MMAP)
- Weighted CSPs (WCSP), Max-CSPs, Max-SAT
- Integer Linear Programs
- Maximum Expected Utility (MEU)

Outline

- **Introduction**
  - Graphical models
  - Optimization tasks for graphical models
  - Solving optimization problems by inference and search
- **Inference**
- **Bounds and heuristics**
- **AND/OR Search**
- **Exploiting parallelism**
- **Software**

Solution Techniques

AND/OR search

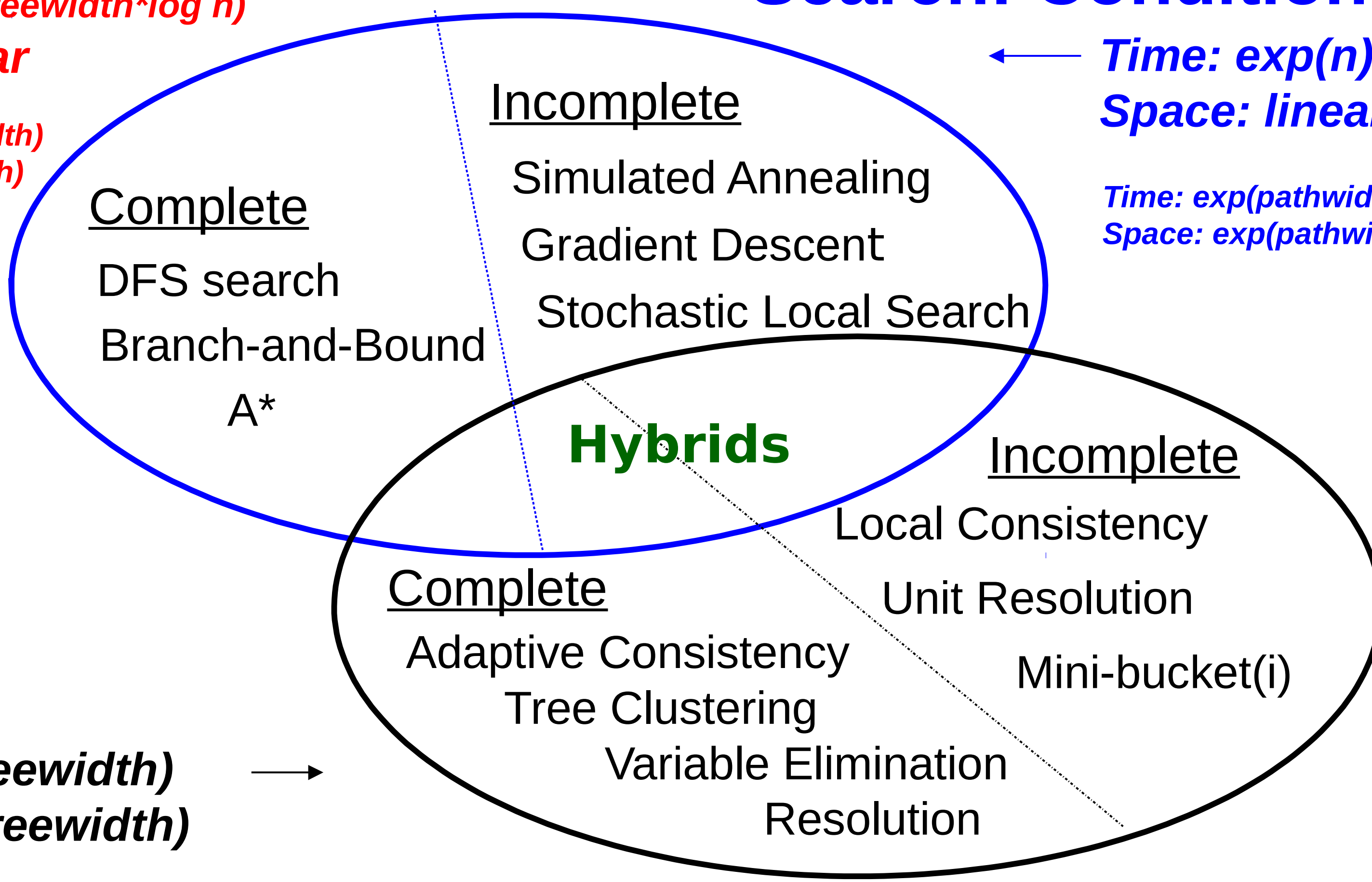
Time: exp(treewidth*log n)
Space: linear

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

Combination of Cost Functions

A	B	f(A,B)
b	b	6
b	g	0
g	b	0
g	g	6

+

B	C	f(B,C)
b	b	6
b	g	0
g	b	0
g	g	6

A	B	C	f(A,B,C)
b	b	b	12
b	b	g	6
b	g	b	0
b	g	g	6
g	b	b	6
g	b	g	0
g	g	b	6
g	g	g	12

= 0 + 6

Elimination in a Cost Function

A	**B**	**f(A,B)**
b	b	4
b	g	6
b	r	1
g	b	2
g	g	6
g	r	3
r	b	1
r	g	1
r	r	6

min

Elim(**f**,**B**)

A	**g(A)**
b	1
g	2
r	1

Elim(**g**,**A**)

$\mathbf{h}_{\varnothing}$
1

Conditioning in a Cost Function

A	B	f(A,B)
b	b	4
b	g	6
b	r	1
g	b	2
g	g	6
g	r	3
r	b	1
r	g	1
r	r	6

Assign **A=b**

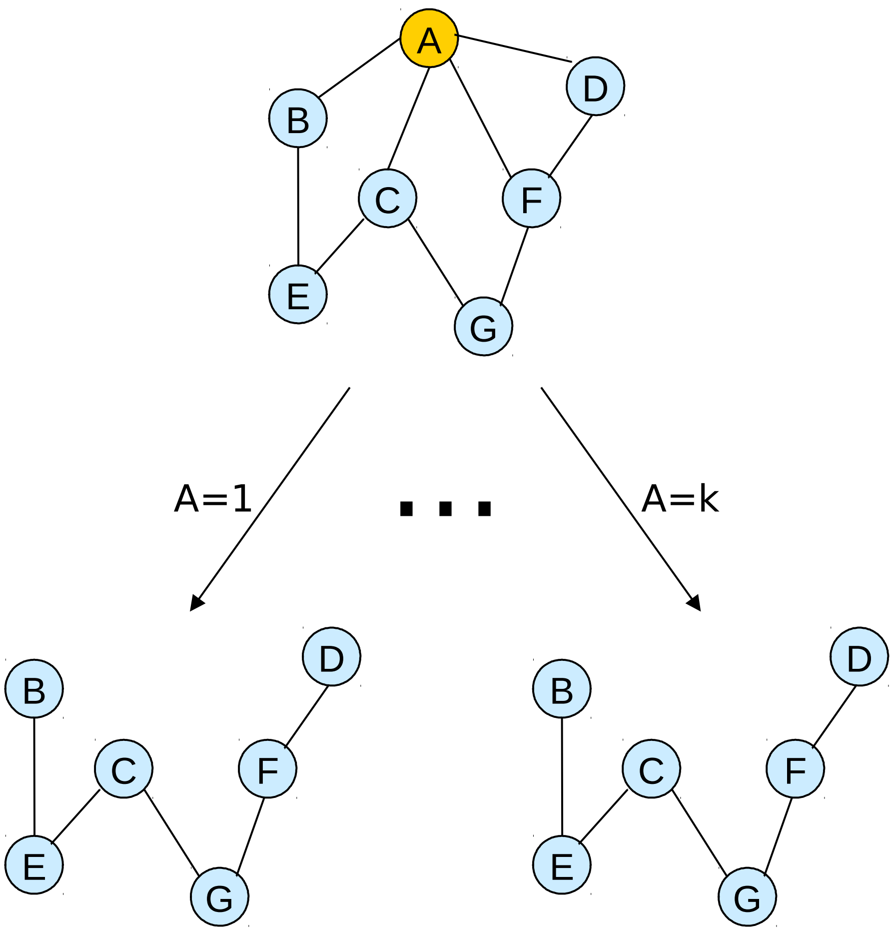
B	g(B)
b	4
g	6
r	1

Assign **B=r**

$h_{\varnothing}$
4

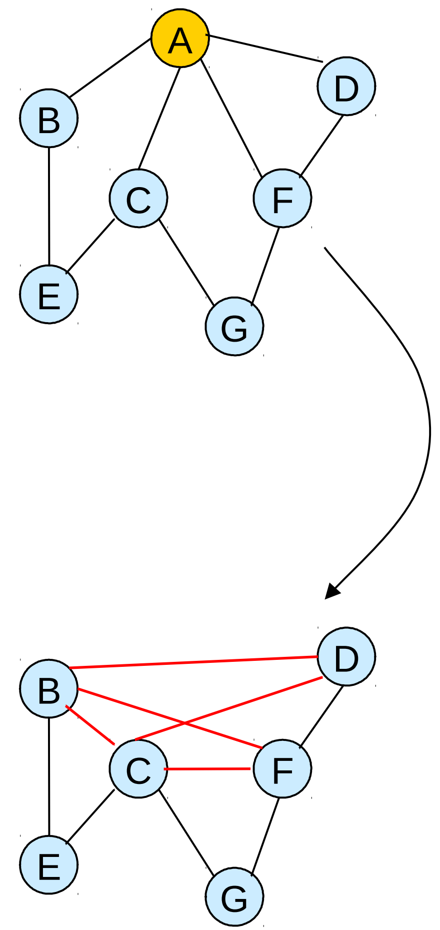
Conditioning vs. Elimination

Conditioning (search)



k “sparser” problems

Elimination (inference)

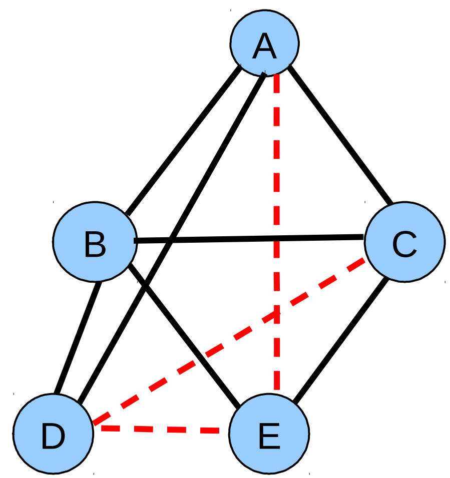


1 “denser” problem

Outline

- Introduction
- **Inference**
  - Variable Elimination, Bucket Elimination
- Bounds and heuristics
- AND/OR Search
- Exploiting parallelism
- Software

Computing the Optimal Cost Solution



Constraint graph

$$\textbf{OPT} = \min_{a,e,d,c,b} f(a) + \underbrace{f(a,b)} + f(a,c) + f(a,d) + \underbrace{f(b,c) + f(b,d) + f(b,e)} + f(c,e)$$

Combination

$$\min_{a} f(a) \min_{e,d} f(a,d) + \min_{c} f(a,c) + f(c,e) + \min_{b} \underbrace{f(a,b) + f(b,c) + f(b,d) + f(b,e)}_{\lambda_B(a,d,c,e)}$$

Variable Elimination

Bucket Elimination

Algorithm **elim-opt** [Dechter, 1996]
Non-serial Dynamic Programming [Bertele & Briochi, 1973]

$$\text{OPT} = \min_{a,e,d,c,b} f(a) + f(a,b) + f(a,c) + f(a,d) + f(b,c) + f(b,d) + f(b,e) + f(c,e)$$

$\min_b \sum$ ← Elimination/Combination operators

bucket B: $\underbrace{f(a,b) \;\; f(b,c) \;\; f(b,d) \;\; f(b,e)}$

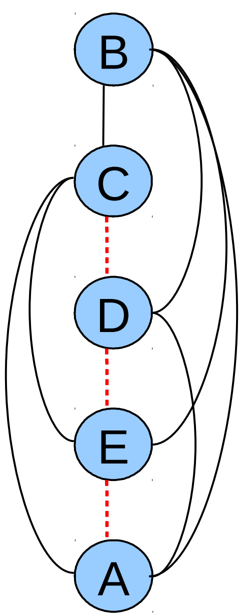
bucket C: $f(c,a) \;\; f(c,e) \;\; \lambda_{B \to C}(a,d,c,e)$

bucket D: $f(a,d) \;\; \lambda_{C \to D}(a,d,e)$

bucket E: $\lambda_{D \to E}(a,e)$

bucket A: $f(a) \;\; \lambda_{E \to A}(a)$

OPT



Generating the Optimal Assignment

$$\mathbf{b}^* = \arg\min_{\mathbf{b}} f(a^*, b) + f(b, c^*) + f(b, d^*) + f(b, e^*)$$

B: $f(a, b)\ f(b, c)\ f(b, d)\ f(b, e)$

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} f(c, a^*) + f(c, e^*) + \lambda_{B\to C}(a^*, d^*, c, e^*)$$

C: $f(c, a)\ f(c, e)$ $\lambda_{B\to C}(a, d, c, e)$

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} f(a^*, d) + \lambda_{C\to D}(a^*, d, e^*)$$

D: $f(a, d)$ $\lambda_{C\to D}(a, d, e)$

$$\mathbf{e}^* = \arg\min_{\mathbf{e}} \lambda_{D\to E}(a^*, e)$$

E: $\lambda_{D\to E}(a, e)$

$$\mathbf{a}^* = \arg\min_{\mathbf{a}} f(a) + \lambda_{E\to A}(a)$$

A: $f(a)$ $\lambda_{E\to A}(a)$

Return: $(\ \mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^*, \mathbf{d}^*, \mathbf{e}^*\ )$

Complexity of Bucket Elimination

Algorithm **elim-opt** [Dechter, 1996]
Non-serial Dynamic Programming [Bertele & Briochi, 1973]

OPT = $\min_{a,e,d,c,b} f(a) + f(a,b) + f(a,c) + f(a,d) + f(b,c) + f(b,d) + f(b,e) + f(c,e)$

$$\underbrace{\min_b \sum}$$ ⟵ Elimination / Combination operators

bucket B: $f(a,b)$ $f(b,c)$ $f(b,d)$ $f(b,e)$

bucket C: $f(c,a)$ $f(c,e)$ $\lambda_{B\to C}(a,d,c,e)$

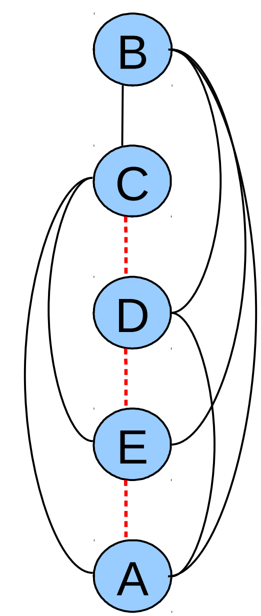
bucket D: $f(a,d)$ $\lambda_{C\to D}(a,d,e)$

bucket E: $\lambda_{D\to E}(a,e)$

bucket A: $f(a)$ $\lambda_{E\to A}(a)$

OPT

exp(w*=4)
“induced width”
(max clique size)



Complexity of Bucket Elimination

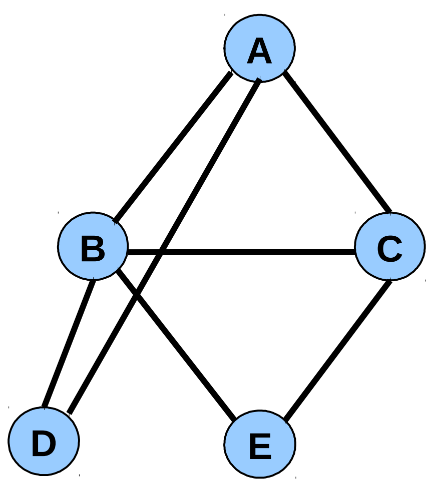
Bucket Elimination is **time** and **space**

$$O\big(r\exp(\,w^*(d)\,)\big)$$

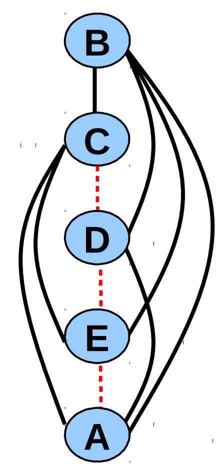
$w^*(d)$: the induced width of the primal graph along ordering *d*

r = number of functions

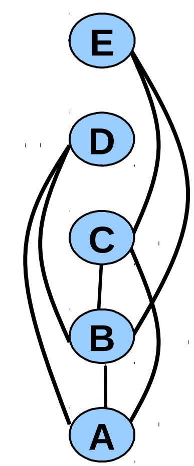
The effect of the ordering:



constraint graph



$w^*(d_1) = 4$



$w^*(d_2) = 2$

Finding the smallest induced width is hard!

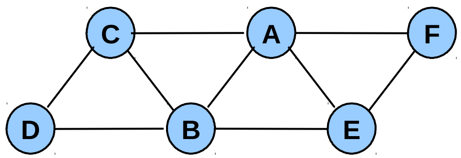
Outline

- Introduction
- Inference
- **Bounds and heuristics**
  - Basics of search: DFS versus BFS
  - Mini-Bucket Elimination
  - Weighted Mini-Buckets and Iterative Cost-Shifting
  - Generating Heuristics using Mini-Bucket Elimination
- AND/OR Search
- Exploiting parallelism
- Software

Outline

- Introduction
- Inference
- **Bounds and heuristics**
  - Basics of search: DFS versus BFS
  - Mini-Bucket Elimination
  - Weighted Mini-Buckets and Iterative Cost-Shifting
  - Generating Heuristics using Mini-Bucket Elimination
- AND/OR Search
- Exploiting parallelism
- Software

OR Search Spaces



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**

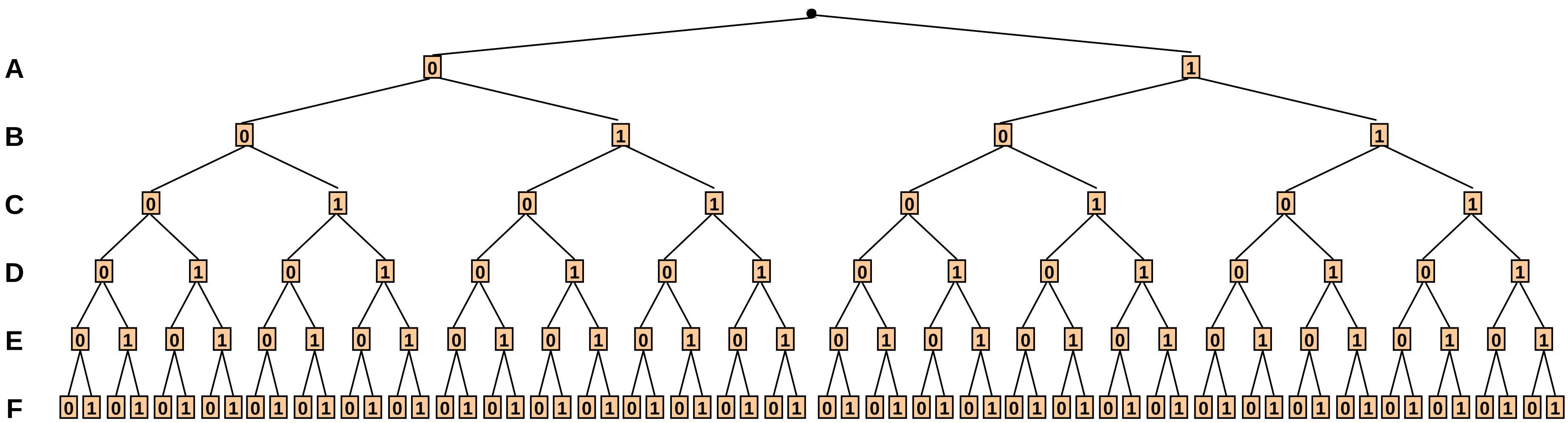
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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

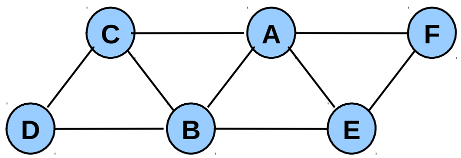
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

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

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



OR Search Spaces



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**

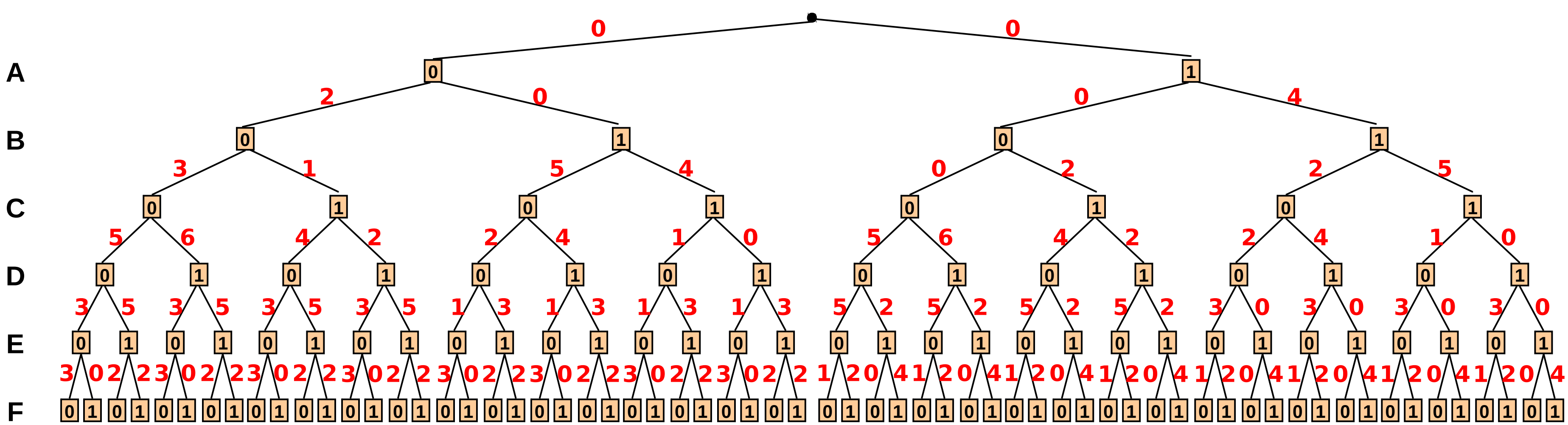
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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**

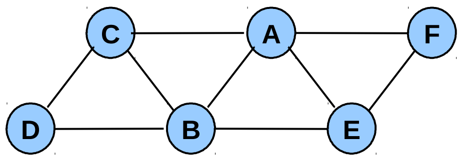
E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

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



Arc-cost is calculated based on cost functions with empty scope (conditioning)

The Value Function



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**

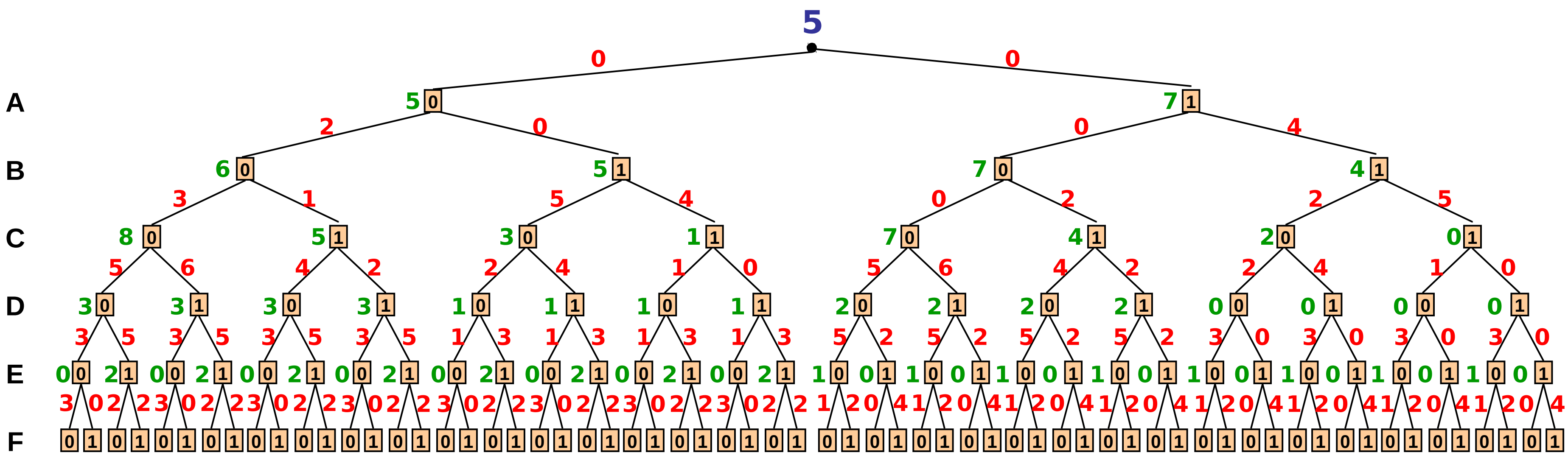
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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**

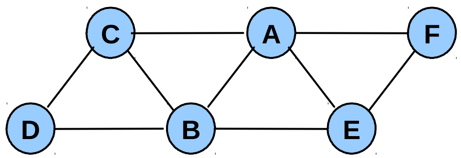
E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

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



Value of node = minimal cost solution below it

The Optimal Solution



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**

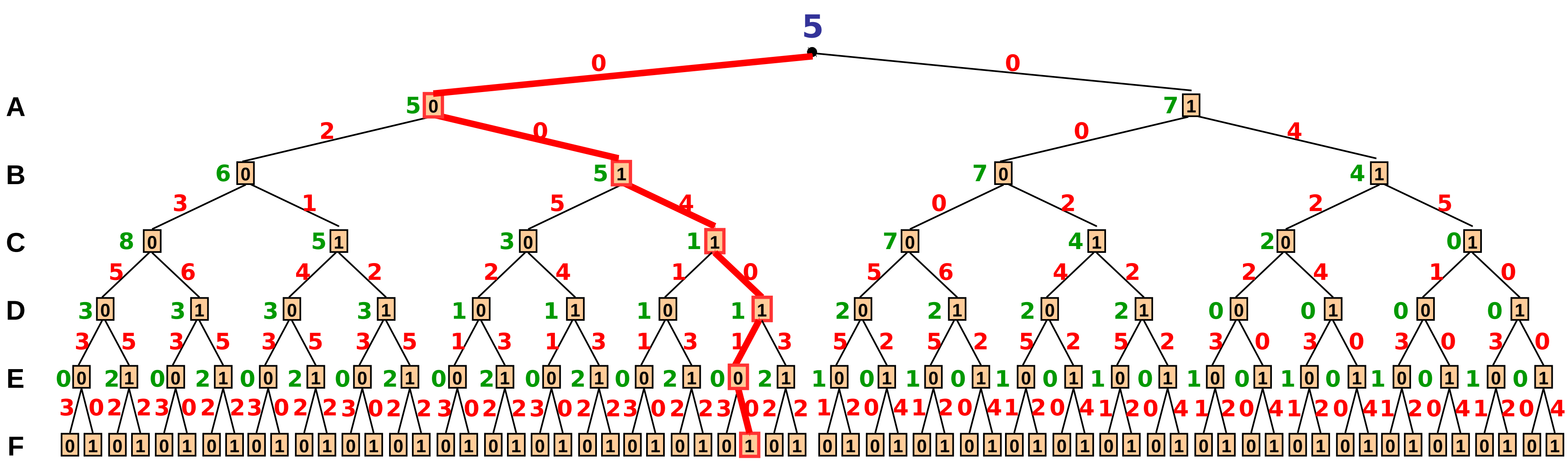
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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



Value of node = minimal cost solution below it

Basic Heuristic Search Schemes

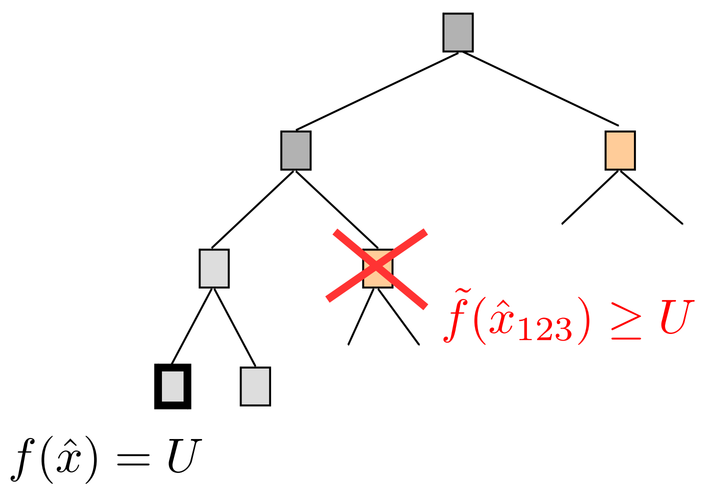
Heuristic function $\tilde{f}(\hat{x}_p)$ computes a lower bound on the best extension of partial configuration $\hat{x}_p$ and can be used to guide heuristic search.

We focus on:

1. Branch-and-Bound

Use heuristic function $\tilde{f}(\hat{x}_p)$ to prune the depth-first search tree

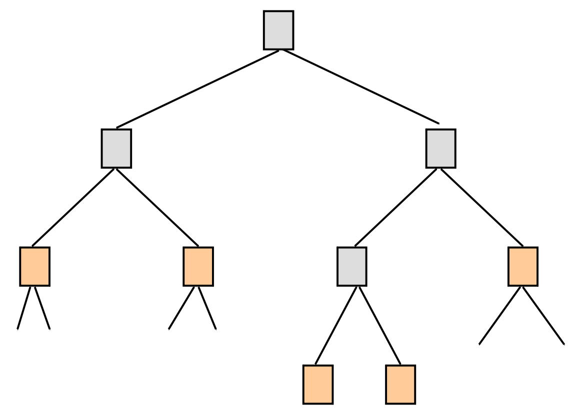
Linear space



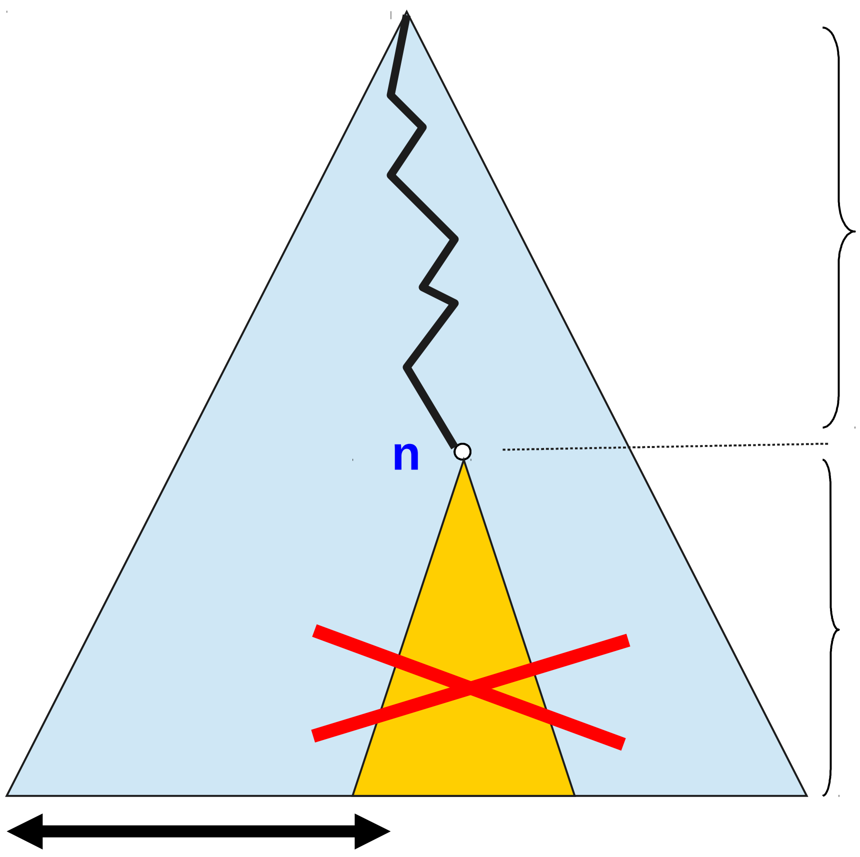
2. Best-First Search

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

Needs lots of memory



Classic Depth-First Branch and Bound



(UB) Upper Bound = best solution so far

Each node is a COP subproblem
(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

Best-First vs. Depth-First Branch and Bound

- Best-First (A*):
  - Expands least number of nodes given h
  - Requires storing full search tree in memory
- Depth-First BnB:
  - Can use linear space
  - If finds an optimal solution early, will expand the same search space as Best-First (if search space is a tree)
  - BnB can improve the heuristic function dynamically

How to Generate Heuristics

- The principle of relaxed models
  - Mini-Bucket Elimination
  - Bounded directional consistency ideas
  - Linear relaxations for integer programs

Outline

- Introduction
- Inference
- **Bounds and heuristics**
  - Basics of search: DFS versus BFS
  - Mini-Bucket Elimination
  - Weighted Mini-Buckets and Iterative Cost-Shifting
  - Generating Heuristics using Mini-Bucket Elimination
- AND/OR Search
- Exploiting parallelism
- Software

Mini-Bucket Approximation

Split a bucket into mini-buckets => bound complexity

bucket (X) =

$$\underbrace{\{\, f_1, \ldots, f_r, f_{r+1}, \ldots, f_n \,\}}$$

$$\lambda_X(\cdot) = \min_x \sum_{i=1}^{n} f_i(x, \ldots)$$

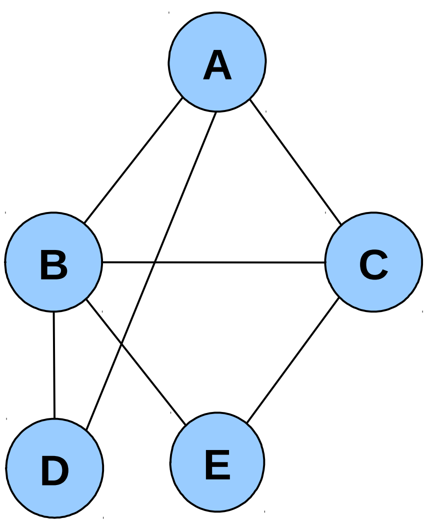
$$\{\, f_1, \ldots, f_r \,\} \qquad \{\, f_{r+1}, \ldots, f_n \,\}$$

$$\lambda'_X(\cdot) = \left(\min_x \sum_{i=1}^{r} f_i(\cdot)\right) + \left(\min_x \sum_{i=r+1}^{n} f_i(\cdot)\right)$$

$$\lambda'_X(\cdot) \;\leq\; \lambda_X(\cdot)$$

Exponential complexity decrease: $O(e^n) \rightarrow O(e^r) + O(e^{n-r})$

Mini-Bucket Elimination



mini-buckets

$\min_{B'} \sum f(\cdot)$ $\min_{B} \sum f(\cdot)$

bucket B: $\underbrace{f(a,b) \quad f(b,c)}$ $\underbrace{f(b,d) \quad f(b,e)}$

bucket C: $\underbrace{\lambda_{B\to C}(a,c) \quad f(a,c) \quad f(c,e)}$

bucket D: $\underbrace{f(a,d) \quad \lambda_{B\to D}(d,e)}$

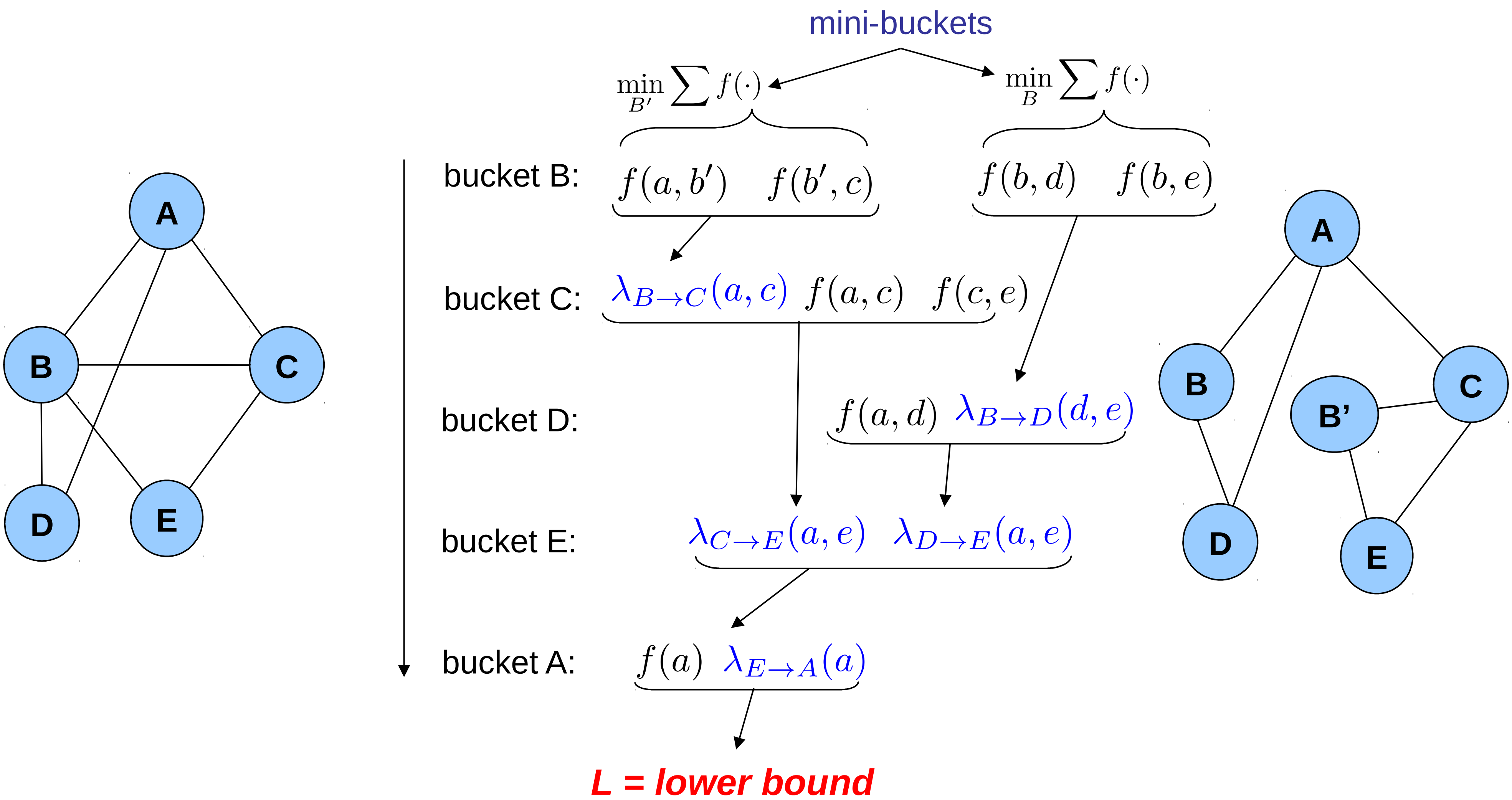
bucket E: $\underbrace{\lambda_{C\to E}(a,e) \quad \lambda_{D\to E}(a,e)}$

bucket A: $\underbrace{f(a) \quad \lambda_{E\to A}(a)}$

L = lower bound

[Dechter and Rish, 2003]

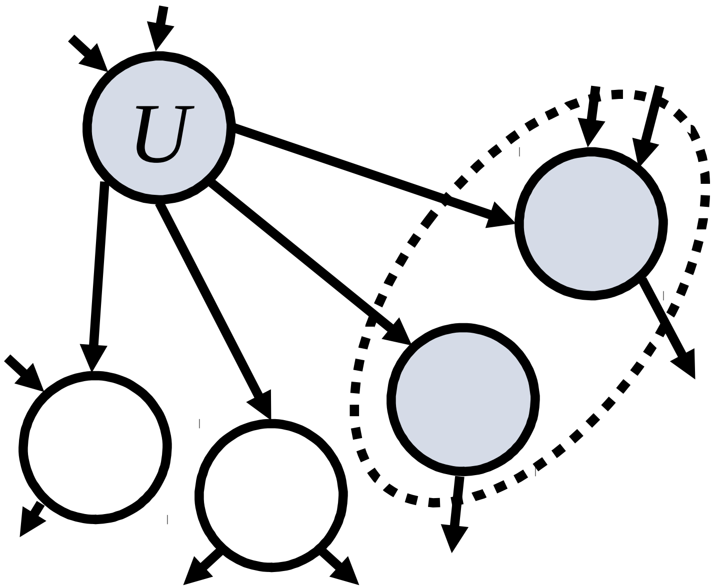
Mini-Bucket Elimination Semantics



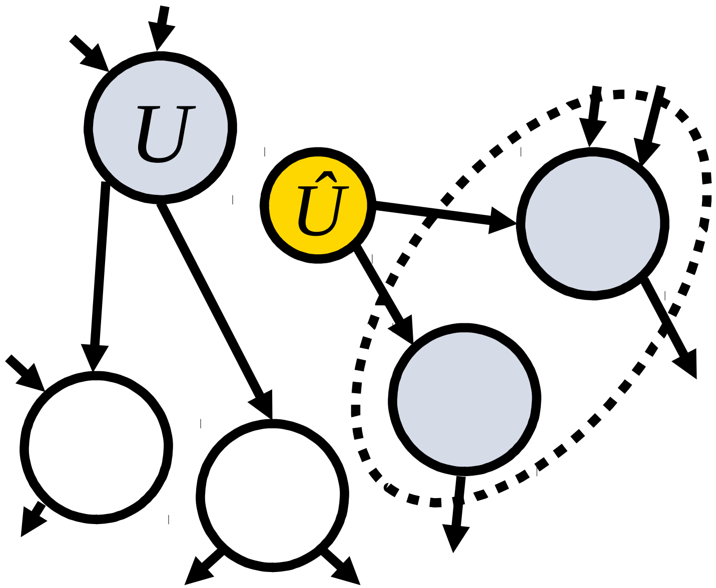
Semantics of Mini-Buckets: Splitting a Node

Variables in different buckets are renamed and duplicated
[Kask *et al.*, 2001], [Geffner *et al.*, 2007], [Choi *et al.*, 2007], [Johnson et al. 2007]

Before Splitting:
Network *N*



After Splitting:
Network *N'*



MBE-MPE(i): Algorithm Approx-MPE

- **Input**: i – max number of variables allowed in a mini-bucket
- **Output**: [lower bound (P of a suboptimal solution), upper bound]

Example: **approx-mpe(3)** versus **elim-mpe**

approx-mpe(3)

		Max variables in a mini-bucket
B:	$\underbrace{f(a,b)\quad f(b,c)}$ $\quad$ $\underbrace{f(b,d)\quad f(b,e)}$	3
C:	$\underbrace{\lambda_{B\to C}(a,c)\ f(a,c)\quad f(c,e)}$	3
D:	$\underbrace{f(a,d)\ \lambda_{B\to D}(d,e)}$	3
E:	$\underbrace{\lambda_{C\to E}(a,e)\quad \lambda_{D\to E}(a,e)}$	2
A:	$\underbrace{f(a)\,\lambda_{E\to A}(a)}$	1

L = lower bound

$$w^* = 2$$

elim-mpe

B:	$\underbrace{f(a,b)\quad f(b,c)\quad f(b,d)\quad f(b,e)}$
C:	$\underbrace{\lambda_{B\to C}(a,c,d,e)\ f(a,c)\quad f(c,e)}$
D:	$\underbrace{f(a,d)\ \lambda_{C\to D}(a,d,e)}$
E:	$\underbrace{\lambda_{D\to E}(a,e)}$
A:	$\underbrace{f(a)\ \lambda_{E\to A}(a)}$

OPT

$$w^* = 4$$

[Dechter and Rish, 1997]

Mini-Bucket Decoding

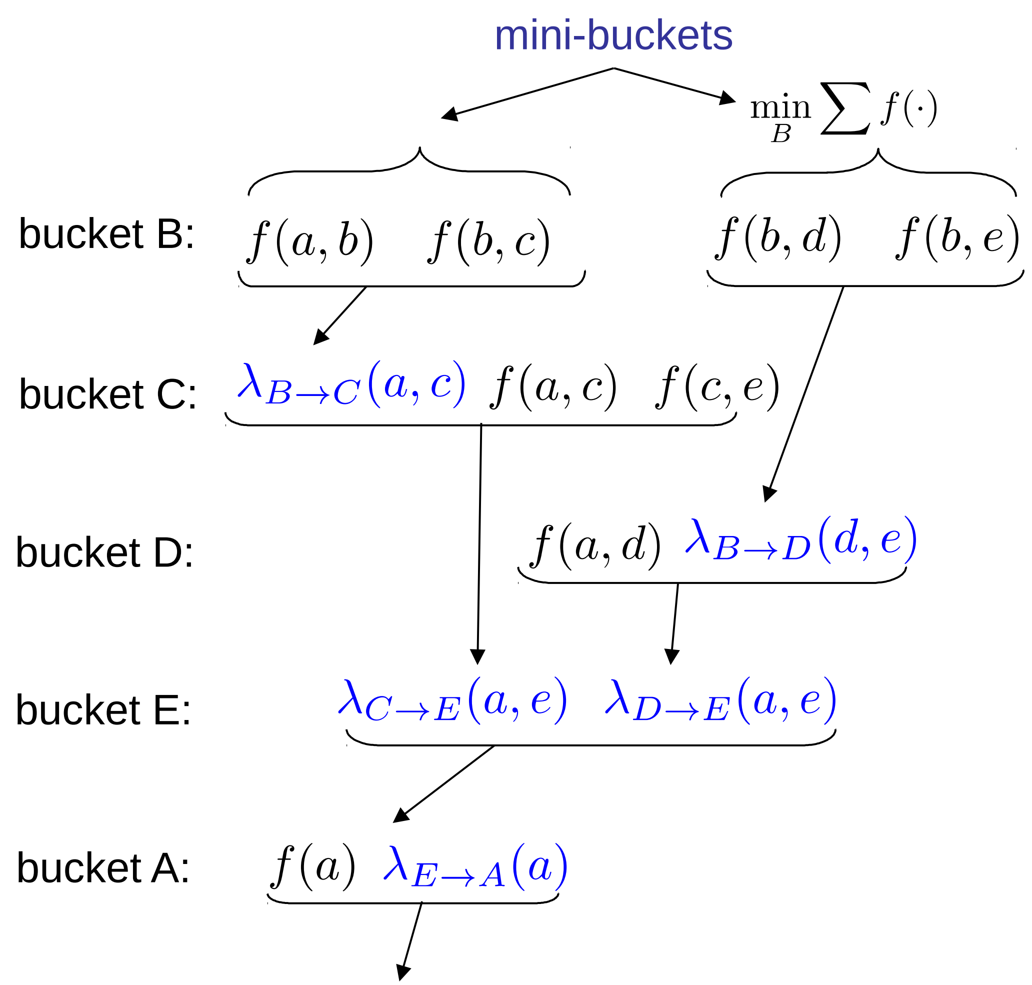
$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} f(\hat{a}, b) + f(b, \hat{c}) + f(b, \hat{d}) + f(b, \hat{e})$$

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}} \lambda_{B\to C}(\hat{a}, c) + f(c, \hat{a}) + f(c, \hat{e})$$

$$\hat{\mathbf{d}} = \arg\min_{\mathbf{d}} f(\hat{a}, d) + \lambda_{B\to D}(d, \hat{e})$$

$$\hat{\mathbf{e}} = \arg\min_{\mathbf{e}} \lambda_{C\to E}(\hat{a}, e) + \lambda_{D\to E}(\hat{a}, e)$$

$$\hat{\mathbf{a}} = \arg\min_{\mathbf{a}} f(a) + \lambda_{E\to A}(a)$$



Greedy configuration = upper bound

L = lower bound

[Dechter and Rish, 2003]

Properties of MBE(i)

- **Complexity**: *O(r exp(i))* time and *O(exp(i))* space
- Yields a lower-bound and an upper-bound
- **Accuracy**: determined by upper/lower (U/L) bound
- Possible use of mini-bucket approximations:
  - As anytime algorithms
  - As heuristics in search
- Other tasks (similar mini-bucket approximations):
  - Belief updating, Marginal MAP, MEU, WCSP, MaxCSP
    [Dechter and Rish, 1997], [Liu and Ihler, 2011], [Liu and Ihler, 2013]

Outline

- Introduction
- Inference
- **Bounds and heuristics**
  - Basics of search: DFS versus BFS
  - Mini-Bucket Elimination
  - Weighted Mini-Buckets and Iterative Cost-Shifting
  - Generating Heuristics using Mini-Bucket Elimination
- AND/OR Search
- Exploiting parallelism
- Software

Cost-Shifting

(Reparameterization)

$+\lambda(\mathbf{B})$

A	B	f(A,B)
b	b	6 + 3
b	g	0 - 1
g	b	0 + 3
g	g	6 - 1

+

$-\lambda(\mathbf{B})$

B	C	f(B,C)
b	b	6 - 3
b	g	0 - 3
g	b	0 + 1
g	g	6 + 1

A	B	C	f(A,B,C)
b	b	b	12
b	b	g	6
b	g	b	0
b	g	g	6
g	b	b	6
g	b	g	0
g	g	b	6
g	g	g	12

= 0 + 6

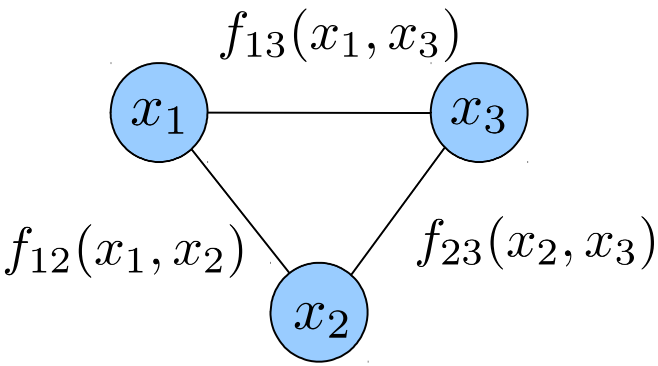
B	$\lambda(\mathbf{B})$
b	3
g	-1

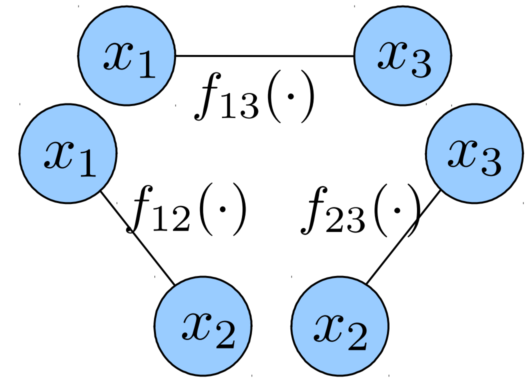
Modify the individual functions

- but -

keep the sum of functions unchanged

Dual Decomposition

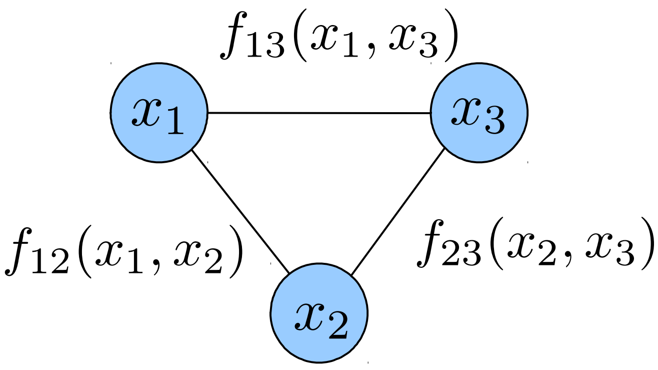


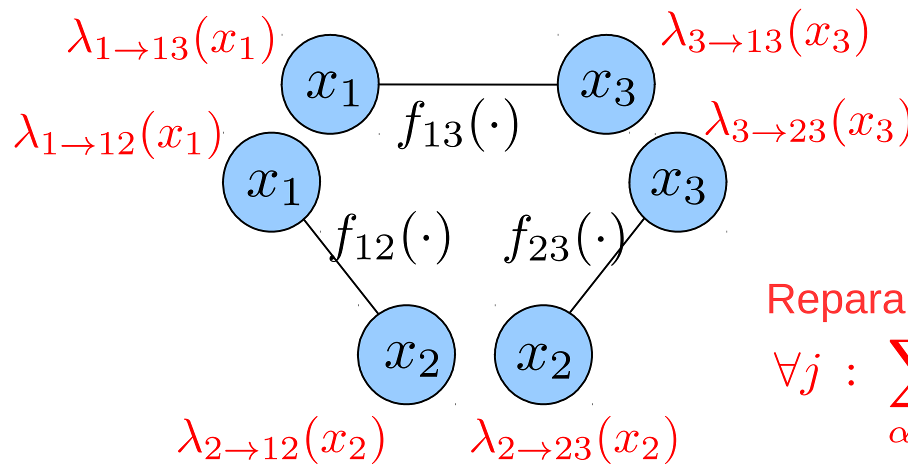


$$F^* = \min_x \sum_\alpha f_\alpha(x) \qquad \geq \qquad \sum_\alpha \min_x \; f_\alpha(x)$$

- Bound solution using decomposed optimization
- Solve independently: optimistic bound

Dual Decomposition





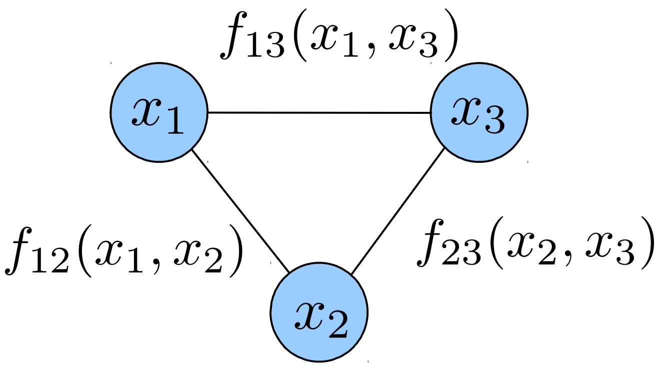
Reparameterization:

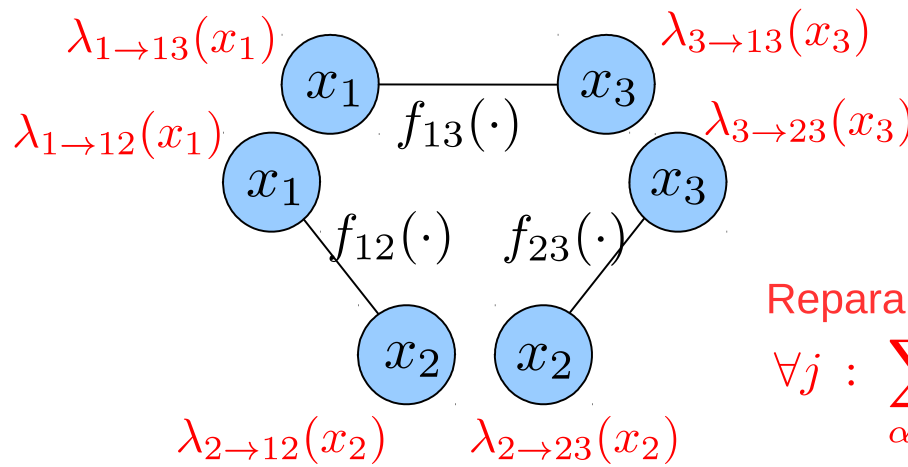
$$\forall j \; : \; \sum_{\alpha \ni j} \lambda_{j\to\alpha}(x_j) = 0$$

$$F^* = \min_x \sum_\alpha f_\alpha(x) \qquad \geq \max_{\lambda_{i\to\alpha}} \sum_\alpha \min_x \left[f_\alpha(x) + \sum_{i\in\alpha} \lambda_{i\to\alpha}(x_i) \right]$$

- Bound solution using decomposed optimization
- Solve independently: optimistic bound
- Tighten the bound by reparameterization
  - Enforce lost equality constraints via Lagrange multipliers

Dual Decomposition





Reparameterization:

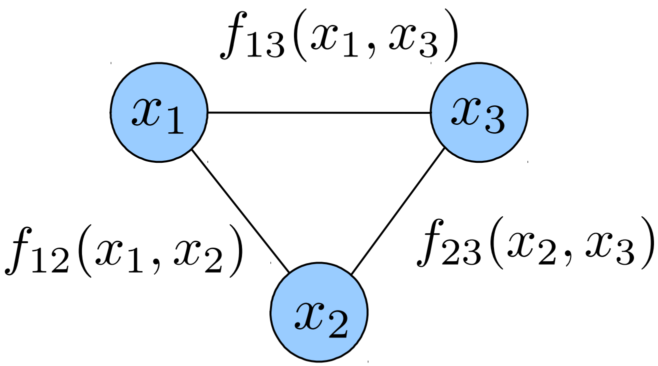
$$\forall j \; : \; \sum_{\alpha \ni j} \lambda_{j\to\alpha}(x_j) = 0$$

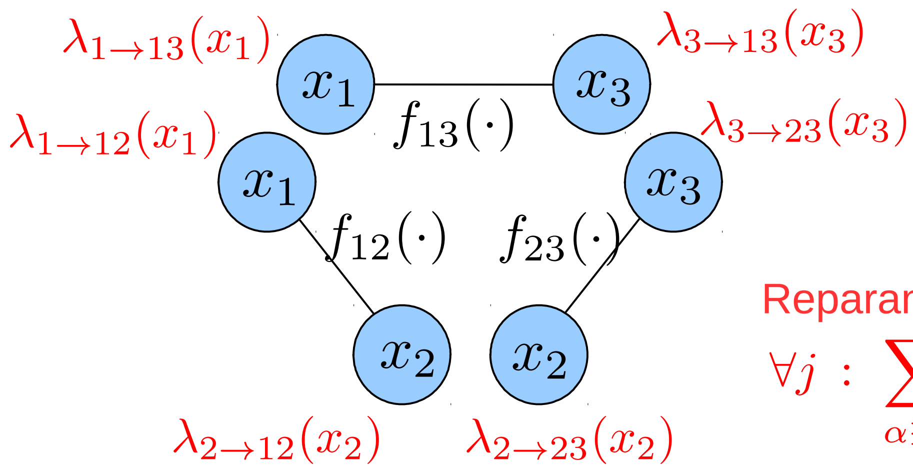
$$F^* = \min_x \sum_\alpha f_\alpha(x) \qquad \geq \max_{\lambda_{i\to\alpha}} \sum_\alpha \min_x \Big[\, f_\alpha(x) + \sum_{i\in\alpha} \lambda_{i\to\alpha}(x_i) \Big]$$

Many names for the same class of bounds:

- Dual decomposition [Komodakis et al. 2007]
- TRW, MPLP [Wainwright et al. 2005, Globerson & Jaakkola 2007]
- Soft arc consistency [Cooper & Schiex 2004]
- Max-sum diffusion [Warner 2007]

Dual Decomposition





Reparameterization:

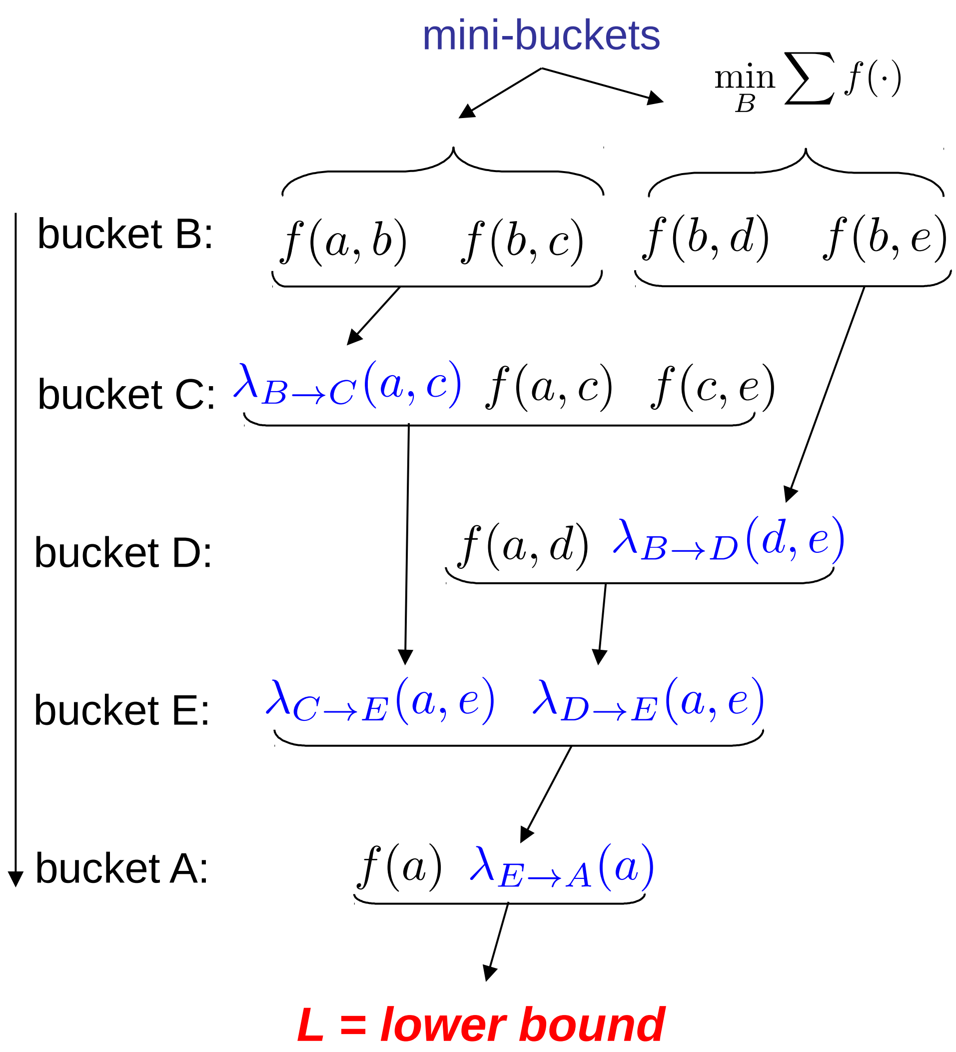
$$\forall j \; : \; \sum_{\alpha \ni j} \lambda_{j\to\alpha}(x_j) = 0$$

$$F^* = \min_x \sum_\alpha f_\alpha(x) \qquad \geq \max_{\lambda_{i\to\alpha}} \sum_\alpha \min_x \left[f_\alpha(x) + \sum_{i\in\alpha} \lambda_{i\to\alpha}(x_i) \right]$$

Many ways to optimize the bound:

- Sub-gradient descent [Komodakis et al. 2007; Jojic et al. 2010]
- Coordinate descent [Warner 2007; Globerson & Jaakkola 2007; Sontag et al. 2009; Ihler et al. 2012]
- Proximal optimization [Ravikumar et al. 2010]
- ADMM [Meshi & Globerson 2011; Martins et al. 2011; Forouzan & Ihler 2013]

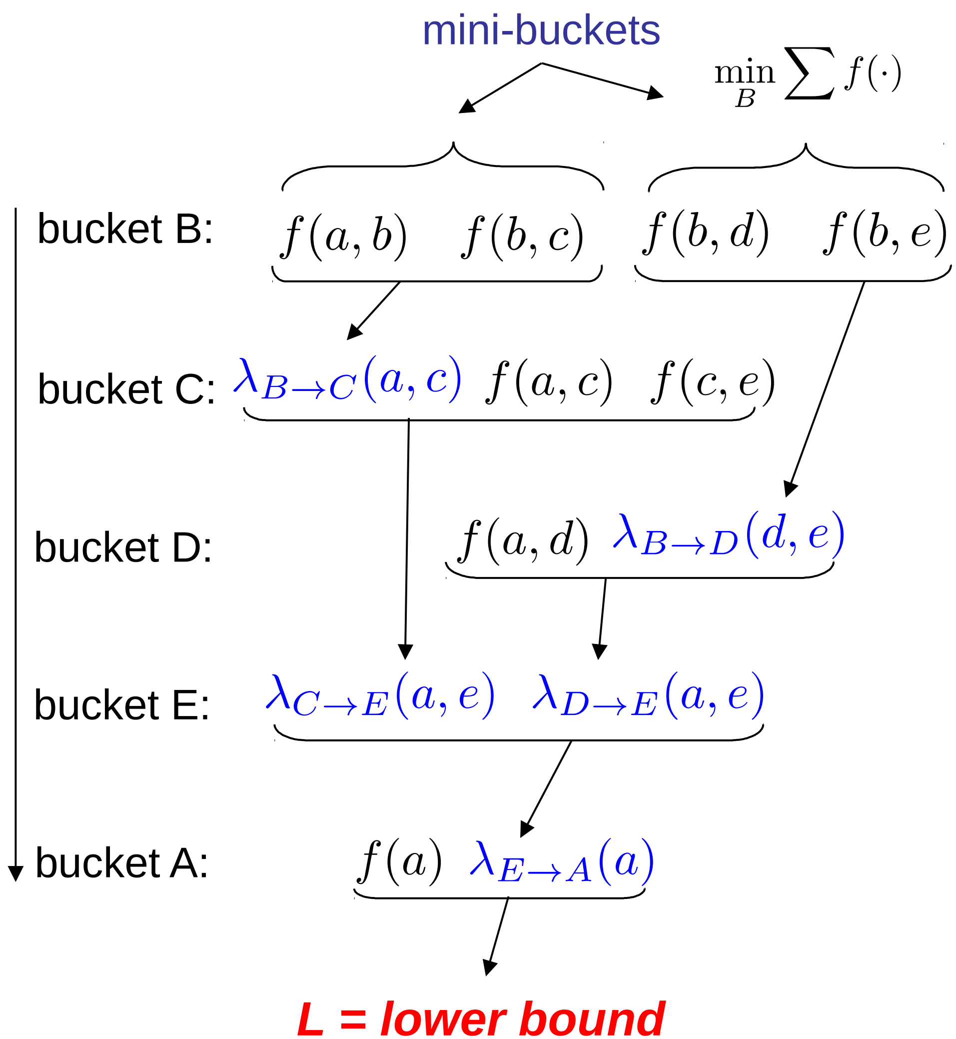
Mini-Bucket as Dual Decomposition



Mini-Bucket as Dual Decomposition

$$\min_{a,c,b} \left[f(a,b) + f(b,c) - \lambda_{B\to C}(a,c) \right] = 0$$

$$\min_{d,e,b} \left[f(b,d) + f(b,e) - \lambda_{B\to D}(d,e) \right] = 0$$

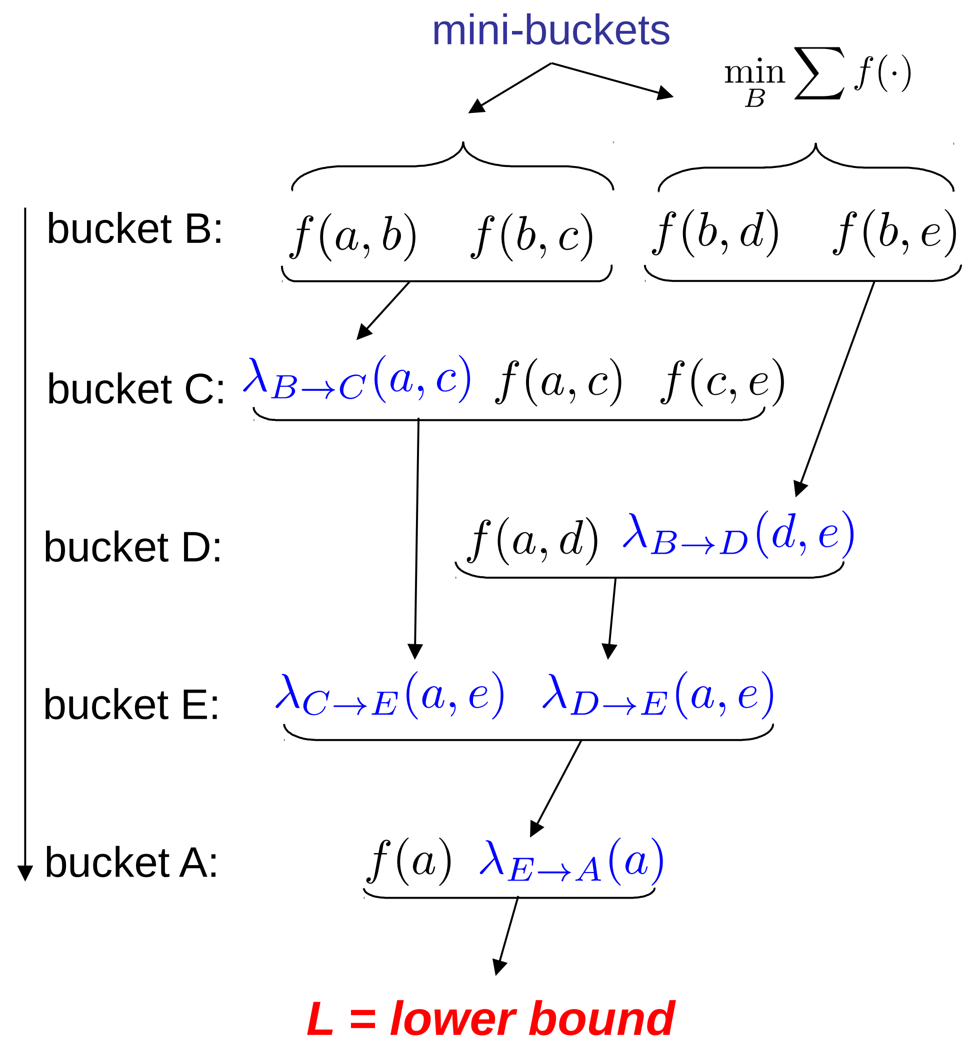


Mini-Bucket as Dual Decomposition

$$\min_{a,c,b}\left[f(a,b)+f(b,c)-\lambda_{B\to C}(a,c)\right]=0$$

$$\min_{d,e,b}\left[f(b,d)+f(b,e)-\lambda_{B\to D}(d,e)\right]=0$$

$$\min_{a,e,c}\left[\lambda_{B\to C}(a,c)+f(a,c)+f(c,e)-\lambda_{C\to E}(a,e)\right]=0$$



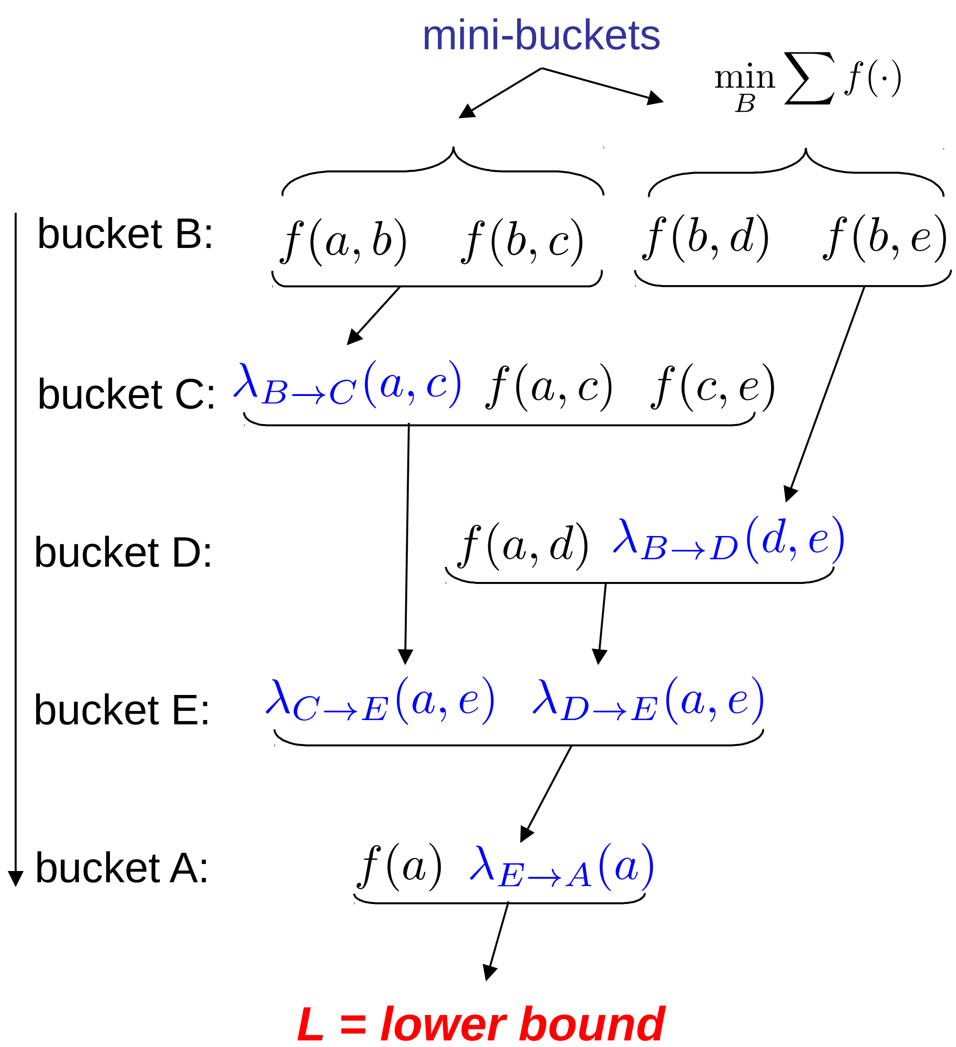
Mini-Bucket as Dual Decomposition

$$\min_{a,c,b} \left[f(a,b) + f(b,c) - \lambda_{B\to C}(a,c) \right] = 0$$

$$\min_{d,e,b} \left[f(b,d) + f(b,e) - \lambda_{B\to D}(d,e) \right] = 0$$

$$\min_{a,e,c} \left[\lambda_{B\to C}(a,c) + f(a,c) + f(c,e) - \lambda_{C\to E}(a,e) \right] = 0$$

$$\min_{a,d} \left[f(a,d) + \lambda_{B\to D}(d,e) - \lambda_{D\to E}(a,e) \right] = 0$$



Mini-Bucket as Dual Decomposition

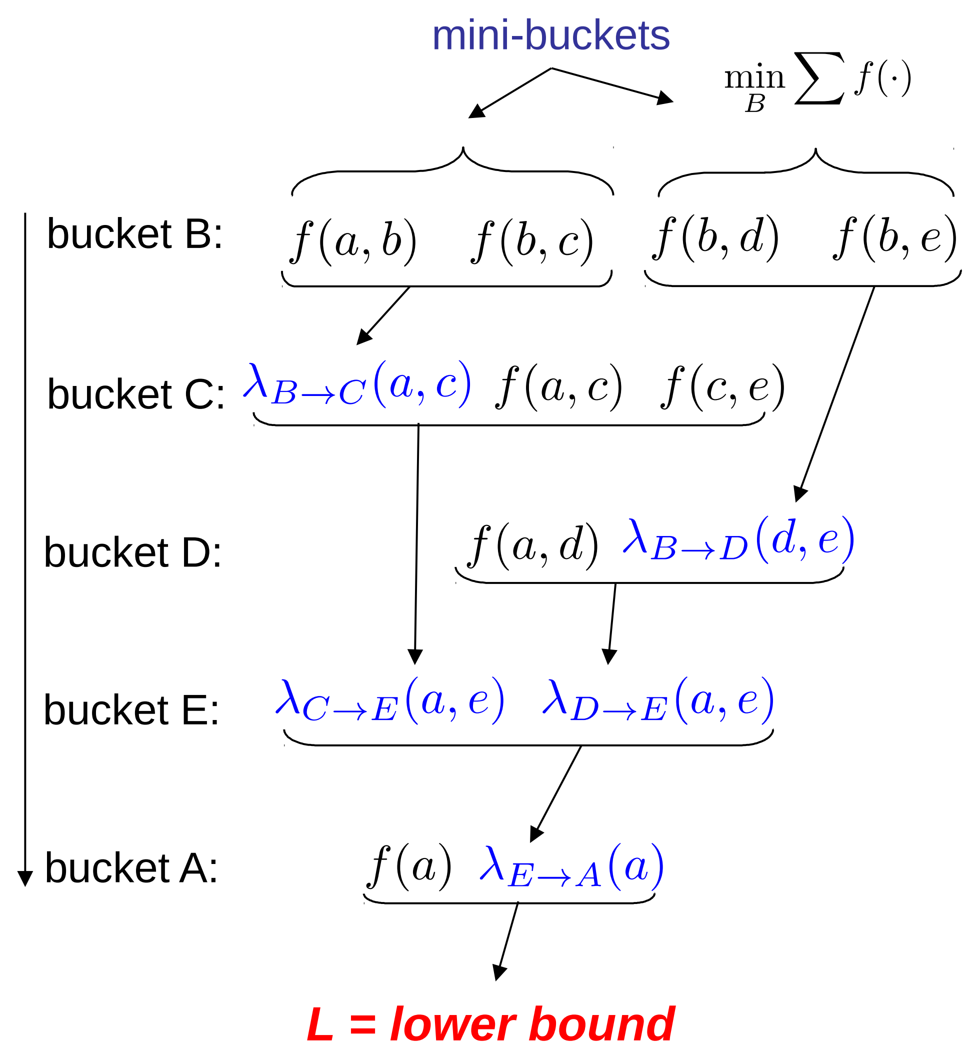
$$\min_{a,c,b} \left[f(a,b) + f(b,c) - \lambda_{B\to C}(a,c) \right] = 0$$

$$\min_{d,e,b} \left[f(b,d) + f(b,e) - \lambda_{B\to D}(d,e) \right] = 0$$

$$\min_{a,e,c} \left[\lambda_{B\to C}(a,c) + f(a,c) + f(c,e) - \lambda_{C\to E}(a,e) \right] = 0$$

$$\min_{a,d} \left[f(a,d) + \lambda_{B\to D}(d,e) - \lambda_{D\to E}(a,e) \right] = 0$$

$$\min_{a,e} \left[\lambda_{C\to E}(a,e) + \lambda_{D\to E}(a,e) - \lambda_{E\to A}(a) \right] = 0$$



Mini-Bucket as Dual Decomposition

$$\min_{a,c,b} \left[f(a,b) + f(b,c) - \lambda_{B \to C}(a,c) \right] = 0$$

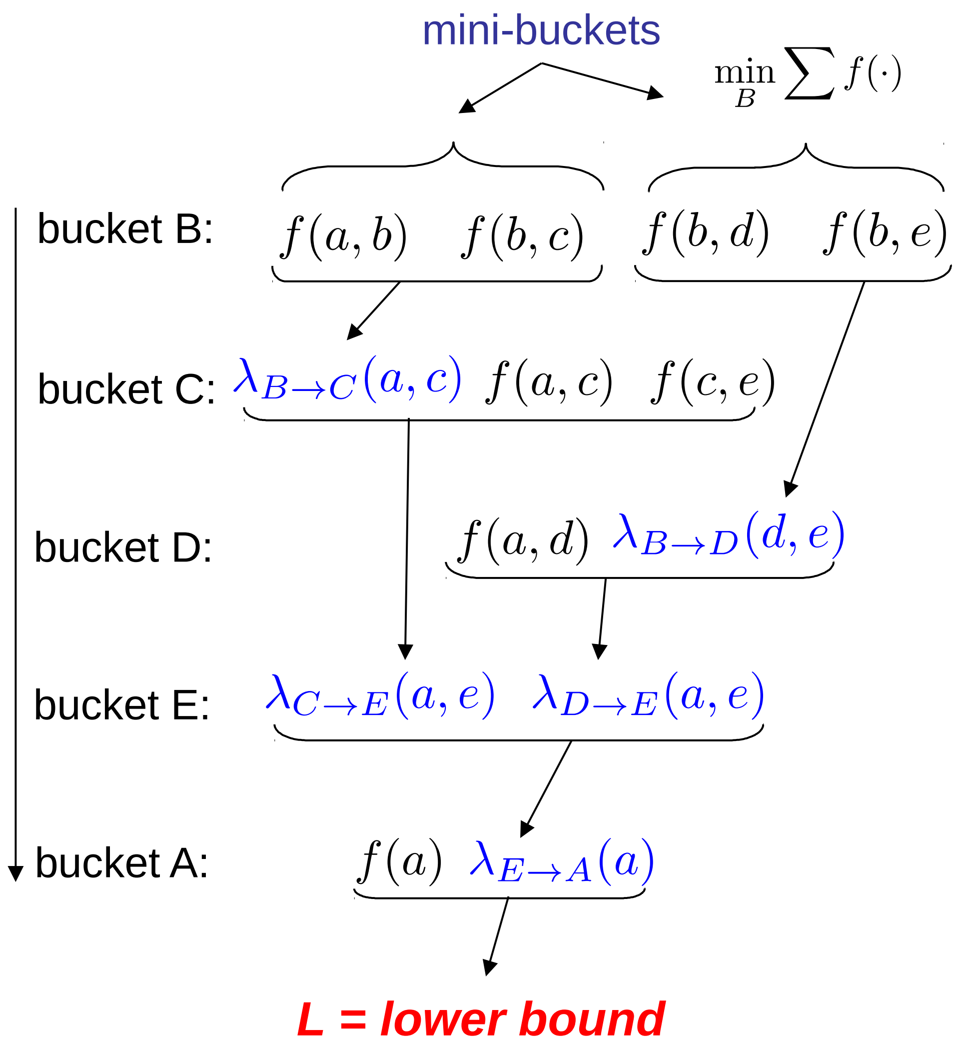
$$\min_{d,e,b} \left[f(b,d) + f(b,e) - \lambda_{B \to D}(d,e) \right] = 0$$

$$\min_{a,e,c} \left[\lambda_{B \to C}(a,c) + f(a,c) + f(c,e) - \lambda_{C \to E}(a,e) \right] = 0$$

$$\min_{a,d} \left[f(a,d) + \lambda_{B \to D}(d,e) - \lambda_{D \to E}(a,e) \right] = 0$$

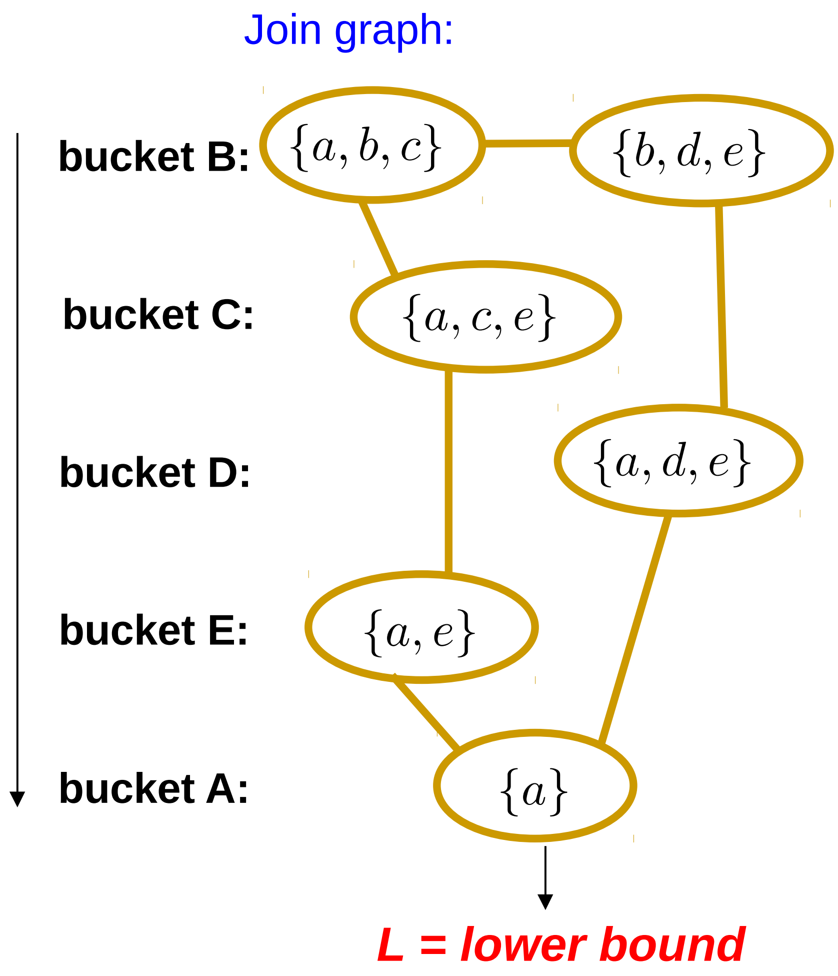
$$\min_{a,e} \left[\lambda_{C \to E}(a,e) + \lambda_{D \to E}(a,e) - \lambda_{E \to A}(a) \right] = 0$$

$$\min_{a} \left[f(a) + \lambda_{E \to A}(a) \right] = L$$



Mini-Bucket as Dual Decomposition

- Downward pass as cost-shifting
- Can also do cost-shifting within mini-buckets
- “Join graph” message passing
- “Moment matching” version: one message update within each bucket during downward sweep.



Anytime Approximation

anytime - mpe(ε)

Initialize : $i = i_0$

While time and space resources are available

$i \leftarrow i + i_{step}$

$U \leftarrow$ upper bound computed by *approx - mpe(i)*

$L \leftarrow$ lower bound computed by *approx - mpe(i)*

keep the best solution found so far

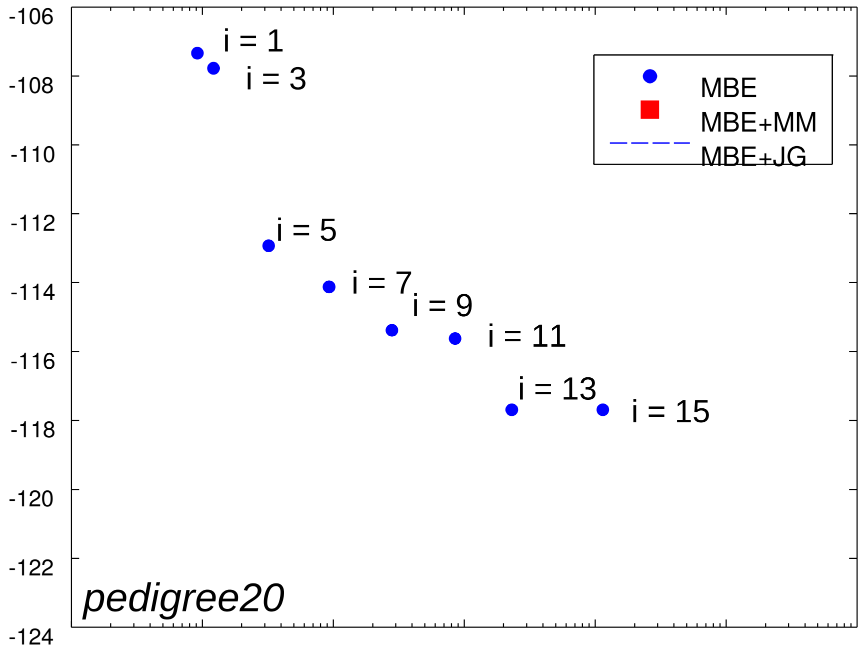
if $1 \le \frac{U}{L} \le 1 + \varepsilon$, return solution

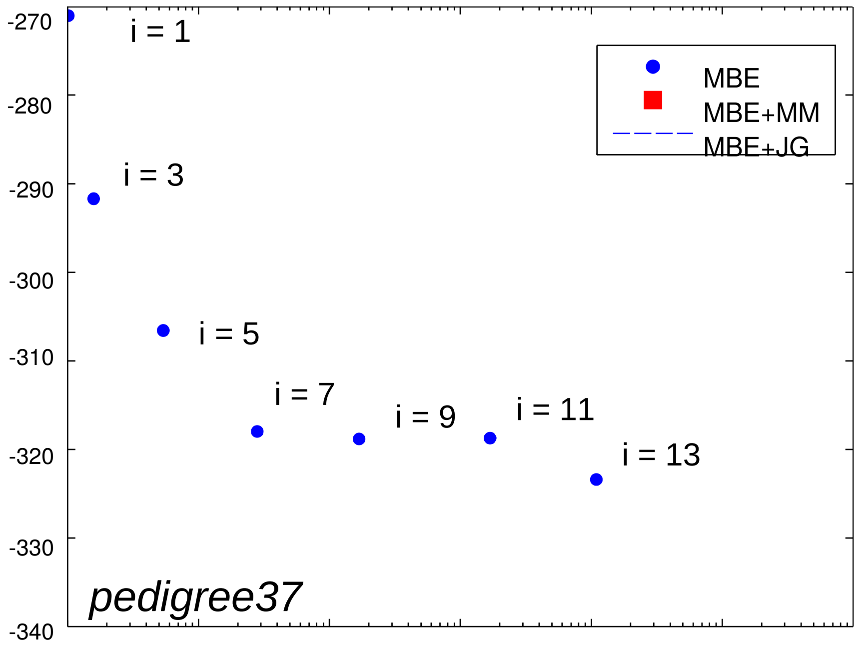
end

return the largest L and the smallest U

[Dechter and Rish, 2003]

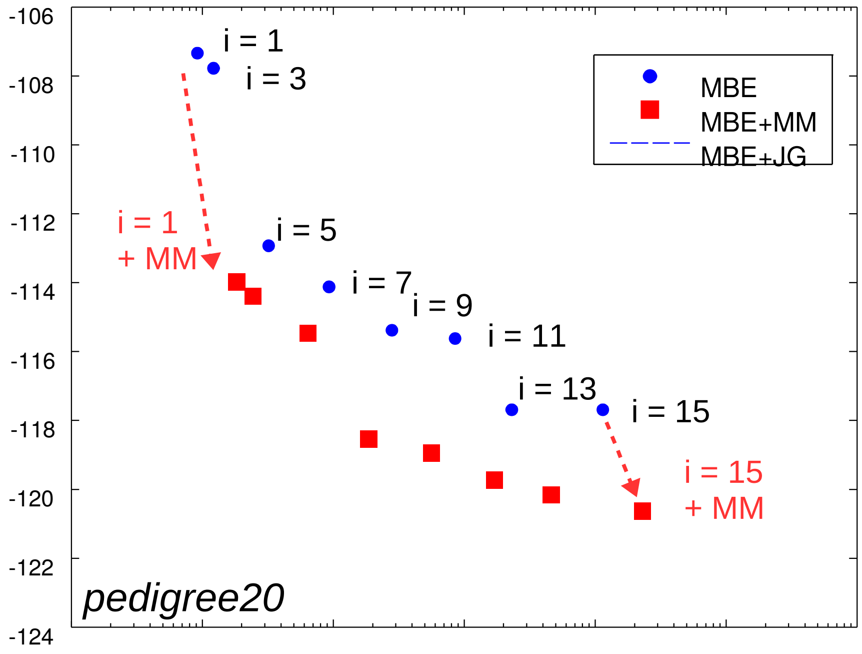
Anytime Approximation

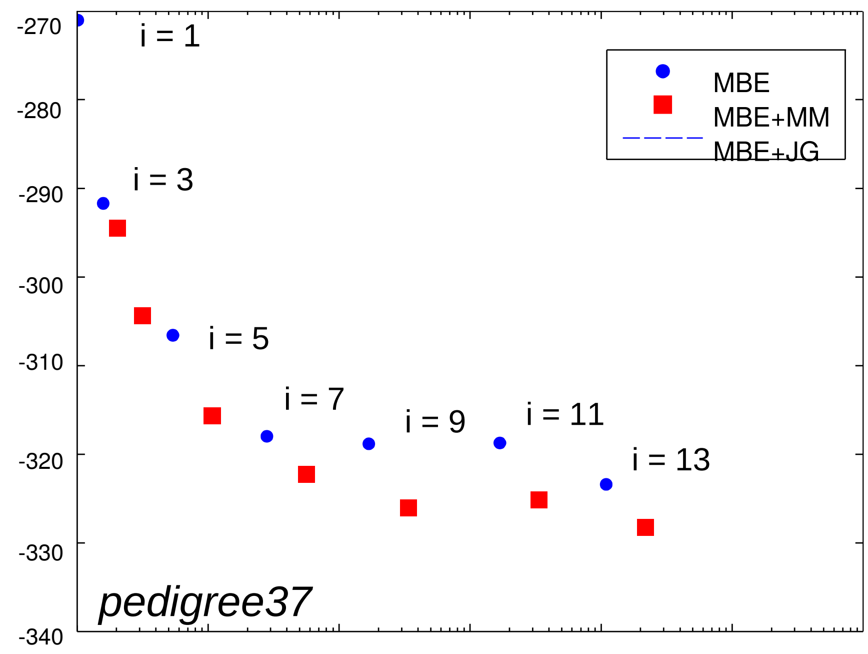




- Can tighten the bound in various ways
  - Cost-shifting (improve consistency between cliques)
  - Increase i-bound (higher order consistency)
- Simple moment-matching step improves bound significantly

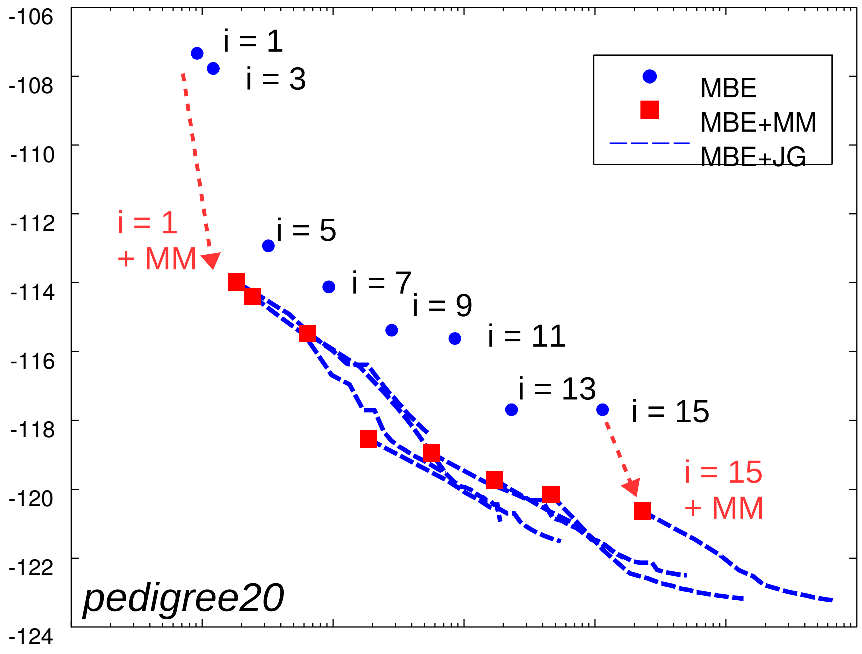
Anytime Approximation

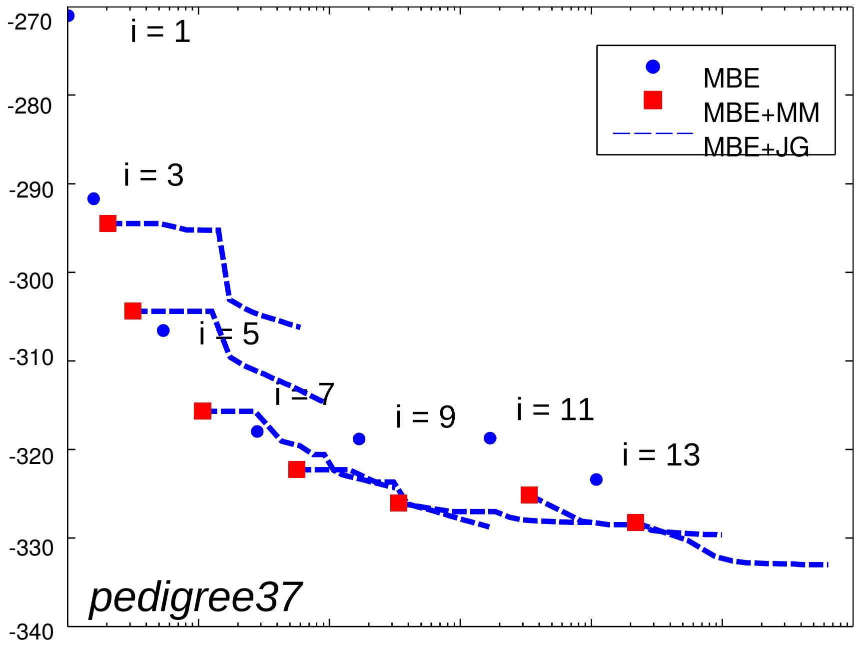




- Can tighten the bound in various ways
  - Cost-shifting (improve consistency between cliques)
  - Increase i-bound (higher order consistency)
- Simple moment-matching step improves bound significantly

Anytime Approximation





- Can tighten the bound in various ways
  - Cost-shifting (improve consistency between cliques)
  - Increase i-bound (higher order consistency)
- Simple moment-matching step improves bound significantly

Weighted Mini-Bucket

(for summation bounds)

Exact bucket elimination:

$$\lambda_B(a,c,d,e) = \sum_b \big[\, f(a,b)\cdot f(b,c)\cdot f(b,d)\cdot f(b,e) \,\big]$$

$$\le \Big[\sum_b^{w_1} f(a,b) f(b,c)\Big] \cdot \Big[\sum_b^{w_2} f(b,d) f(b,e)\Big]$$

$$= \lambda_{B\to C}(a,c) \qquad\qquad \cdot\, \lambda_{B\to D}(d,e)$$

(mini-buckets)

where $\sum_x^{w} f(x) = \Big[\sum_x f(x)^{1/w}\Big]^{w}$

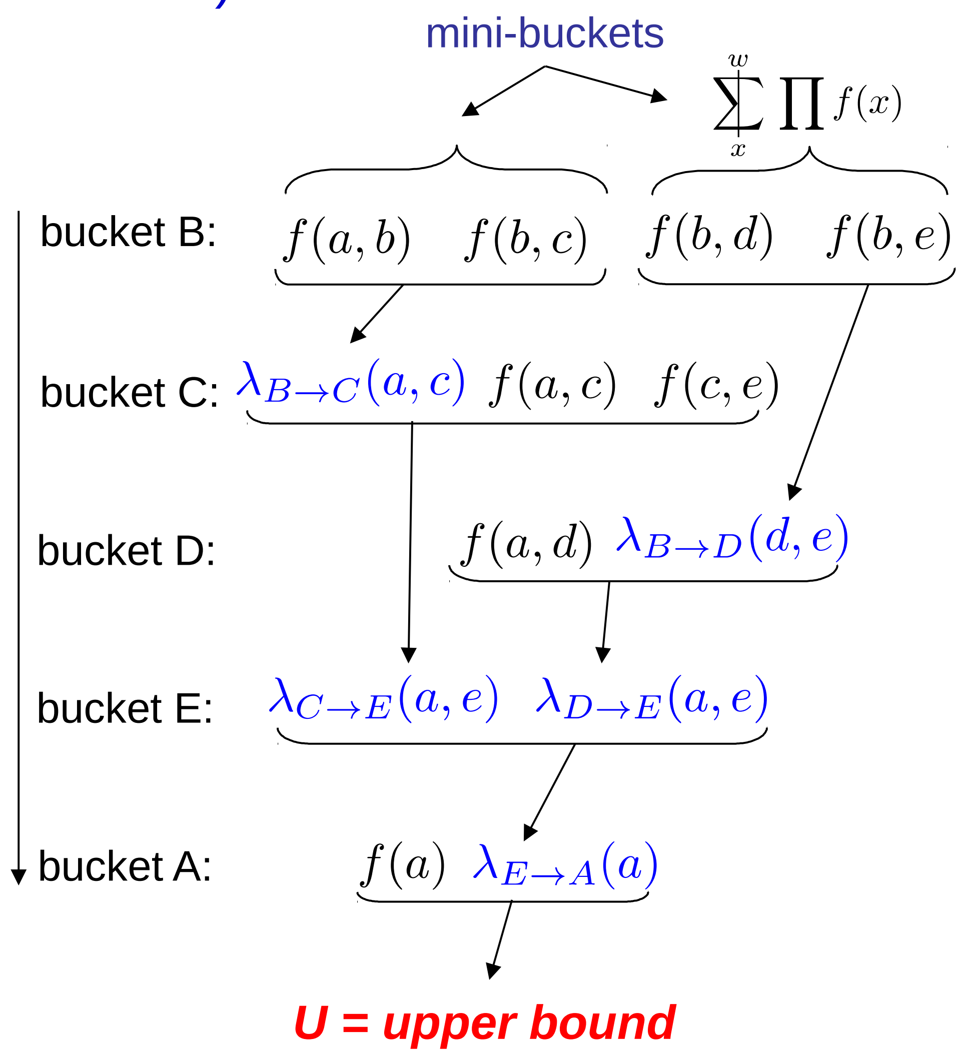
is the weighted or “power” sum operator

By Holder's inequality,

$$\sum_x^{w} f_1(x) f_2(x) \le \Big[\sum_x^{w_1} f_1(x)\Big]\Big[\sum_x^{w_2} f_2(x)\Big]$$

where $w_1 + w_2 = w$ and $w_1 > 0,\ w_2 > 0$

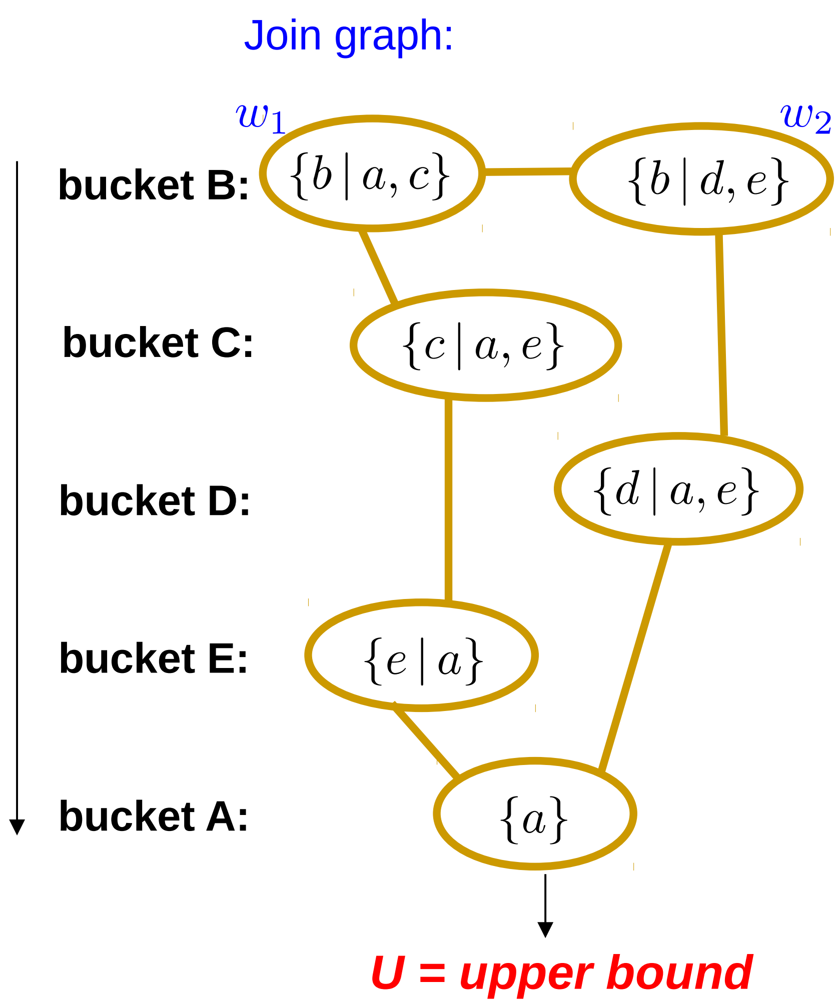
(lower bound if $w_1 > 0,\ w_2 < 0$)



[Liu & Ihler 2011]

Weighted Mini-Bucket

- Related to conditional entropy decomposition
  [Globerson & Jaakkola 2008]
  but, with an efficient, "primal" bound form
- We can optimize the bound over:
  - Cost-shifting
  - Weights
- Again, involves message passing on JG
- Similar, one-pass "moment matching" variant



[Liu & Ihler 2011]

WMB for Marginal MAP

Weighted mini-bucket is applicable more generally, since

$$\sum_x^w f(x) = \Big[\sum_x f(x)^{1/w}\Big]^w \Rightarrow \begin{cases} \lim_{w\to 0^+} \sum_x^w f(x) = \max_x f(x) \\ \lim_{w\to 0^-} \sum_x^w f(x) = \min_x f(x) \end{cases}$$

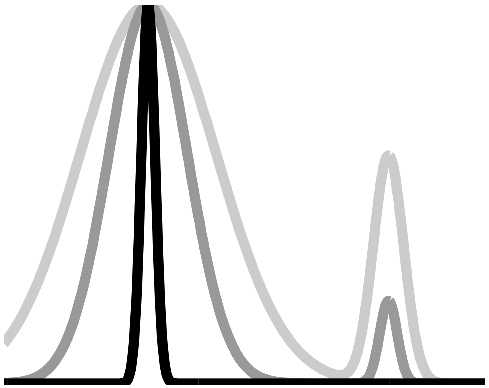
$(\, f(x) \geq 0 \,)$

(w = “temperature”)

So, when w=0+, WMB reduces to max-inference.

For marginal MAP problems, just use different w's:

$$\max_{x_B} \sum_{x_A} \prod_j f_j(x) = \sum_{x_B}^{0^+} \sum_{x_A}^{1} \prod_j f_j(x)$$



WMB for Marginal MAP

$$\lambda_{B\to C}(a,c) = \sum_b^{w_1} f(a,b)f(b,c)$$

$$\lambda_{B\to D}(d,e) = \sum_b^{w_2} f(b,d)f(b,e)$$

$$(\, w_1 + w_2 = 1 \,)$$

$$\vdots$$

$$\lambda_{E\to A}(a) = \max_e \lambda_{C\to E}(a,e)\lambda_{D\to E}(a,e)$$

$$U = \max_a f(a)\lambda_{E\to A}(a)$$

Marginal MAP:

mini-buckets

$\sum_b$	bucket B:	$\underbrace{f(a,b) \quad f(b,c)}$ $\quad \underbrace{f(b,d) \quad f(b,e)}$
$\sum_c$	bucket C:	$\underbrace{\lambda_{B\to C}(a,c)\, f(a,c) \quad f(c,e)}$
$\max_d$	bucket D:	$\underbrace{f(a,d) \;\; \lambda_{B\to D}(d,e)}$
$\max_e$	bucket E:	$\underbrace{\lambda_{C\to E}(a,e) \quad \lambda_{D\to E}(a,e)}$
$\max_a$	bucket A:	$\underbrace{f(a) \;\; \lambda_{E\to A}(a)}$

U = upper bound

Can optimize over cost-shifting and weights
(single-pass “MM” or with iterative message passing)

[Liu & Ihler 2011, 2013]

Outline

- Introduction
- Inference
- **Bounds and heuristics**
  - Basics of search: DFS versus BFS
  - Mini-Bucket Elimination
  - Weighted Mini-Buckets and Iterative Cost-Shifting
  - Generating Heuristics using Mini-Bucket Elimination
- AND/OR Search
- Exploiting parallelism
- Software

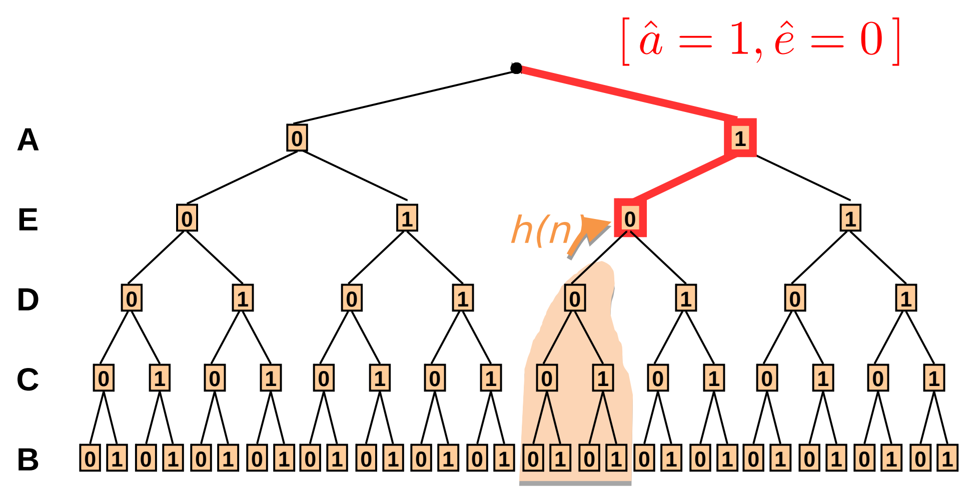
Generating Heuristics for Graphical Models

Given a cost function:

$$f(a,\ldots,e) = f(a) + f(a,b) + f(a,c) + f(a,d) + f(b,c) + f(b,d) + f(b,e) + f(c,e)$$

define an evaluation function over a partial assignment as the cost of its best extension:

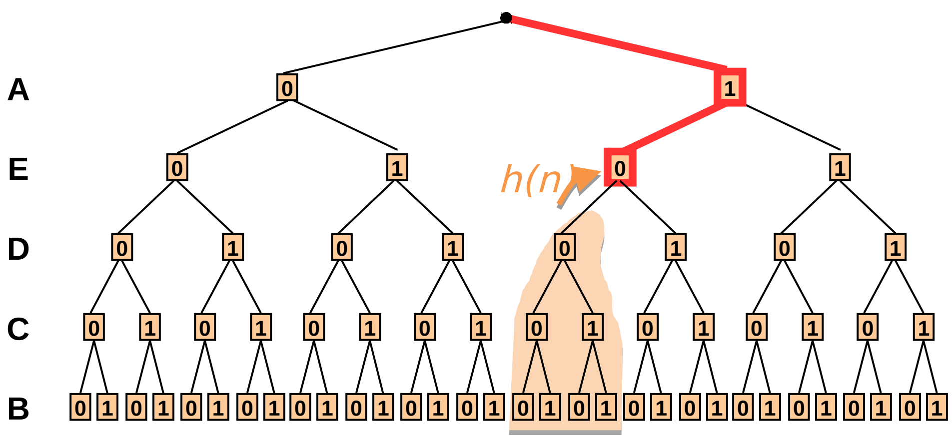
$$\begin{aligned} f^*(\hat{a},\hat{e},D) &= \min_{b,c} F(\hat{a},b,c,D,\hat{e}) \\ &= \underbrace{f(\hat{a})} + \underbrace{\min_{b,c} f(\hat{a},b) + f(\hat{a},c) + \ldots} \\ &= g(\hat{a},\hat{e},D) + h^*(\hat{a},\hat{e},D) \end{aligned}$$



[Kask and Dechter, 2001]

Static Mini-Bucket Heuristics

Given a partial assignment, $[\hat{a}=1, \hat{e}=0]$
(weighted) mini-bucket gives an admissible heuristic:



mini-buckets

$\min_B \sum f(\cdot)$

bucket B: $\underbrace{f(a,b) \quad f(b,c)} \quad \underbrace{f(b,d) \quad f(b,e)}$

bucket C: $\lambda_{B\to C}(a,c)\, f(a,c) \quad f(c,e)$

bucket D: $f(a,d) \ \lambda_{B\to D}(d,e)$

bucket E: $\lambda_{C\to E}(a,e) \quad \lambda_{D\to E}(a,e)$

bucket A: $f(a) \ \lambda_{E\to A}(a)$

L = lower bound

cost to go:

$$\tilde{h}(\hat{a},\hat{e},D) = \lambda_{C\to E}(\hat{a},\hat{e}) + f(\hat{a},D) + \lambda_{B\to D}(D,\hat{e})$$

(admissible: $\tilde{h}(\hat{a},\hat{e},D) \le h^*(\hat{a},\hat{e},D)$)

cost so far:

$$g(\hat{a},\hat{e}) = f(A=\hat{a})$$

Properties of the Heuristic

- MB heuristic is monotone, admissible
- Computed in linear time
- **IMPORTANT**
  - Heuristic strength can vary by MB(i)
  - Higher i-bound → more pre-processing → more accurate heuristic → less search
- Allows controlled trade-off between pre-processing and search

Dynamic Mini-Bucket Heuristics

- Rather than pre-compile, compute the heuristics, dynamically, during search
- **Dynamic MB**: use the Mini-Bucket algorithm to produce a bound for any node during search
- **Dynamic MBTE**: compute heuristics simultaneously for all un-instantiated variables using Mini-Bucket-Tree Elimination (MBTE)
- **MBTE** is an approximation scheme defined over cluster trees. It outputs multiple bounds for each variable and value extension at once

[Marinescu, Kask and Dechter, 2003]

Outline

- Introduction
- Inference
- Bounds and heuristics
- **AND/OR Search**
- Exploiting parallelism
- Software

Outline

- Introduction
- Inference
- Bounds and heuristics
- **AND/OR Search**
  - AND/OR Search Spaces
  - AND/OR Branch and Bound
  - Best-First AND/OR Search
  - Advanced Searches and Tasks
- Exploiting parallelism
- Software

Solution Techniques

AND/OR search

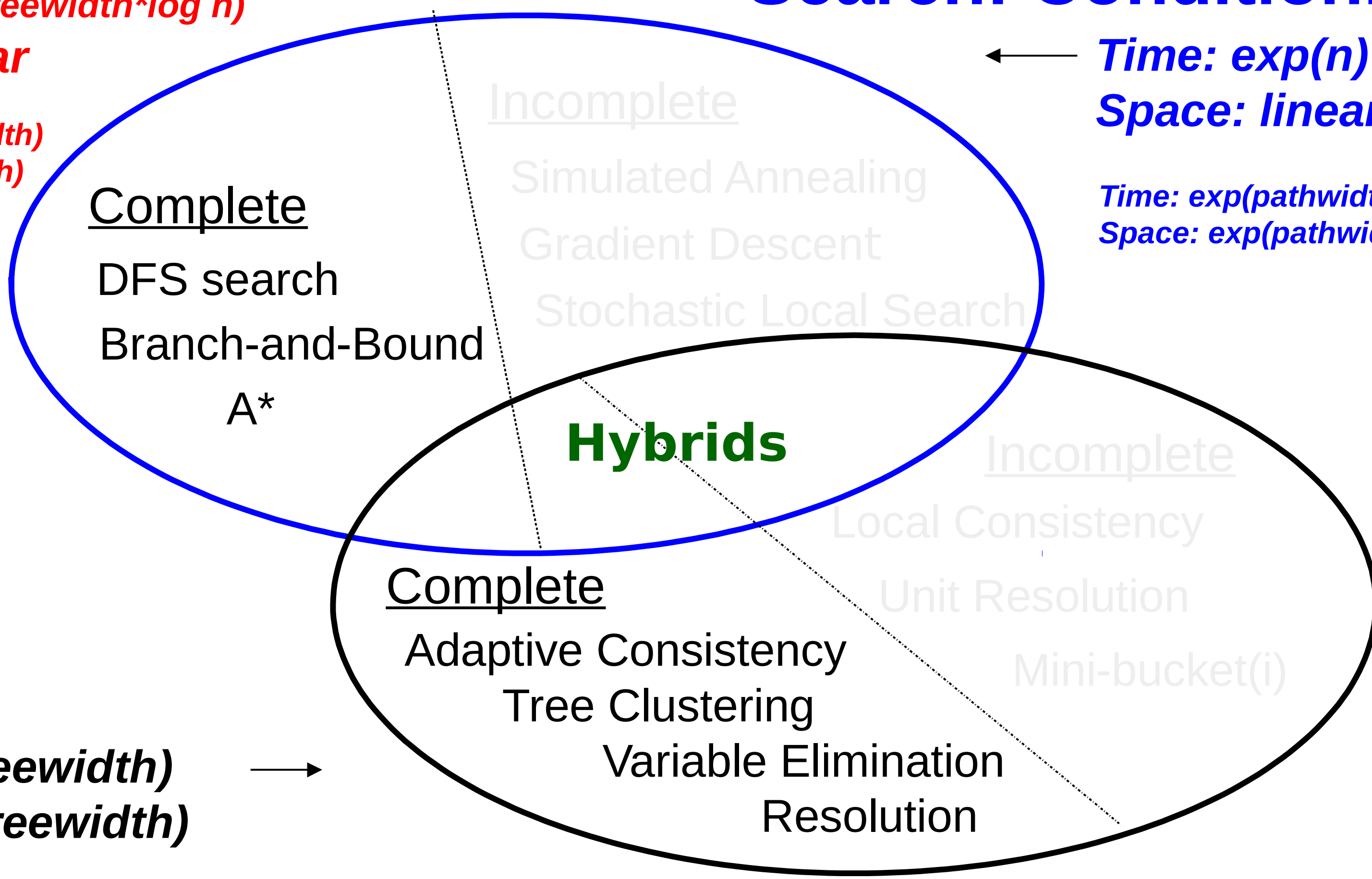
Time: exp(treewidth*log n)
Space: linear

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

Search: Conditioning

Time: exp(n)
Space: linear

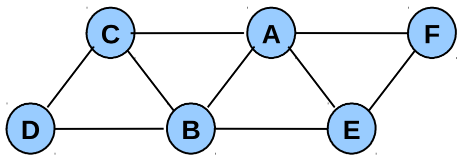
Time: exp(pathwidth)
Space: exp(pathwidth)



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**

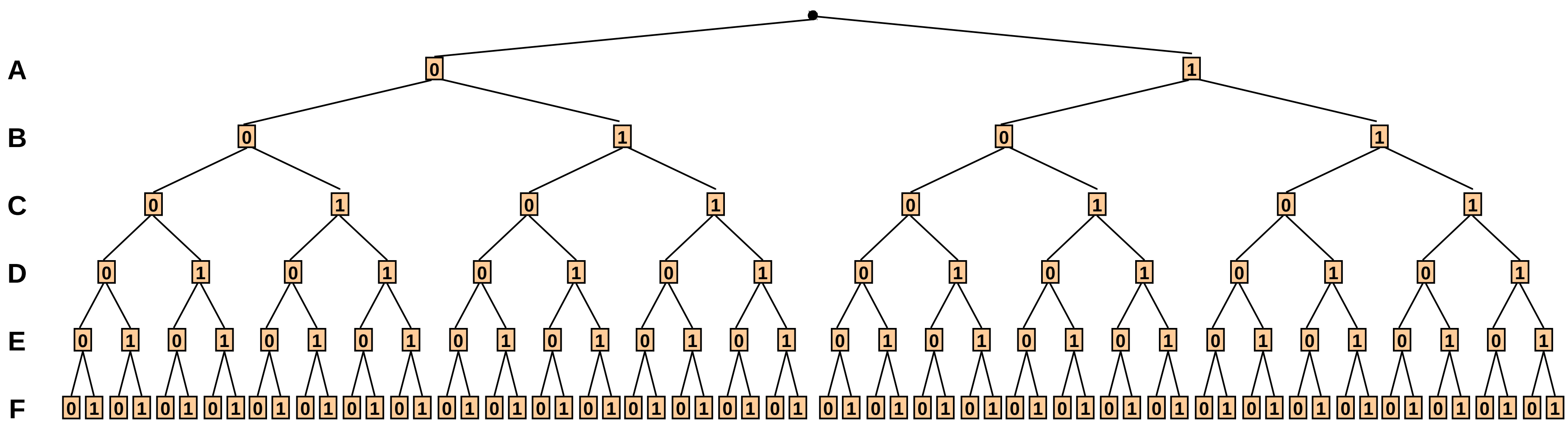
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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

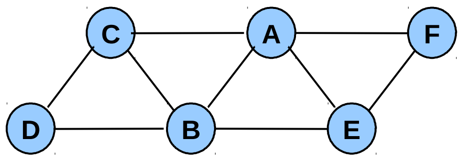
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

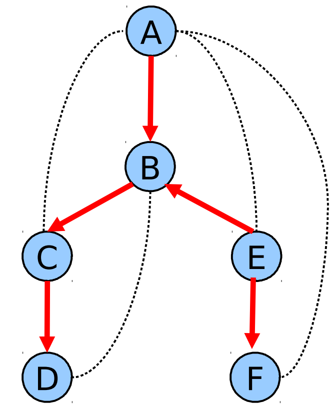
E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

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

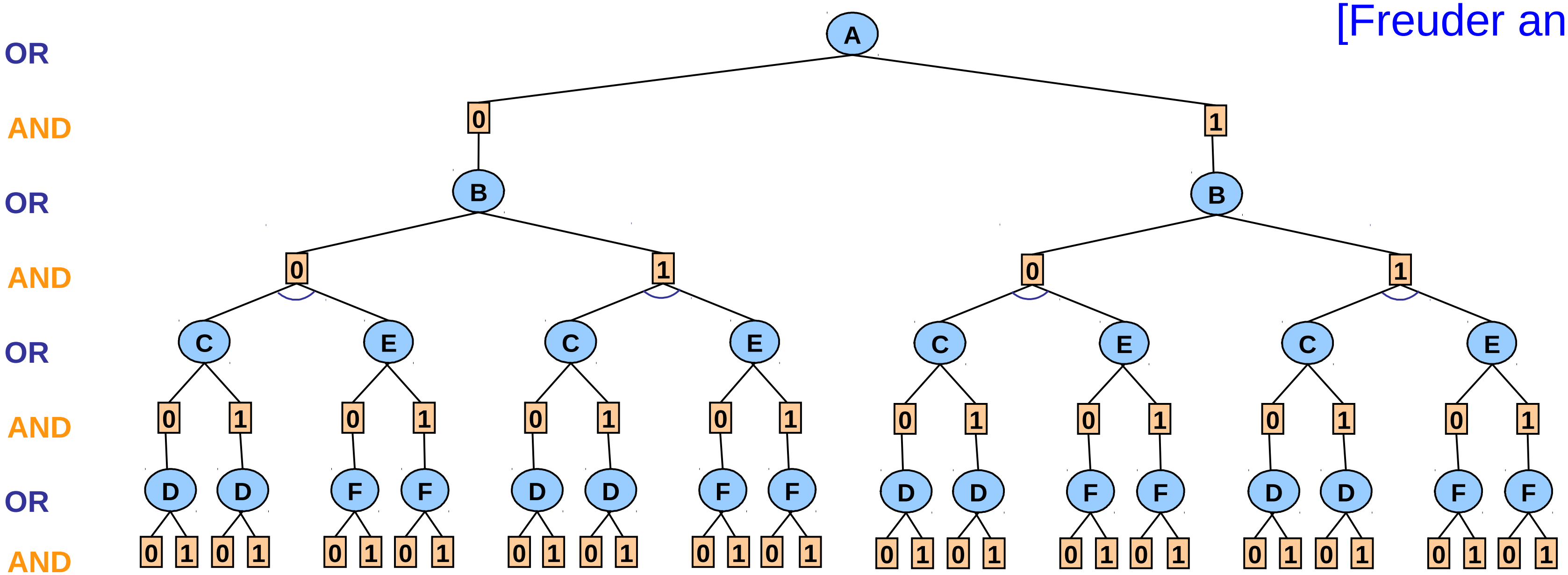


The AND/OR Search Tree



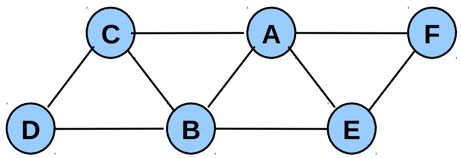


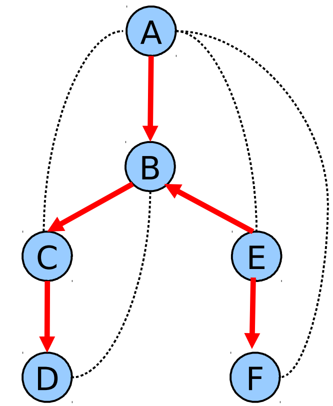
Pseudo tree
[Freuder and Quinn, 1985]



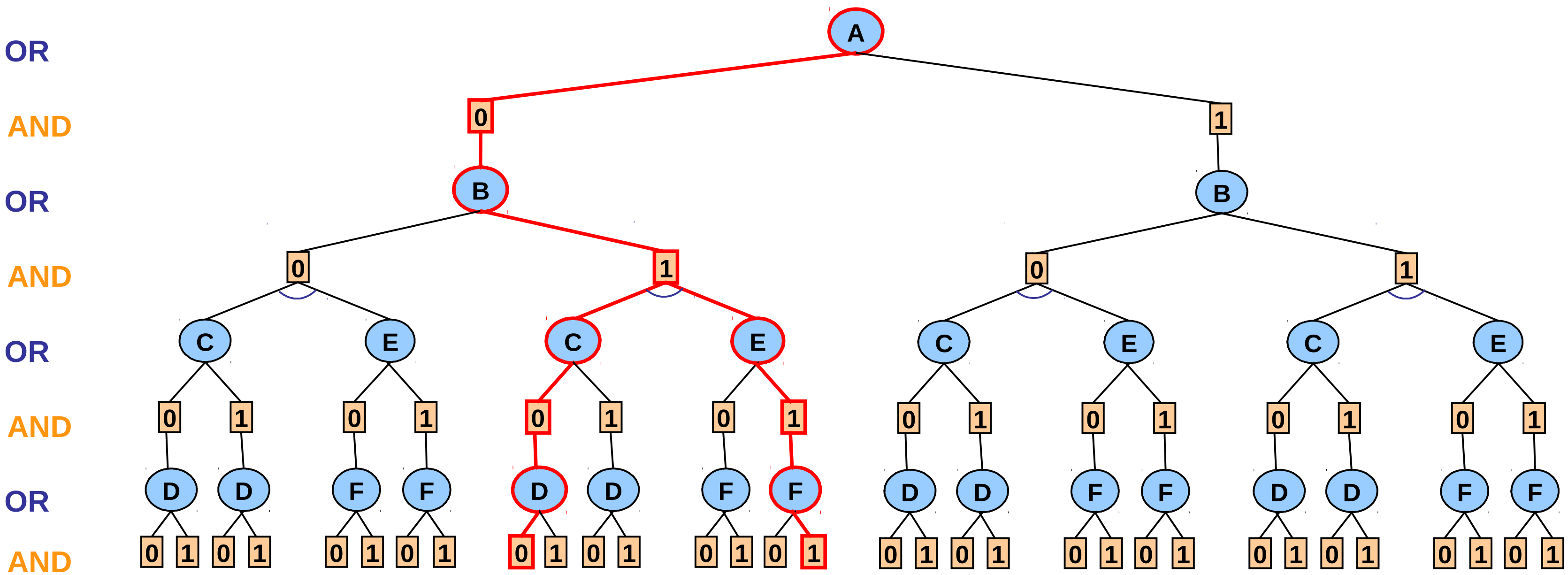
[Dechter and Mateescu, 2007]

The AND/OR Search Tree



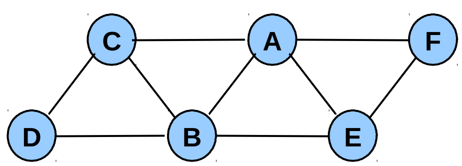


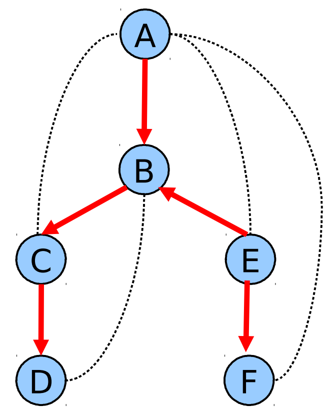
Pseudo tree



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**

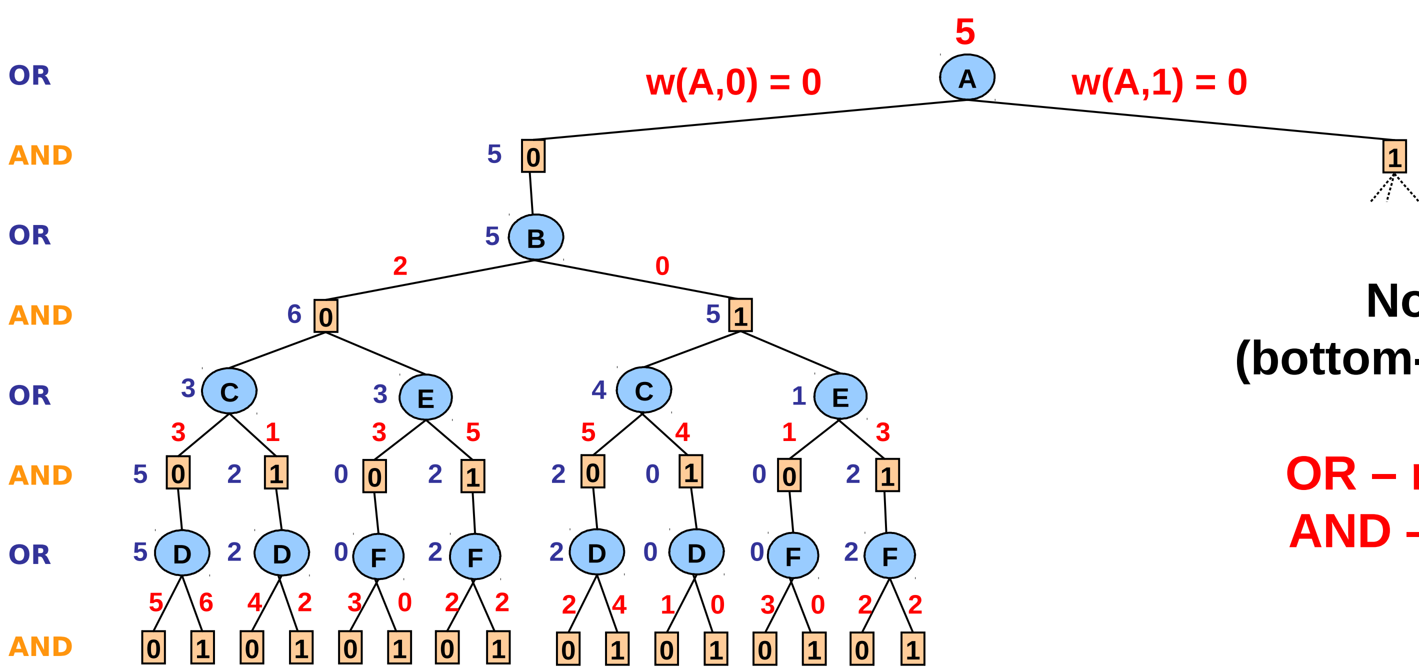
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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**

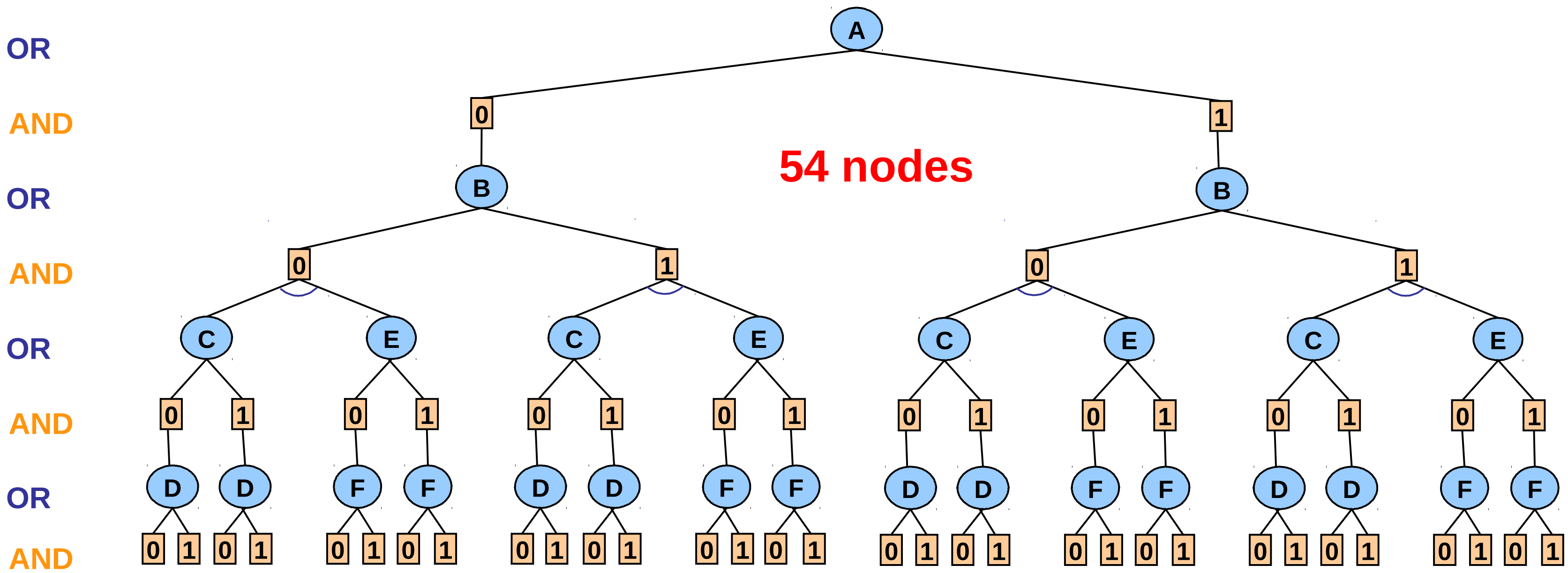
Objective function: $F^* = \min_X \sum_i f_i(X)$

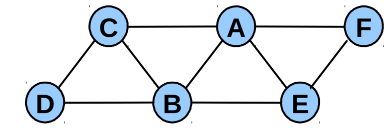


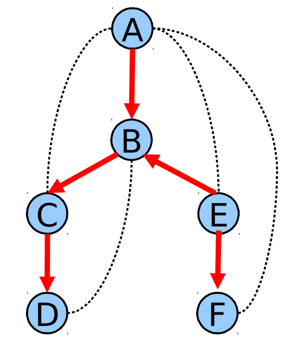
Node Value (bottom-up evaluation)

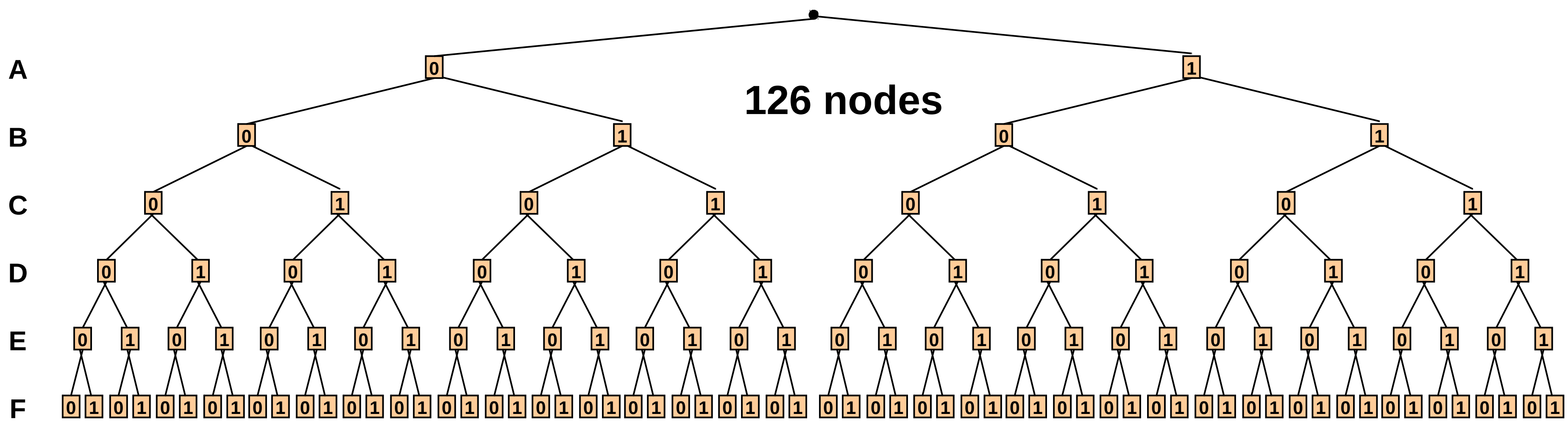
OR – minimization
AND – summation

AND/OR versus OR Spaces









AND/OR versus OR Spaces

width	depth	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

Complexity of AND/OR Tree Search

	AND/OR tree	OR tree
Space	$O(n)$	$O(n)$
Time	$O(n\,d^t)$ $O(n\,d^{(w^* \log n)})$	$O(d^n)$

[Freuder & Quinn85], [Collin, Dechter & Katz91], [Bayardo & Miranker95], [Darwiche01]

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

Constructing Pseudo Trees

- AND/OR serch algorithms are influenced by the **quality** of the pseudo tree
- Finding minimal induced width / depth pseudo tree is NP-hard
- Heuristics
  - Min-Fill (min induced width)
  - Hypergraph partitioning (min depth)

Constructing Pseudo Trees

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

Quality of the Pseudo Trees

Network	hypergraph		min-fill	
	w*	depth	w*	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

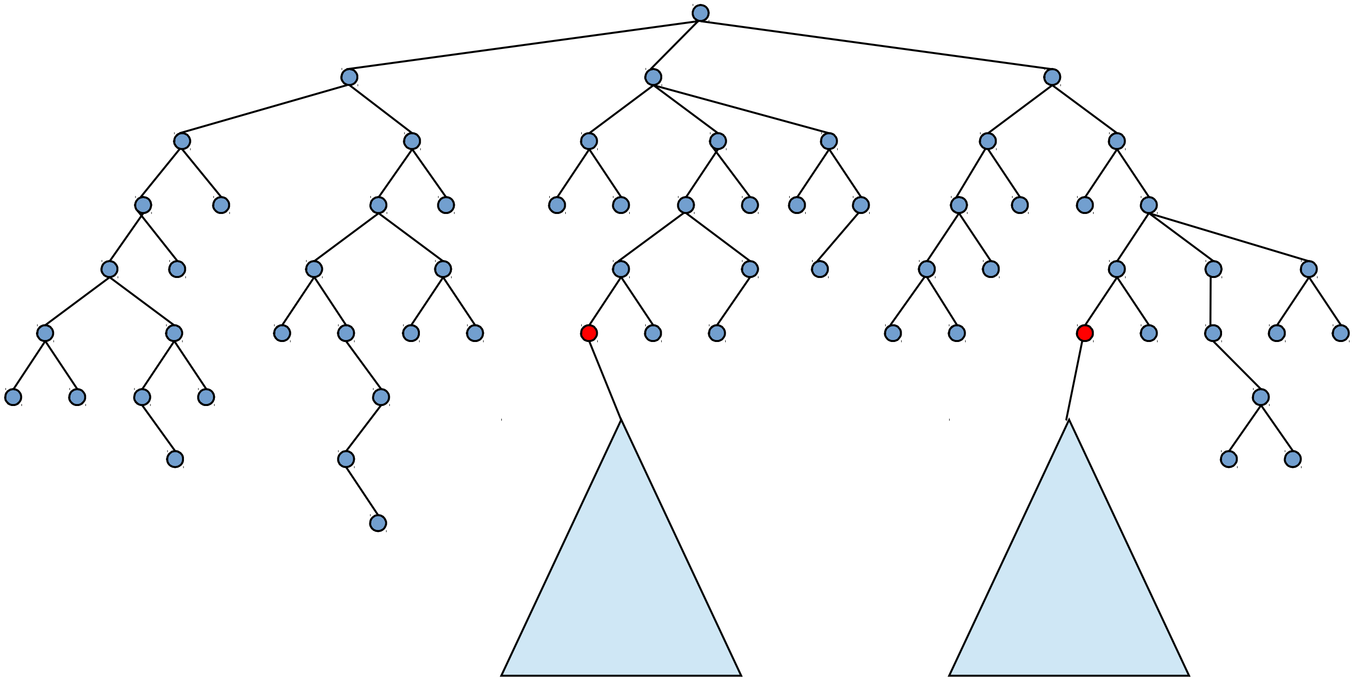
Bayesian Networks Repository

Network	hypergraph		min-fill	
	w*	depth	w*	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 Benchmark

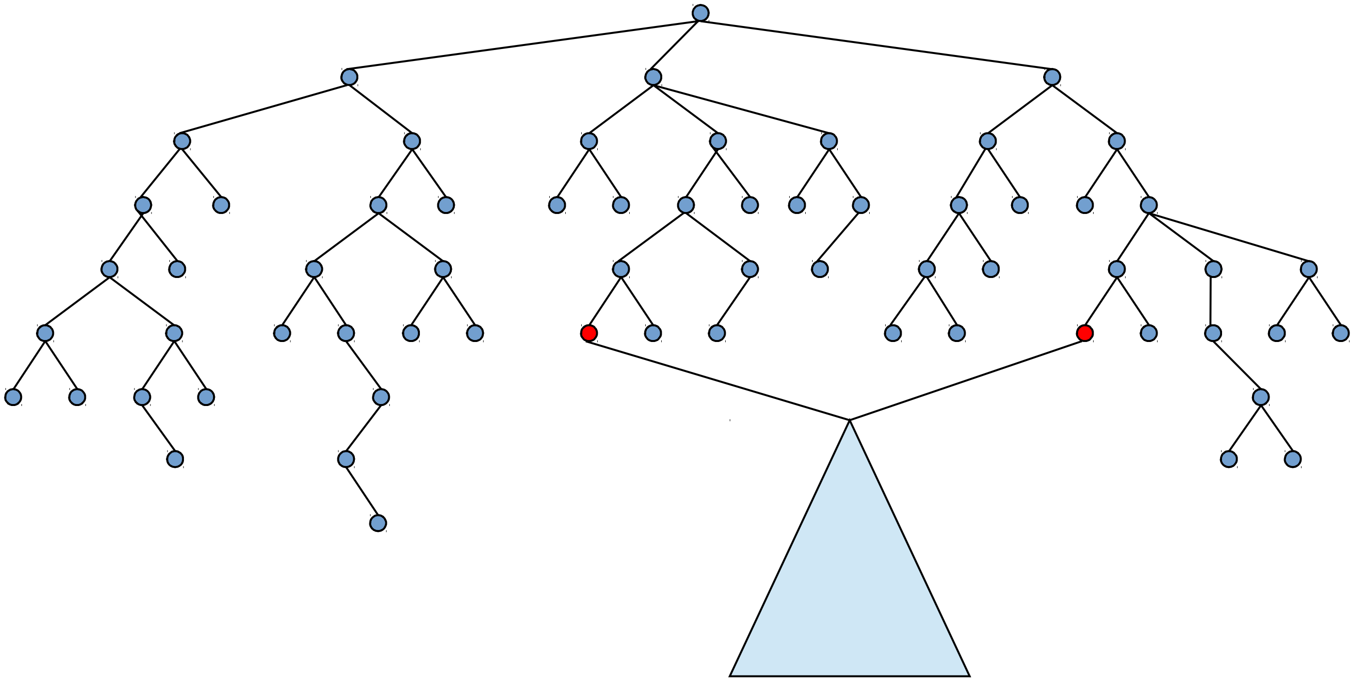
From Search Trees to Search Graphs

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



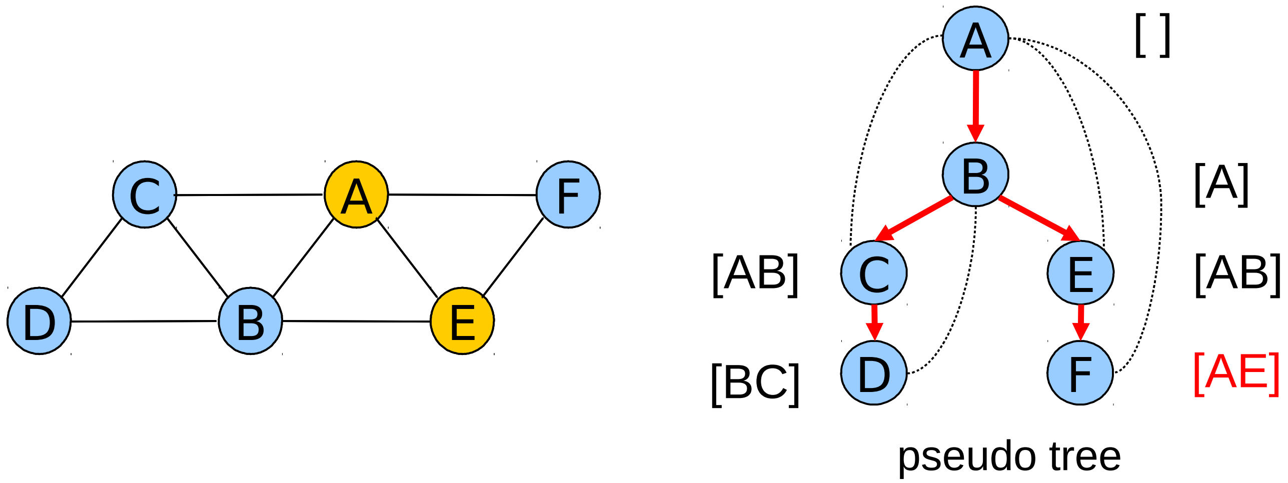
From Search Trees to Search Graphs

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

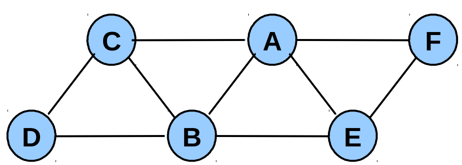


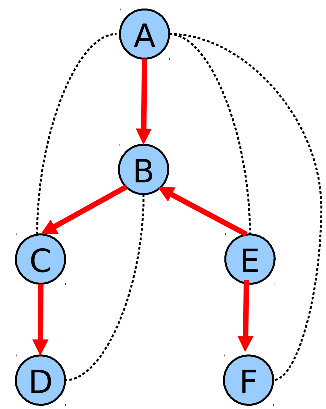
Merging Based on Contexts

- 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**

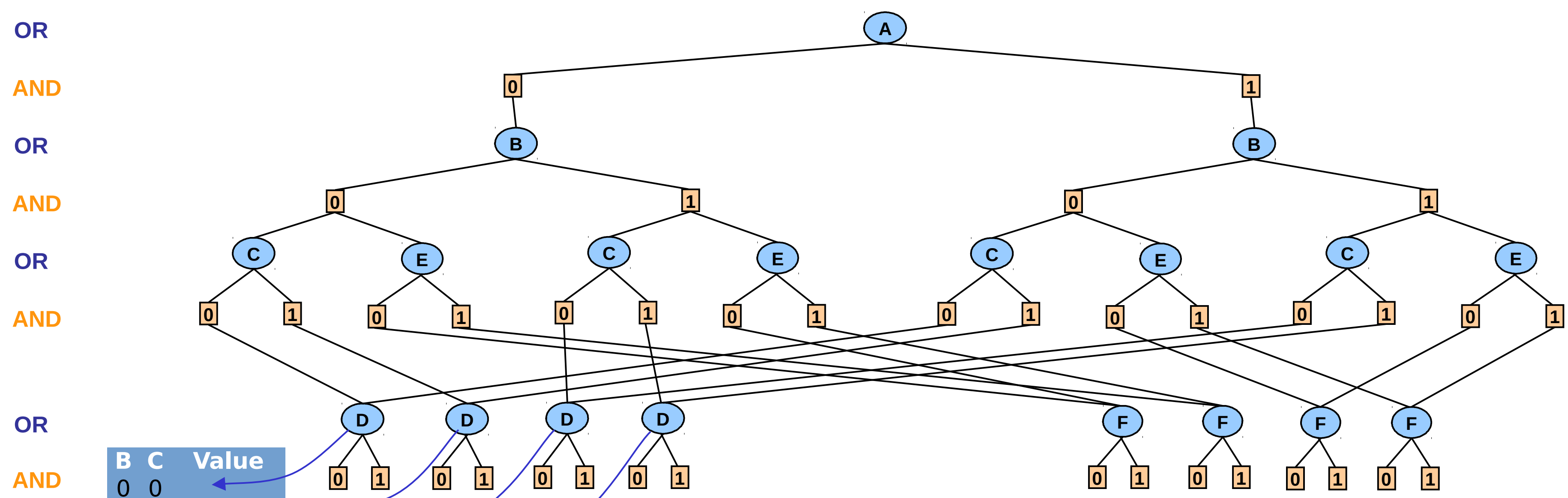
B	D	f_{bd}
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

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



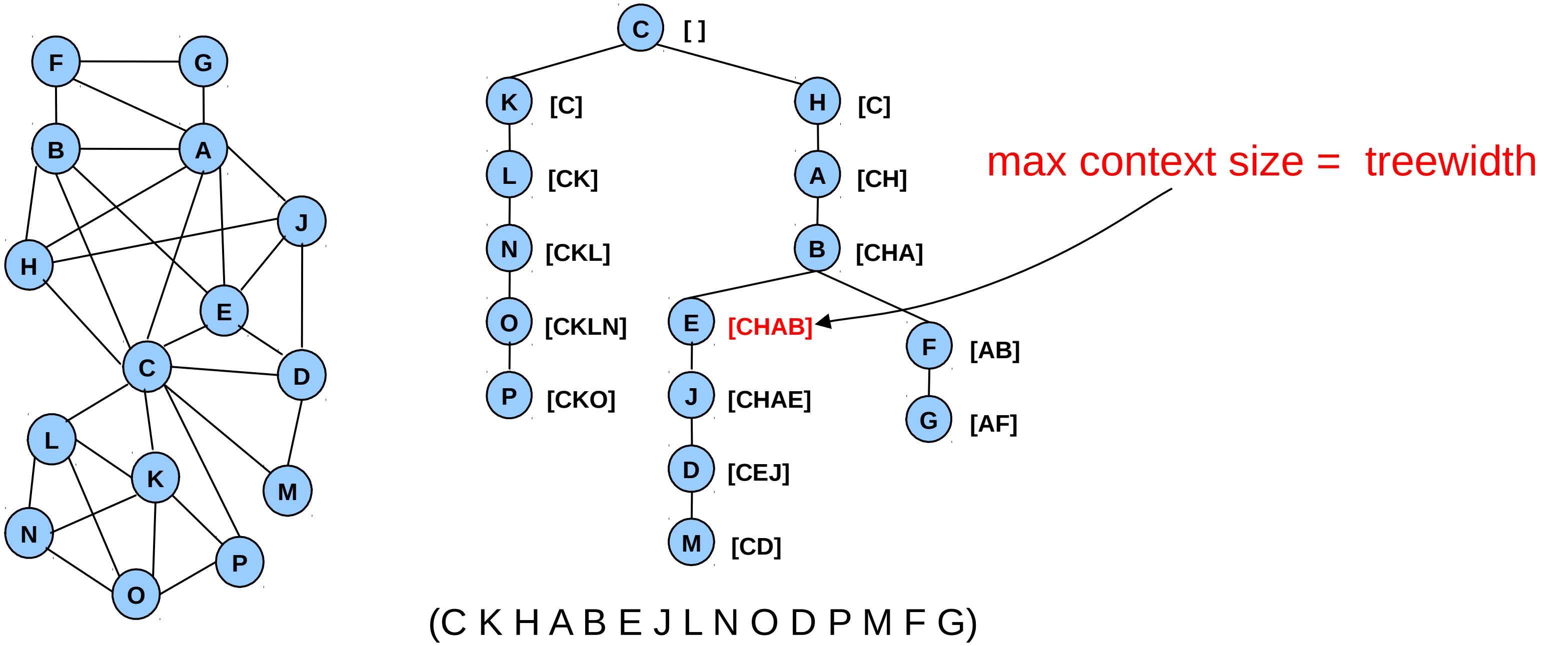
B	C	Value
0	0	
0	1	
1	0	
1	1	

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.



Complexity of AND/OR Graph Search

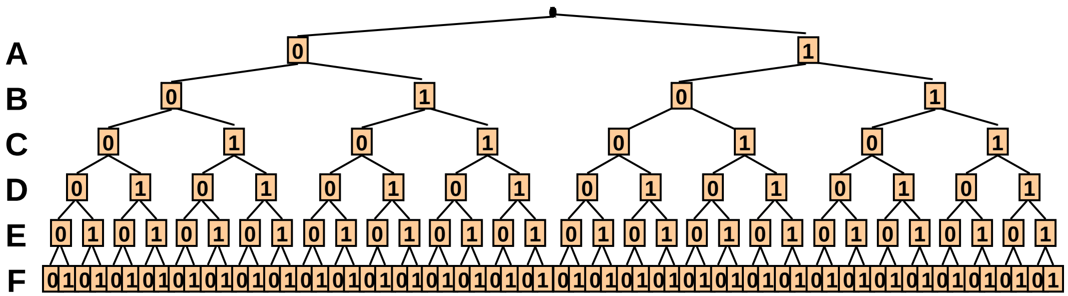
	AND/OR graph	OR graph
Space	$O(n\, d^{w^*})$	$O(n\, d^{pw^*})$
Time	$O(n\, d^{w^*})$	$O(n\, d^{pw^*})$

d = domain size
w* = induced width

n = number of variables
pw* = pathwidth

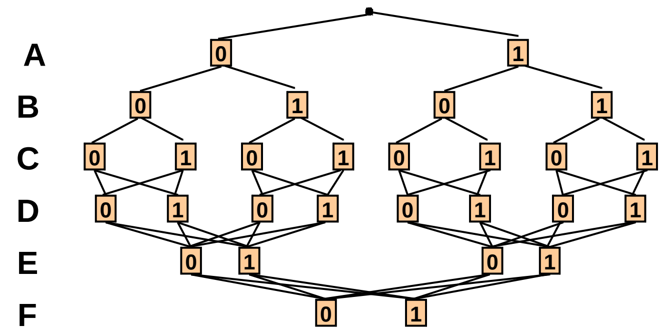
$w^* \leq pw^* \leq w^* \log n$

All Four Search Spaces



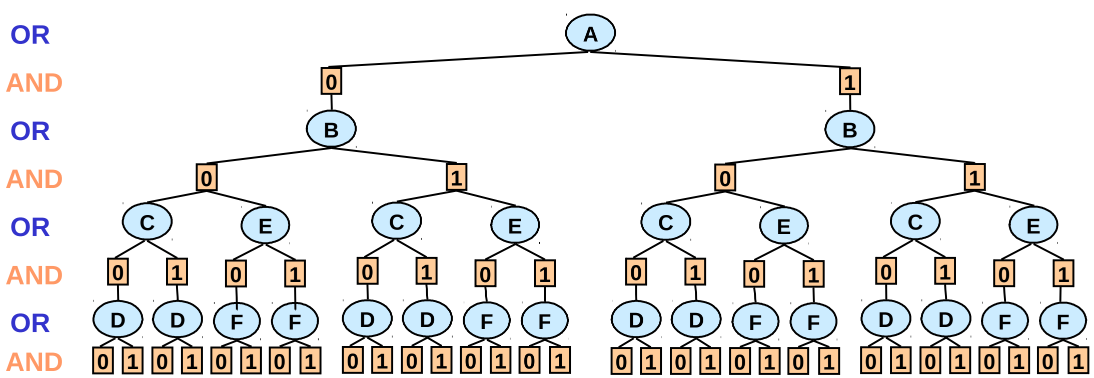
Full OR search tree

126 nodes



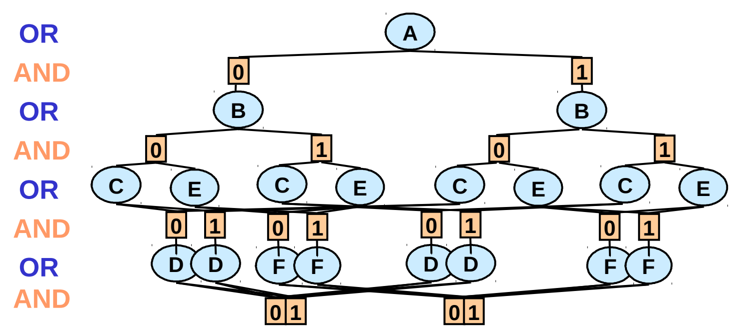
Context minimal OR search graph

28 nodes



Full AND/OR search tree

54 AND nodes



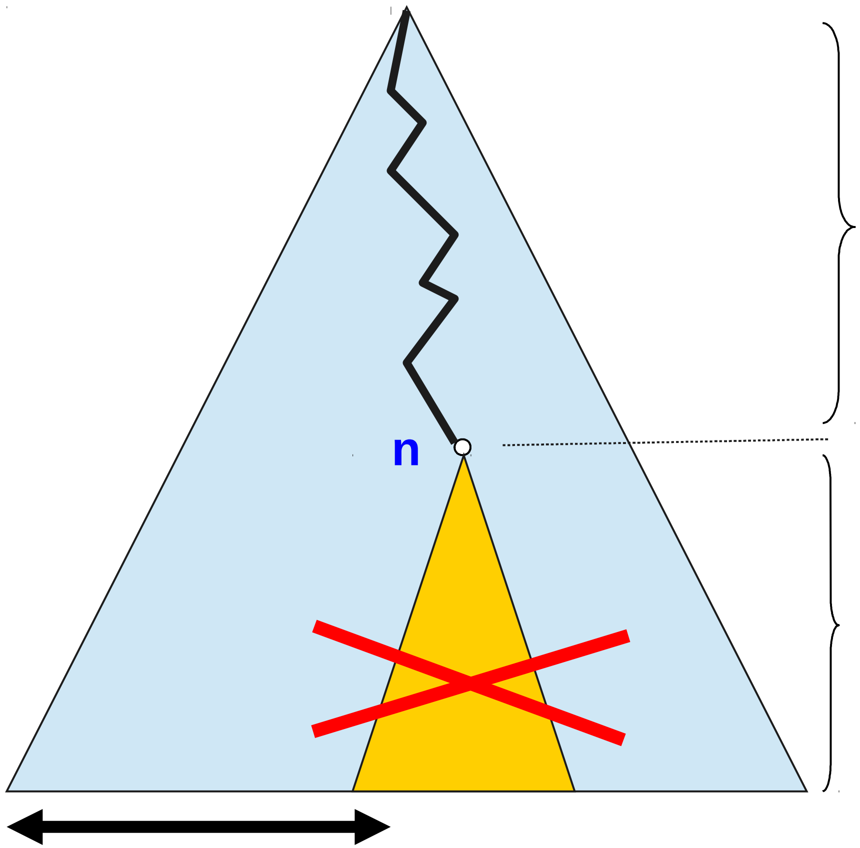
Context minimal AND/OR search graph

18 AND nodes

Outline

- **Introduction**
- **Inference**
- **Bounds and heuristics**
- **AND/OR Search**
  - AND/OR Search Spaces
  - AND/OR Branch and Bound
  - Best-First AND/OR Search
  - Advanced Searches and Tasks
- **Exploiting parallelism**
- **Software**

Classic Depth-First Branch and Bound



Each node is a COP subproblem
(defined by current conditioning)

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

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

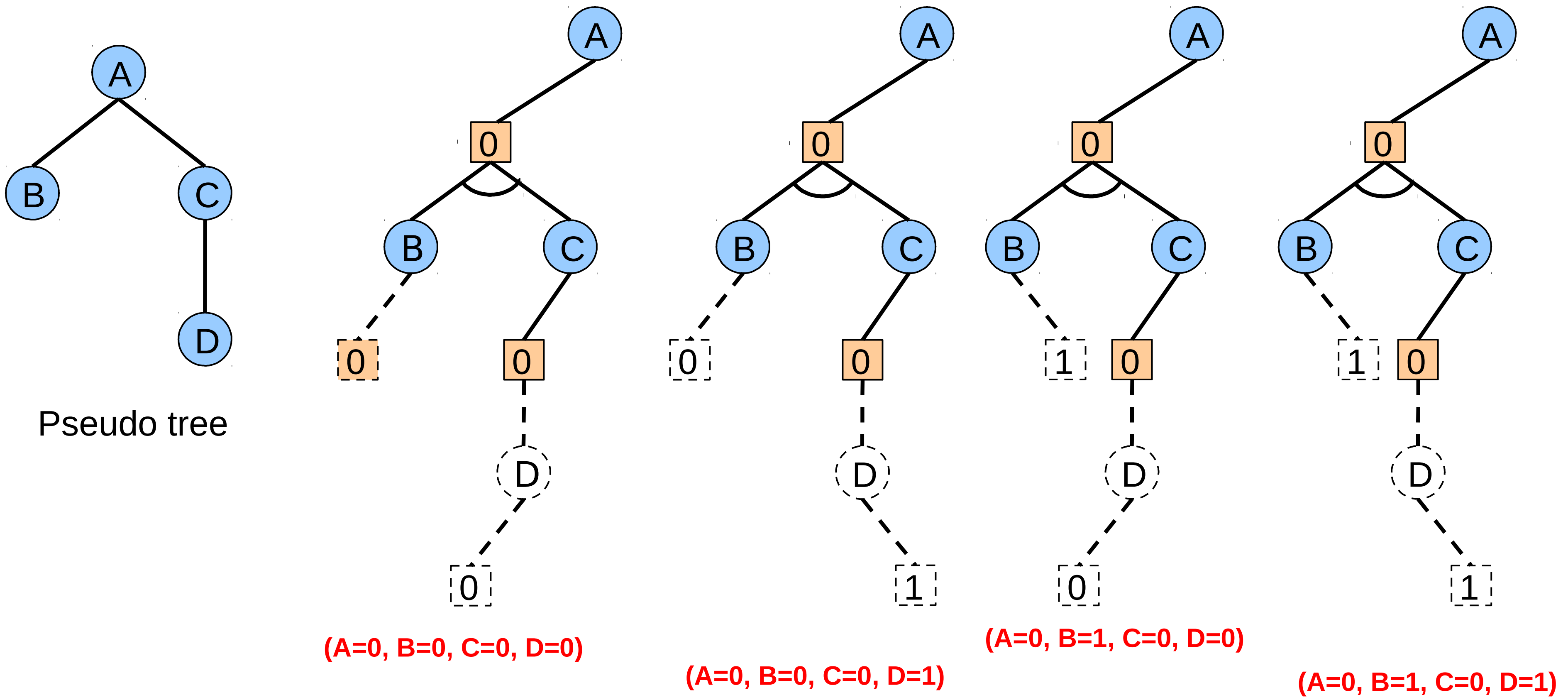
(lower bound)

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

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

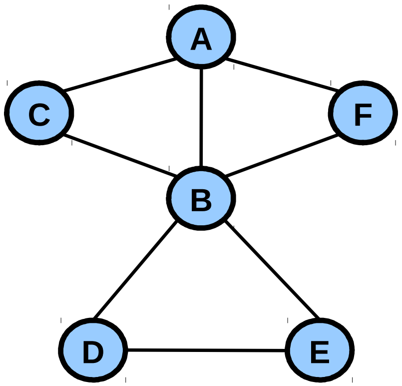
(UB) Upper Bound = best solution so far

Partial Solution Tree



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

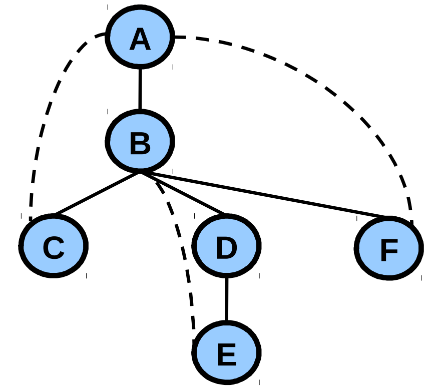
Exact Evaluation Function

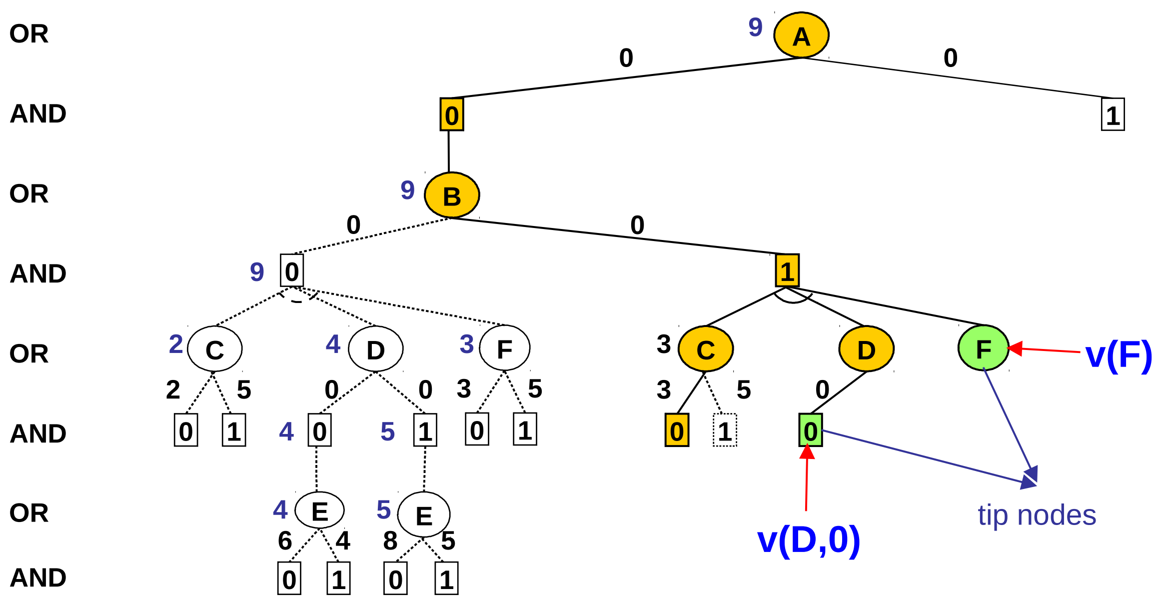


A	B	C	$f_1(ABC)$
0	0	0	2
0	0	1	5
0	1	0	3
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	2

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

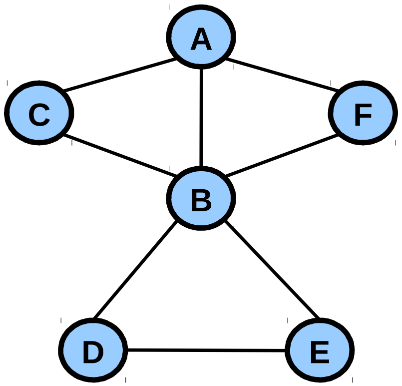
B	D	E	$f_3(BDE)$
0	0	0	6
0	0	1	4
0	1	0	8
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	4





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

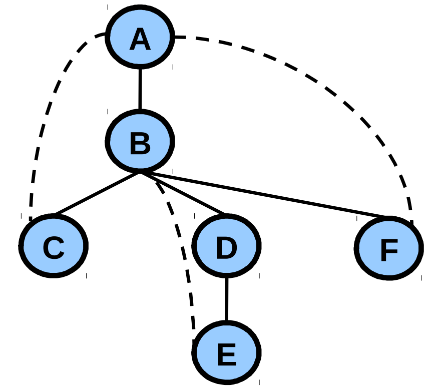
Exact Evaluation Function

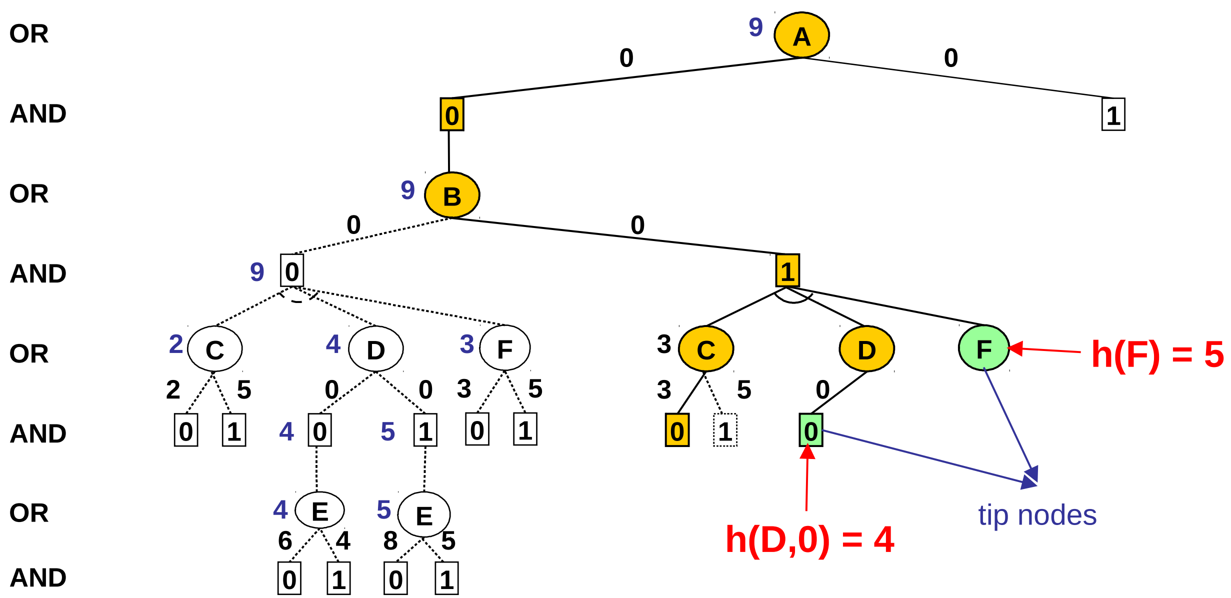


A	B	C	$f_1(ABC)$
0	0	0	2
0	0	1	5
0	1	0	3
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	2

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

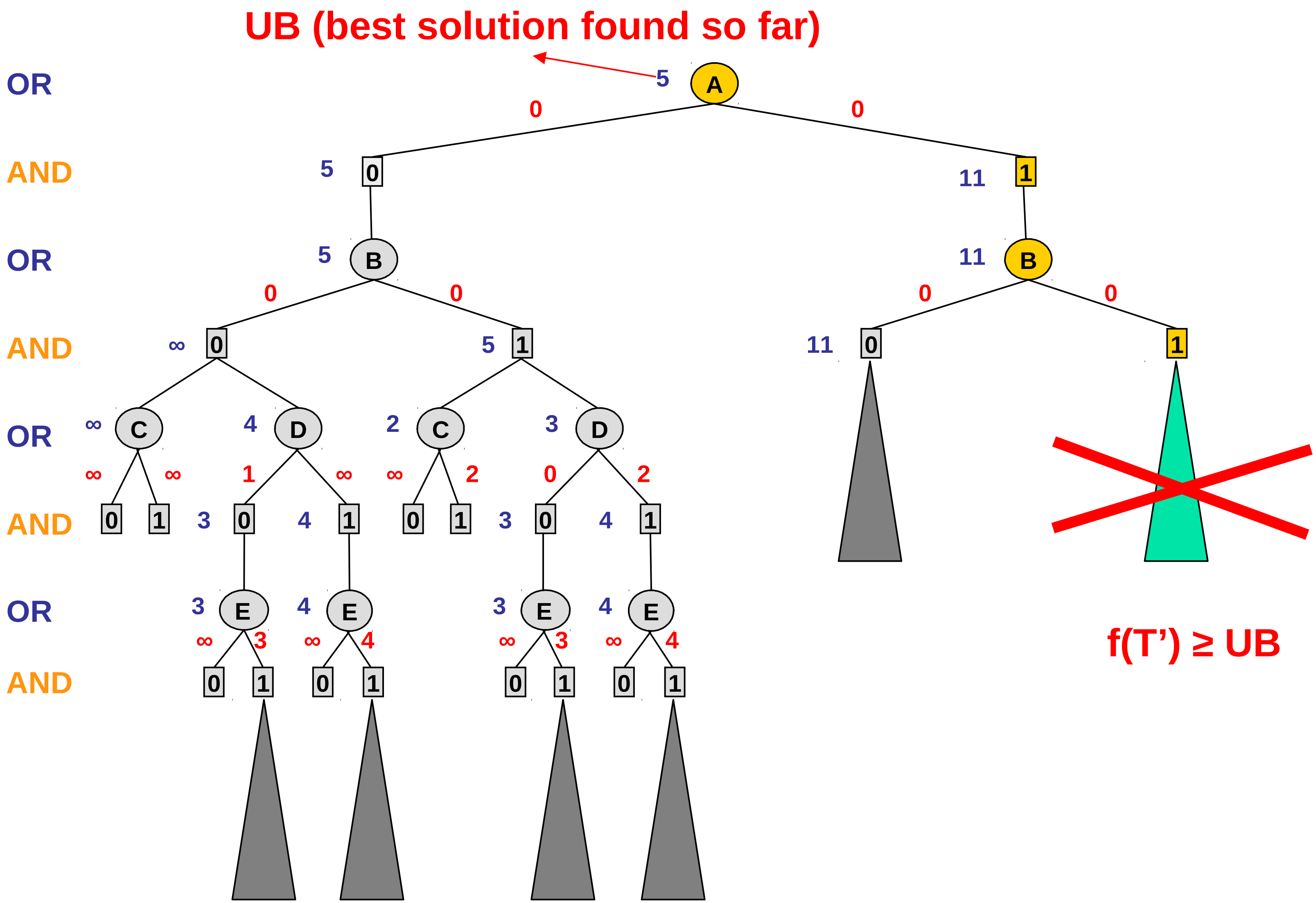
B	D	E	$f_3(BDE)$
0	0	0	6
0	0	1	4
0	1	0	8
0	1	1	5
1	0	0	9
1	0	1	3
1	1	0	7
1	1	1	4





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

AND/OR Branch and Bound Search



AND/OR Branch and Bound (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 $f(T') \geq UB$
  - Generate successors of the tip node n
- UPDATE (bottom-up)
  - Update value of the parent p of n
    - OR nodes: minimization
    - AND nodes: summation

[Marinescu and Dechter, 2005; 2009]

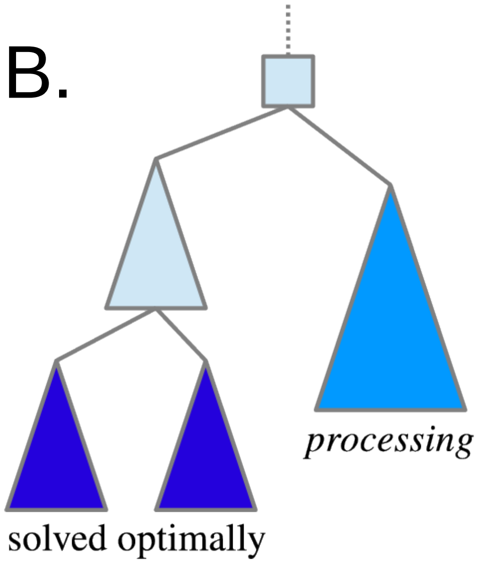
AND/OR Branch and Bound with Caching

- 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
  - 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]

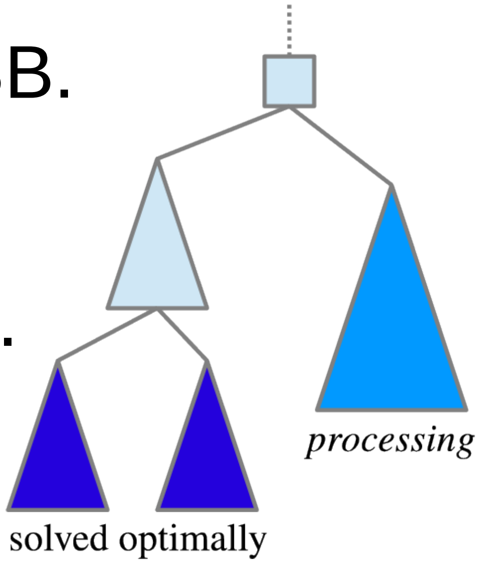
Breadth-Rotating AOBB

- AND/OR decomposition *vs.* depth-first search:
  - Compromises anytime property of AOBB.



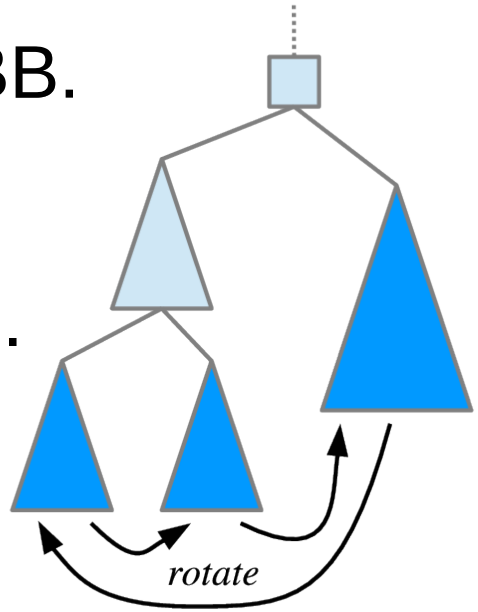
Breadth-Rotating AOBB

- 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.



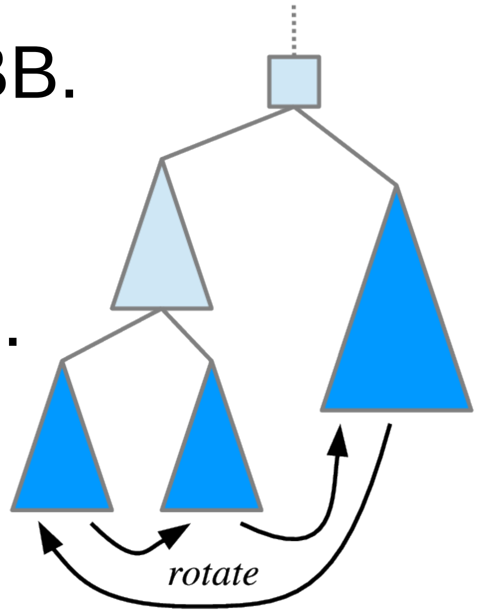
Breadth-Rotating AOBB

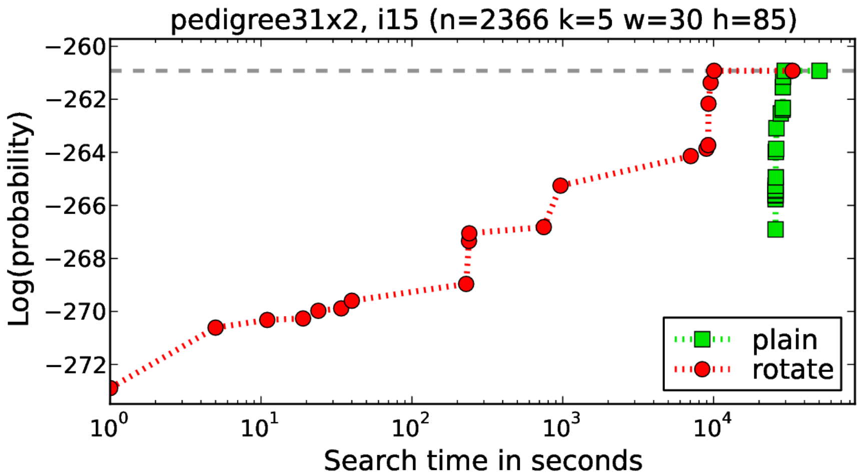
- AND/OR decomposition *vs.* depth-first search:
  - Compromises anytime property of AOBB.
- **Breadth-Rotating AOBB**:
  - Combined breadth/depth-first schedule.
  - Maintains depth-first complexity.
  - Superior experimental results.

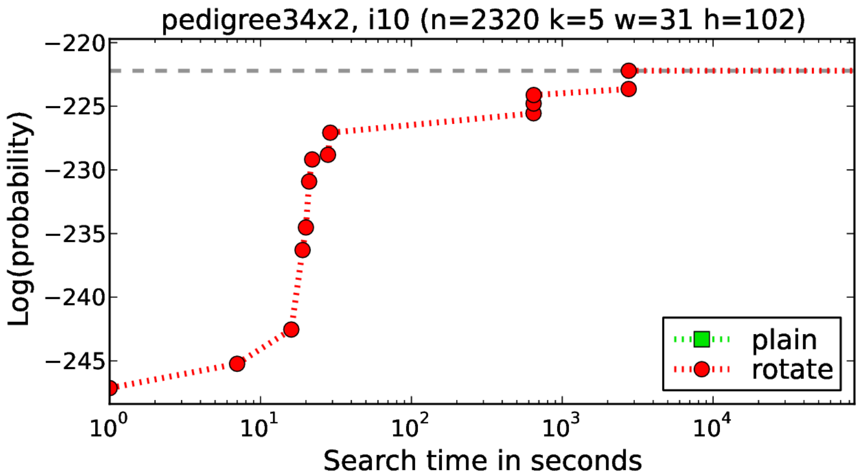


Breadth-Rotating AOBB

- 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.

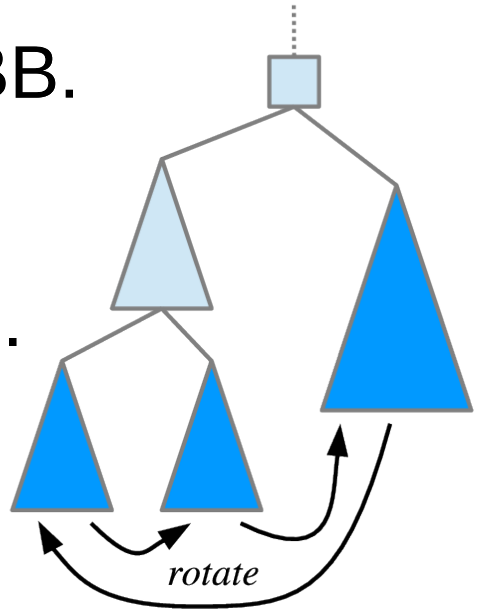






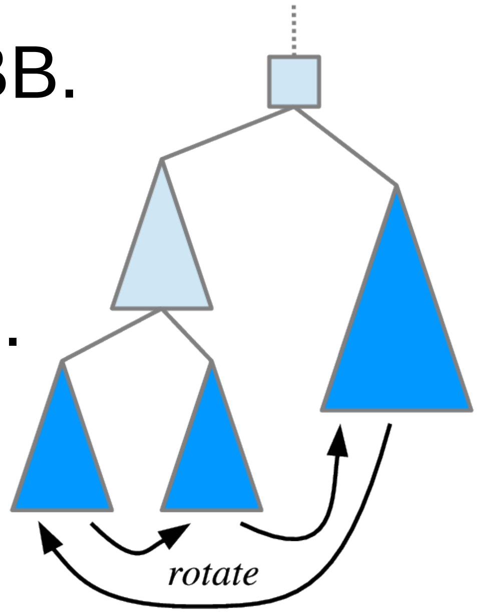
Breadth-Rotating AOBB

- 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.



Breadth-Rotating AOBB

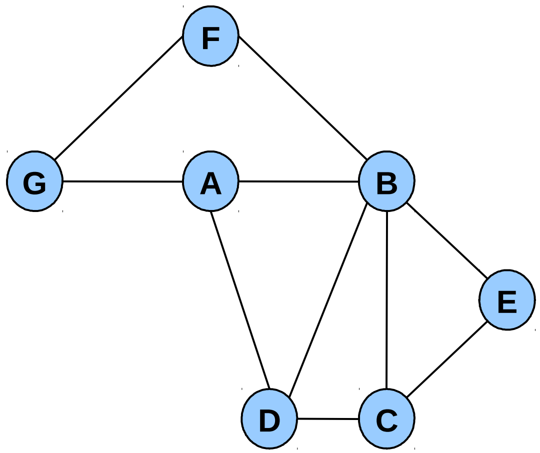
- AND/OR decomposition *vs.* depth-first search:
  - Compromises anytime property of AOBB.
- **Breadth-Rotating AOBB**:
  - Combined breadth/depth-first schedule.
  - Maintains depth-first complexity.
  - Superior experimental results.
- Won PASCAL'11 Inference Challenge MPE track.

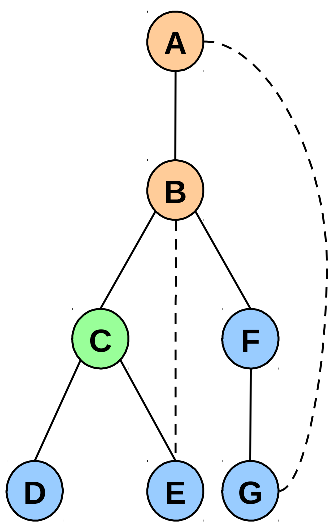


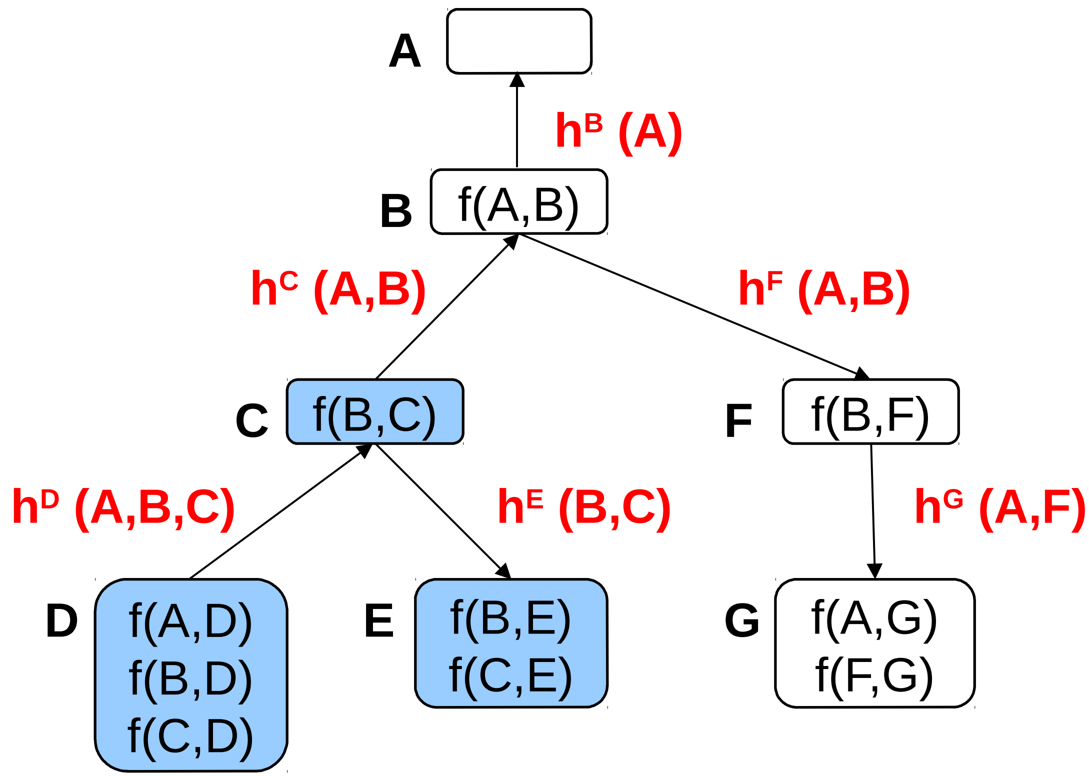
Mini-Bucket Heuristics for AND/OR Search

- 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

Bucket Elimination





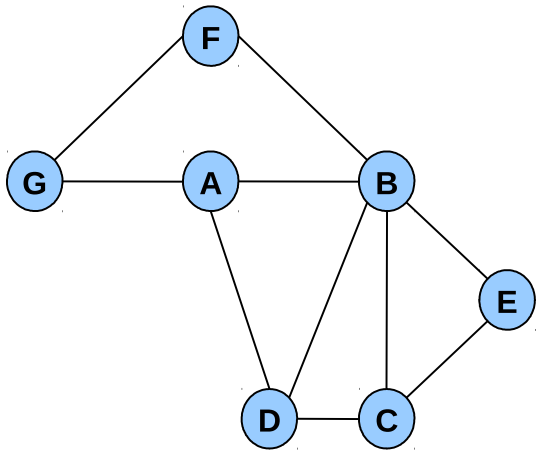


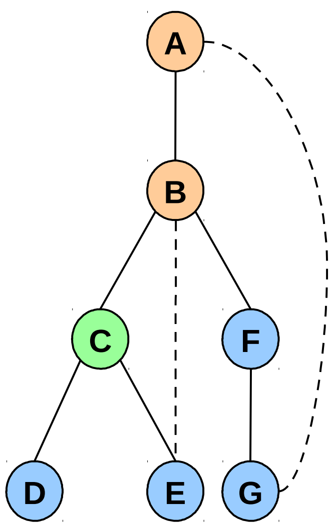
Exact evaluation of (A=a, B=b) below C:

$h^*(a, b, C) = h^D(a, b, C) + h^E(b, C)$

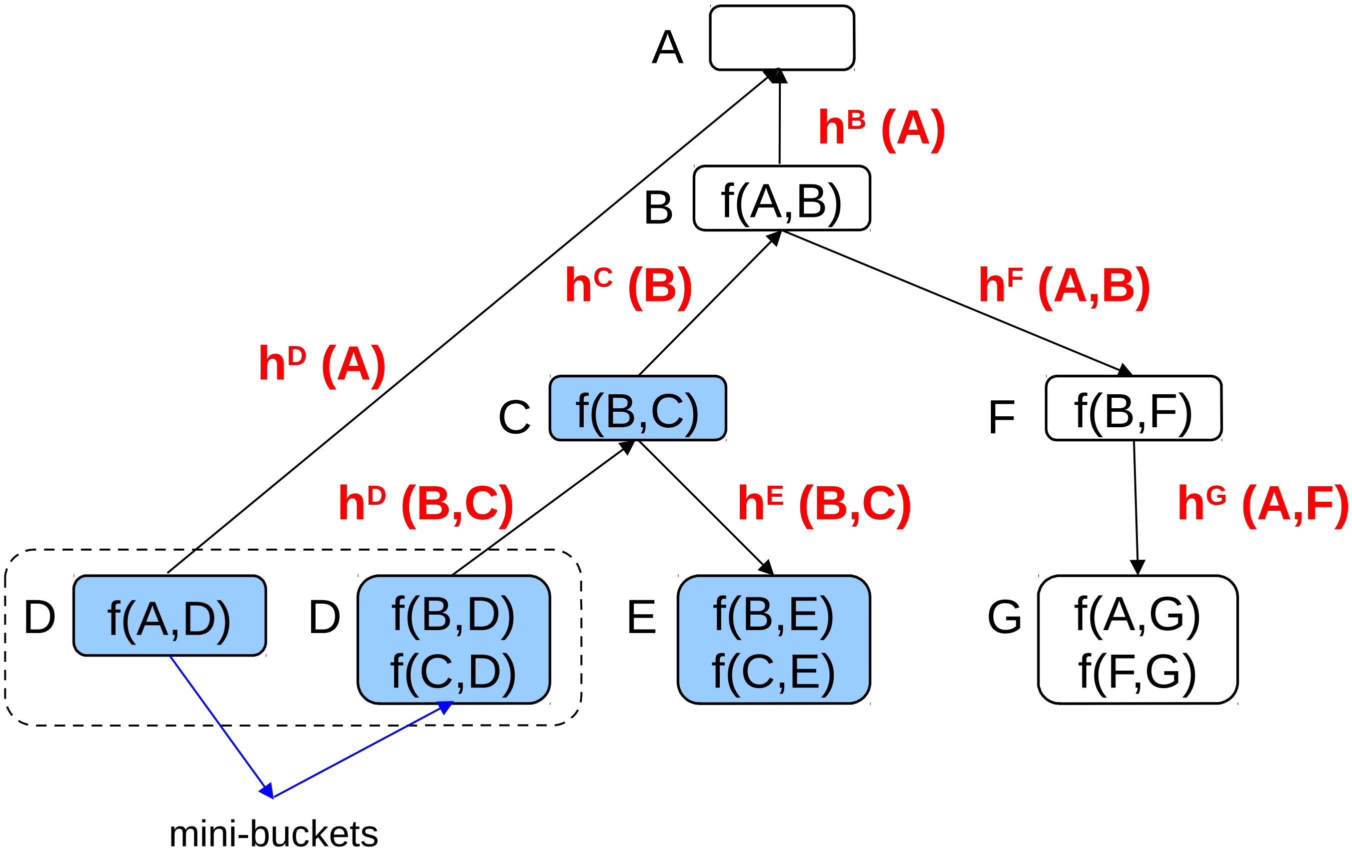
Ordering: (A, B, C, D, E, F, G)

Static Mini-Bucket Heuristics



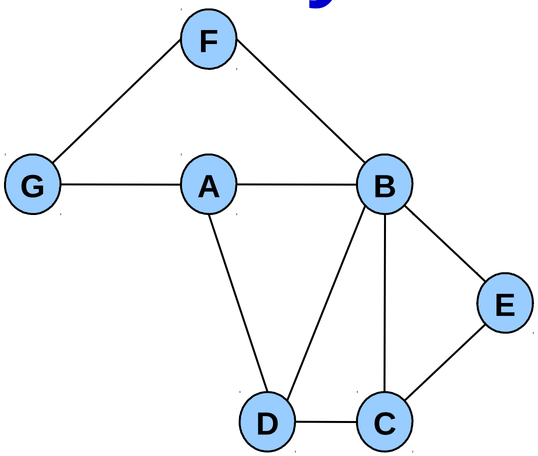


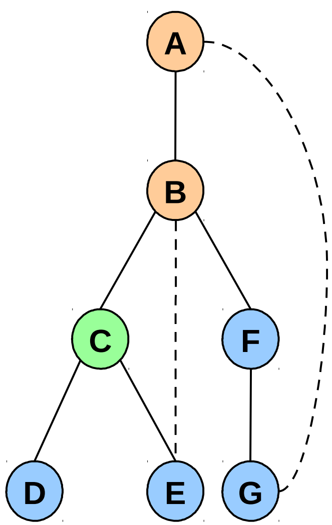
Ordering: (A, B, C, D, E, F, G)

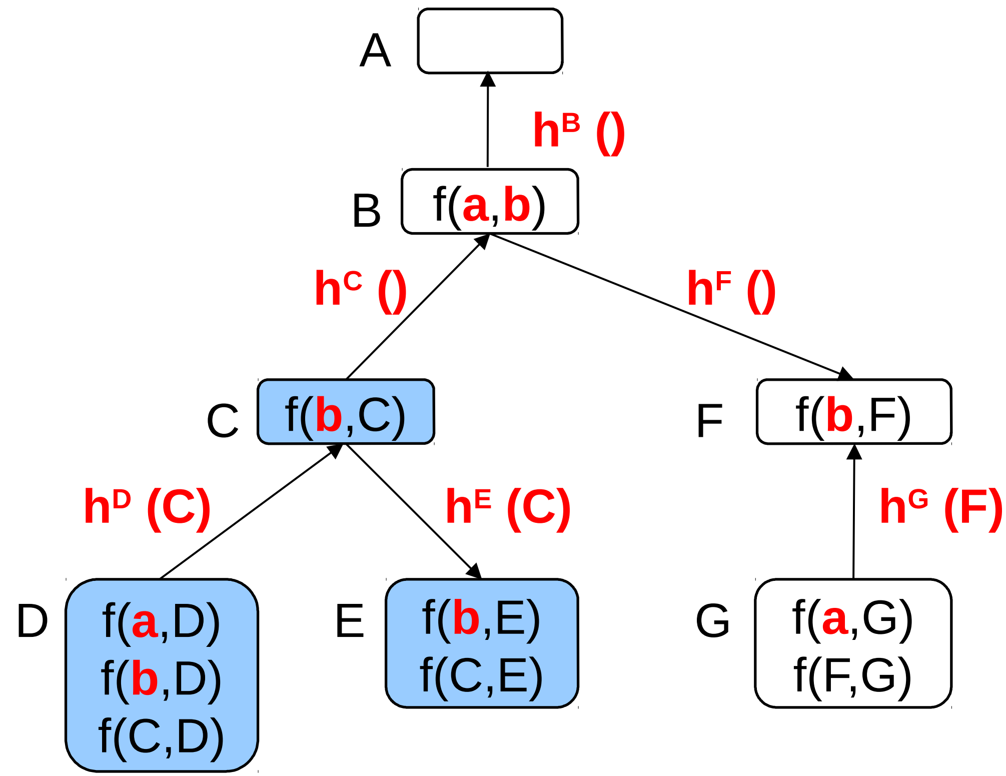


$$h(a, b, C) = h^D(a) + h^D(b, C) + h^E(b, C)$$
$$\leq h^*(a, b, C)$$

Dynamic Mini-Bucket Heuristics







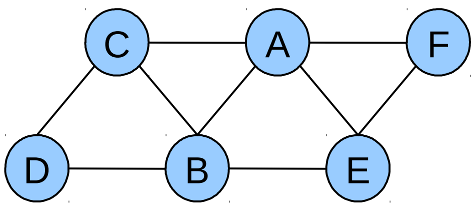
Ordering: (A, B, C, D, E, F, G)

$$h(a, b, C) = h^D(C) + h^E(C)$$
$$= h^*(a, b, C)$$

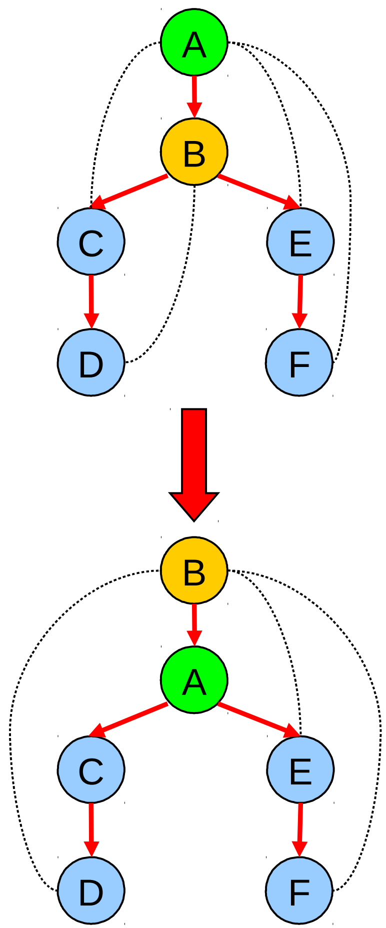
Dynamic Variable Orderings

- Variable ordering heuristics
  - Semantic-based
    - Aim at shrinking the size of the search space based on context and current value assignments
      - e.g., min-domain, min-dom/wdeg, min reduced cost
  - Graph-based
    - Aim at maximizing the problem decomposition
      - e.g., pseudo tree arrangement

Partial Variable Orderings (PVO)



Primal graph



Variable Groups/Chains:

- {A,B}
- {C,D}
- {E,F}

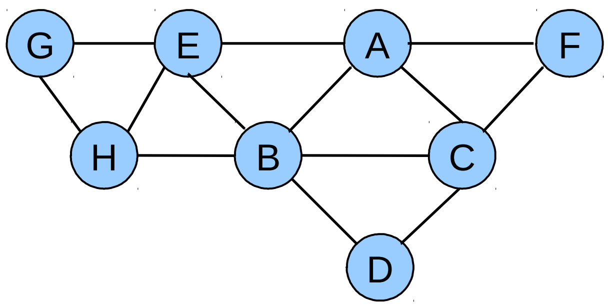
Instantiate {A,B}
before {C,D} and {E,F}

*{A,B} is a separator/chain

Variables on **chains**
in the pseudo tree
can be instantiated
dynamically, based
on some semantic
ordering heuristic

* Similar idea is exploited by **BTD** (Backtracking with Tree Decomposition)
[Jegou and Terrioux, 2004]

Full Dynamic Variable Ordering (DVO)

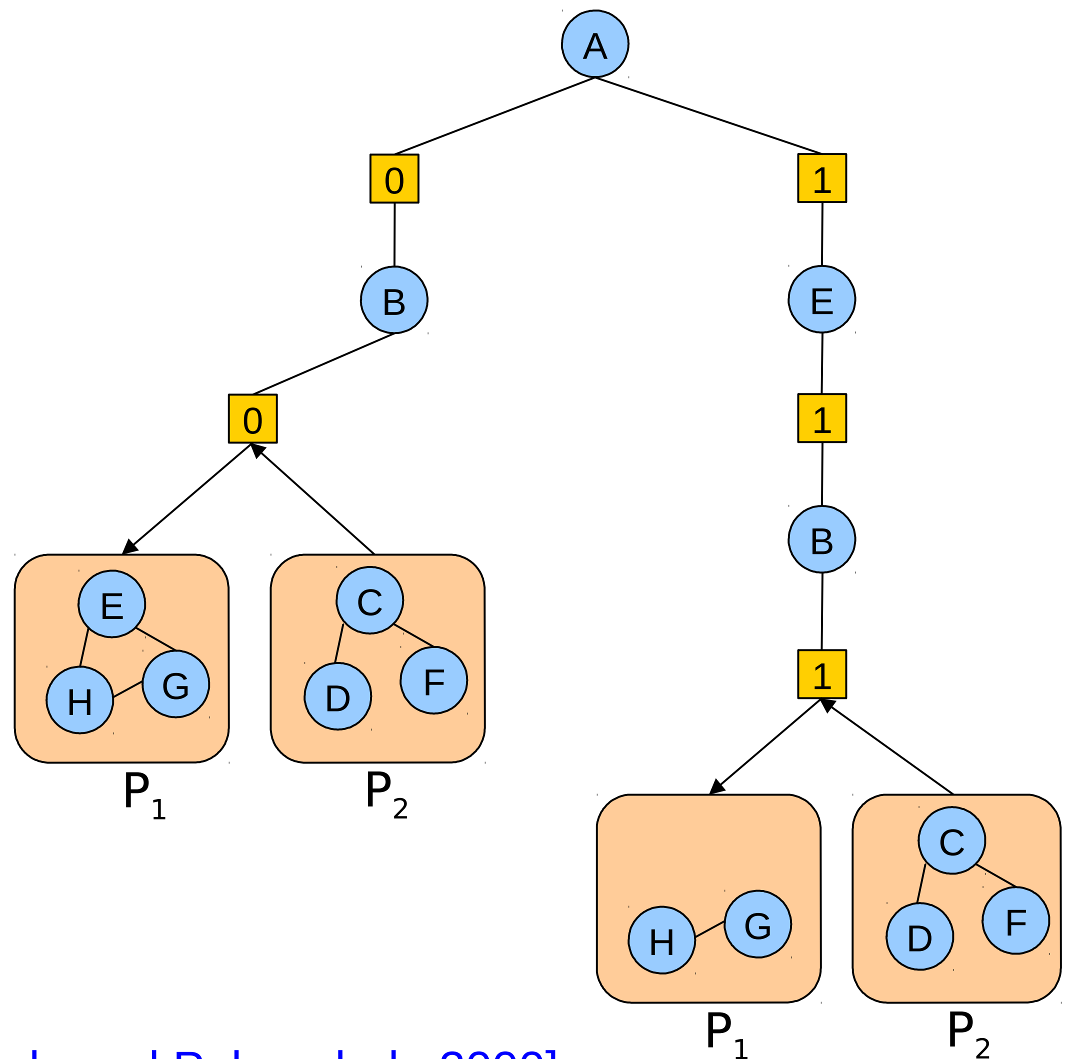


Domains $D_A=\{0,1\}$ $D_B=\{0,1,2\}$
$D_E=\{0,1,2,3\}$
$D_C=D_D=D_F=D_G=D_H=D_E$

Cost functions

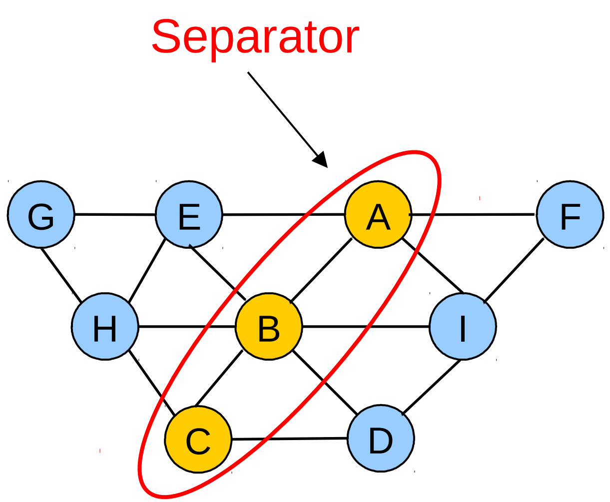
A	B	f(AB)
0	0	**3**
0	1	**8**
0	2	**8**
1	0	**4**
1	1	**0**
1	2	**6**

A	E	f(AE)
0	0	**0**
0	1	**5**
0	2	**1**
0	3	**4**
1	0	**8**
1	1	**8**
1	2	**0**
1	3	**5**

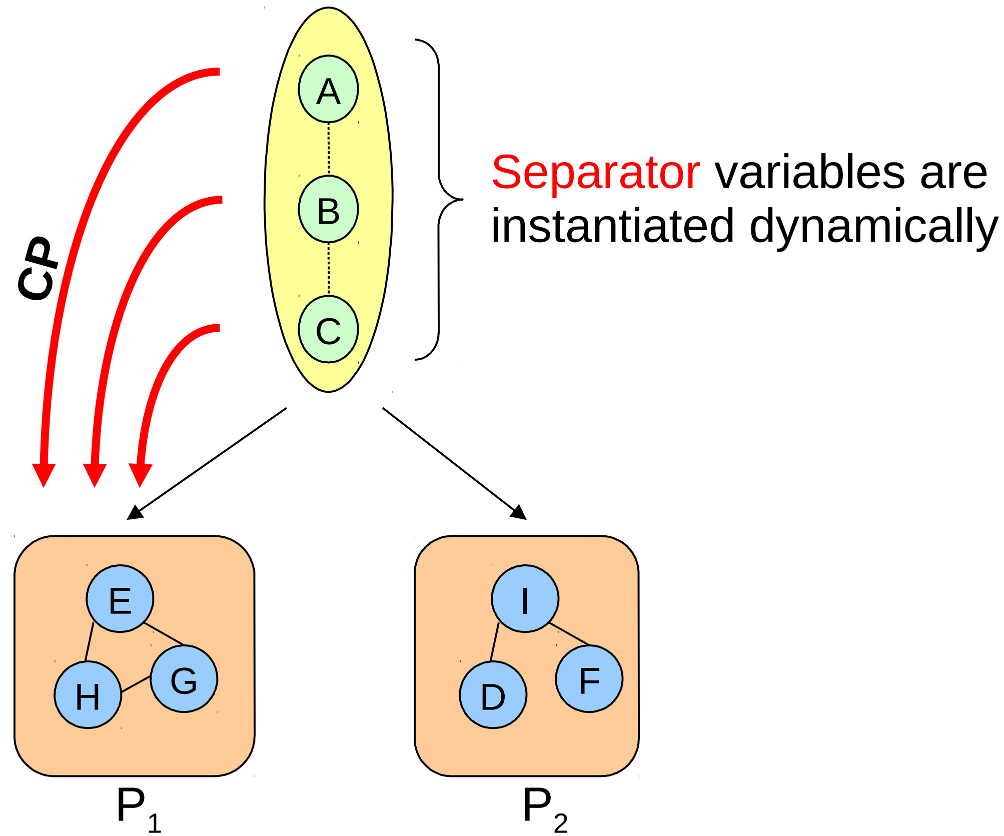


* Similar idea exploited in #SAT [Bayardo and Pehoushek, 2000]

Dynamic Separator Ordering (DSO)



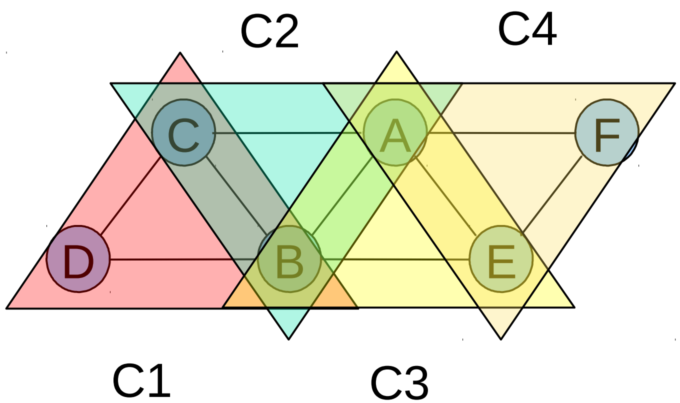
Primal graph

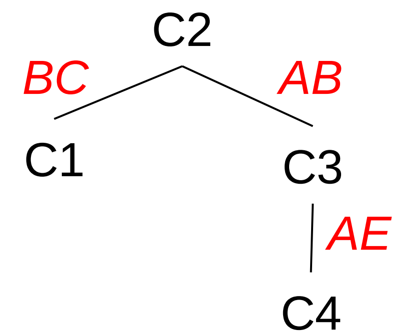


Constraint Propagation may create **singleton** variables in **P1** and **P2** (changing the problem's structure), which in turn may yield smaller separators

* Similar idea exploited in SAT [Li and val Beek, 2004]

Backtrack with Tree Decomposition

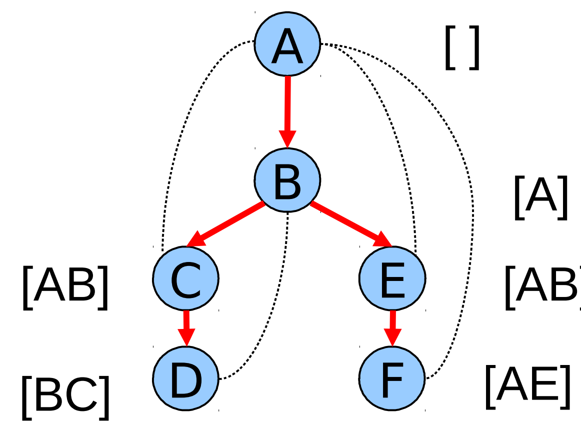




tree decomposition (w=2)

BTD:

- AND/OR graph search (caching on separators)
- Partial variable ordering (dynamic inside clusters)
- Maintaining local consistency



pseudo tree (w=2)

[Jegou and Terrioux, 2004]

Backtrack with Tree Decomposition

- Before the search
  - Merge clusters with a separator size > p
  - Time O(k exp(w*)), Space O(exp(p))
  - More freedom for variable ordering heuristics
- Properties
  - BTD(-1) is Depth-First Branch and Bound
  - BTD(0) solves connected components independently
  - BTD(1) exploits bi-connected components
  - BTD(s) is Backtrack with Tree Decomposition

    (s: largest separator size)

Outline

- Introduction
- Inference
- Bounds and heuristics
- **AND/OR Search**
  - AND/OR Search Spaces
  - 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
  - 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

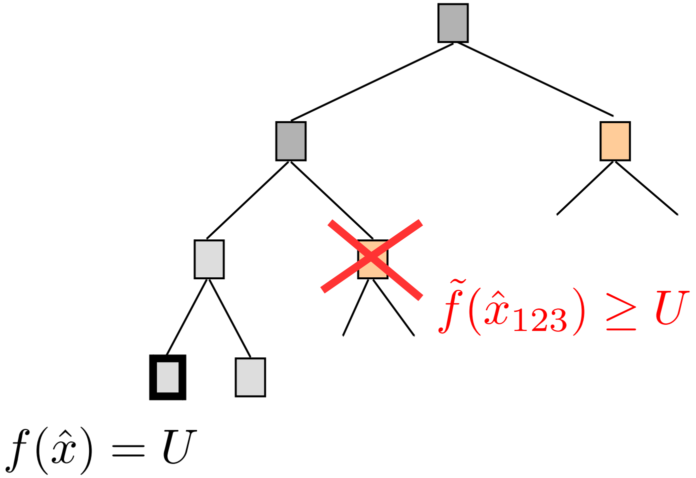
Use heuristic function $\tilde{f}(\hat{x}_p)$ to prune the depth-first search tree

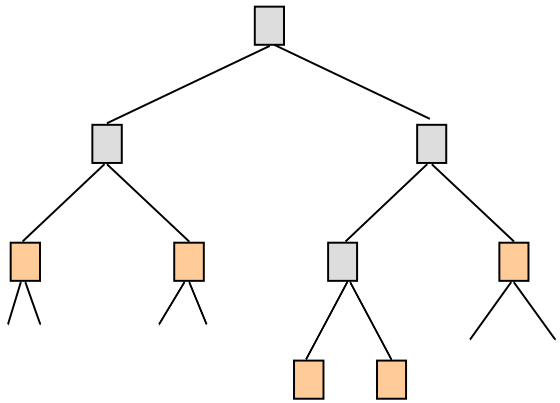
Linear space

2. Best-First Search

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

Needs lots of memory





Best-First Principle

- Best-first search expands first the node with the best heuristic evaluation function among all nodes encountered so far
- Never expands nodes whose cost is beyond the optimal one, unlike depth-first algorithms [Dechter and Pearl, 1985]
- Superior among memory intensive algorithms employing the same heuristic evaluation function

Best-First AND/OR Search (AOBF)

- Maintains the explicated AND/OR search graph in memory
- **Top-Down Step (EXPAND)**
  - Trace down marked connectors from root
    - E.g., **best partial solution tree**
  - Expand a tip node n by generating its successors n'
  - Associate each successor with heuristic estimate h(n')
    - Initialize **q(n) = h(n')** (q-value q(n) is a lower bound on v(n)
- **Bottom-Up Step (UPDATE)**
  - Update node values q(n)
    - OR nodes: minimization
    - AND nodes: summation
  - Mark the most promising partial solution tree from the root
  - Label the nodes as SOLVED:
    - OR node is SOLVED if marked child is SOLVED
    - AND node is SOLVED if all children are SOLVED
- Terminate when root node is SOLVED

[Marinescu and Dechter, 2006; 2009]

AOBF versus AOBB

- **AOBF** with the same heuristic as **AOBB** is likely to expand the smallest search space
  - This translates into significant time savings
- **AOBB** can use far less memory by avoiding for example dead-caches, whereas **AOBF** keeps in memory the explicated search graph
- **AOBB** is anytime, whereas **AOBF** is not

Recursive Best-First AND/OR Search

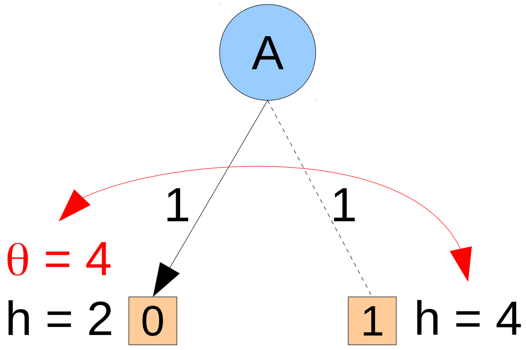
- AND/OR search algorithms (AOBB and AOBF)
  - **AOBB** (depth-first): memory efficient but may explore many suboptimal subspaces
  - **AOBF** (best-first): explores the smallest search space but may require huge memory
- Recursive best-first search for AND/OR graphs
  - Requires limited memory (even linear)
  - Nodes are explored in best-first order
  - Main issue: some nodes will be re-expanded (want to minimize this)

Recursive Best-First AND/OR Search

- Transform best-first search (AO* like) into depth-first search using a threshold controlling mechanism (explained next)
  - Based on Korf's classic RBFS
  - Adapted to the context minimal AND/OR graph
- Nodes are still expanded in best-first order
- Node values are updated in the usual manner based on the values of their successors
  - OR nodes by minimization
  - AND nodes by summation
- Some nodes will be re-expanded
  - Use caching (limited memory) based on contexts
  - Use overestimation of the threshold to minimize node re-expansions

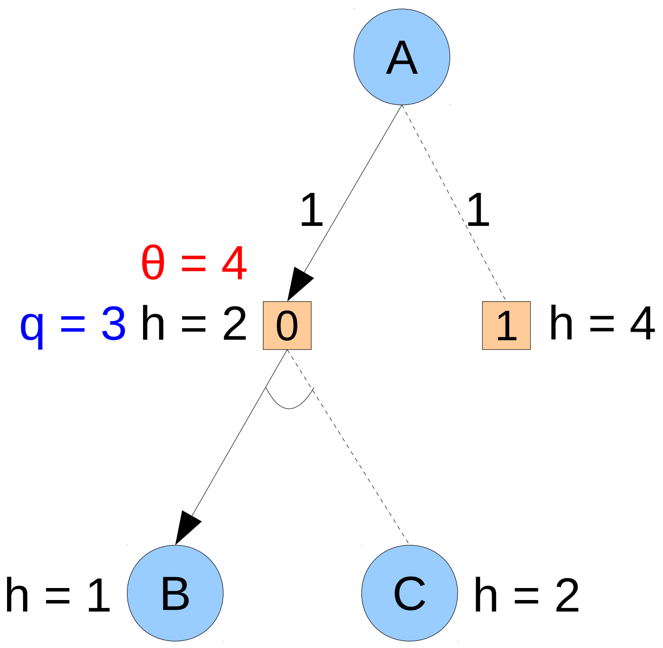
[Kishimoto and Marinescu, 2014]

RBFAOO – Example (1)



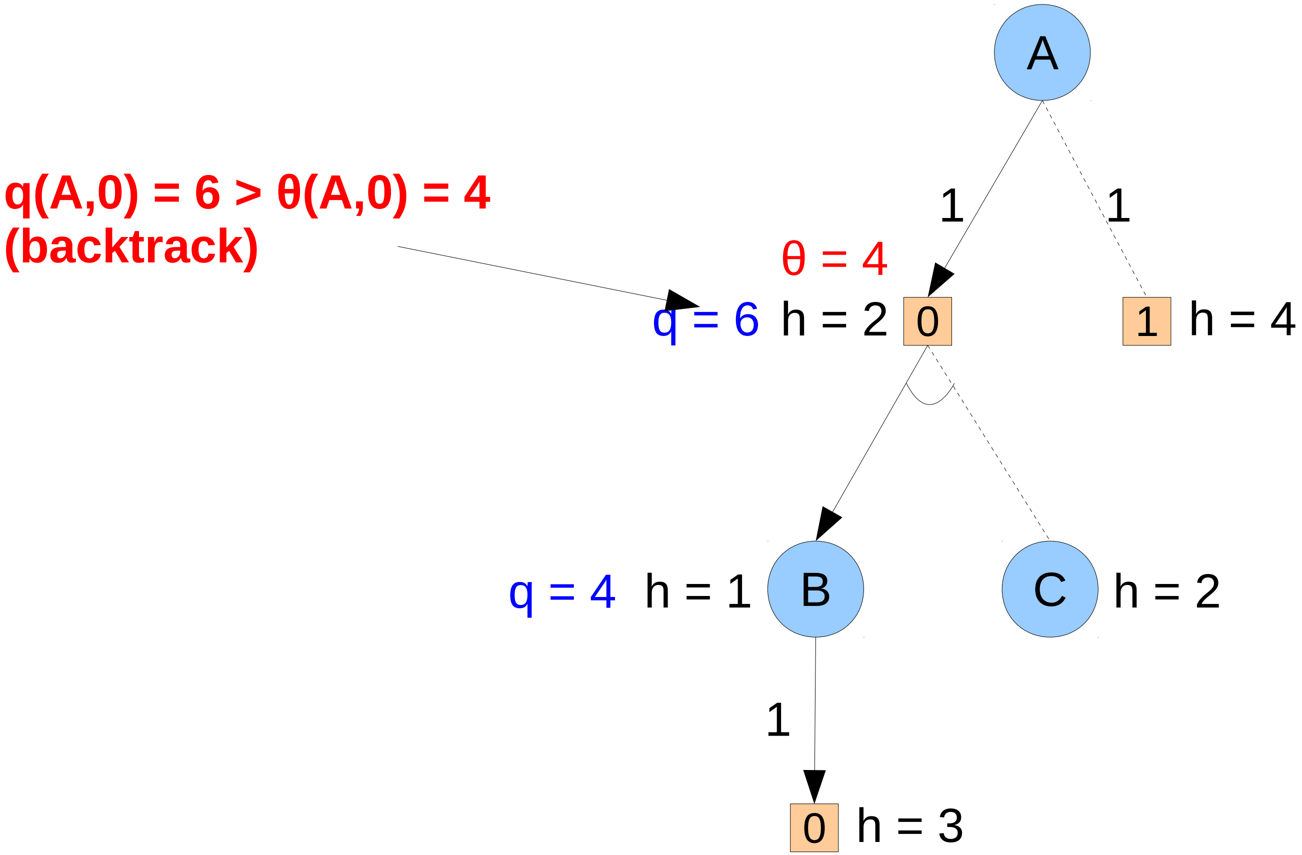
- Expand OR node A by generating its AND successors: (A,0) and (A,1)
- Best successor is (A,0)
- Set threshold θ(A,0) = 4 – indicates next best successor is (A,1)
  - We can backtrack to (A,1) if the updated cost of the subtree below (A,0) exceeds the threshold θ = 4

RBFAOO – Example (2)



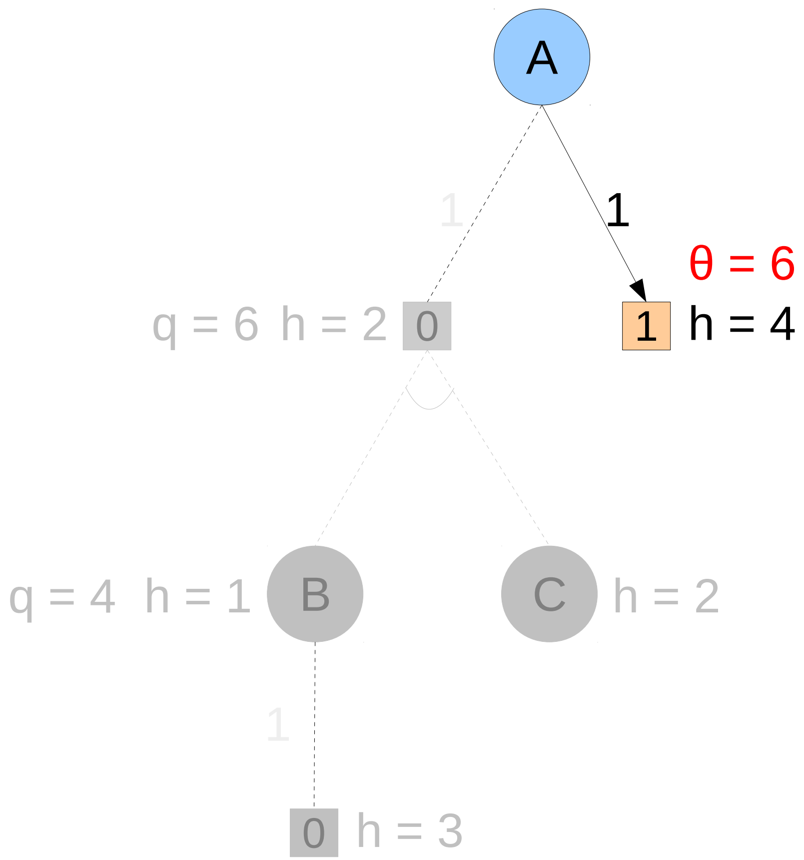
- Expand AND node (A,0) by generating its OR successors: B and C
- Update node value q(A,0) = h(B) + h(C) = 3 – threshold OK

RBFAOO – Example (3)



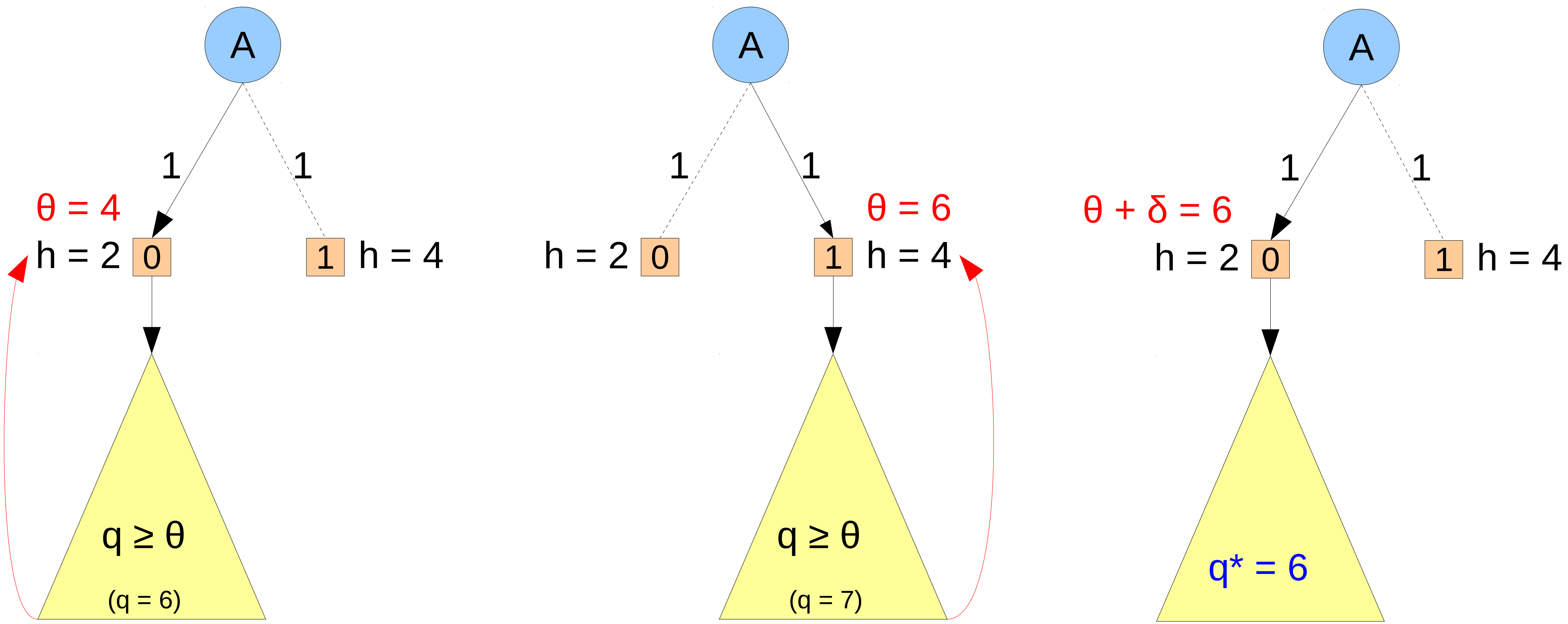
- Expand OR node B by generating its AND successor: (B,0)
- Update node values q(B) = 4 and q(A,0) = 6 – threshold NOT OK

RBFAOO – Example (4)



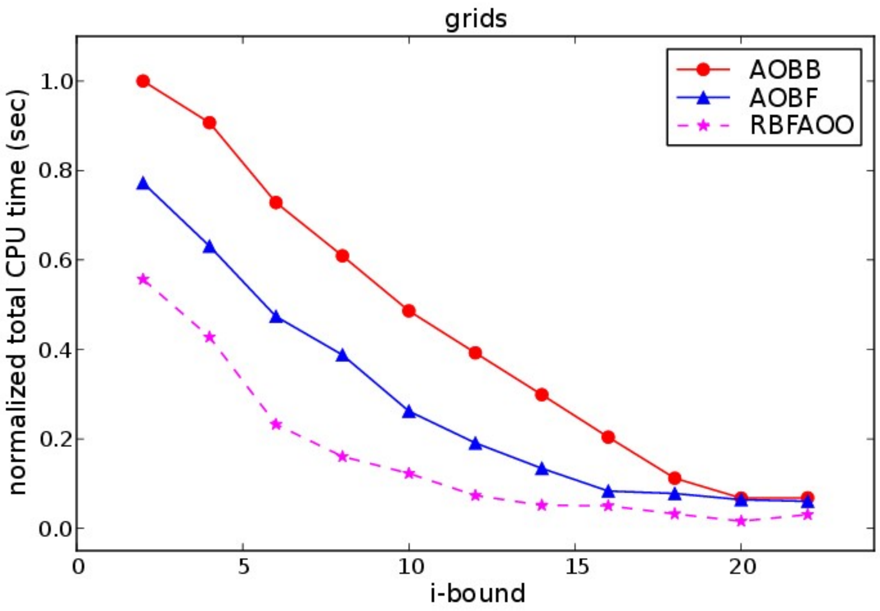
- Backtrack to (A,0) and select next best node (A,1)
- Set threshold θ(A,1) = 6 (updated value of the left subtree)
- Cache (minimize re-expansion) or discard left subtree

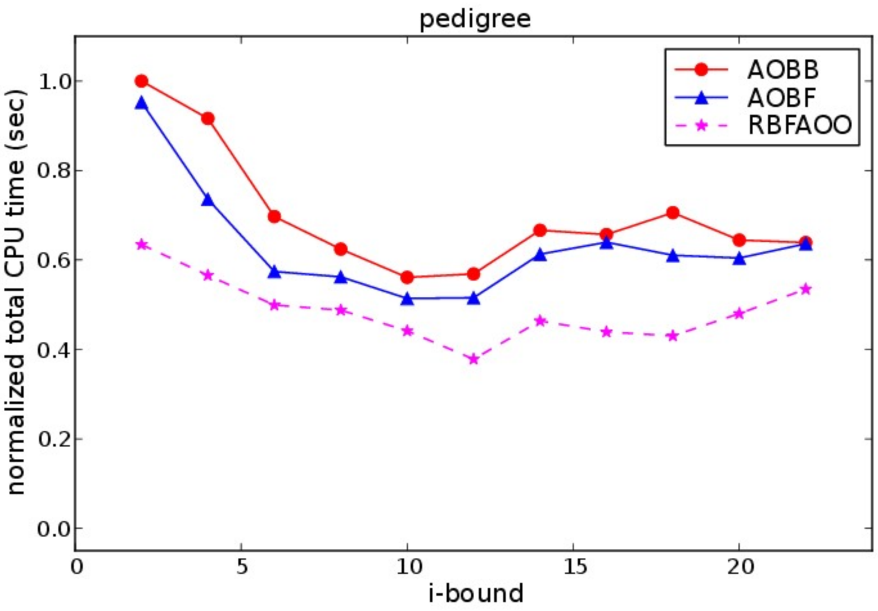
RBFAOO – Overestimation



- Some of the nodes in the subtree below (A,0) may be re-expanded
- Simple overestimation scheme for minimizing the node re-expansions
- Inflate the threshold with some small δ: θ' = θ + δ (δ > 0)
  - In practice, we determine δ experimentally (e.g., δ = 1 worked best)

Empirical Evaluation





Grid and Pedigree benchmarks; Time limit 1 hour.

Outline

- Introduction
- Inference
- Bounds and heuristics
- **AND/OR Search**
  - AND/OR Search Spaces
  - AND/OR Branch and Bound
  - Best-First AND/OR Search
  - Advanced Searches and Tasks
- Exploiting parallelism
- Software

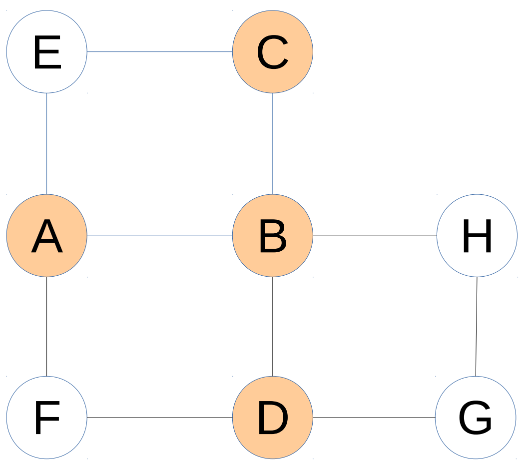
Marginal MAP

- Occurs in many applications involving hidden variables
- Seeks a partial configuration of variables with maximum marginal probability
- Complexity: NP^{PP}-complete
- State-of-the-art is DFS BnB (over the MAP variables)
  - Guided by unconstrained join-tree based upper bounds
- Advances
  - AND/OR Branch and Bound and Best-First AND/OR Search algorithms
  - Heuristics based on Weighted Mini-Buckets
    - WMB-MM: single pass with cost-shifting by moment matching
    - WMB-JG: iterative updates by message passing along the join-graph

Marginal MAP

- Occurs in many applications involving hidden variables
- Seeks a partial configuration of variables with maximum marginal probability
- Complexity: NP^{PP}-complete
- State-of-the-art is DFS BnB (over the MAP variables)
  - Guided by unconstrained join-tree based upper bounds
- Advances
  - AND/OR Branch and Bound and Best-First AND/OR Search algorithms
  - Heuristics based on Weighted Mini-Buckets
    - WMB-MM: single pass with cost-shifting by moment matching
    - WMB-JG: iterative updates by message passing along the join-graph

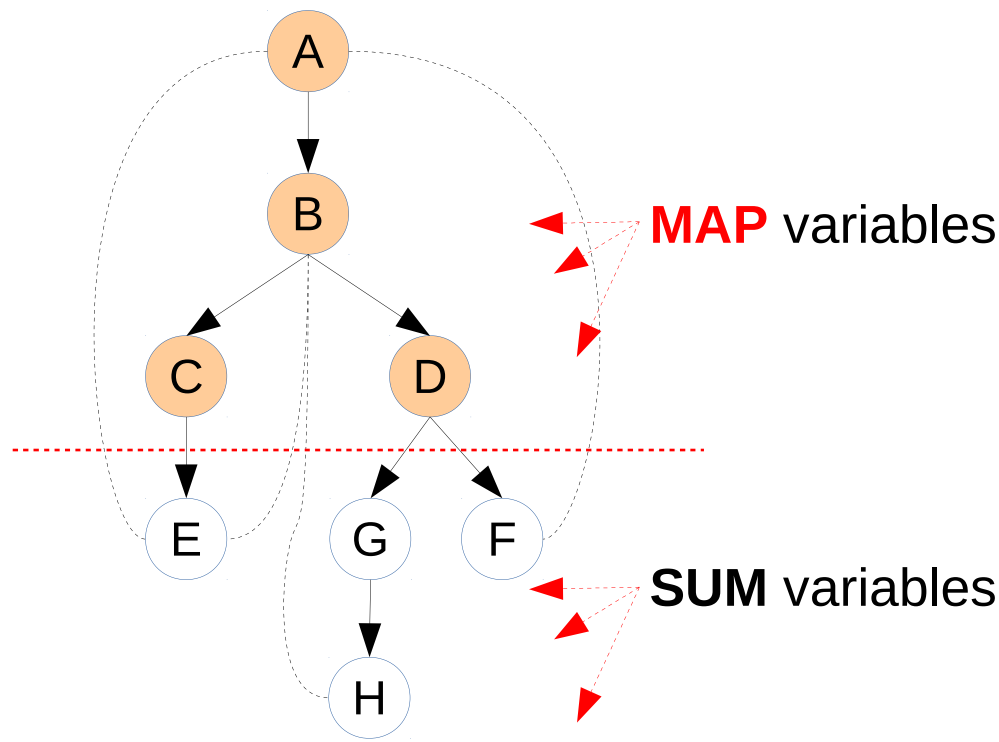
AND/OR Search Space for MMAP



primal graph

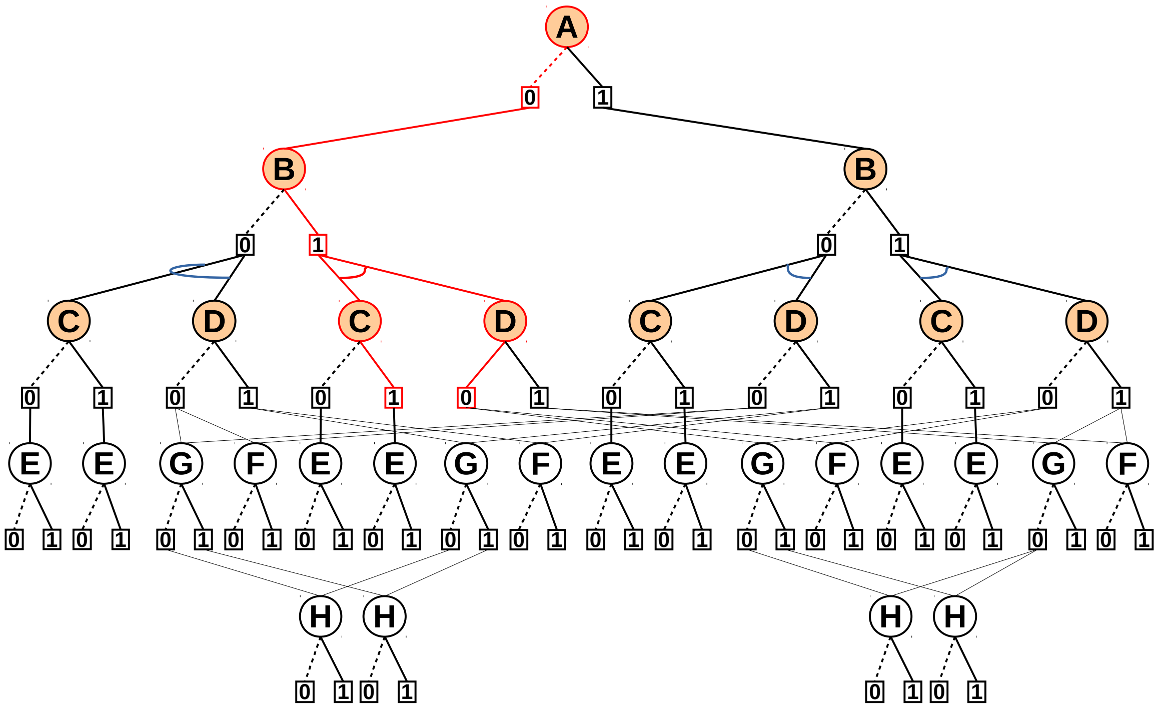
$$X_M = \{A, B, C, D\}$$

$$X_S = \{E, F, G, H\}$$



constrained pseudo tree

AND/OR Search Space for MMAP



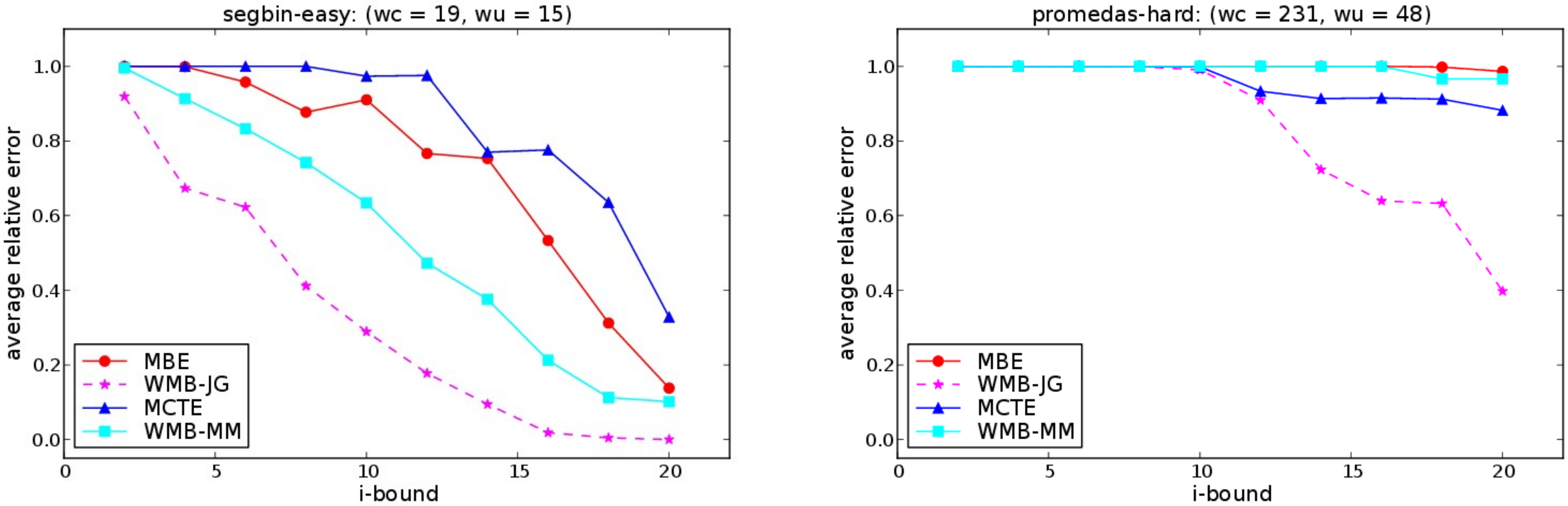
- Node types
  - OR (MAP): max
  - OR (SUM): sum
  - AND: multiplication
- Arc weights
  - derived from input **F**
- Problem decomposition over MAP variables

AND/OR Search Algorithms

- **AOBB: Depth-First AND/OR Branch and Bound**
  - Depth-first traversal of the AND/OR search graph
  - Prune only at OR nodes that correspond to MAP variables
  - Cost of MAP assignment obtained by searching the SUM sub-problem
- **AOBF: Best First AND/OR Search**
  - Best-first (AO*) traversal of the AND/OR space corresponding to the MAP variables
  - SUM subproblem solved exactly
- **RBFAOO: Recursive Best-First AND/OR Search**
  - Recursive best-first traversal of the AND/OR graph
  - For SUM subproblems, the threshold is set to ∞ (equivalent to depth-first search)

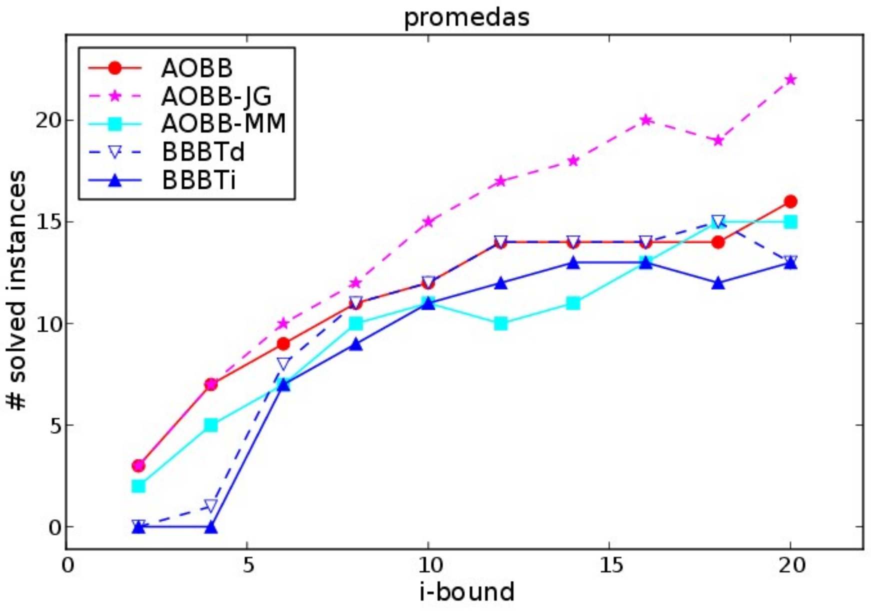
[Marinescu, Dechter and Ihler; 2014; 2015]

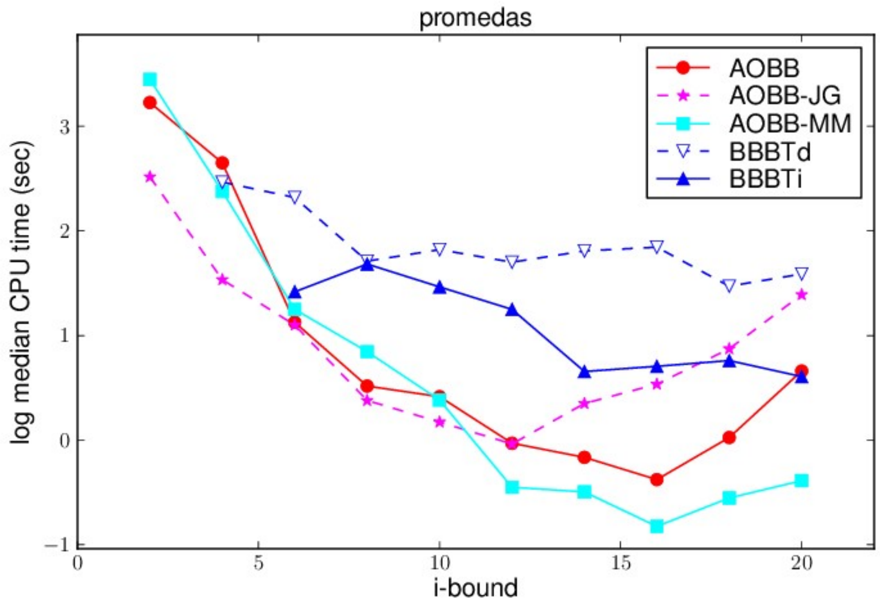
Quality of the Upper Bounds



Average relative error wrt tightest upper bound. 10 iterations for WMB-JG(i).

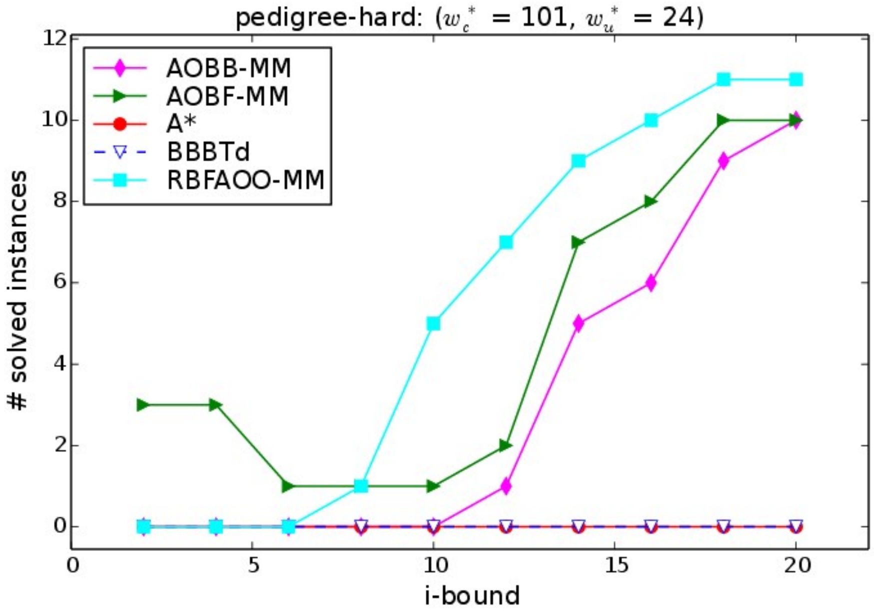
AOBB versus BB

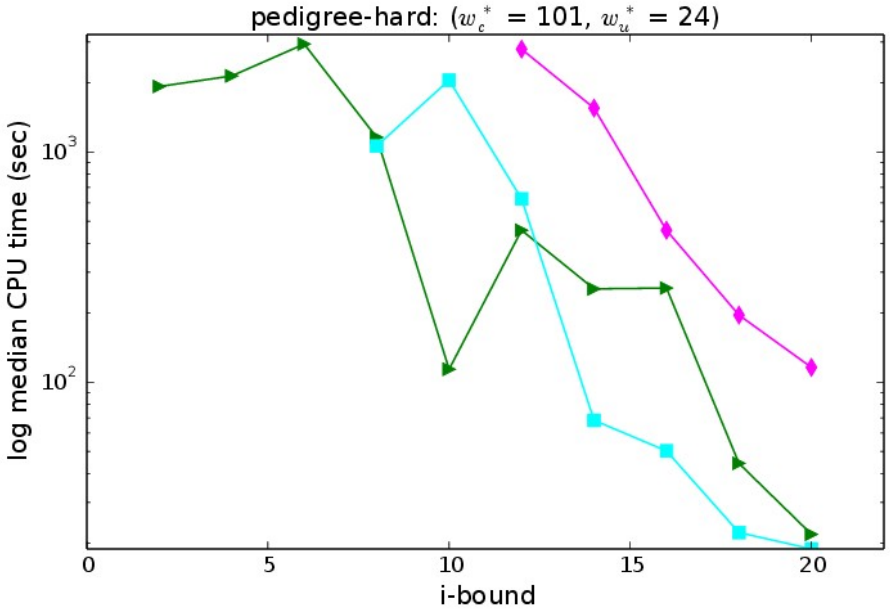




Number of instances solved and median CPU time (sec). 10 iterations for WMB-JG(i).

AOBF/RBFAOO versus AOBB





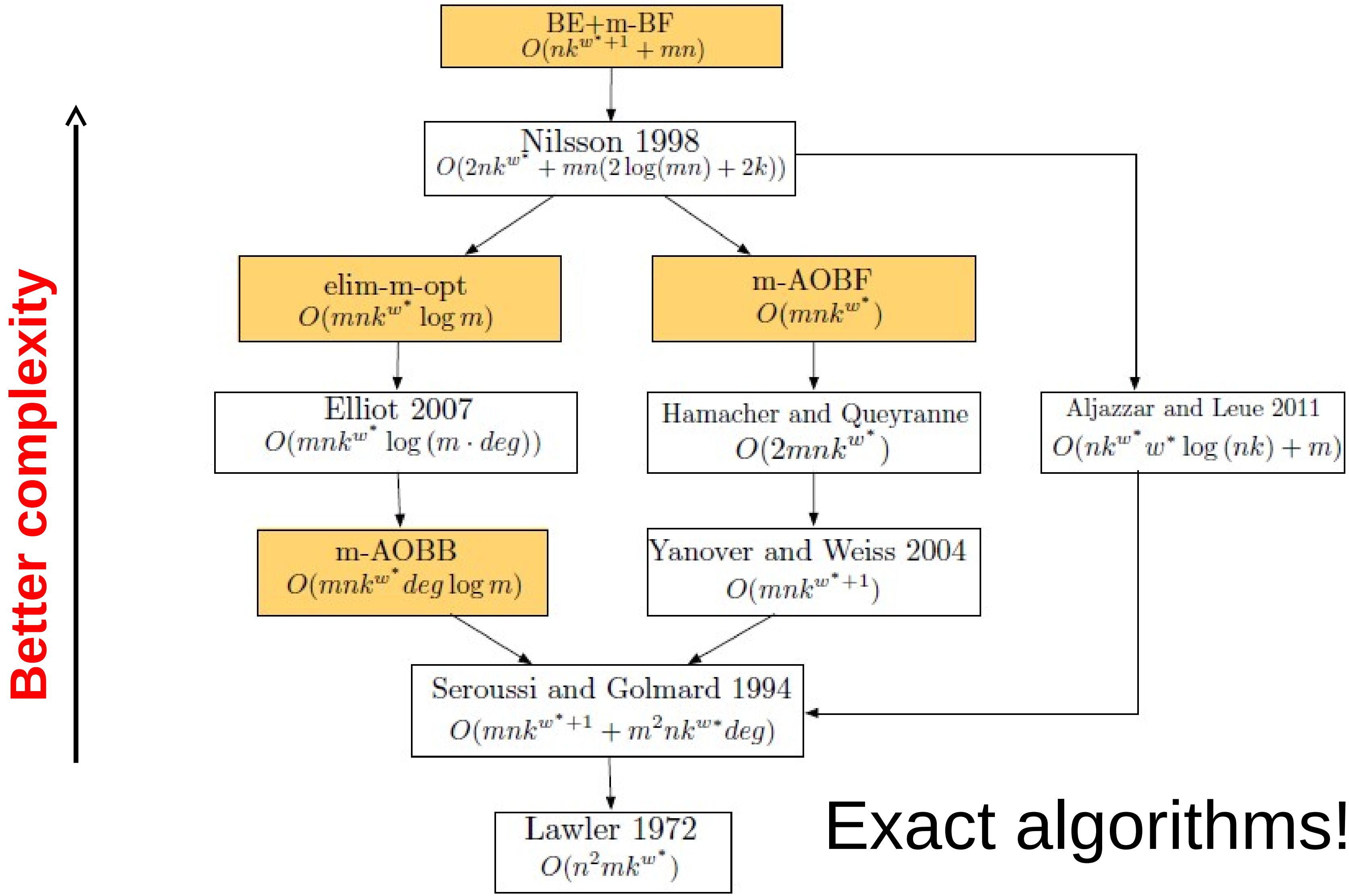
Number of instances solved and median CPU time (sec). Time limit 1 hour.

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, yielding m-AOBB and m-AOBF
- Competitive and often superior to alternative (approximate) approaches based on LP relaxations
  - e.g., [Fromer and Globerson, 2009], [Batra, 2012]

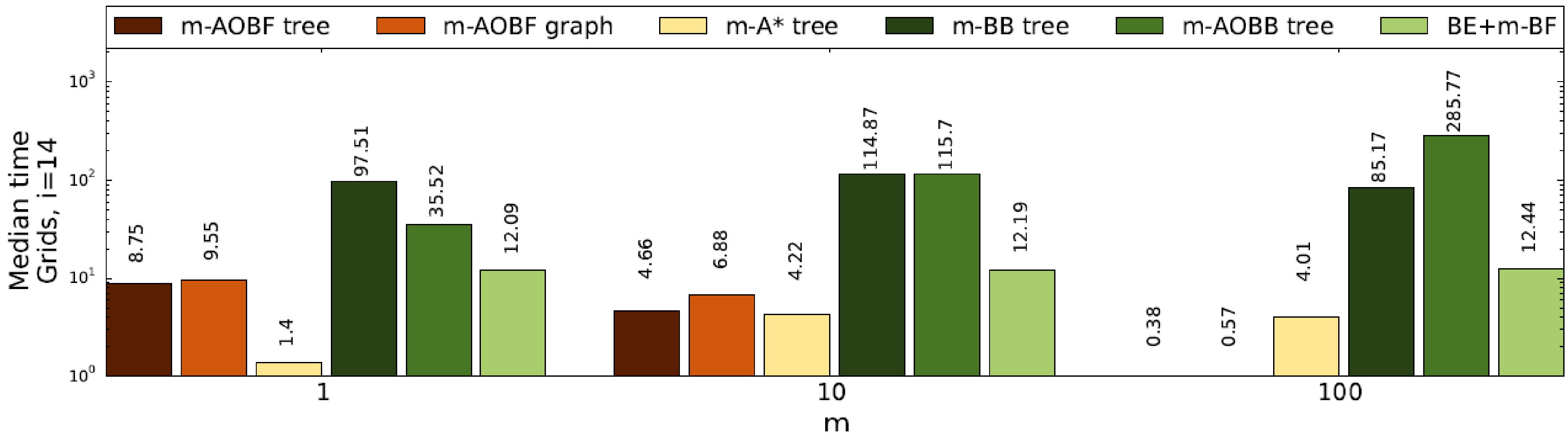
[Dechter, Flerova and Marinescu; 2012]

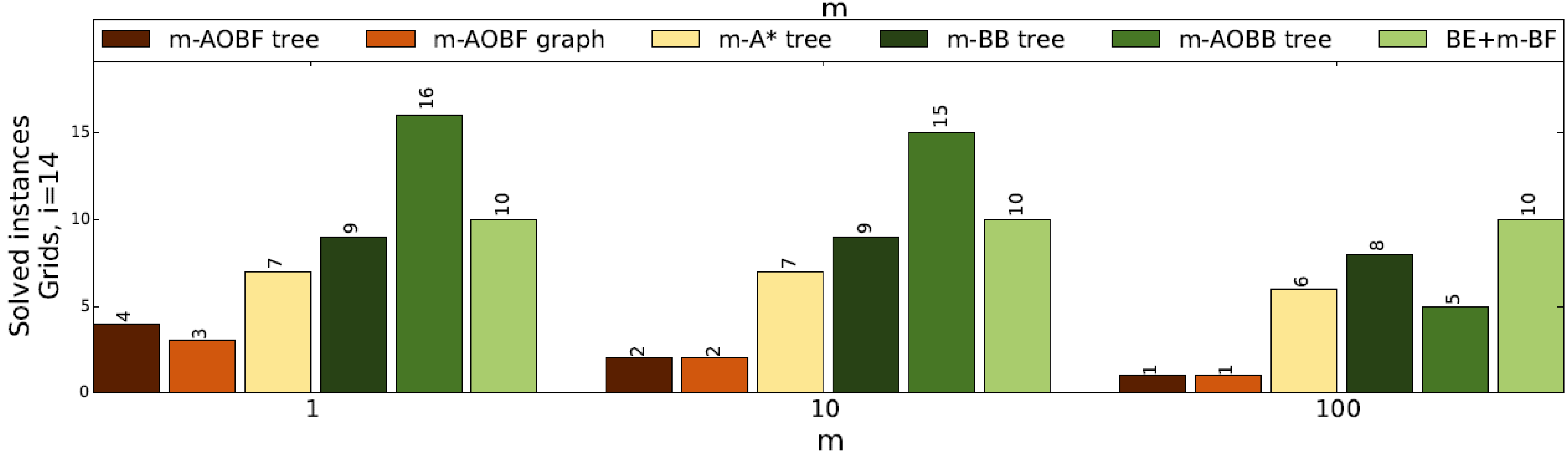
Searching for M Best Solutions



Exact algorithms!

Empirical Evaluation





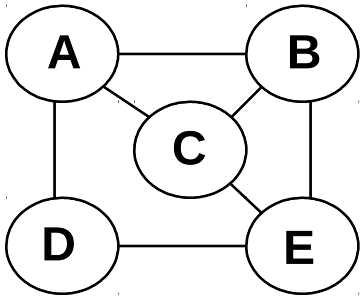
Grid instances; Time limit = 3h; Memory bound = 4 GB

Hybrid of Variable Elimination and Search

- Tradeoff space and time

Search Basic Step: Conditioning

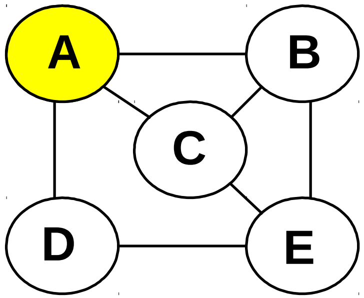
Variable Branching by Conditioning



Search Basic Step: Conditioning

Variable Branching by Conditioning

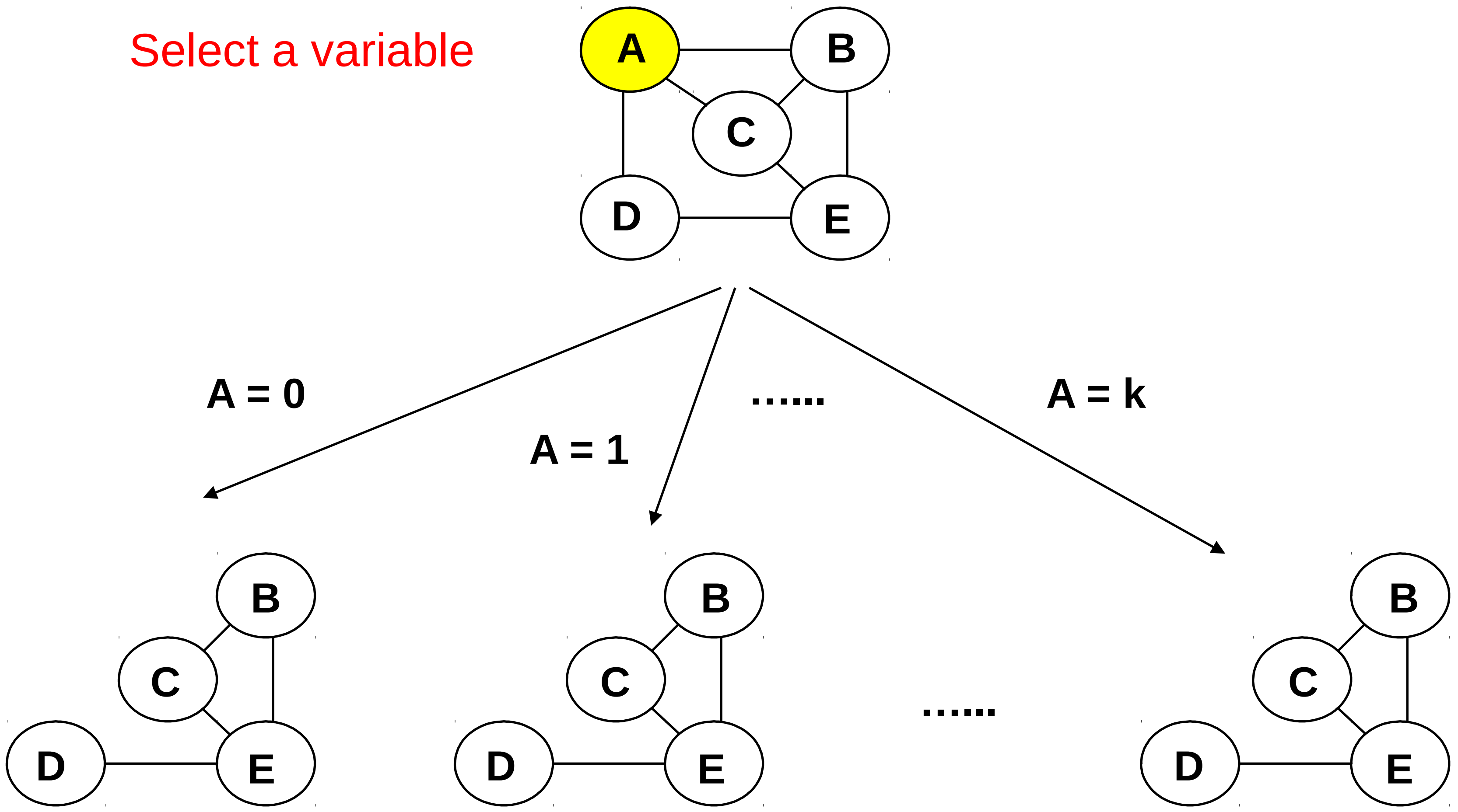
Select a variable



Search Basic Step: Conditioning

Variable Branching by Conditioning

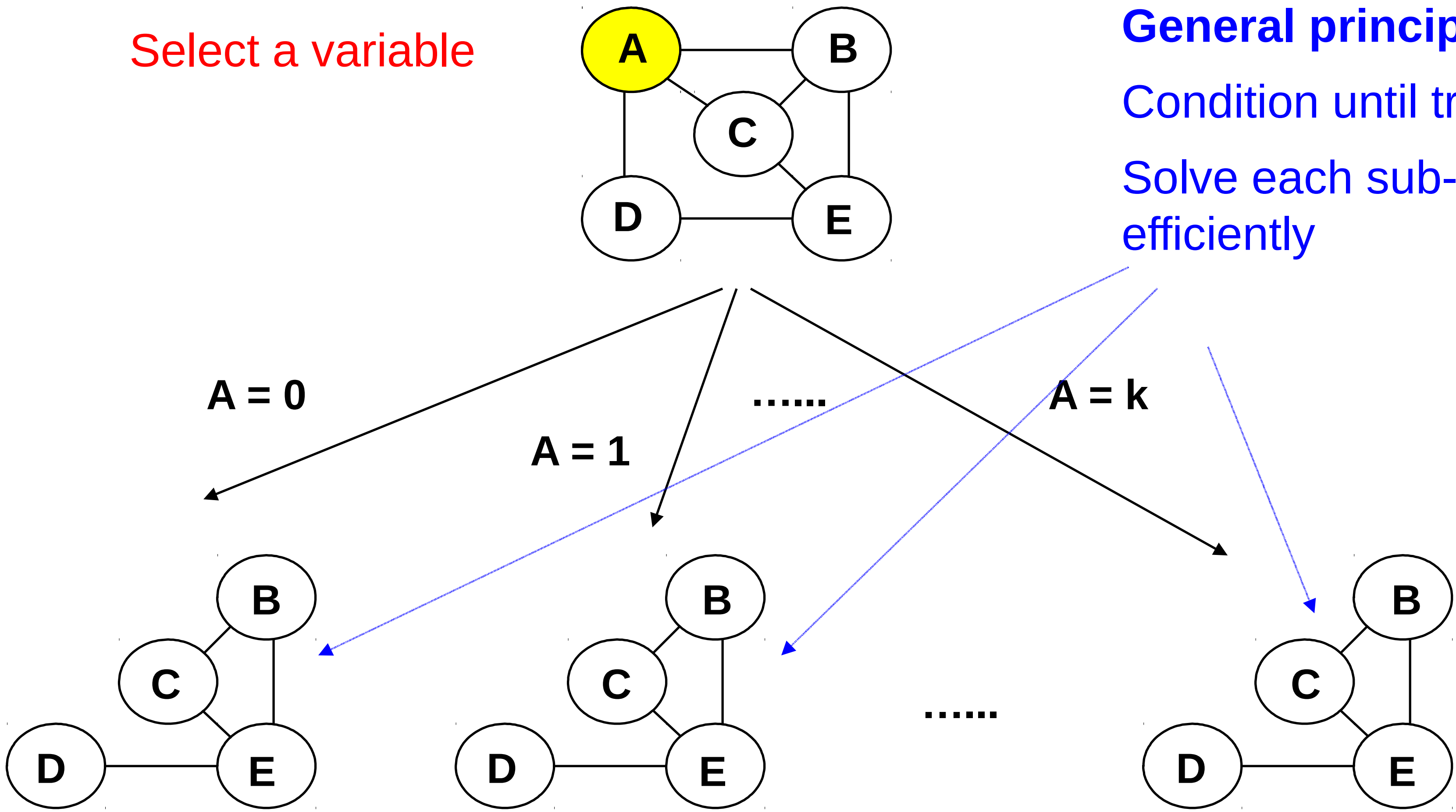
Select a variable



Search Basic Step: Conditioning

Variable Branching by Conditioning

Select a variable



General principle:

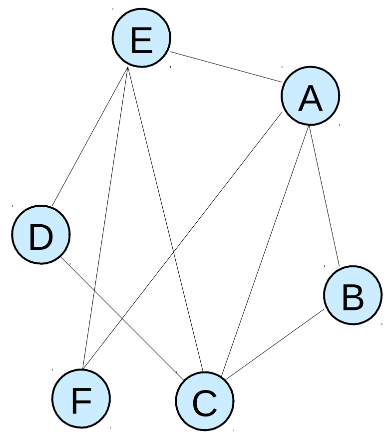
Condition until tractable

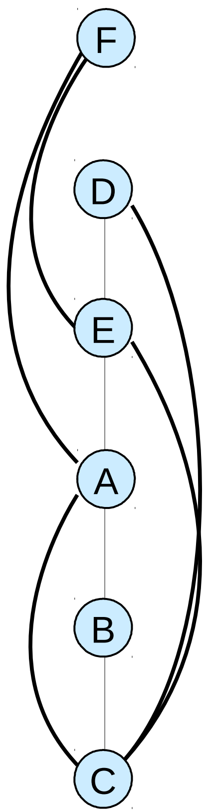
Solve each sub-problem efficiently

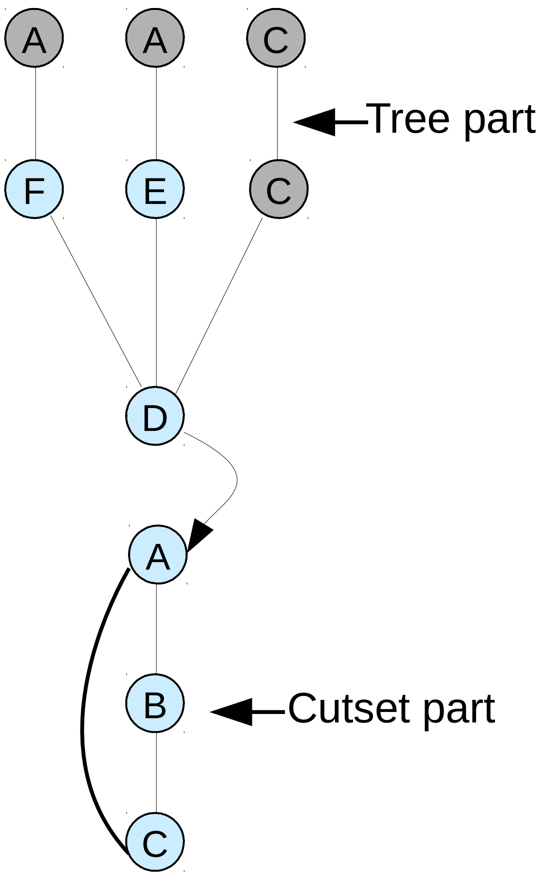
The Cycle-Cutset Scheme

Condition until Treeness

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

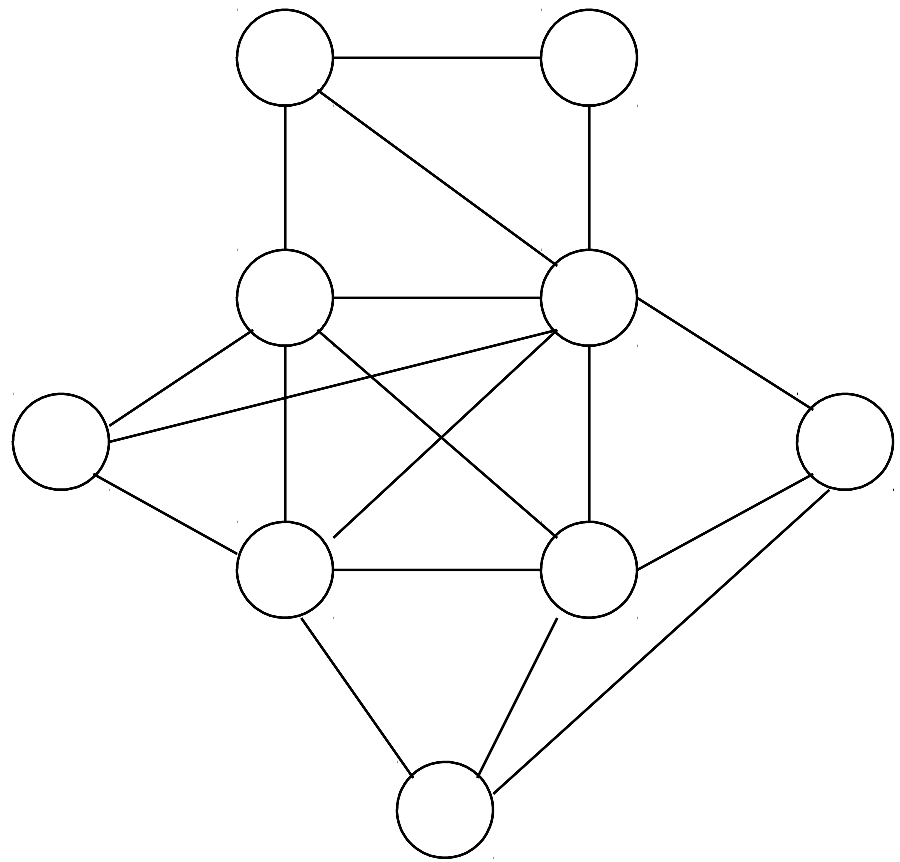




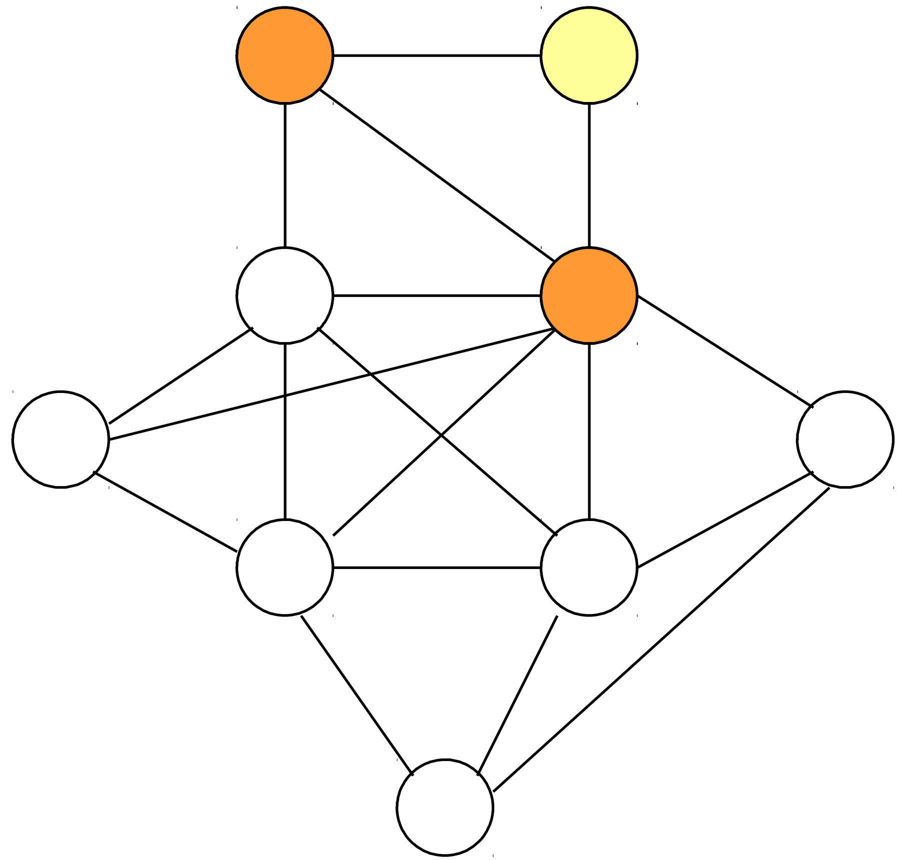


Space: exp(i), Time: O(exp(i+c(i))

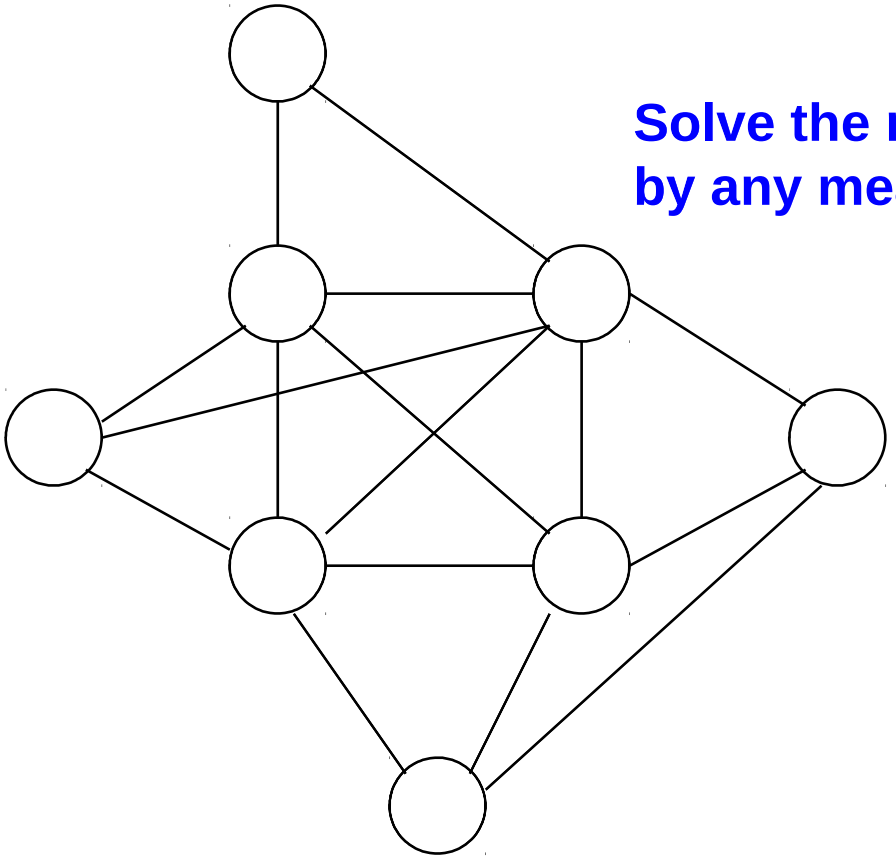
Eliminate First



Eliminate First



Eliminate First

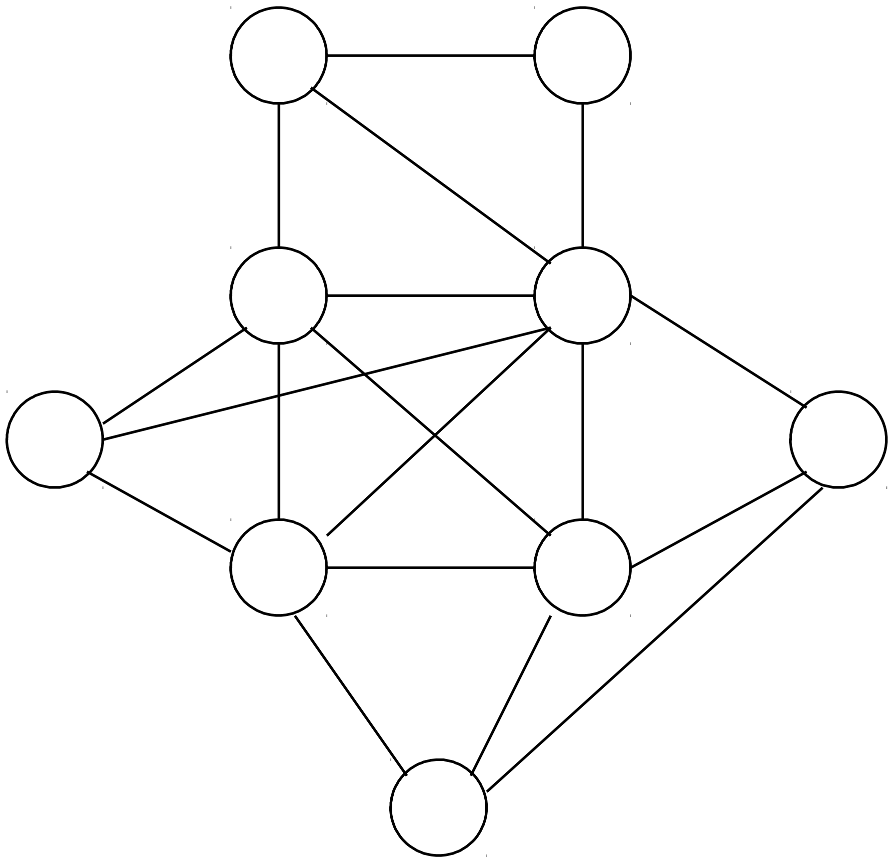


Solve the rest of the problem by any means

Hybrid Variants

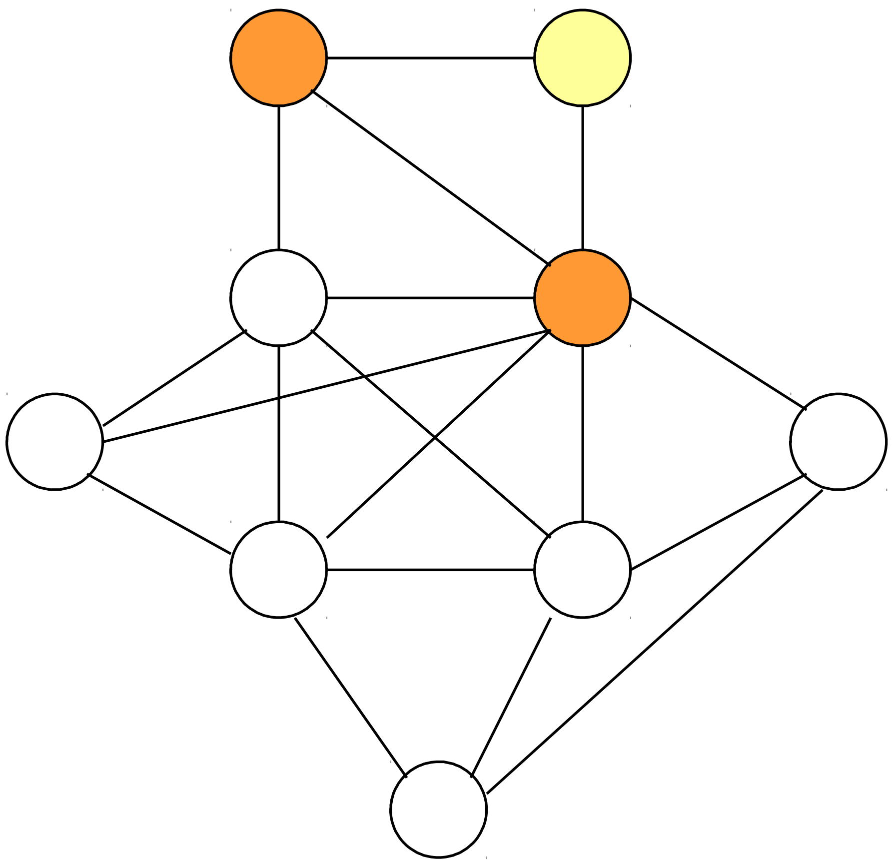
- **Condition**, **condition**, **condition**, ... and then only eliminate (w-cutset, cycle-cutset)
- **Eliminate**, **eliminate**, **eliminate**, ... and then only search
- **Interleave** conditioning and elimination steps (elim-cond(i), VE+C)

Interleaving Conditioning and Elimination

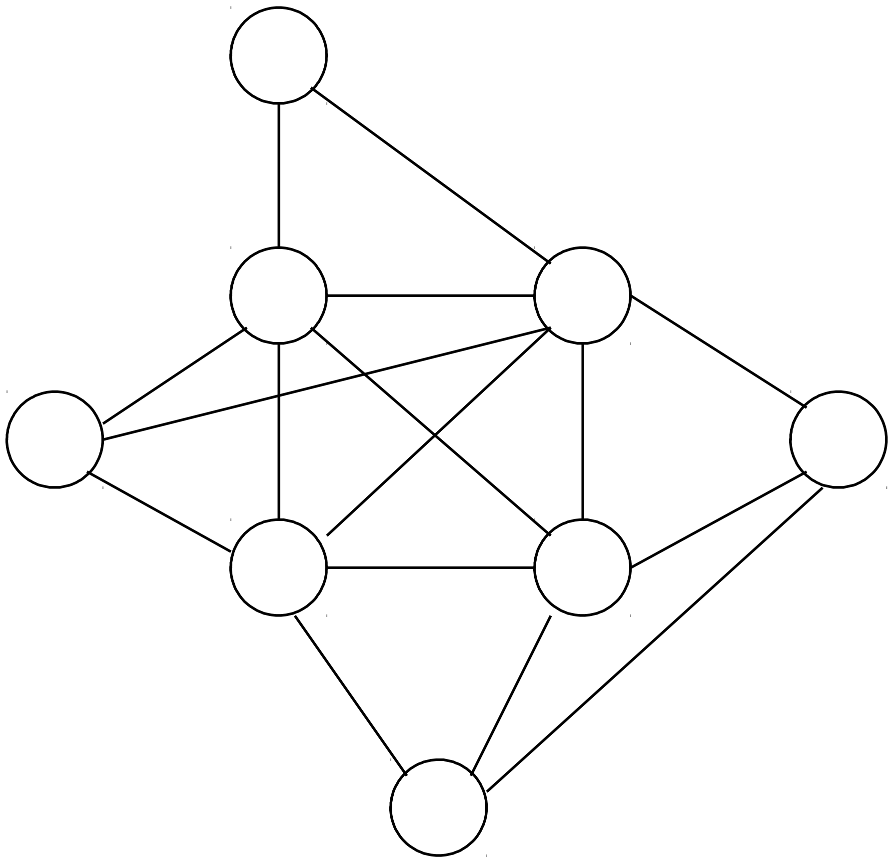


[Larrosa and Dechter, 2002]

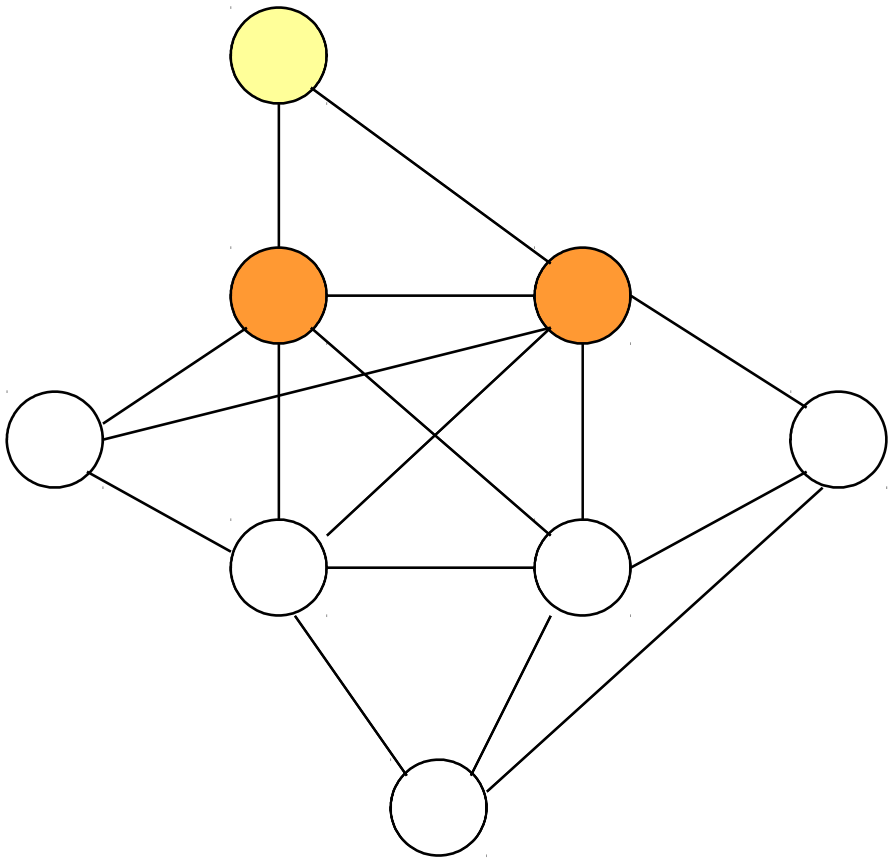
Interleaving Conditioning and Elimination



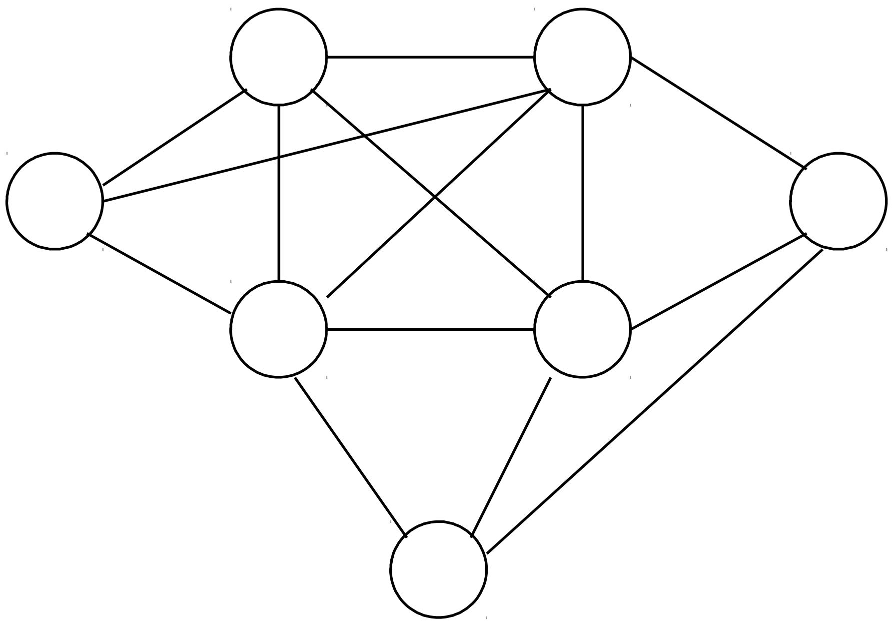
Interleaving Conditioning and Elimination



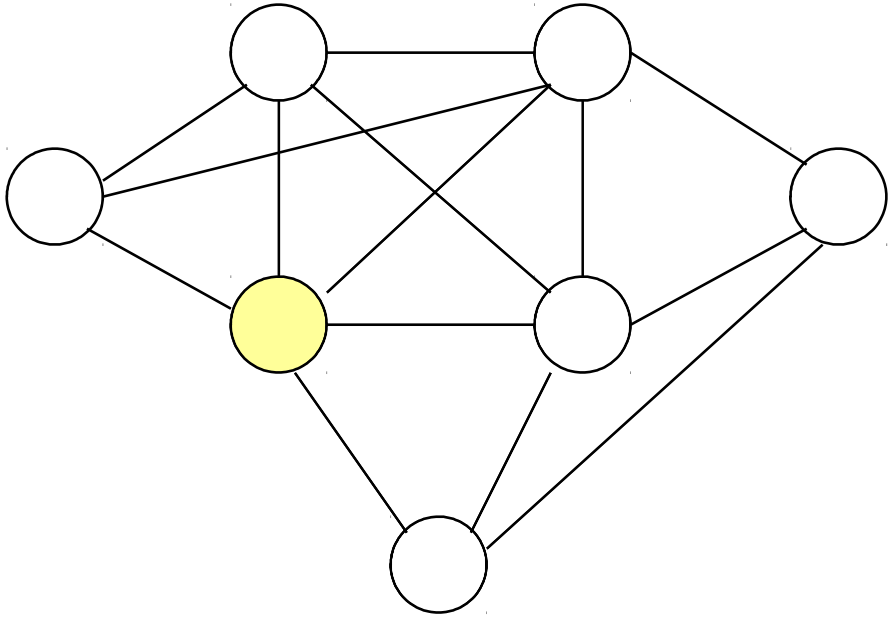
Interleaving Conditioning and Elimination



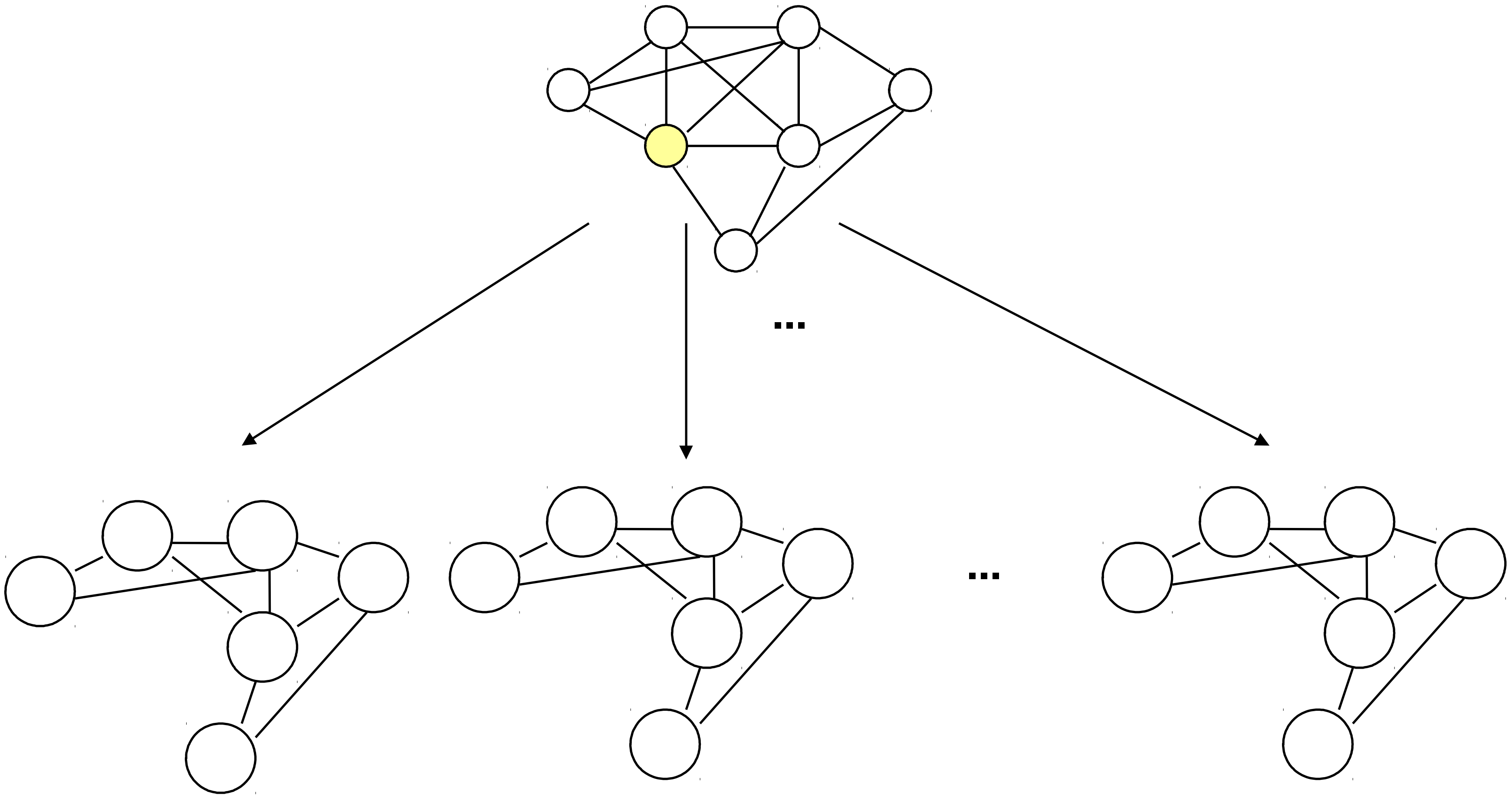
Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination

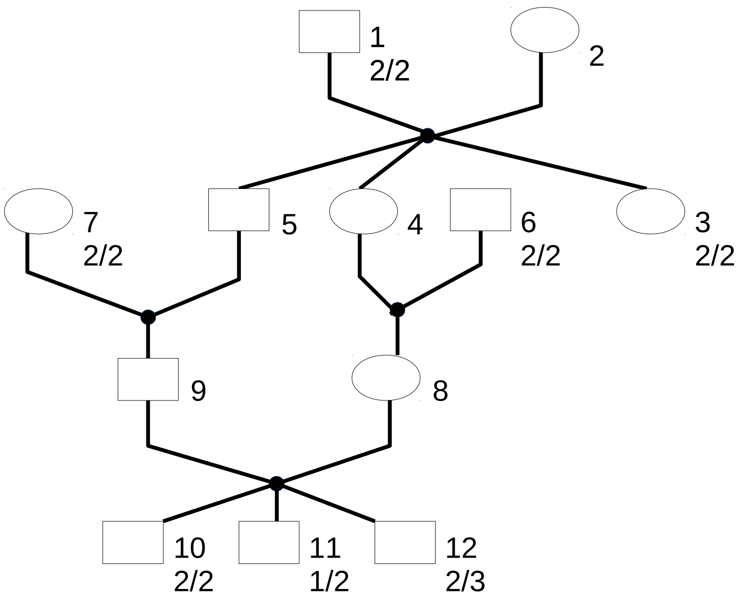


Boosting Search with Variable Elimination

- At each search node
  - Eliminate all unassigned variables with degree ≤ p
  - Select an assigned variable A
  - Branch on the values of A
- Properties
  - BB+VE(-1) is Depth-First Branch and Bound
  - BB+VE(w) is Variable Elimination
  - BB+VE(1) is similar to Cycle-Cutset
  - BB+VE(2) is well suited with soft local consistencies (add binary constraints only, independent of elimination order)

[Larrosa and Dechter, 2003]

Mendelian Error Detection



- Given a pedigree and partial observations (genotypings)
- Find the erroneous genotypings, such that their removal restores consistency

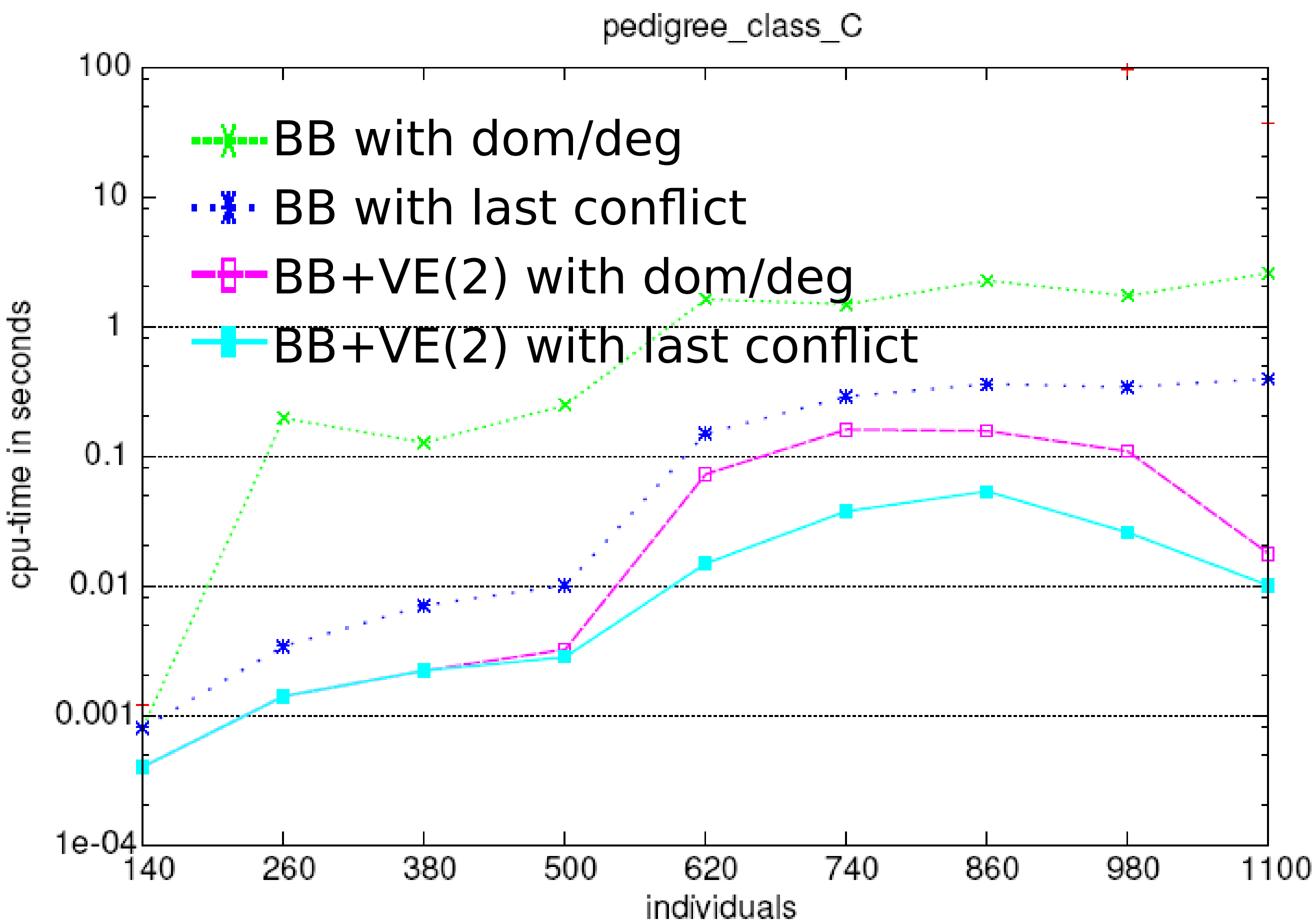
- Checking consistency is NP-complete [Aceto et al, 2004]
- Minimize the number of genotypings to be removed
- Maximize the joint probability of true genotypes (MPE/MAP)

Pedigree problem size: n ≤ 20,000; k = 3-66; e(3) ≤ 30,000

[Sanchez et al, 2008]

Pedigree

- toulbar2 v0.5 with EDAC and binary branching
- Minimize the number of genotypings to be removed
- CPU time to find and prove optimality on a 3 GHz computer with 16 GB



Outline

- Introduction
- Inference
- Bounds and heuristics
- AND/OR Search
- **Exploiting parallelism**
- Software

Outline

- Introduction
- Inference
- Search
- Lower bounds and relaxations
- **Exploiting parallelism**
  - Distributed and parallel search
- Software

Contributions

- Propose parallel AOBB, first of its kind.
  - Runs on computational grid.
  - Extends parallel tree search paradigm.
  - Two variants with different parallelization logic.
- Analysis of schemes' properties:
  - Performance considerations and trade-offs.
    - Granularity vs. overhead and redundancies.
- Large-scale experimental evaluation:
  - Good parallel performance in many cases.
  - Analysis of some potential performance pitfalls.

Context and Related Work

- Task parallelism (vs. data parallelism):
  - Extensive computation on small input.

Context and Related Work

- Task parallelism (vs. data parallelism):
  - Extensive computation on small input.
- Computational grid framework:
  - Independent hosts, limited or no communication.

Context and Related Work

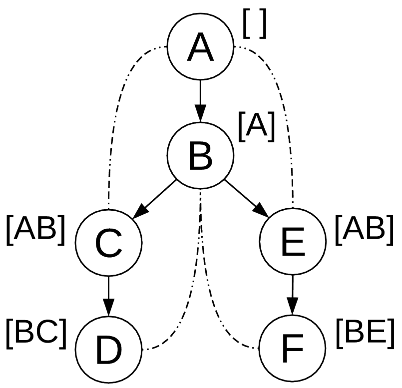
- Task parallelism (vs. data parallelism):
  - Extensive computation on small input.
- Computational grid framework:
  - Independent hosts, limited or no communication.
- Parallel tree search (“stack splitting”):
  - Typically uses *shared memory* for dynamic load balancing and cost bound updates for BaB.
    - Not feasible in grid setup.

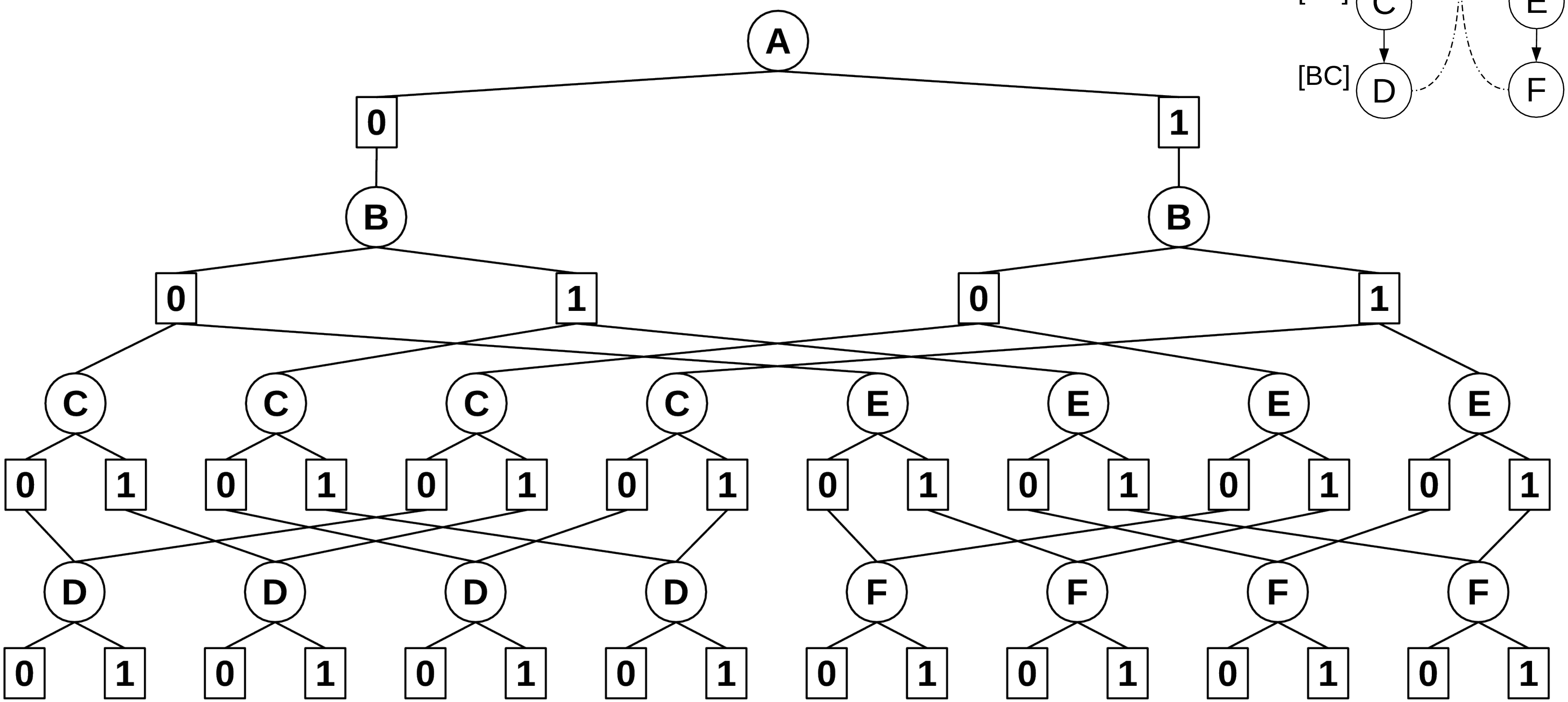
Context and Related Work

- Task parallelism (vs. data parallelism):
  - Extensive computation on small input.
- Computational grid framework:
  - Independent hosts, limited or no communication.
- Parallel tree search (“stack splitting”):
  - Typically uses *shared memory* for dynamic load balancing and cost bound updates for BaB.
    - Not feasible in grid setup.
- Motivation: *Superlink Online*.
  - Distributed linkage (likelihood) computation.

Parallel AOBB Illustrated

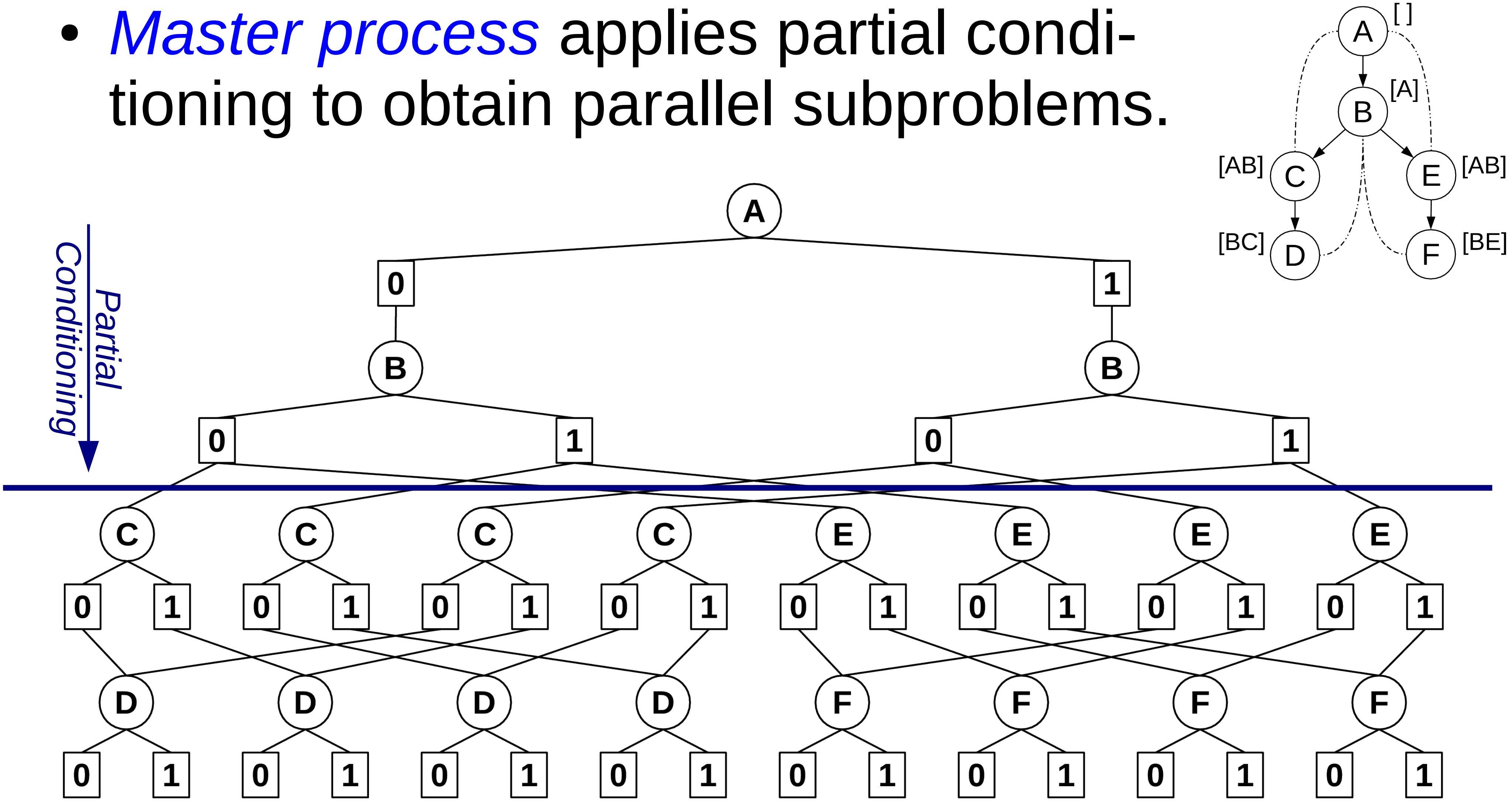
- *Master process* applies partial conditioning to obtain parallel subproblems.





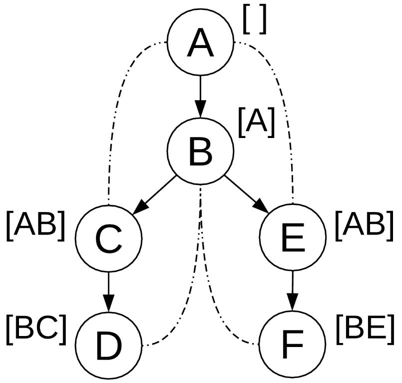
Parallel AOBB Illustrated

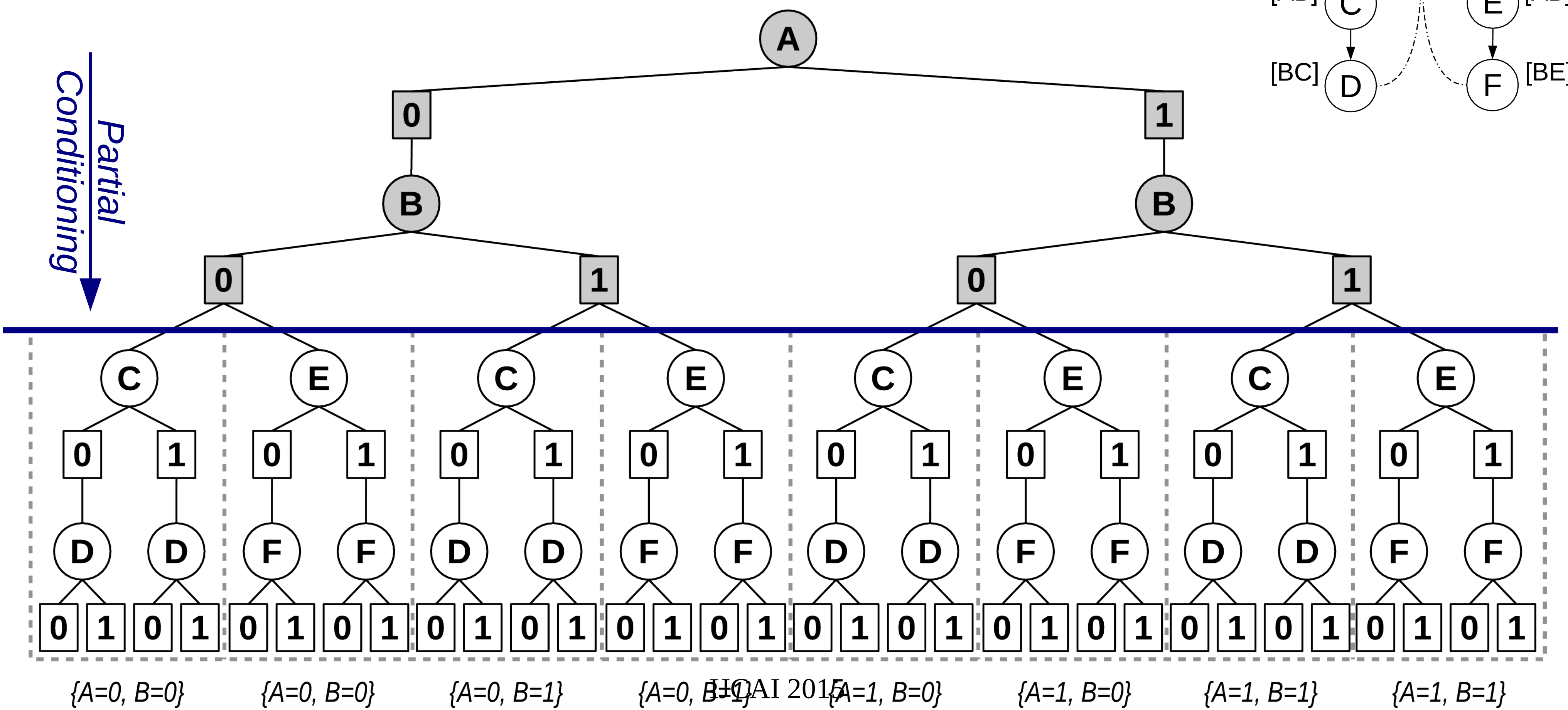
- *Master process* applies partial conditioning to obtain parallel subproblems.



Parallel AOBB Illustrated

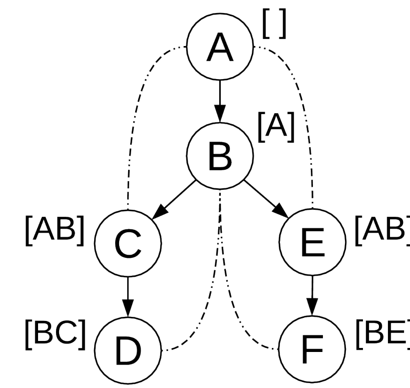
- *Master process* applies partial conditioning to obtain parallel subproblems.

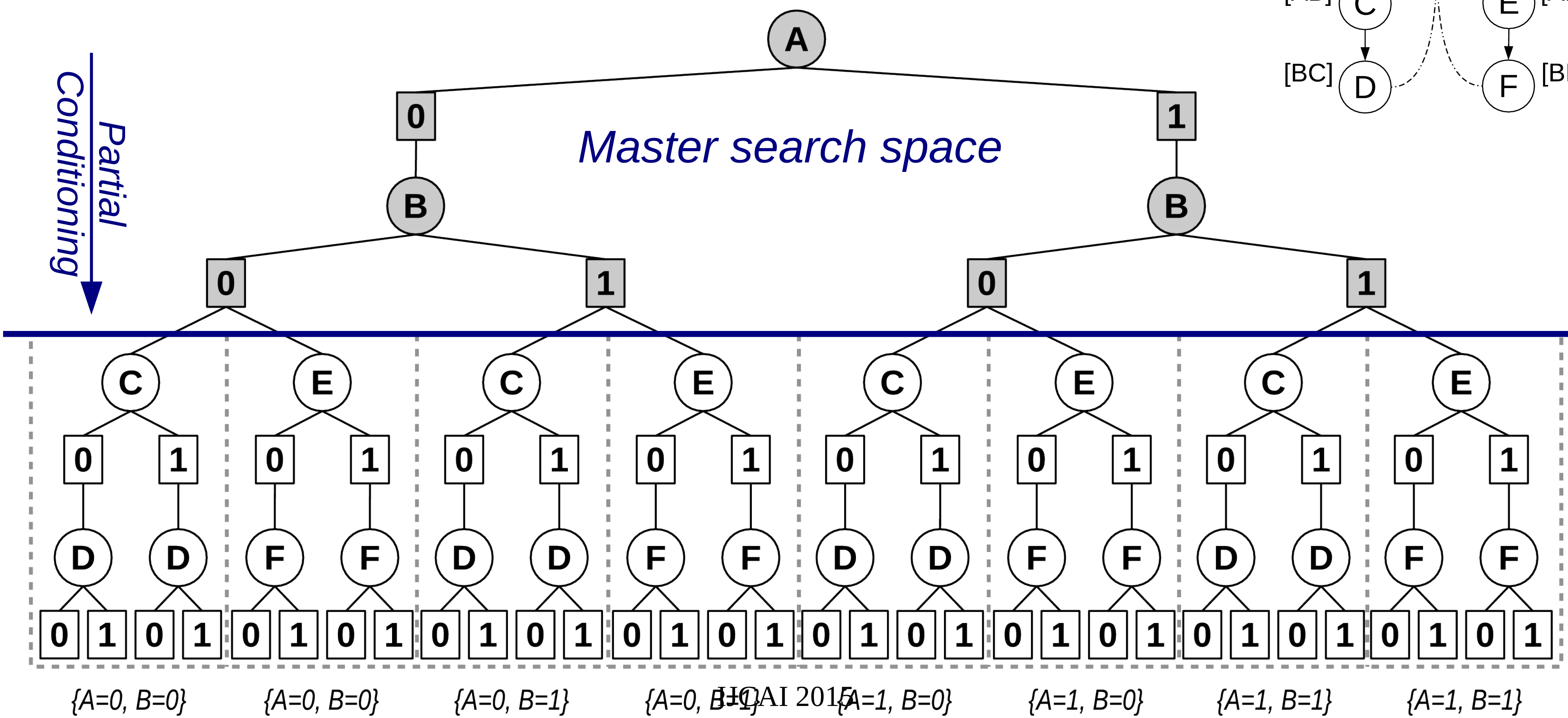




Parallel AOBB Illustrated

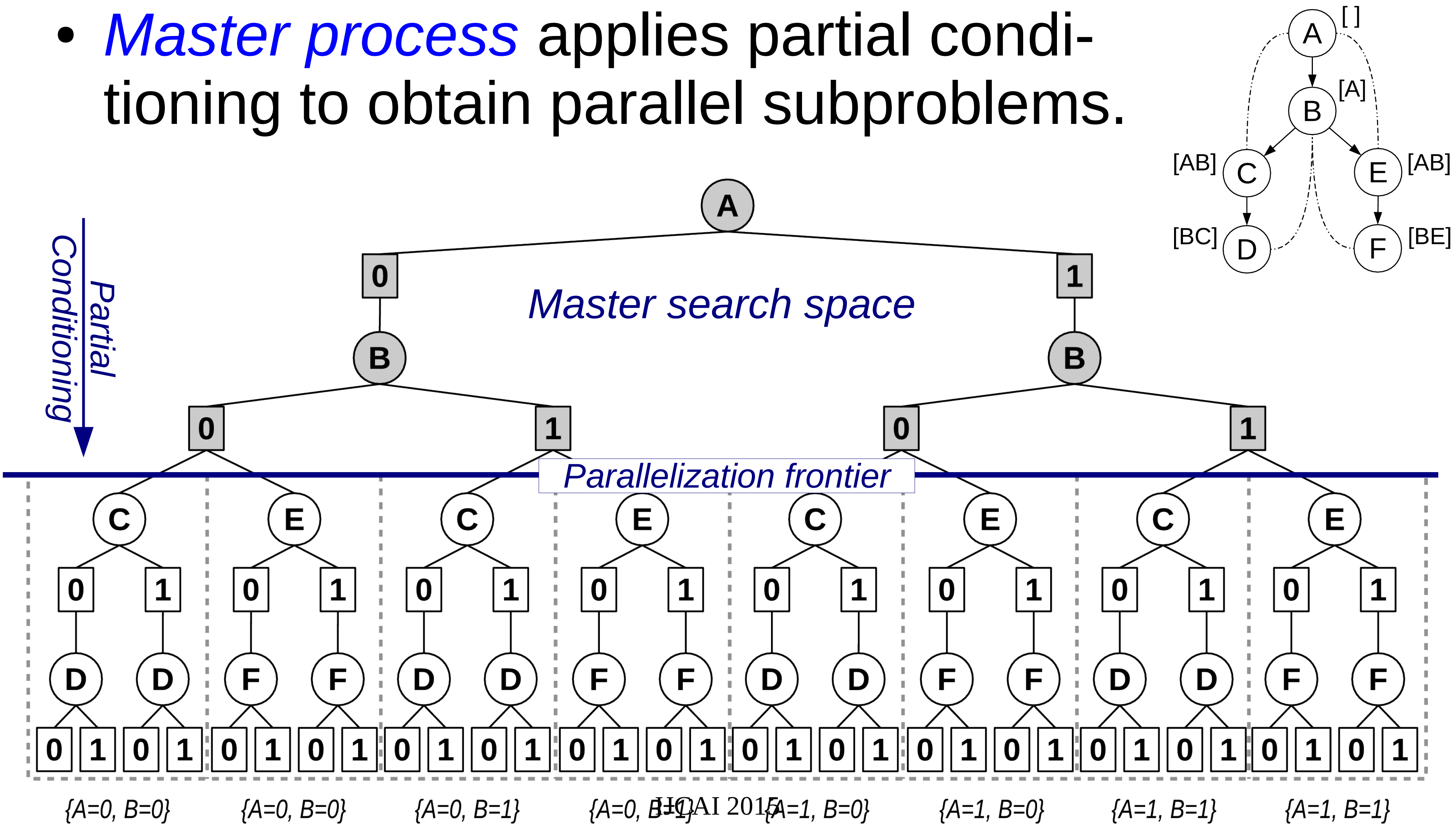
- *Master process* applies partial conditioning to obtain parallel subproblems.





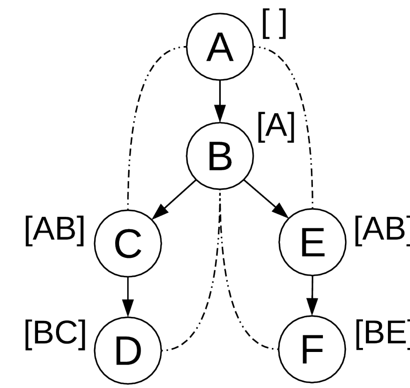
Parallel AOBB Illustrated

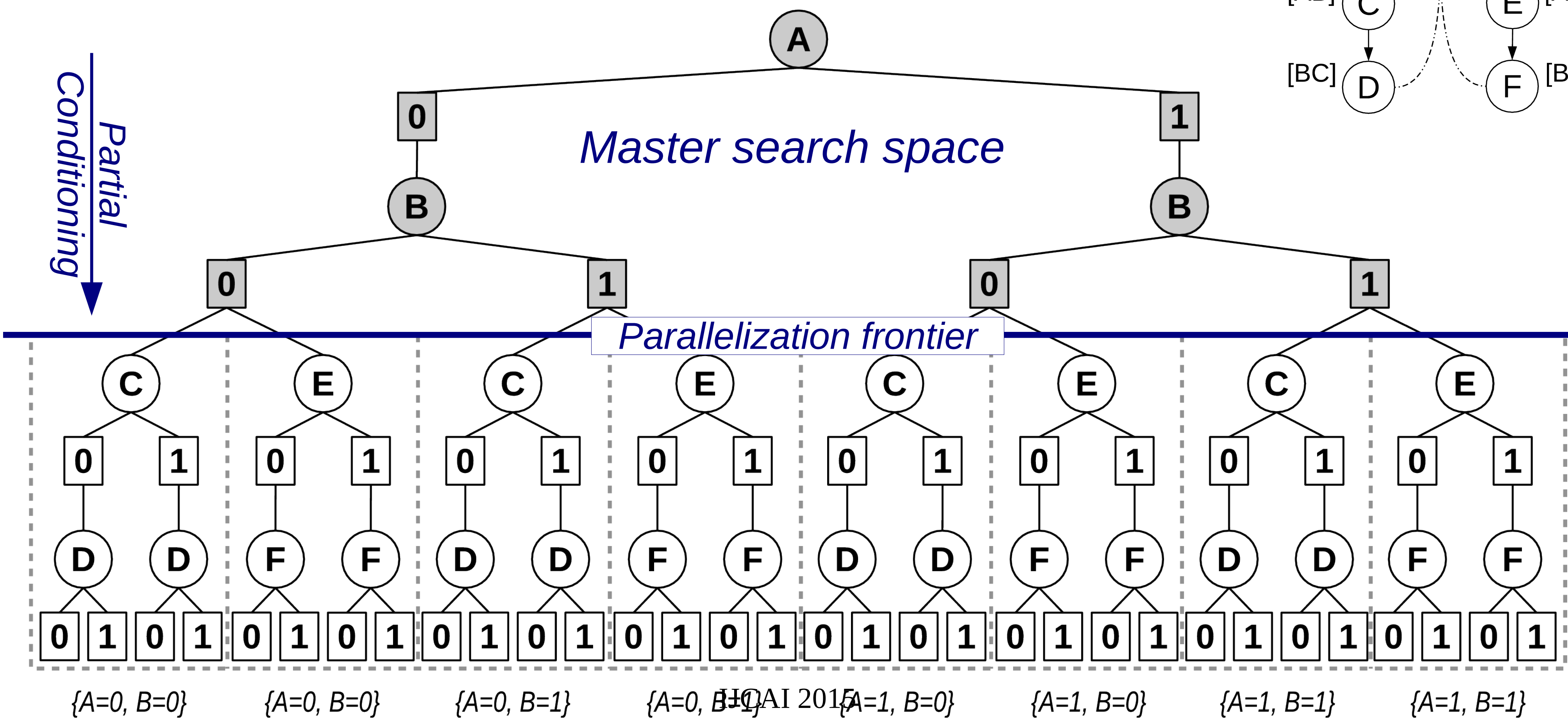
- *Master process* applies partial conditioning to obtain parallel subproblems.



Parallel AOBB Illustrated

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

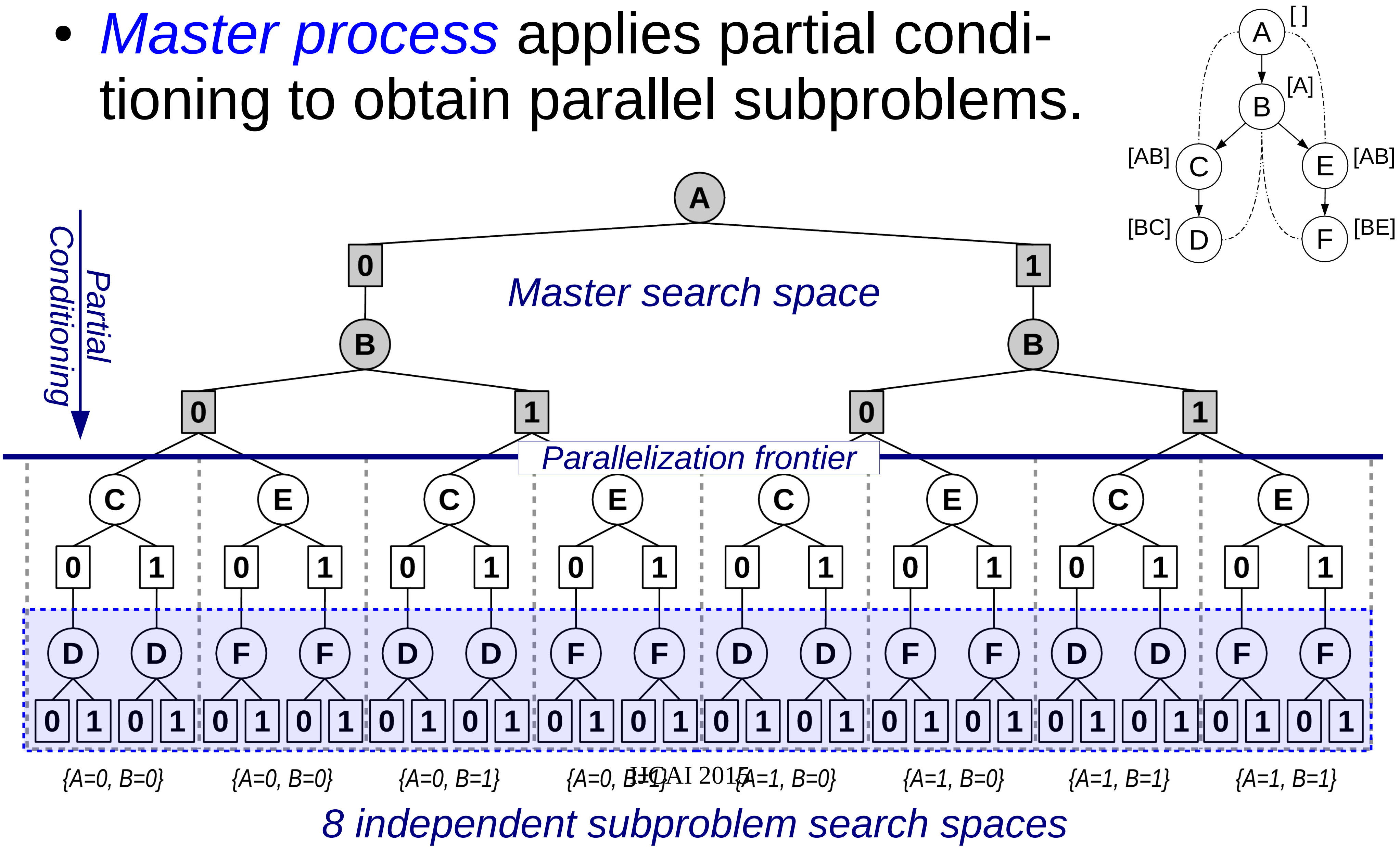




8 independent subproblem search spaces

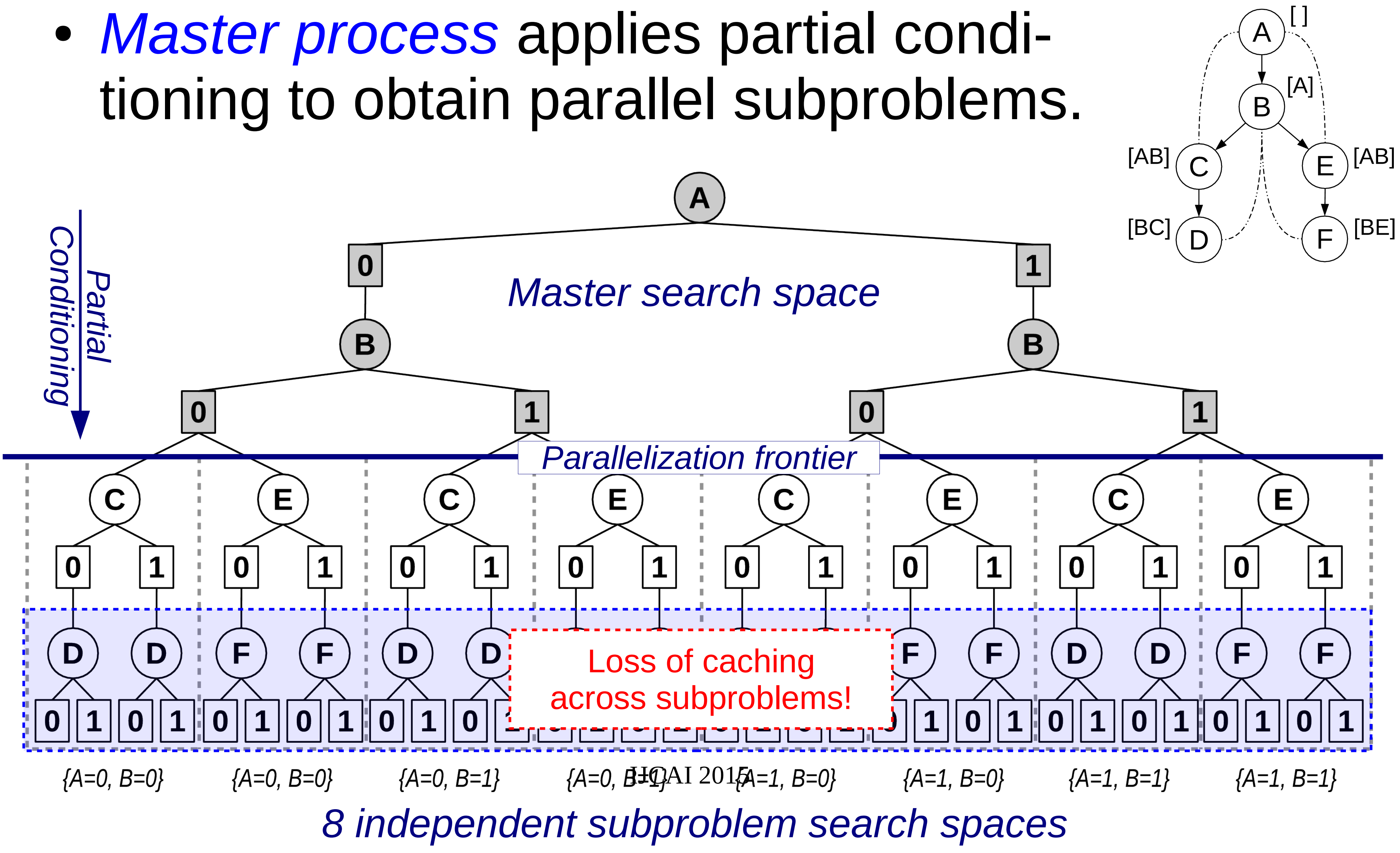
Parallel AOBB Illustrated

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



Parallel AOBB Illustrated

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

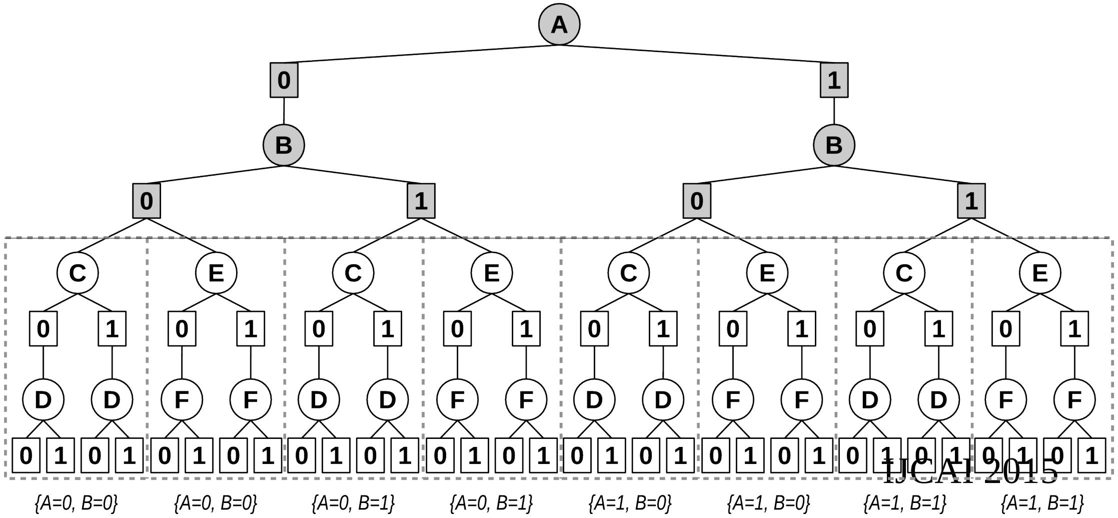


Fixed-depth Parallel AOBB

- Algorithm receives cutoff depth *d* as input:
  - Expand nodes centrally until depth *d*.
  - At depth *d*, submit to grid job queue.

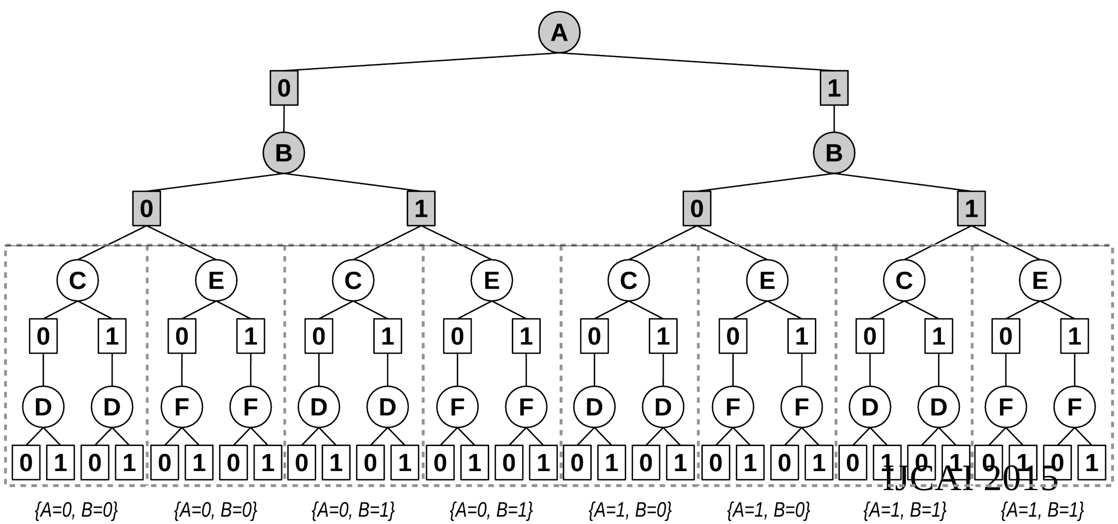
Fixed-depth Parallel AOBB

- Algorithm receives cutoff depth *d* as input:
  - Expand nodes centrally until depth *d*.
  - At depth *d*, submit to grid job queue.



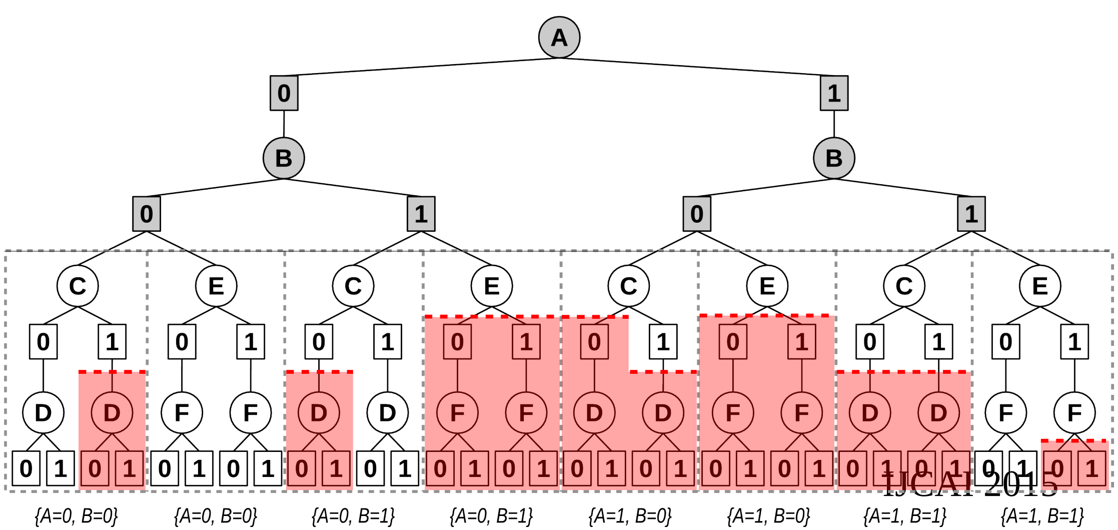
Fixed-depth Parallel AOBB

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



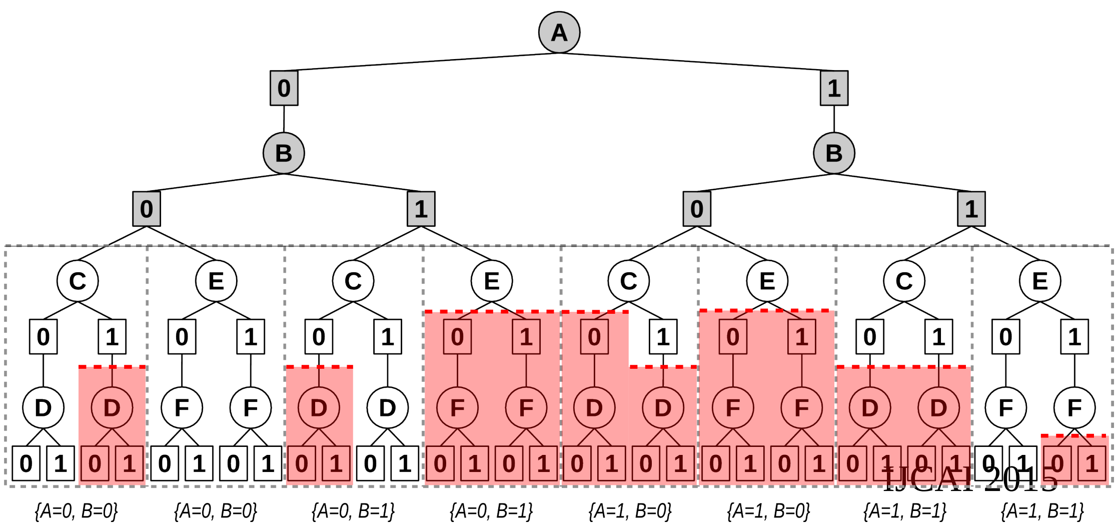
Fixed-depth Parallel AOBB

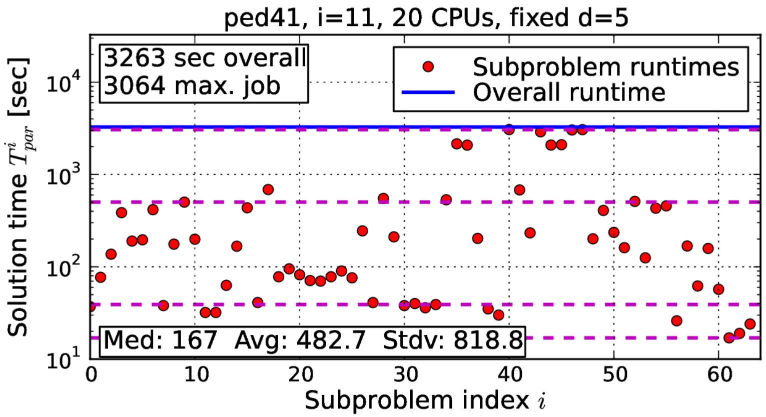
- Algorithm receives cutoff depth *d* as input:
  - Expand nodes centrally until depth *d*.
  - At depth *d*, submit to grid job queue.
- Explored subproblem search spaces potentially very unbalanced.



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 subproblem search spaces potentially very unbalanced.





Variable-depth Parallel AOBB

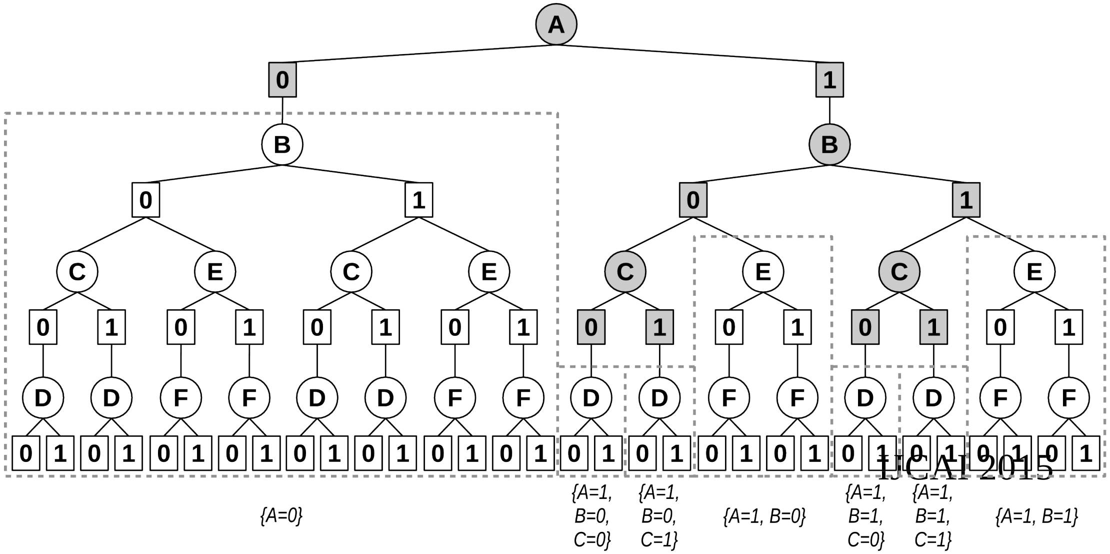
- Given subproblem count *p* and estimator *N* :
  - Iteratively deepen frontier until size *p* reached:
    - Pick subproblem *n* with largest estimate *N(n)* and split.
  - Submit subproblems into job queue by descending complexity estimates.

Variable-depth Parallel AOBB

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

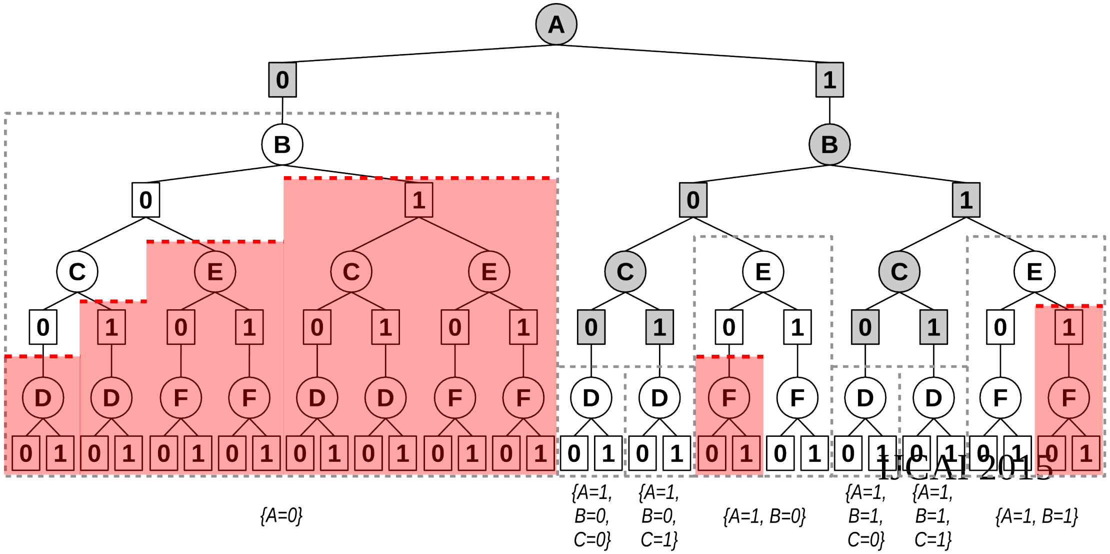
Variable-depth Parallel AOBB

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



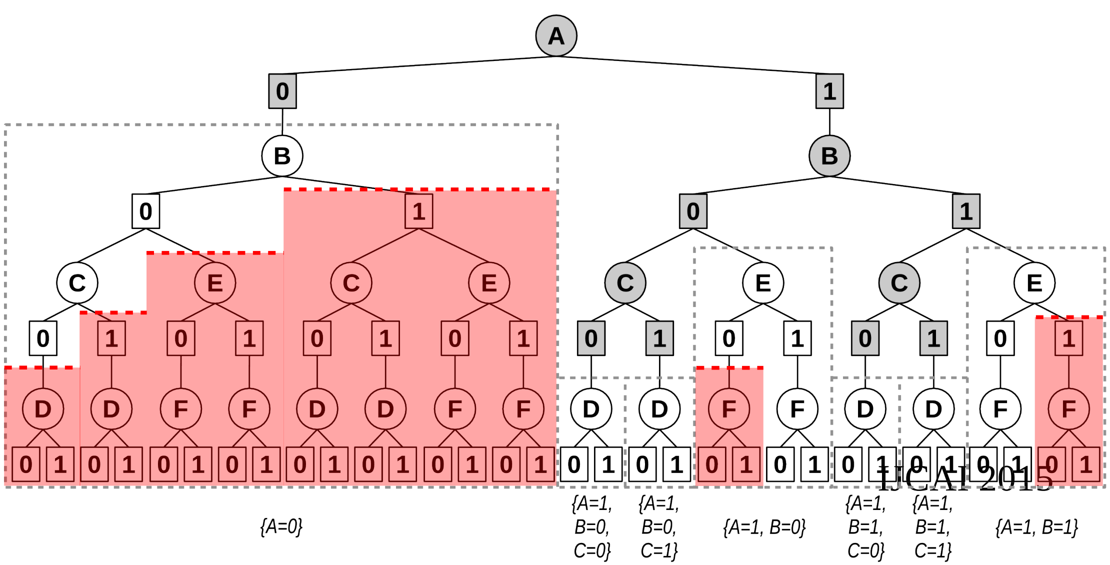
Variable-depth Parallel AOBB

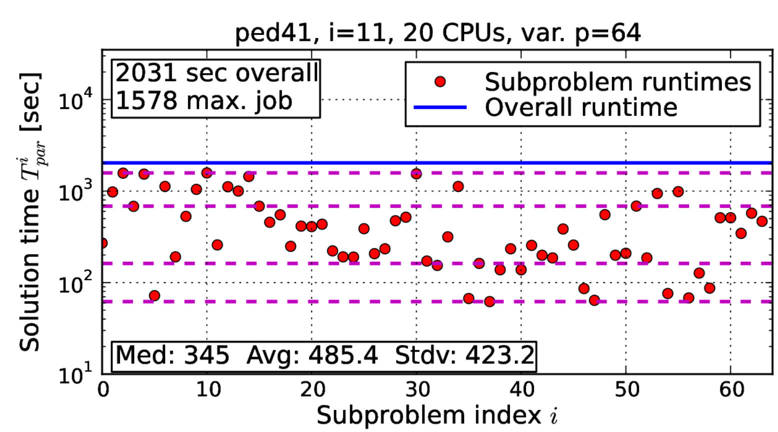
- Given subproblem count *p* and estimator *N* :
  - Iteratively deepen frontier until size *p* reached:
    - Pick subproblem *n* with largest estimate *N(n)* and split.
  - Submit subproblems into job queue by descending complexity estimates.
- Hope to achieve better subproblem balance.



Variable-depth Parallel AOBB

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





Aside: Modeling AOBB Complexity

- Model number of nodes $N(n)$ in subproblem as exp. function of subproblem features $\varphi_i(n)$:

$$N(n) = \exp\left(\sum_i \lambda_i \varphi_i(n)\right)$$

Aside: Modeling AOBB Complexity

- Model number of nodes *N(n)* in subproblem as exp. function of subproblem features $\varphi_i(n)$:

$$N(n) = \exp\left(\sum_i \lambda_i \varphi_i(n)\right)$$

- Logarithm yields *linear regression* problem.
  - Minimize MSE with Lasso regularization. [Tibshirani]

$$\frac{1}{m}\sum_{j=1}^{m}\left(\sum_i \lambda_i \varphi_i(n_k) - \log N(n_k)\right)^2 + \alpha \sum_i |\lambda_i|$$

  - Full details:
    - “A Case Study in Complexity Estimation: Towards Parallel Branch-and-Bound over Graphical Models”, UAI 2012.

35 Subproblem Features

- Characterize subproblem:
  - Static, structural properties:
    - Number of variables.
    - Avg. and max. width.
    - Height of sub pseudo tree.
    - State space bound *SS* .
  - Dynamic, runtime properties:
    - Upper and lower bound on subproblem cost.
    - Pruning ratio and depth of small AOBB probe.
      - only *5n* nodes, very fast.

Subproblem variable statistics (static):
1: Number of variables in subproblem.
2-6: Min, Max, mean, average, and std. dev. of variable domain sizes in subproblem.

Pseudo tree depth/leaf statistics (static):
7: Depth of subproblem root in overall search space.
8-12: Min, max, mean, average, and std. dev. of depth of subproblem pseudo tree leaf nodes, counted from subproblem root.
13: Number of leaf nodes in subproblem pseudo tree.

Pseudo tree width statistics (static):
14-18: Min, max, mean, average, and std. dev. of induced width of variables within subproblem.
19-23: Min, max, mean, average, and std. dev. of induced width of variables within subproblem, *conditioned on subproblem root context*.

State space bound (static):
24: State space size upper bound on subproblem search space size.

Subproblem cost bounds (dynamic):
25: Lower bound L on subproblem solution cost, derived from current best overall solution.
26: Upper bound U on subproblem solution cost, provided by mini bucket heuristics.
27: Difference $U - L$ between upper and lower bound, expressing "constrainedness" of the subproblem.

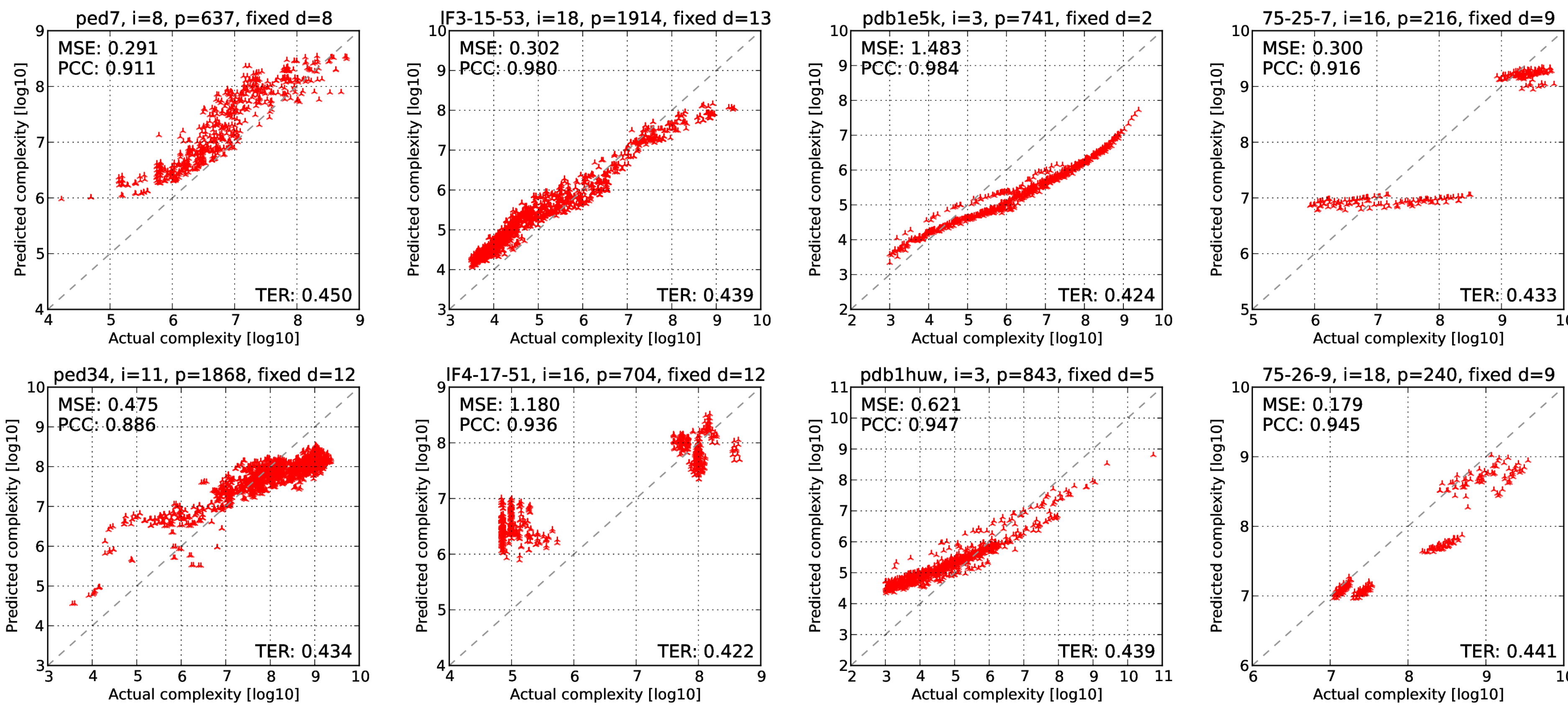
Pruning ratios (dynamic), based on running AOBB for $5n$ node expansions:
28: Ratio of nodes pruned using the heuristic.
29: Ratio of nodes pruned due of determinism (zero probabilities, e.g.)
30: Ratio of nodes corresponding to pseudo tree leaf.

AOBB sample (dynamic), based on running AOBB for $5n$ node expansions:
31: Average depth of terminal search nodes within probe.
32: Average node depth within probe (denoted $\bar{d}$).
33: Average branching degree, defined as $\sqrt[\bar{d}]{5n}$.

Various (static):
34: Mini bucket i-bound parameter.
35: Max. subproblem variable context size minus mini bucket i-bound.

Example Estimation Results

- Across subproblems from several domains.
  - Hold out test data for model learning.



Assessing Parallel Performance

- Sequential AOBB performance baseline:
  - T_{seq}: sequential runtime.
  - N_{seq}: number of sequential node expansions.

Assessing Parallel Performance

- Sequential AOBB performance baseline:
  - T_{seq}: sequential runtime.
  - N_{seq}: number of sequential node expansions.
- Parallel AOBB performance metrics:
  - T_{par}: parallel runtime including central preprocessing.
  - S_{par}: Parallel speedup T_{seq} / T_{par} .

Assessing Parallel Performance

- Sequential AOBB performance baseline:
  - T_{seq}: sequential runtime.
  - N_{seq}: number of sequential node expansions.
- Parallel AOBB performance metrics:
  - T_{par}: parallel runtime including central preprocessing.
  - S_{par}: Parallel speedup T_{seq} / T_{par} .
  - N_{par}: Node expansions across all subproblems.
  - O_{par}: Relative parallel overhead N_{par} / N_{seq} .

Assessing Parallel Performance

- Sequential AOBB performance baseline:
  - T_{seq}: sequential runtime.
  - N_{seq}: number of sequential node expansions.
- Parallel AOBB performance metrics:
  - T_{par}: parallel runtime including central preprocessing.
  - S_{par}: Parallel speedup T_{seq} / T_{par} .
  - N_{par}: Node expansions across all subproblems.
  - O_{par}: Relative parallel overhead N_{par} / N_{seq} .
  - U_{par}: Avg. processor utilization, relative to longest.

Performance Considerations

- Amdahl's Law [1967]:
  - “If a fraction *p* of a computation can be sped up by a factor or *s*, the overall speedup cannot exceed $1/(1–p+p/s)$.”
    - Example: 20 minute computation, 30 sec preprocessing. Best speedup **40x** (regardless of parallel CPUs).

Performance Considerations

- Amdahl's Law [1967]:
  - “If a fraction p of a computation can be sped up by a factor or s, the overall speedup cannot exceed $1/(1–p+p/s)$.”
    - Example: 20 minute computation, 30 sec preprocessing. Best speedup **40x** (regardless of parallel CPUs).
- Implication of overhead O_{par} :
  - Proposition: assuming parallel overhead o and execution on p CPUs, speedup is bounded by p/o .
    - Example: 500 CPUs, overhead 2 → best speedup 250.
    - In practice even lower due to load balancing, communication delays, etc.

Parallel AOBB Performance Factors

- Distributed System Overhead:
  - Master preprocessing and parallelization decision.
  - Repeated preprocessing in workers (mini-buckets).
  - Communication and scheduling delays.

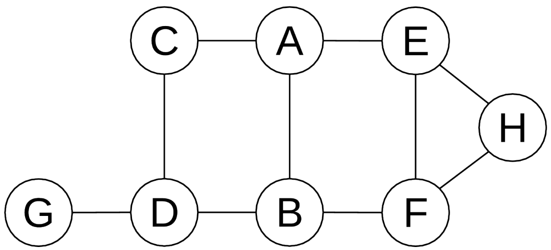
Parallel AOBB Performance Factors

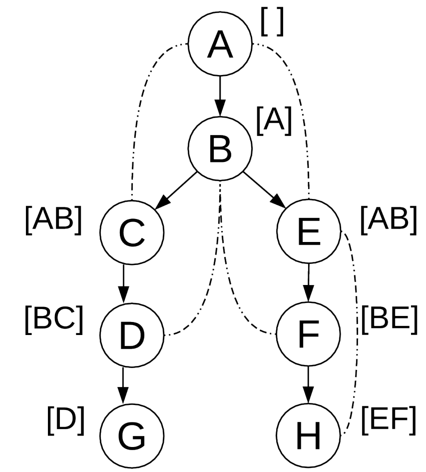
- Distributed System Overhead:
  - Master preprocessing and parallelization decision.
  - Repeated preprocessing in workers (mini-buckets).
  - Communication and scheduling delays.
- Parallel search space redundancies:
  - Impacted pruning, lack of bounds propagation.
    - Local search for near-optimal initial bound.
  - Loss of caching across parallel subproblems.
    - Analyzed subsequently.

Parallel AOBB Performance Factors

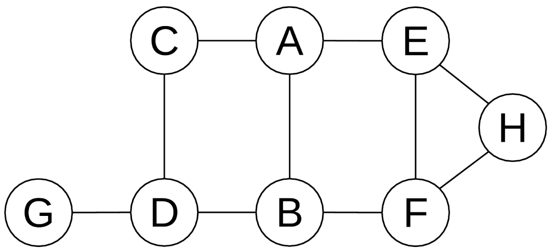
- Distributed System Overhead:
  - Master preprocessing and parallelization decision.
  - Repeated preprocessing in workers (mini-buckets).
  - Communication and scheduling delays.
- Parallel search space redundancies:
  - Impacted pruning, lack of bounds propagation.
    - Local search for near-optimal initial bound.
  - Loss of caching across parallel subproblems.
    - Analyzed subsequently.
- Parallel AOBB is <u>not</u> *“embarrassingly parallel”*.

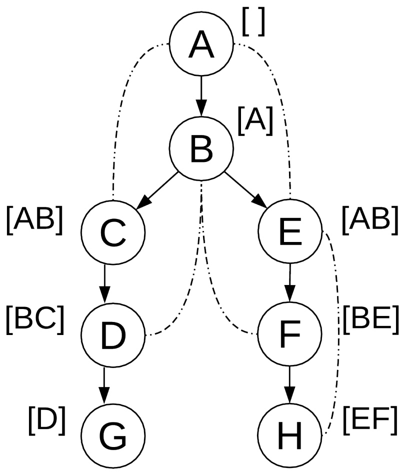
Redundancy Analysis

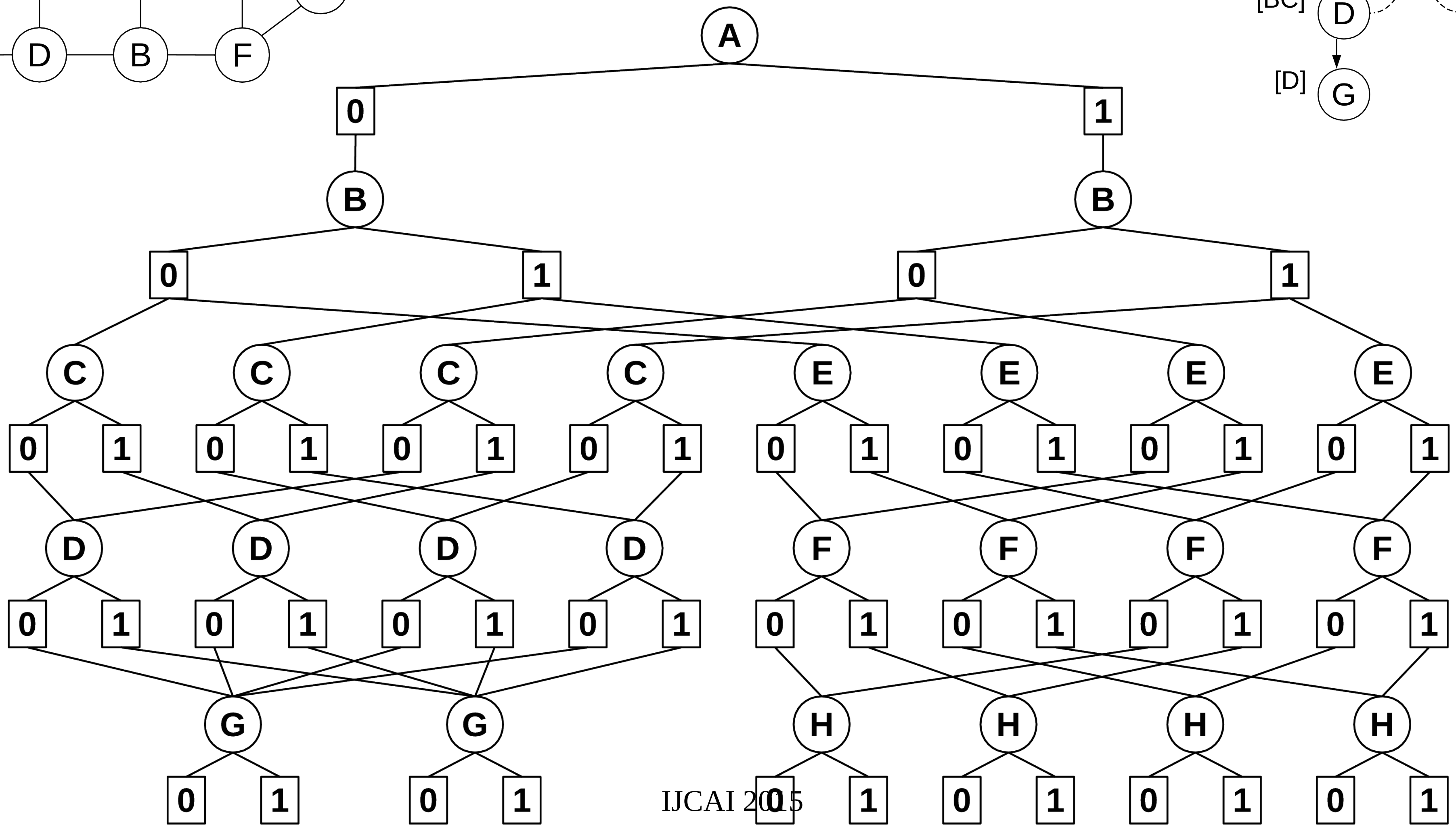




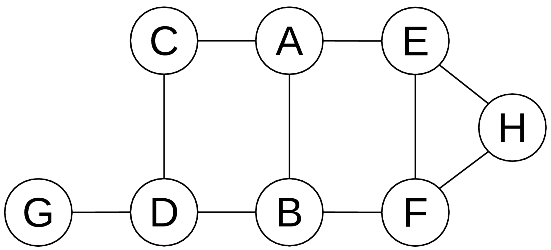
Redundancy Analysis

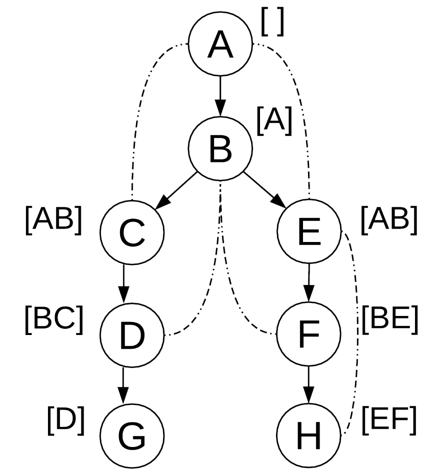


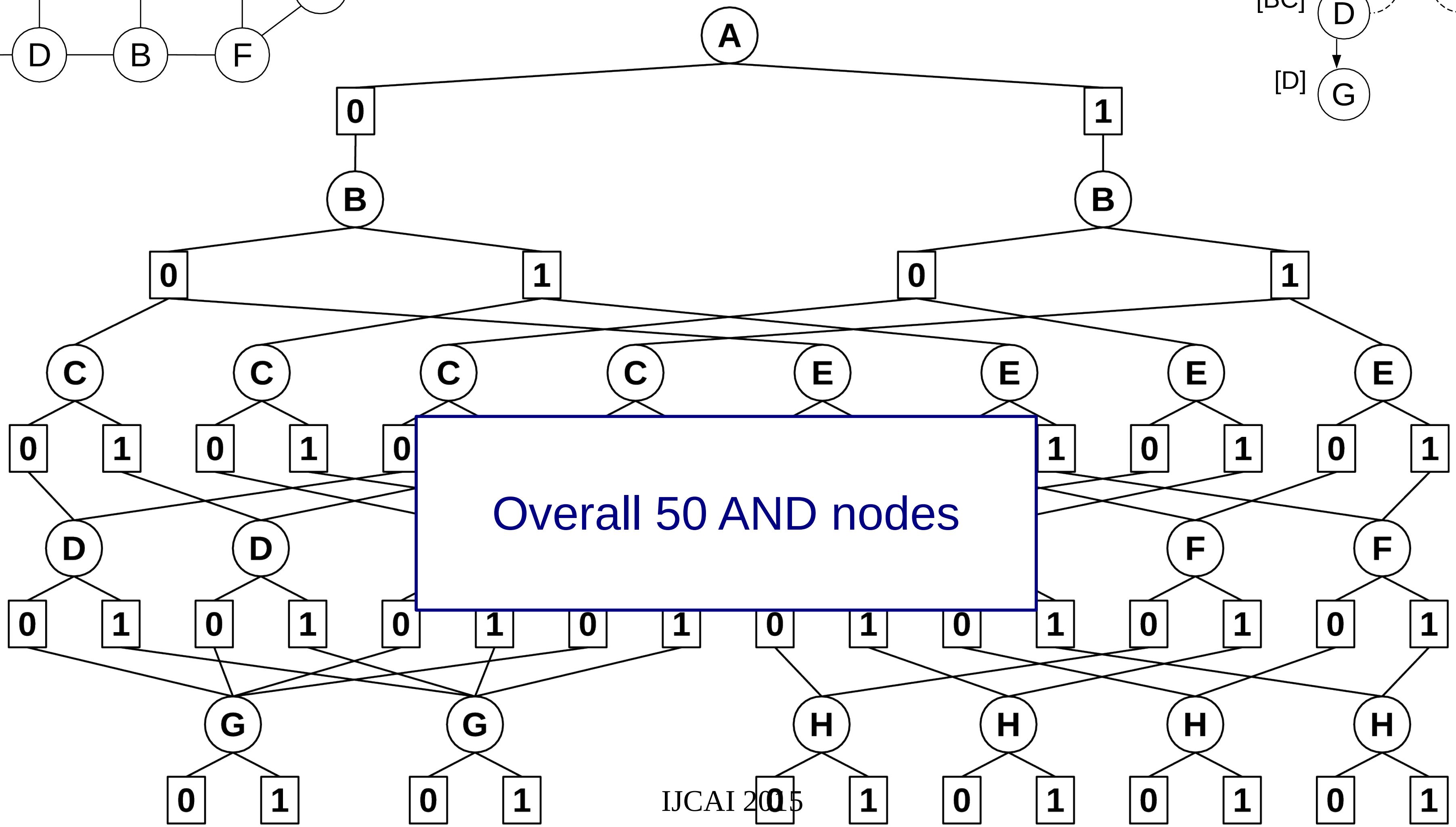




Redundancy Analysis

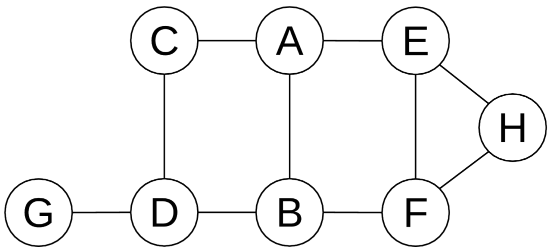


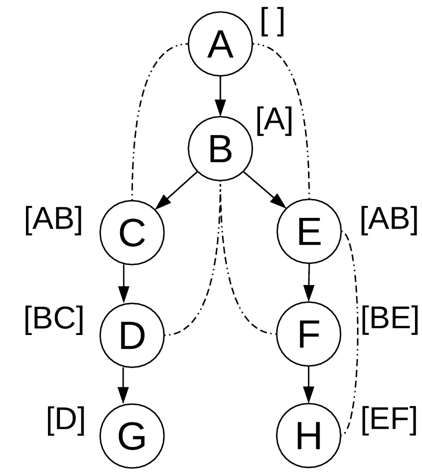


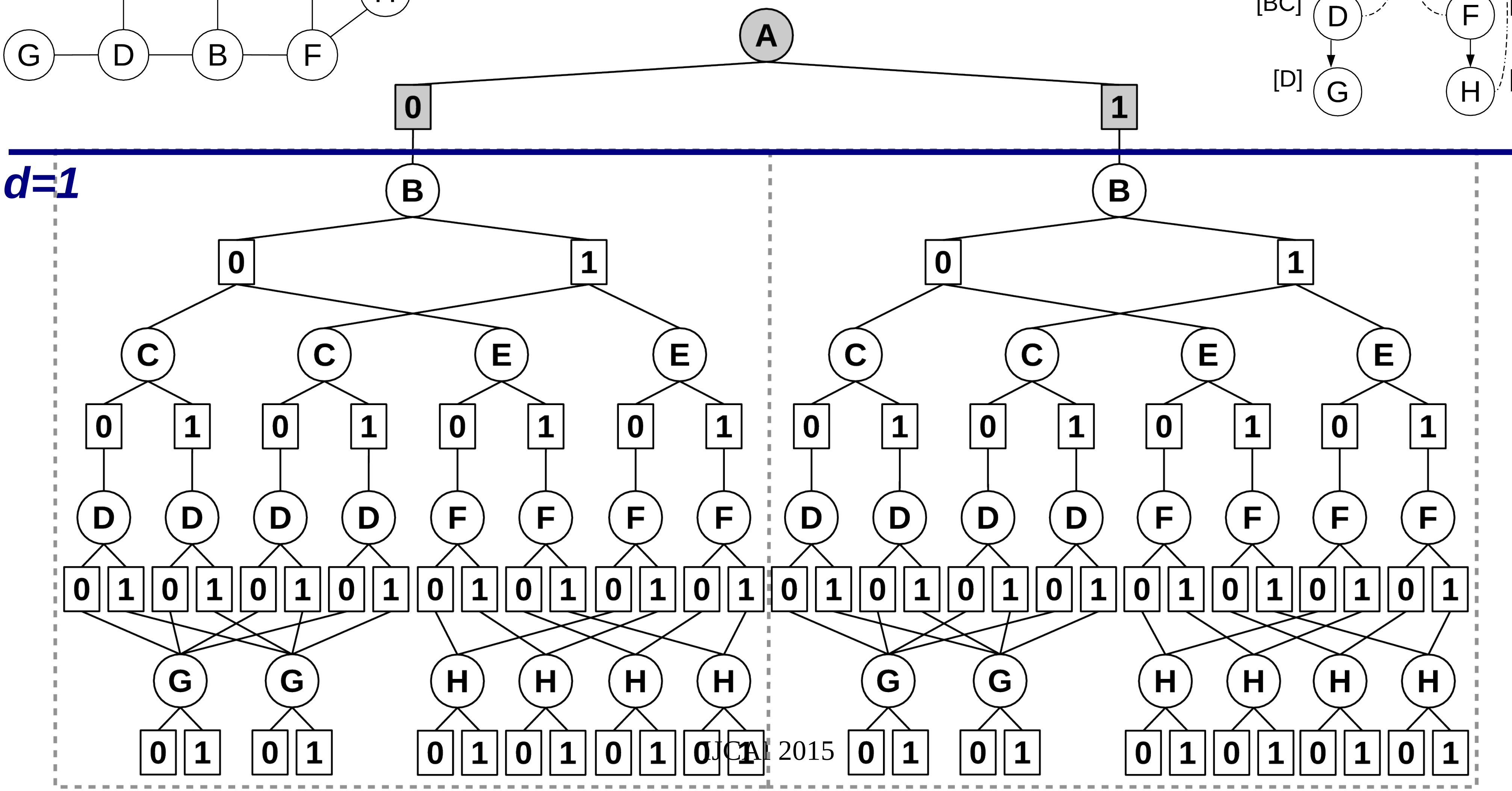


Overall 50 AND nodes

Redundancy Analysis



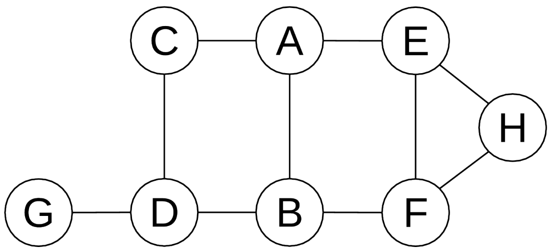


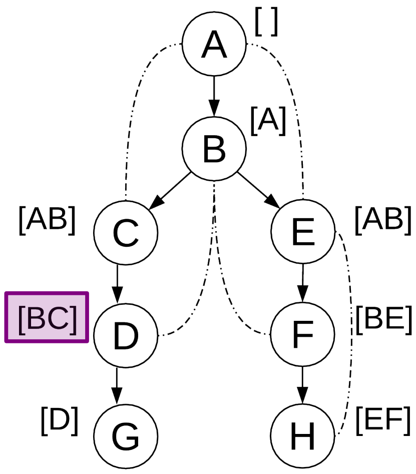


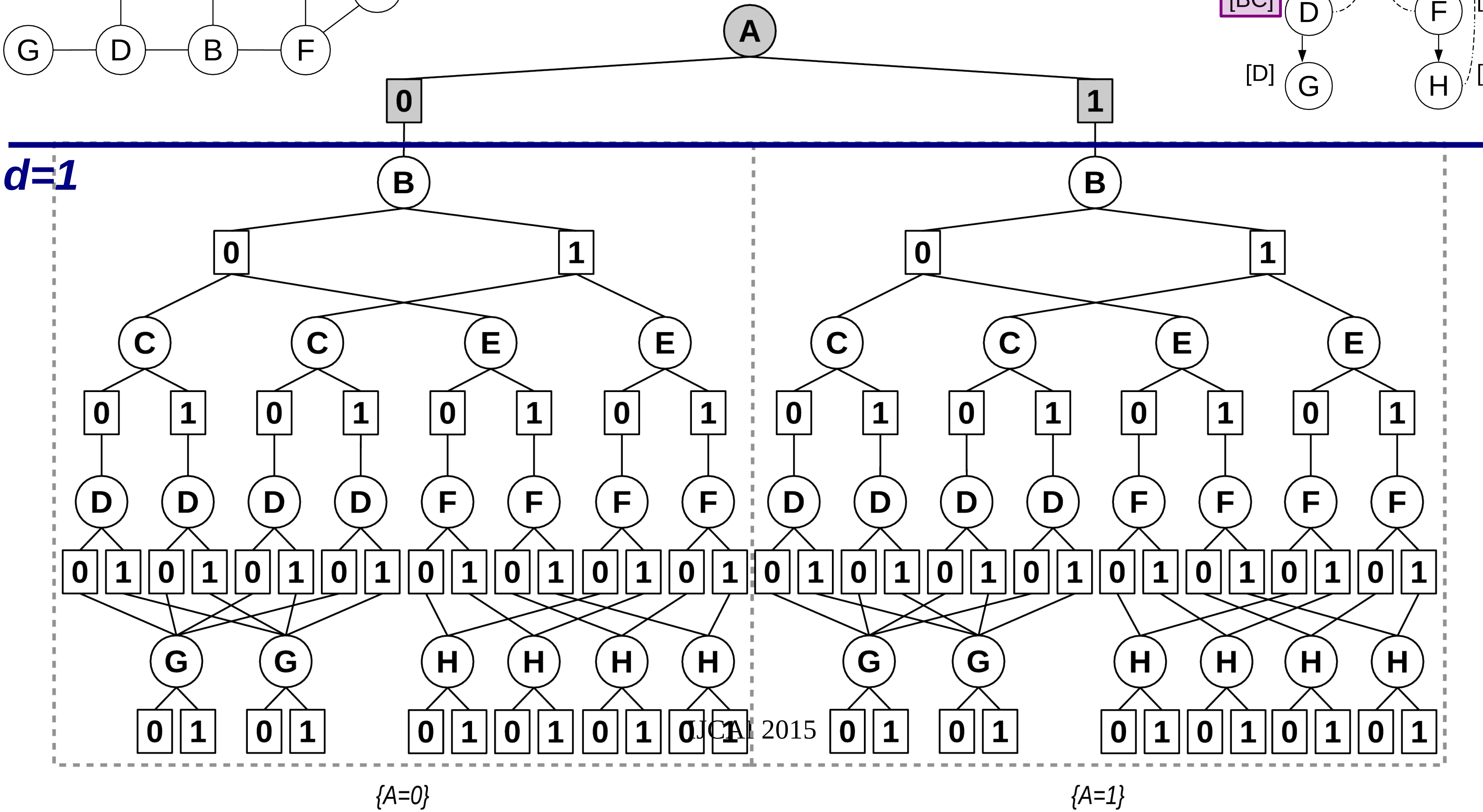
{A=0}

{A=1}

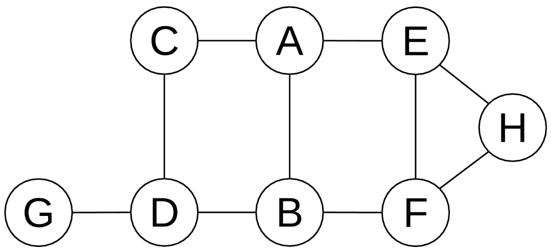
Redundancy Analysis

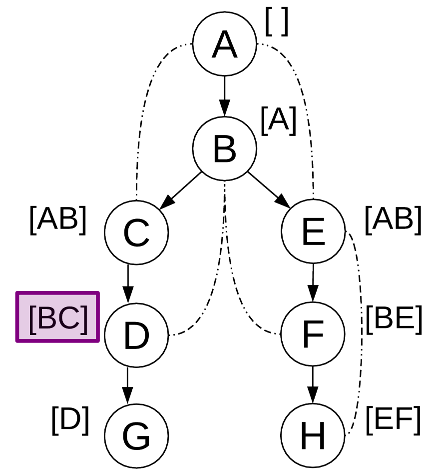


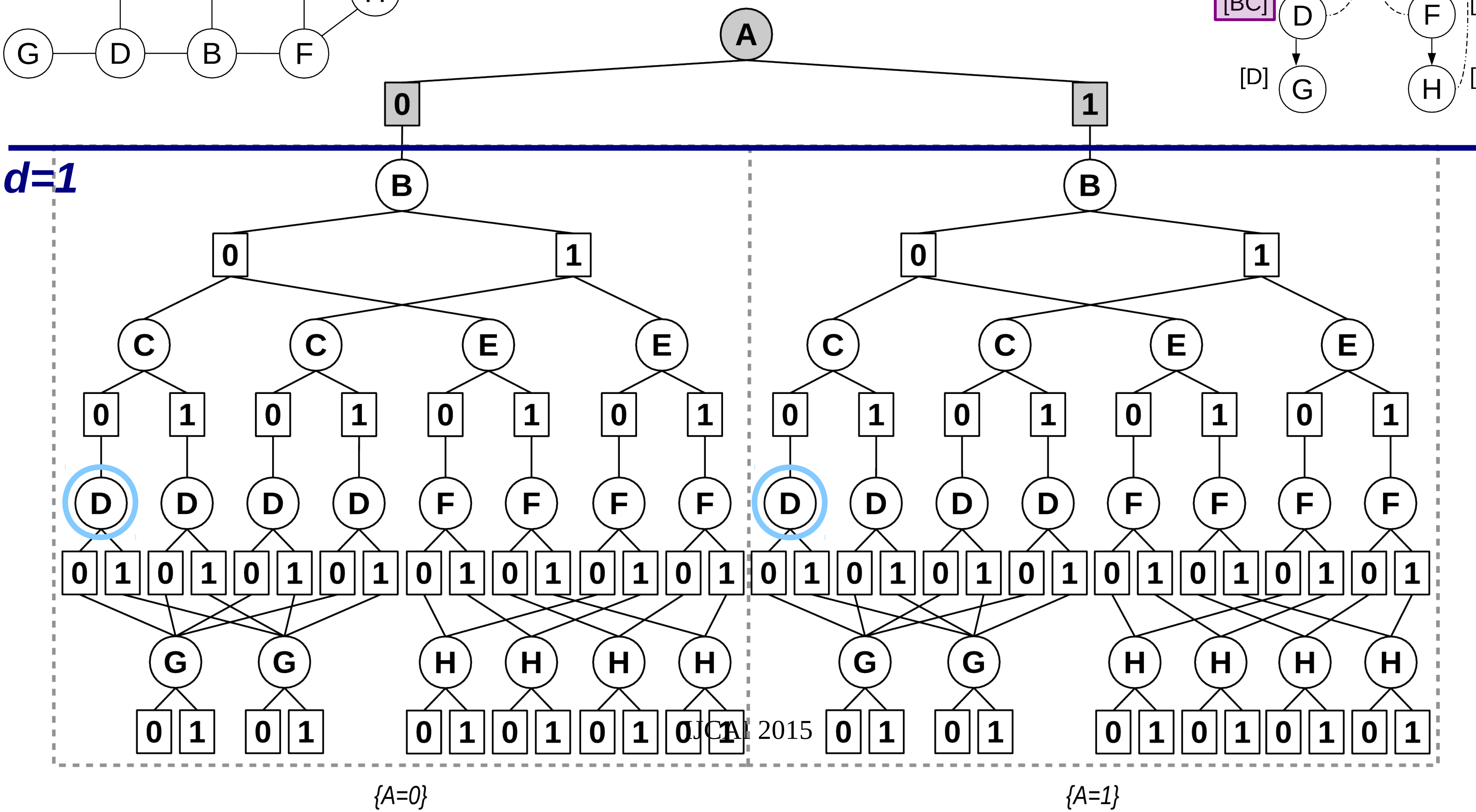




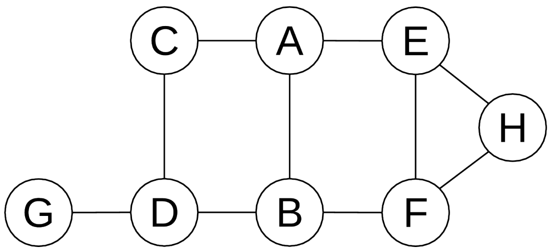
Redundancy Analysis

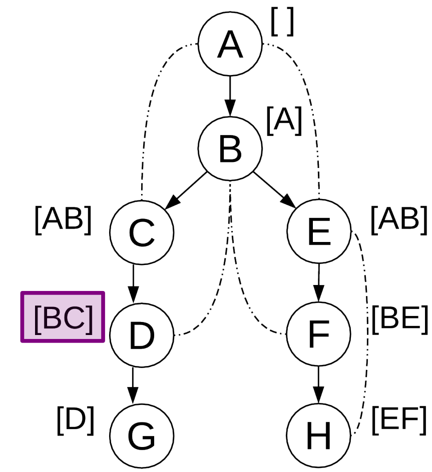


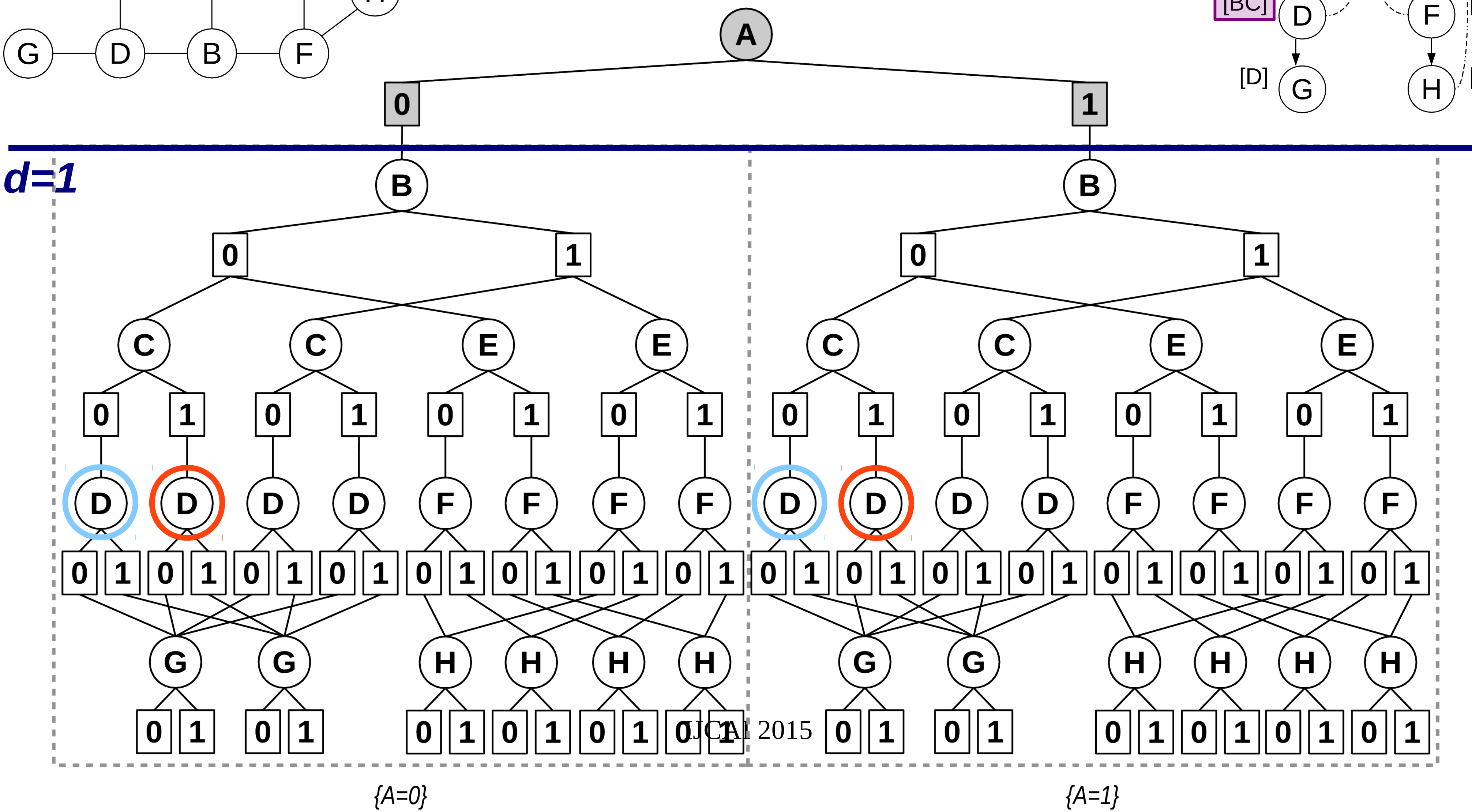




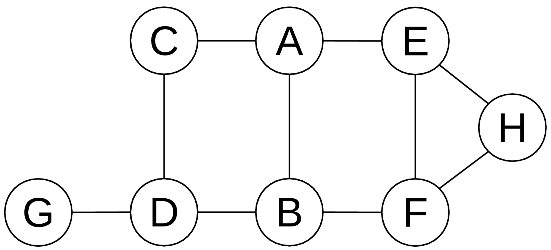
Redundancy Analysis

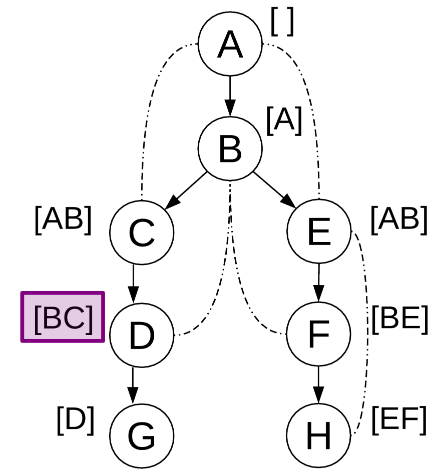


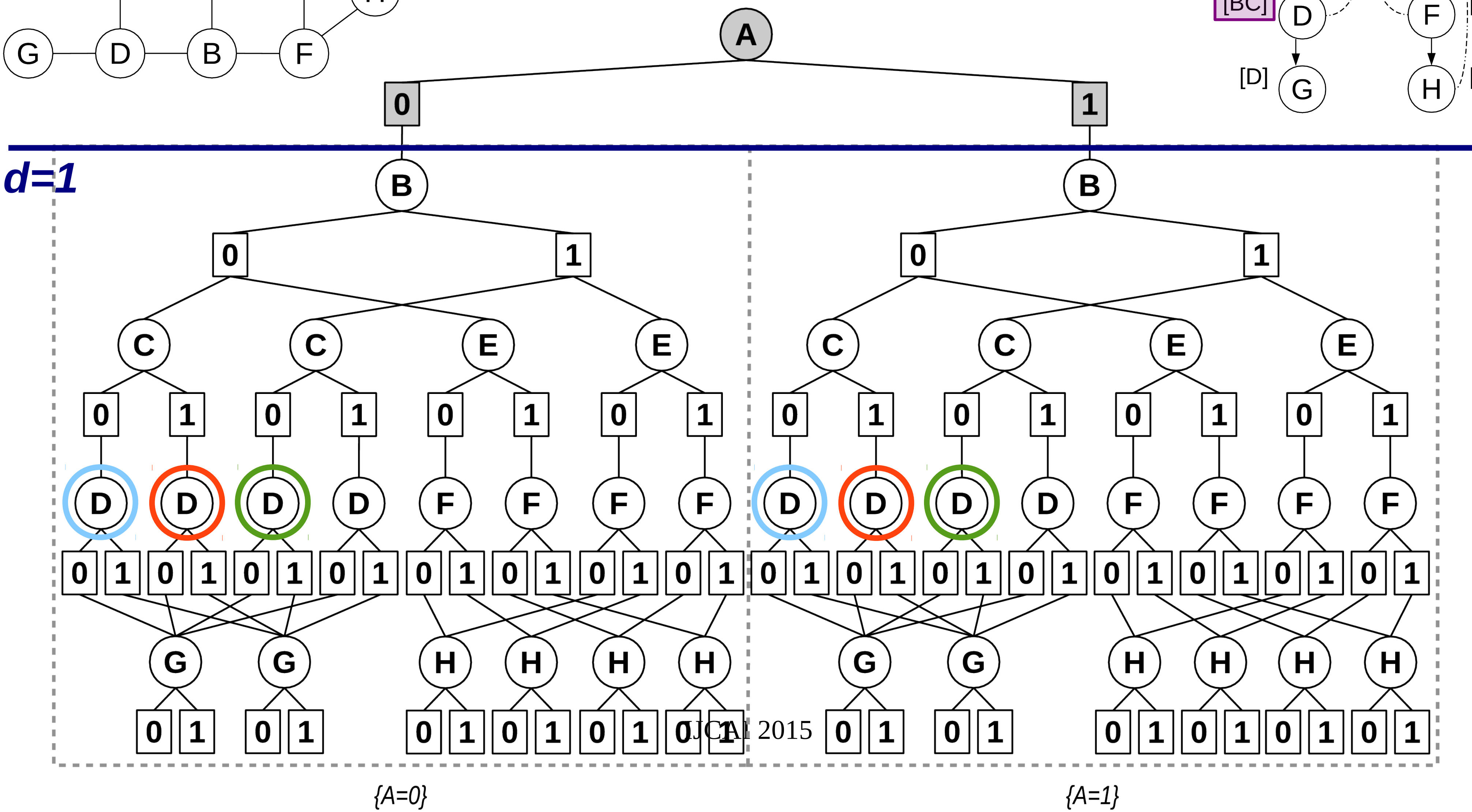




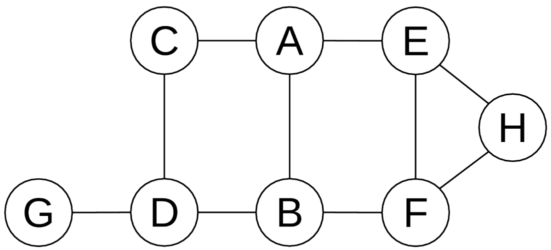
Redundancy Analysis

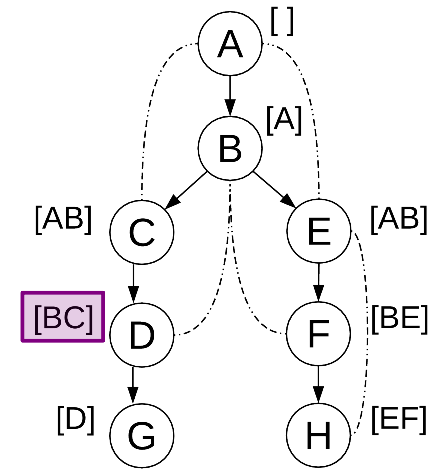


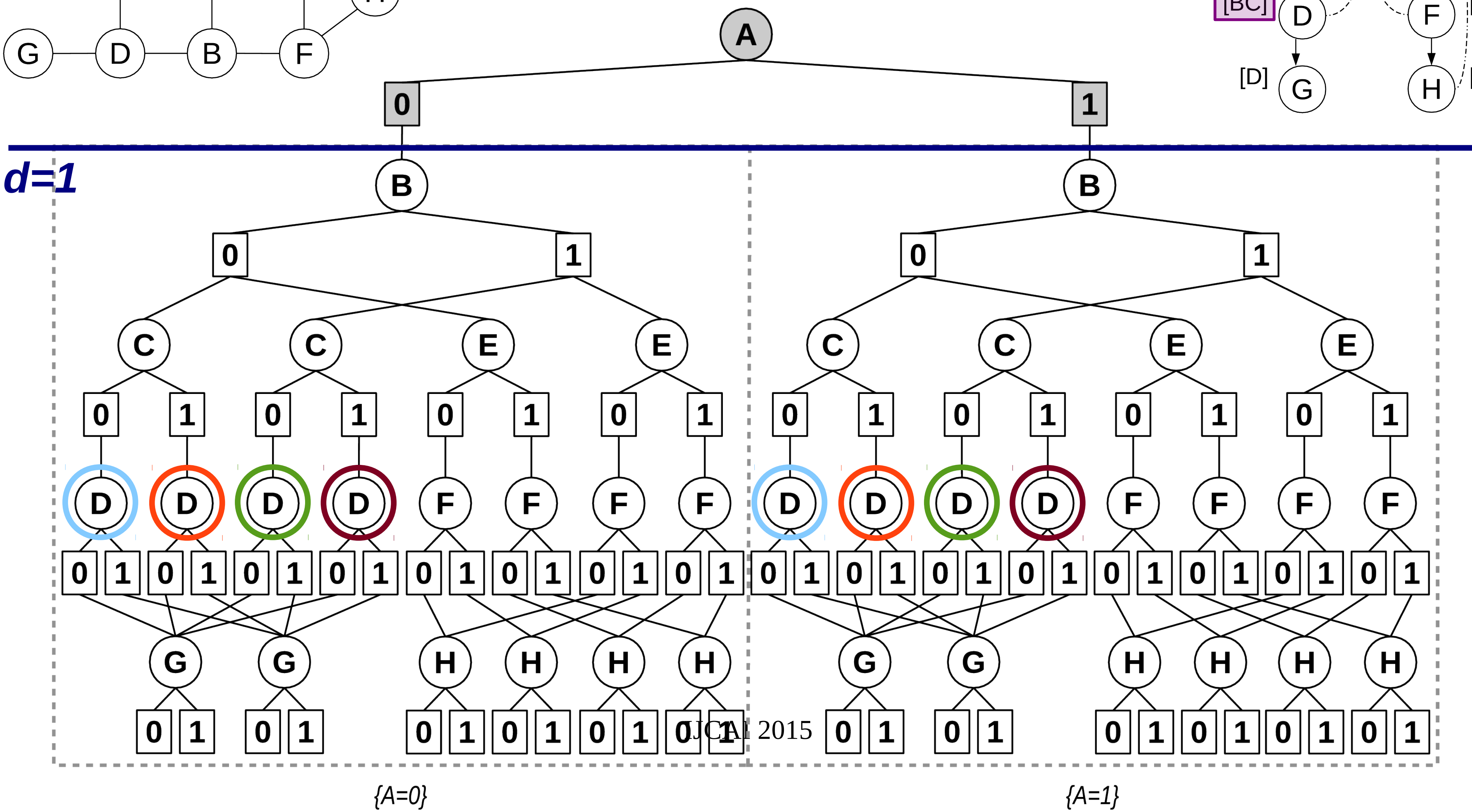




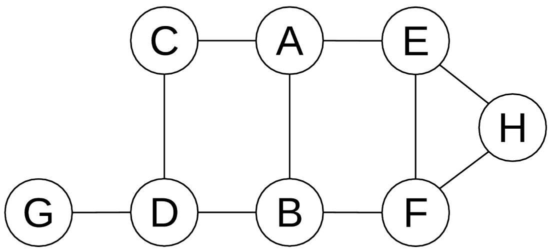
Redundancy Analysis

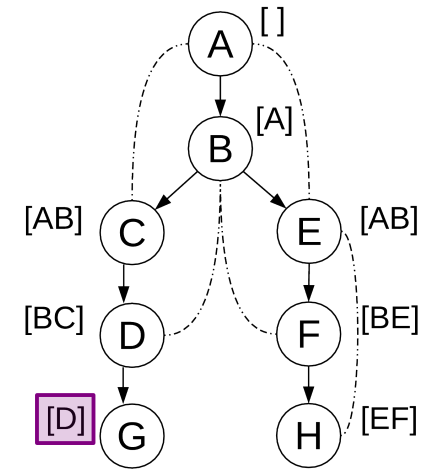


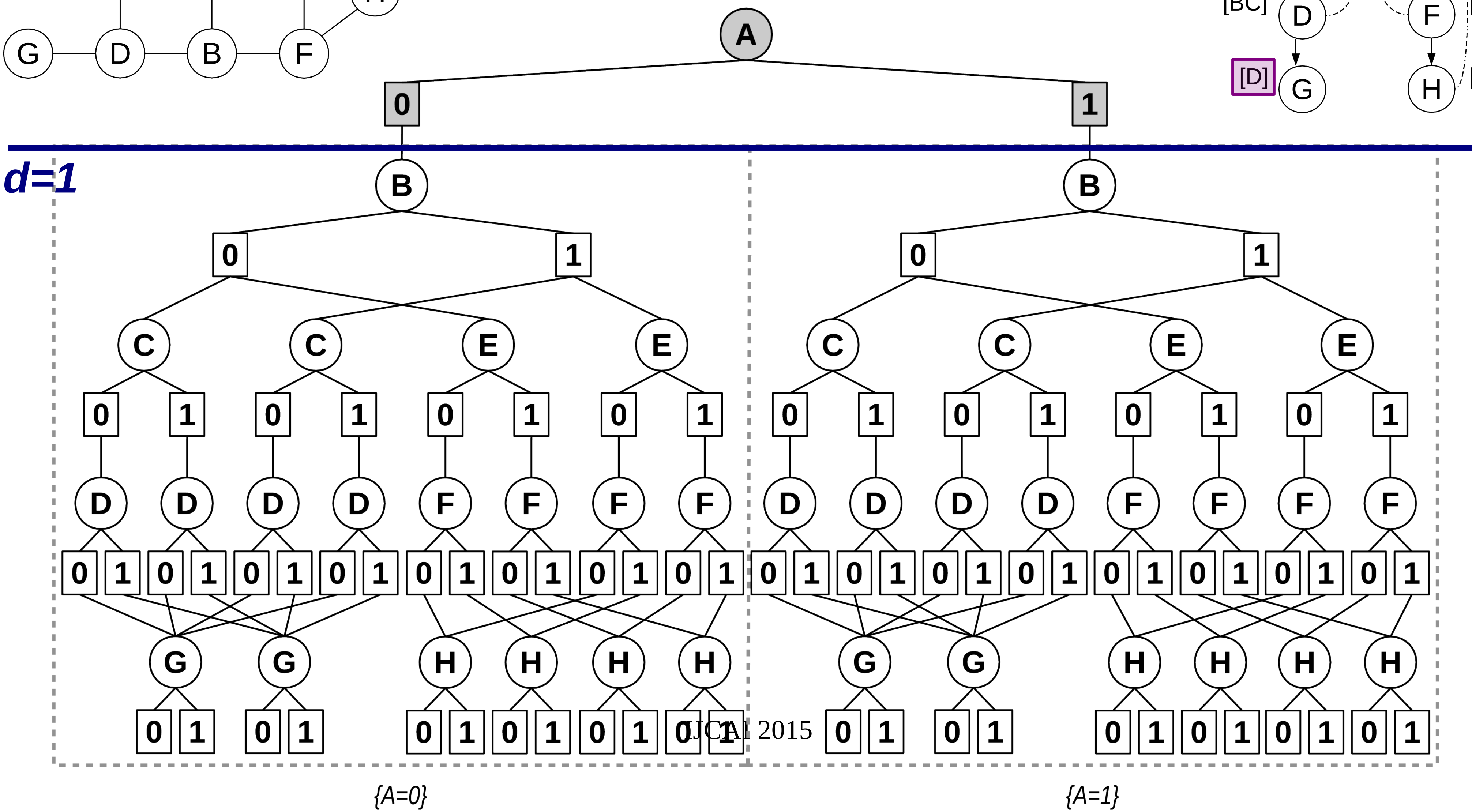




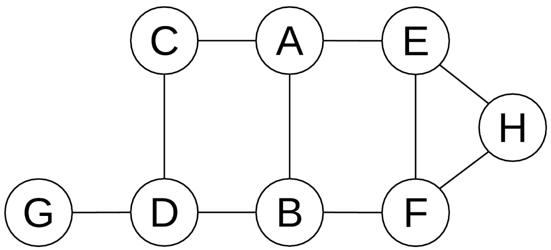
Redundancy Analysis

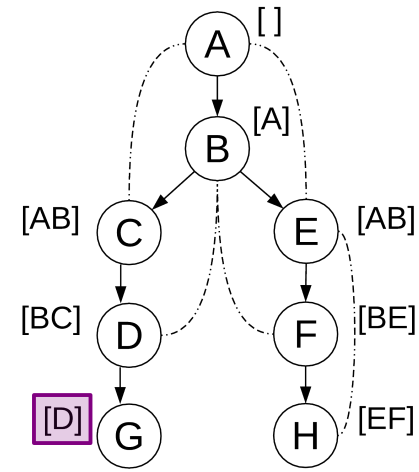


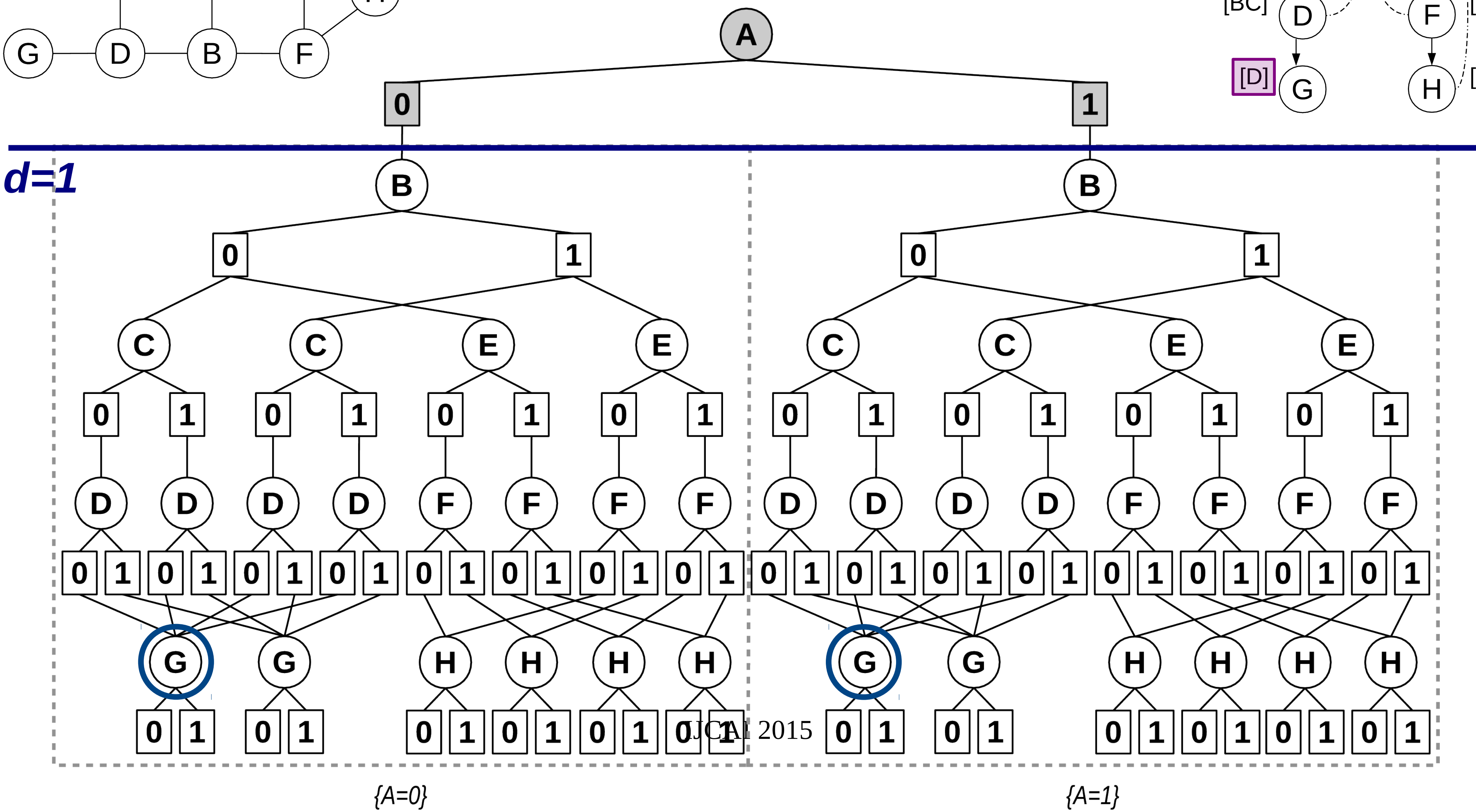




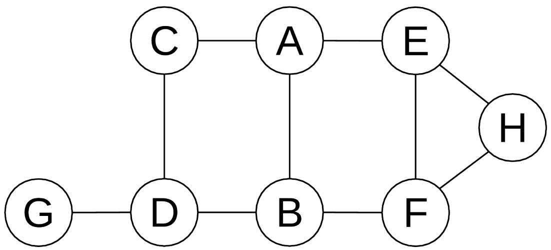
Redundancy Analysis

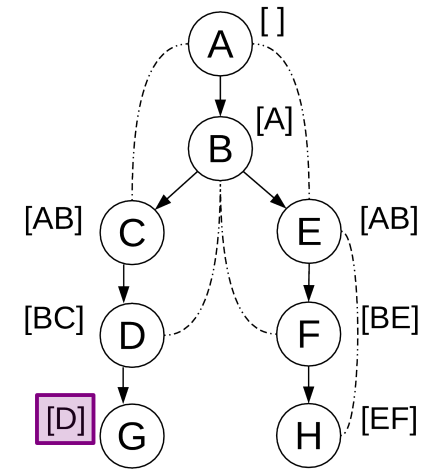


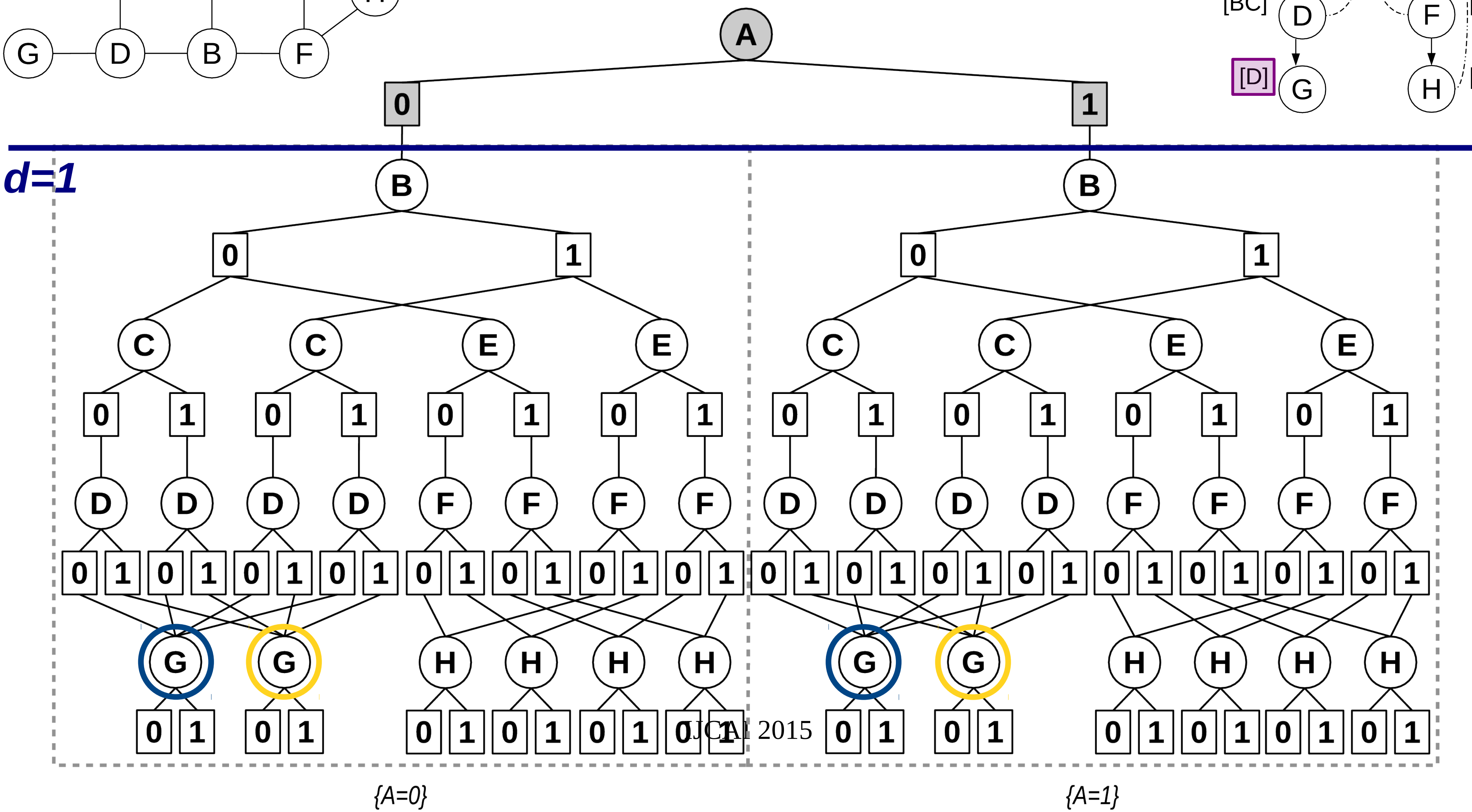




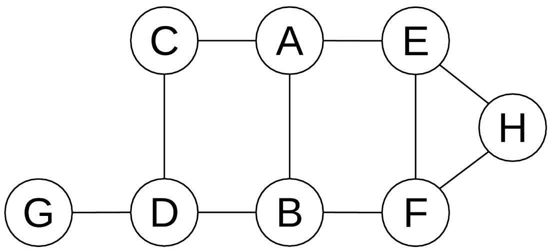
Redundancy Analysis

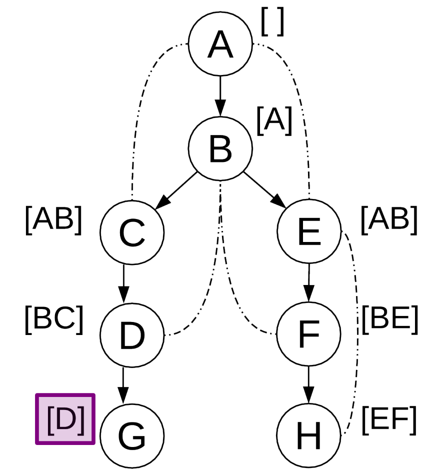


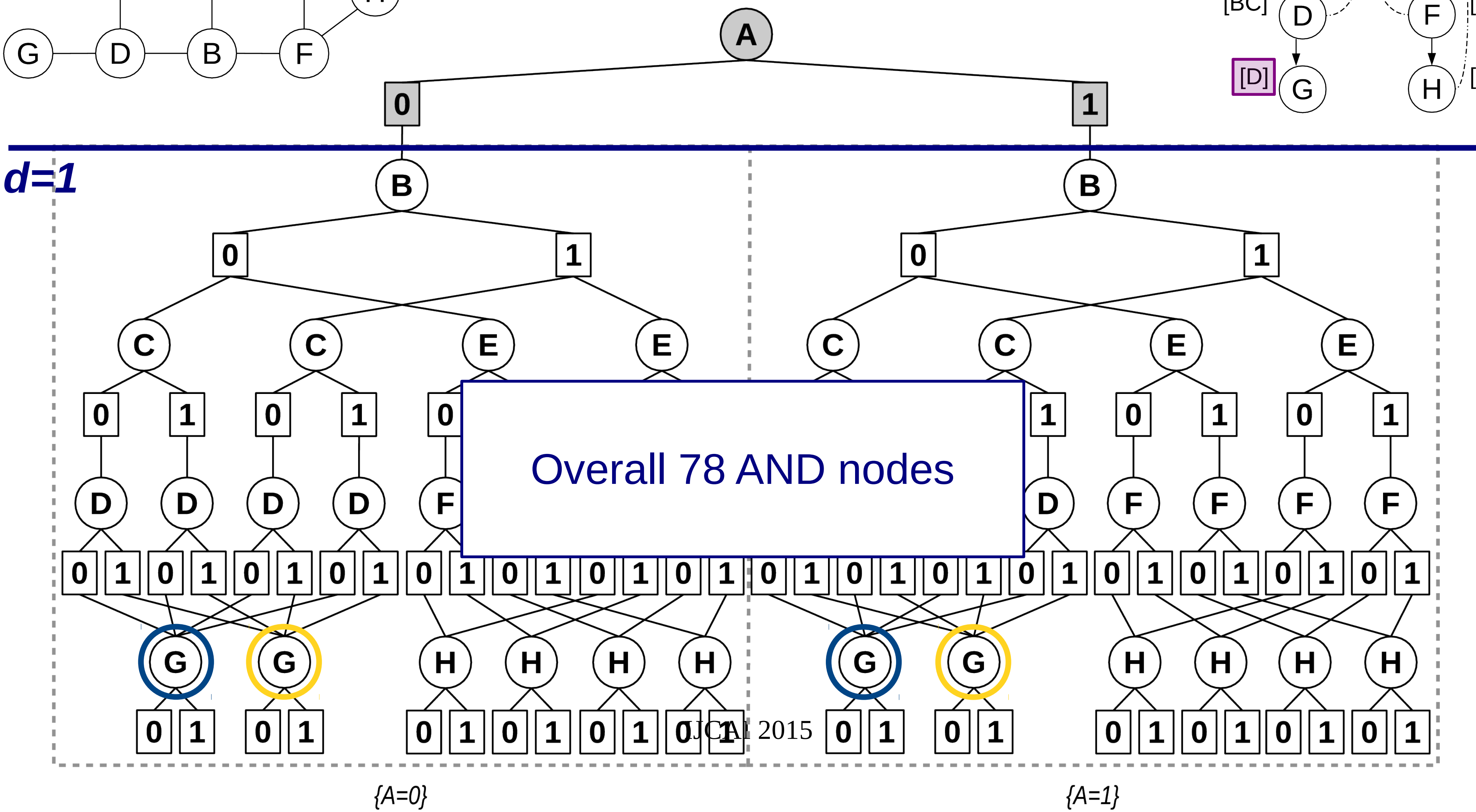




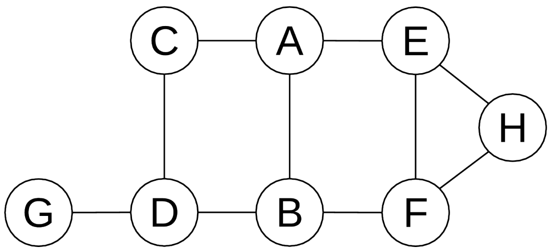
Redundancy Analysis

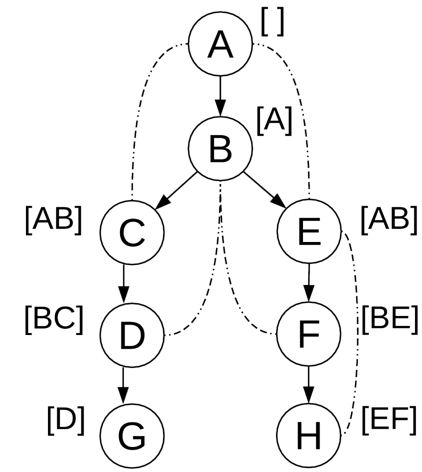


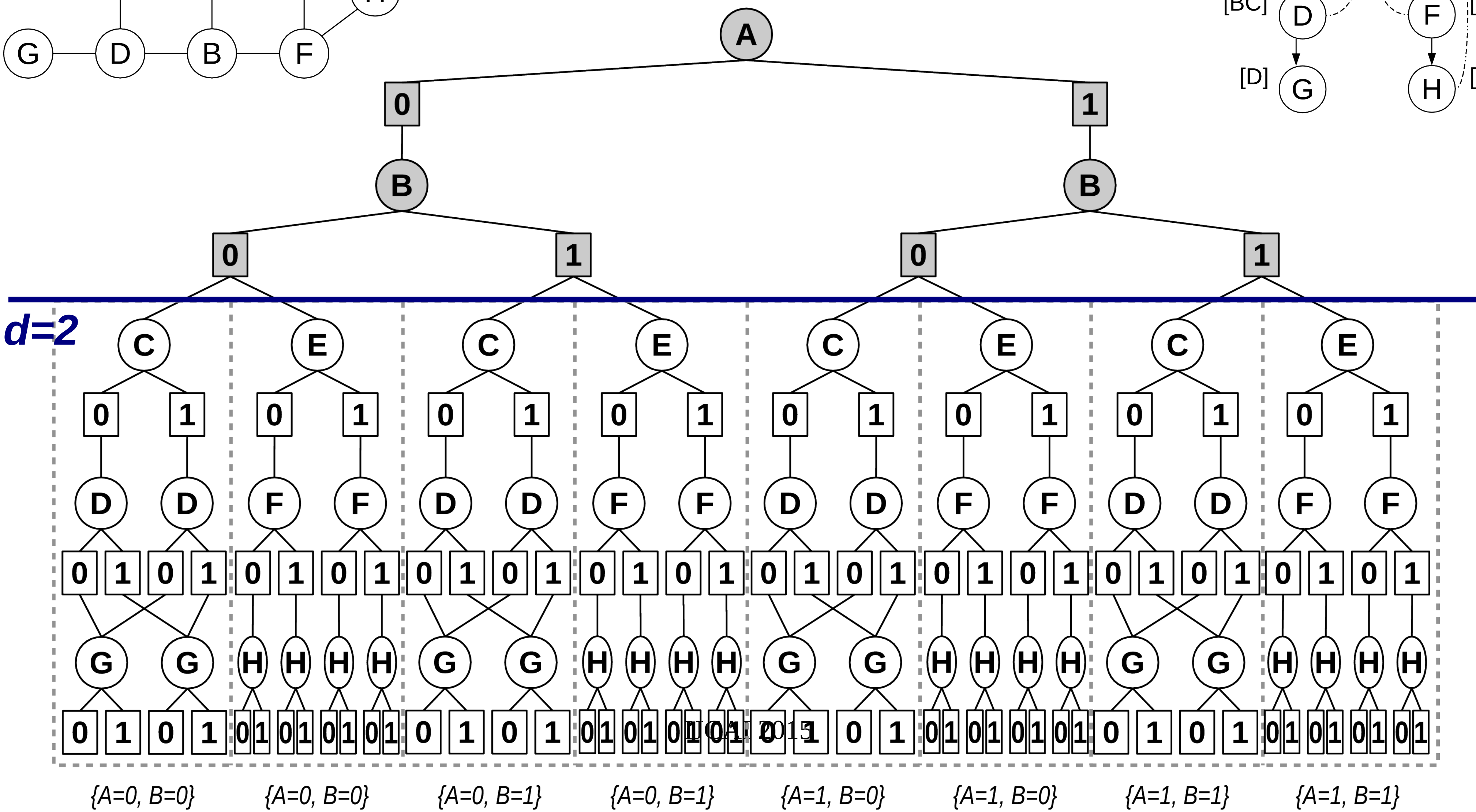




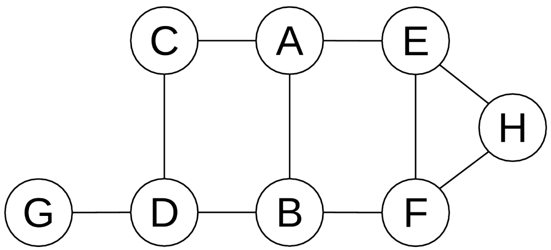
Redundancy Analysis

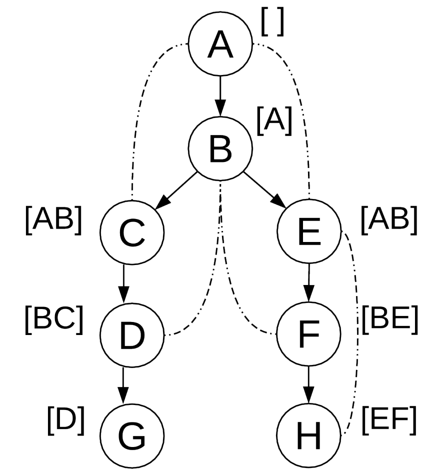


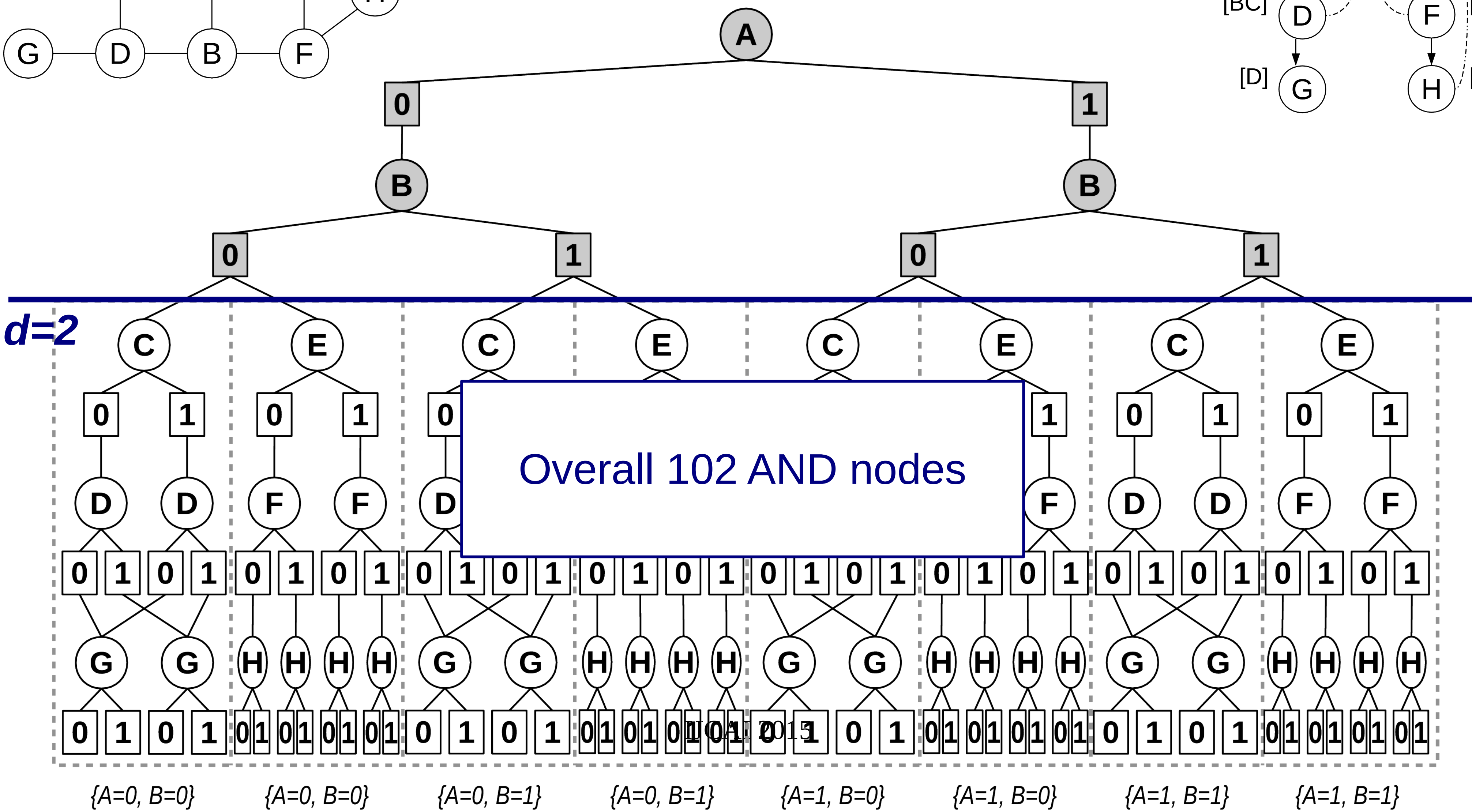




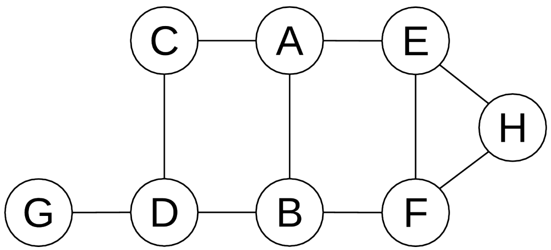
Redundancy Analysis

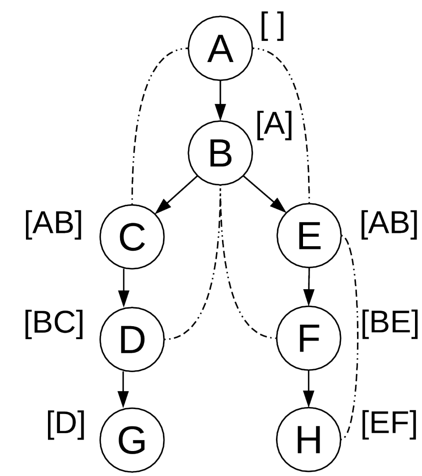


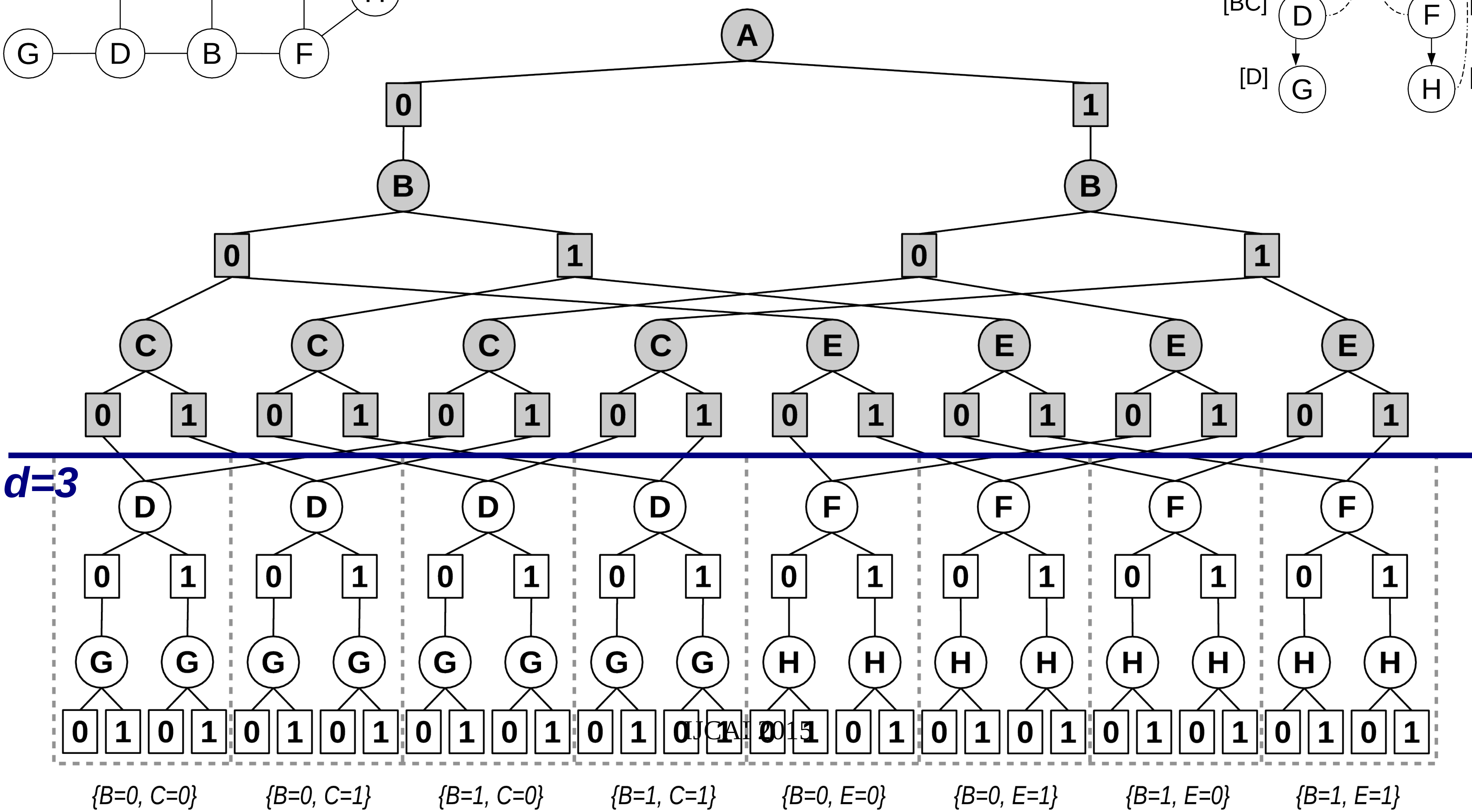




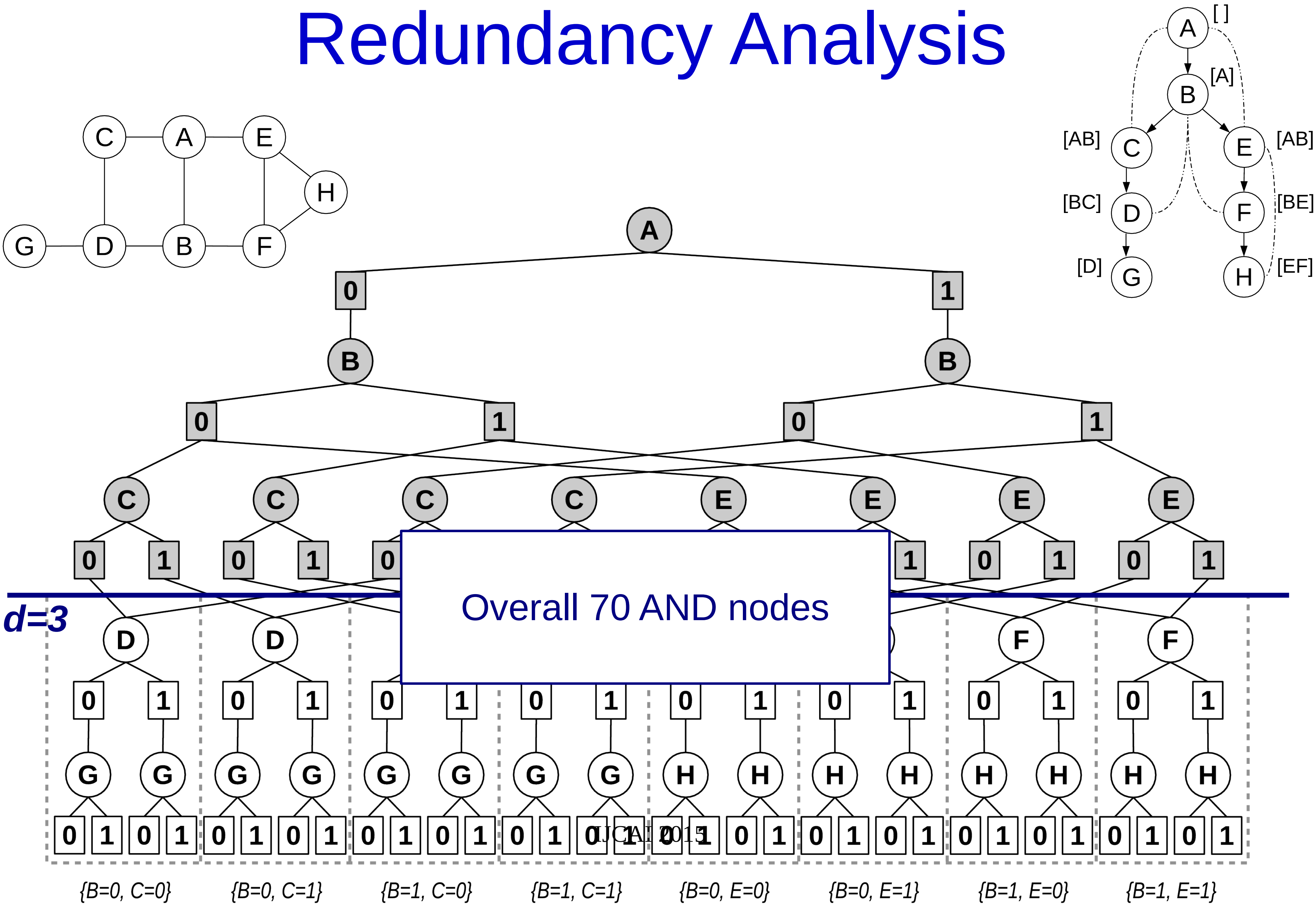
Redundancy Analysis







Redundancy Analysis



Redundancy Quantied

- Definitions:
  - $w_d(X)$ is size of context of X below level d.
  - $\pi_d(X)$ is ancestor of X at level d.
- Underlying parallel search space size SS_{par} :

$$SS_{par}(d) \;=\; \sum_{j=0}^{d} \sum_{X' \in L_j} k^{w(X')+1} + \sum_{j=d+1}^{h} \sum_{X' \in L_j} k^{w(\pi_d(X'))+w_d(X')+1}$$

Redundancy Quantied

- Definitions:
  - $w_d(X)$ is size of context of X below level d.
  - $\pi_d(X)$ is ancestor of X at level d.
- Underlying parallel search space size SS_{par} :

Conditioning space

$$SS_{par}(d) = \boxed{\sum_{j=0}^{d} \sum_{X' \in L_j} k^{w(X')+1}} + \sum_{j=d+1}^{h} \sum_{X' \in L_j} k^{w(\pi_d(X'))+w_d(X')+1}$$

Redundancy Quantied

- Definitions:
  - $w_d(X)$ is size of context of X below level d.
  - $\pi_d(X)$ is ancestor of X at level d.
- Underlying parallel search space size SS_{par} :

$$SS_{par}(d) = \underbrace{\sum_{j=0}^{d} \sum_{X' \in L_j} k^{w(X')+1}}_{\text{Conditioning space}} + \underbrace{\sum_{j=d+1}^{h} \sum_{X' \in L_j} k^{w(\pi_d(X'))+w_d(X')+1}}_{\text{Overall subproblem space}}$$

Redundancy Quantied

- Definitions:
  - $w_d(X)$ is size of context of X below level d.
  - $\pi_d(X)$ is ancestor of X at level d.
- Underlying parallel search space size SS_{par} :

Conditioning space *Overall subproblem space*

$$SS_{par}(d) = \sum_{j=0}^{d} \sum_{X' \in L_j} k^{w(X')+1} + \sum_{j=d+1}^{h} \sum_{X' \in L_j} k^{\boxed{w(\pi_d(X'))} + w_d(X')+1}$$

Redundancy Quantied

- Definitions:
  - $w_d(X)$ is size of context of X below level d.
  - $\pi_d(X)$ is ancestor of X at level d.
- Underlying parallel search space size SS_{par} :

Conditioning space — *Overall subproblem space*

$$SS_{par}(d) = \underbrace{\sum_{j=0}^{d} \sum_{X' \in L_j} k^{w(X')+1}}_{\text{Conditioning space}} + \underbrace{\sum_{j=d+1}^{h} \sum_{X' \in L_j} k^{\boxed{w(\pi_d(X'))} + w_d(X')+1}}_{\text{Overall subproblem space}}$$

  - $SS_{par}(0) = SS_{par}(h) = SS_{seq}$.
  - $SS_{par}(d) \geq SS_{par}(0)$ for all d .

Redundancy vs. Parallelism

- Assume parallelism with sufficient CPUs.
  - Consider conditioning space + max. subproblem.

Example revisited:

	d					
	0	1	2	3	4	5
parallel space $SS_{par}(d)$	50	78	102	70	50	50
conditioning space	0	2	6	22	38	50
no. of subproblems	1	2	8	8	6	0
max. parallel subproblem	50	38	14	6	2	–
cond. space + max. subprob	50	40	20	28	40	50

Redundancy vs. Parallelism

- Assume parallelism with sufficient CPUs.
  - Consider conditioning space + max. subproblem.

Example revisited:

	d					
	0	1	2	3	4	5
parallel space $SS_{par}(d)$	50	78	102	70	50	50
conditioning space	0	2	6	22	38	50
no. of subproblems	1	2	8	8	6	0
max. parallel subproblem	50	38	14	6	2	–
cond. space + max. subprob	50	40	20	28	40	50

Redundancy vs. Parallelism

- Assume parallelism with sufficient CPUs.
  - Consider conditioning space + max. subproblem.

Example revisited:

	d					
	0	1	2	3	4	5
parallel space $SS_{par}(d)$	50	78	$\boxed{102}$	70	50	50
conditioning space	0	2	6	22	38	50
no. of subproblems	1	2	8	8	6	0
max. parallel subproblem	50	38	14	6	2	–
cond. space + max. subprob	50	40	$\boxed{20}$	28	40	50

Redundancy vs. Parallelism

- Assume parallelism with sufficient CPUs.
  - Consider conditioning space + max. subproblem.

Example revisited:

	d					
	0	1	2	3	4	5
parallel space $SS_{par}(d)$	50	78	$\boxed{102}$	70	50	50
conditioning space	0	2	6	22	38	50
no. of subproblems	1	2	8	8	6	0
max. parallel subproblem	50	38	14	6	2	–
cond. space + max. subprob	50	40	$\boxed{20}$	28	40	50

- <u>But:</u> doesn't capture explored search space.
  - Can pruning compensate for redundancies?

Empirical Evaluation

- Parallel experiments over 75 benchmarks.
  - Instances from four classes, with varying *i*-bound.
  - T_{seq} from under 1 hour to over 2 weeks.
  - Run with 20, 100, and 500 parallel CPUs.

Empirical Evaluation

- Parallel experiments over 75 benchmarks.
  - Instances from four classes, with varying *i*-bound.
  - T_{seq} from under 1 hour to over 2 weeks.
  - Run with 20, 100, and 500 parallel CPUs.
- Experimental Methodology:
  - Apply different fixed-depth cutoff depths *d*.
  - Use subproblem count *p* as var-depth input.

Empirical Evaluation

- Parallel experiments over 75 benchmarks.
  - Instances from four classes, with varying *i*-bound.
  - T_{seq} from under 1 hour to over 2 weeks.
  - Run with 20, 100, and 500 parallel CPUs.
- Experimental Methodology:
  - Apply different fixed-depth cutoff depths *d*.
  - Use subproblem count *p* as var-depth input.
- ~91 thousand CPU hours – over 10 years!
  - Over 1400 parallel runs.
  - Can only summarize some aspects here.

Example Results

- Record overall parallel runtime / speedup.
  - Lots of data!

				Cutoff depth d											
instance	i	T_{seq}	#cpu	2		4		6		8		10		12	
				fix	var	fix	var	fix	var	fix	var	fix	var	fix	var
lF3-15-59	19	43307		*(p=4)*		*(p=20)*		*(p=80)*		*(p=476)*		*(p=1830)*		*(p=6964)*	
n=3730			20	15858	15694	5909	5470	3649	2845	2744	2501	3482	3505	7222	7238
k=3, w=31			100	15858	15694	5909	5470	3434	2247	1494	723	928	741	1540	1536
h=84			500	15858	15694	5909	5470	3434	2247	1414	573	692	260	415	399
ped44	6	95830		*(p=4)*		*(p=16)*		*(p=112)*		*(p=560)*		*(p=2240)*		*(p=8960)*	
n=811			20	26776	26836	9716	9481	6741	6811	7959	7947	10103	9763	12418	12472
k=4, w=25			100	26776	26836	9716	9481	2344	3586	1799	1700	2126	2276	2545	2543
h=65			500	26776	26836	9716	9481	1659	3586	583	886	536	905	569	824
ped7	6	118383		*(p=4)*		*(p=32)*		*(p=160)*		*(p=640)*		*(p=1280)*		*(p=3840)*	
n=1068			20	35387	58872	12338	58121	9031	8515	9654	7319	8705	7582	8236	7693
k=4, w=32			100	35387	58872	11956	58121	5122	7690	4860	2306	3929	1814	2644	1649
h=90			500	35387	58872	11956	58121	4984	7690	4359	2086	3294	1301	1764	943

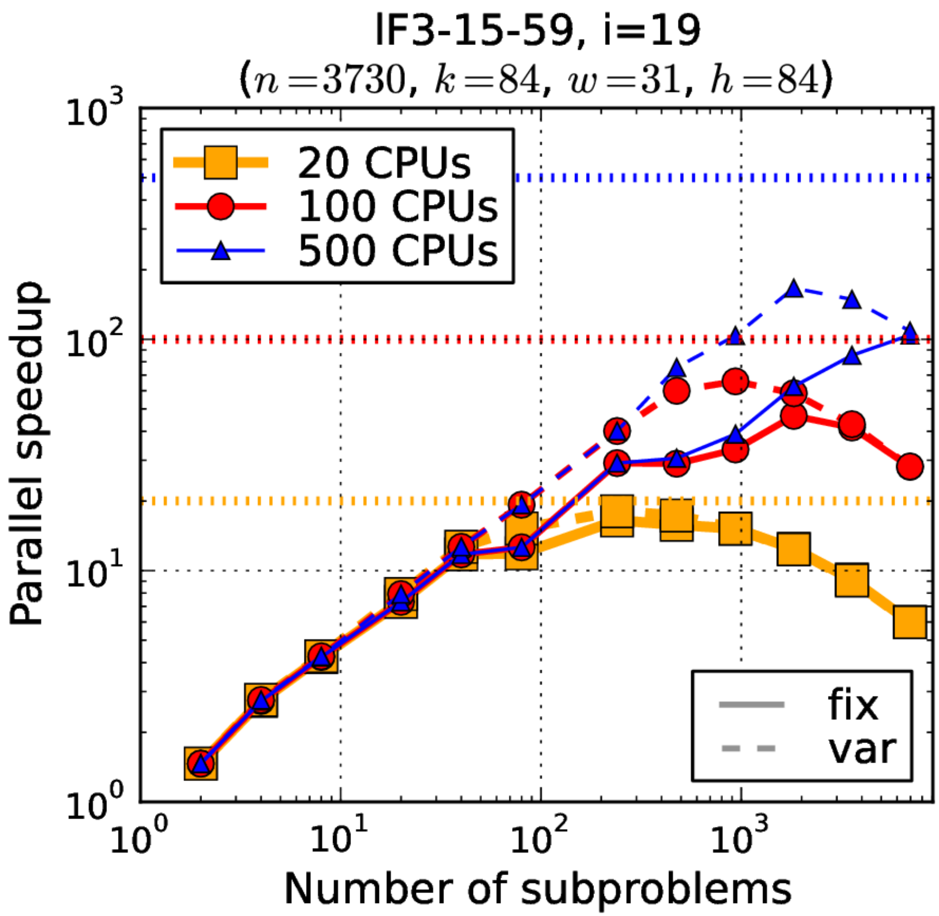
Example Results

- Record overall parallel runtime / speedup.
  - Lots of data!

instance	i	T_{seq}	#cpu	Cutoff depth d: 2 fix	2 var	4 fix	4 var	6 fix	6 var	8 fix	8 var	10 fix	10 var	12 fix	12 var
lF3-15-59 (n=3730, k=3, w=31, h=84)	19	43307		(p=4)		(p=20)		(p=80)		(p=476)		(p=1830)		(p=6964)	
			20	2.73	2.76	7.33	7.92	11.87	15.22	15.78	17.32	12.44	12.36	6.00	5.98
			100	2.73	2.76	7.33	7.92	12.61	19.27	28.99	59.90	46.67	58.44	28.12	28.19
			500	2.73	2.76	7.33	7.92	12.61	19.27	30.63	75.58	62.58	166.57	104.35	108.54
ped44 (n=811, k=4, w=25, h=65)	6	95830		(p=4)		(p=16)		(p=112)		(p=560)		(p=2240)		(p=8960)	
			20	3.58	3.57	9.86	10.11	14.22	14.07	12.04	12.06	9.49	9.82	7.72	7.68
			100	3.58	3.57	9.86	10.11	40.88	26.72	53.27	56.37	45.08	42.10	37.65	37.68
			500	3.58	3.57	9.86	10.11	57.76	26.72	164.37	108.16	178.79	105.89	168.42	116.30
ped7 (n=1068, k=4, w=32, h=90)	6	118383		(p=4)		(p=32)		(p=160)		(p=640)		(p=1280)		(p=3840)	
			20	3.35	2.01	9.59	2.04	13.11	13.90	12.26	16.17	13.60	15.61	14.37	15.39
			100	3.35	2.01	9.90	2.04	23.11	15.39	24.36	51.34	30.13	65.26	44.77	71.79
			500	3.35	2.01	9.90	2.04	23.75	15.39	27.16	56.75	35.94	90.99	67.11	125.54

Example: LargeFam3-15-59, *i*=19

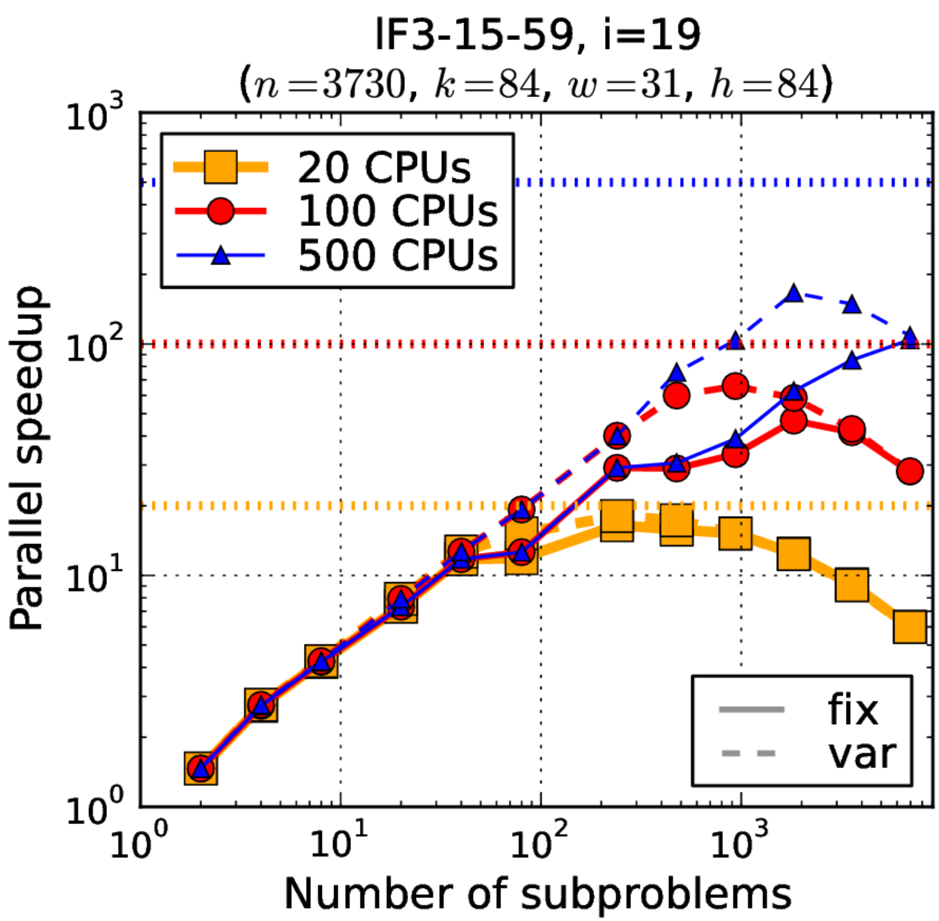
- Sequential runtime
  T_{seq} = 43,307 sec.



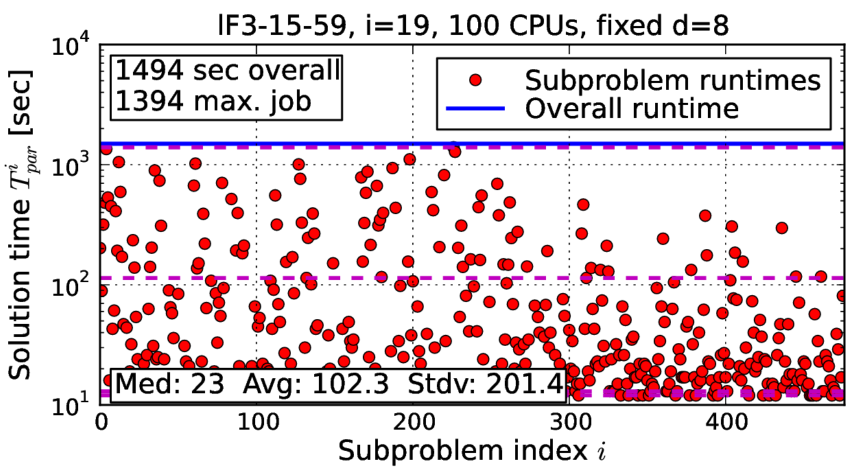
)15

Example: LargeFam3-15-59, *i*=19

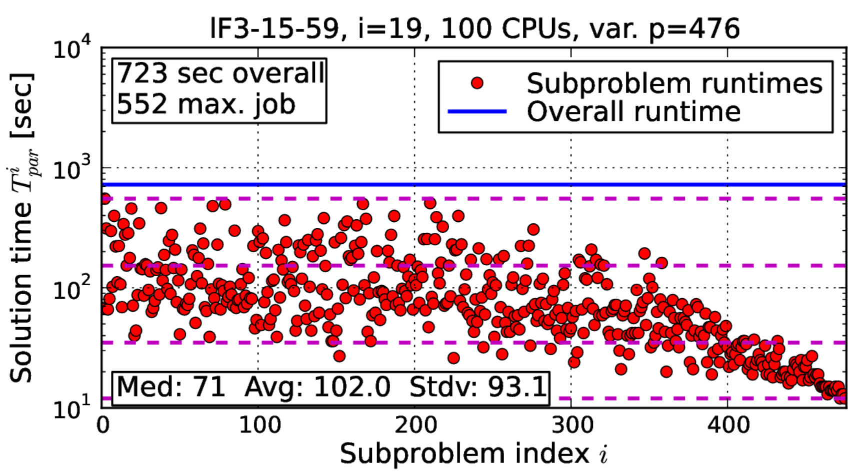
- Sequential runtime
  T_{seq} = 43,307 sec.



(Fixed-depth)

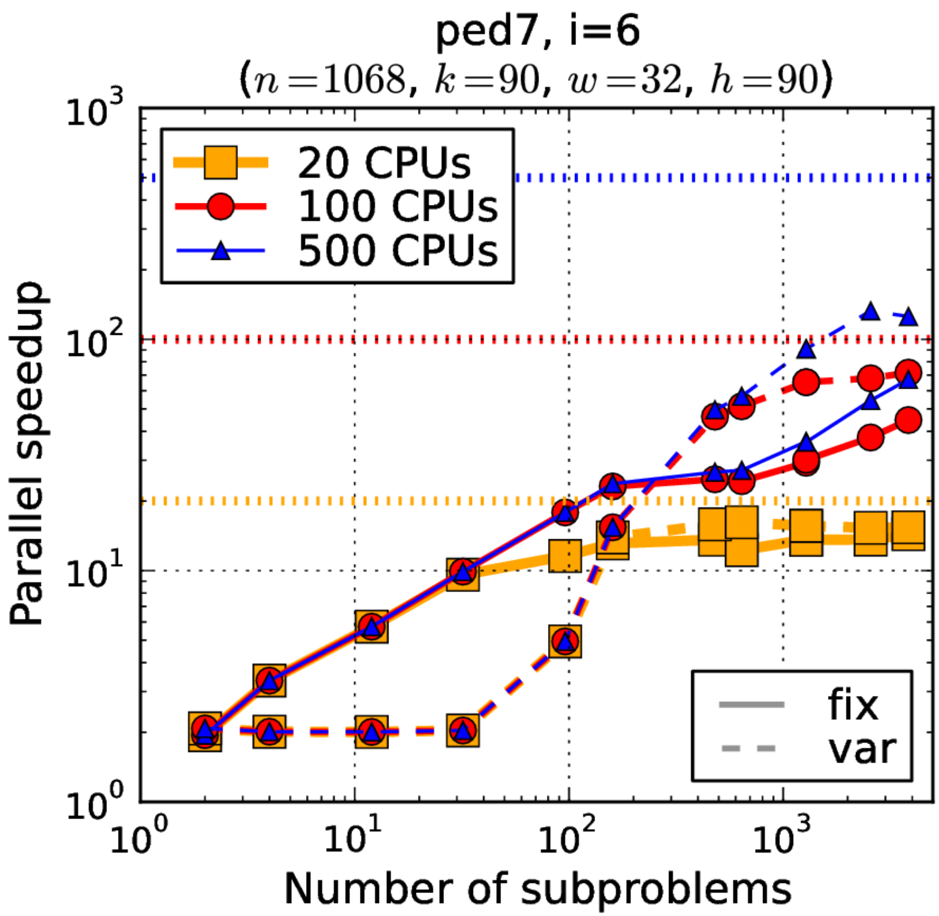


(Variable-depth)



Example: Pedigree7, *i*=6

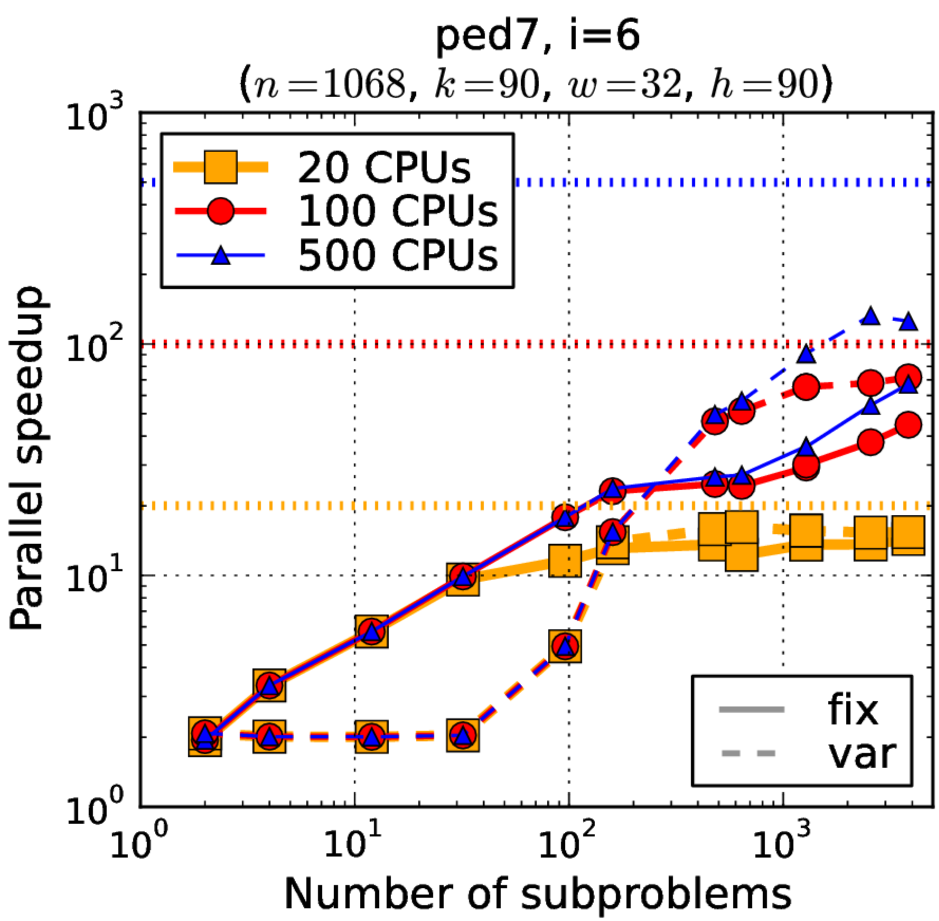
- Sequential runtime
  T_{seq} = 118,383 sec.



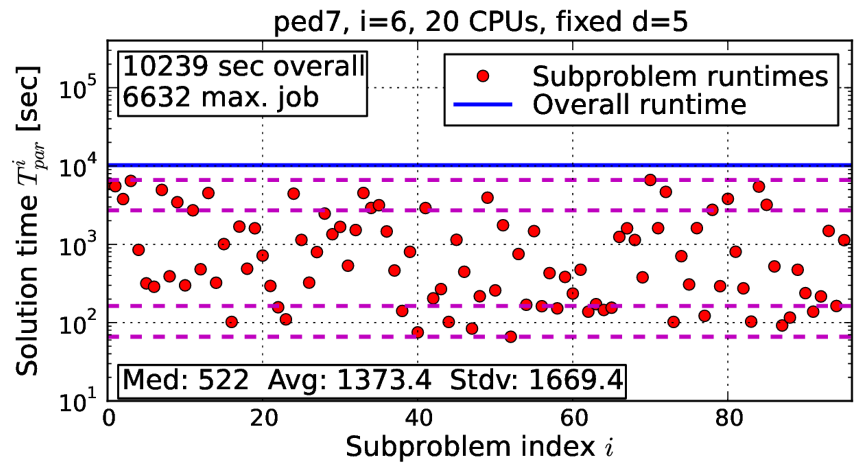
)15

Example: Pedigree7, *i*=6

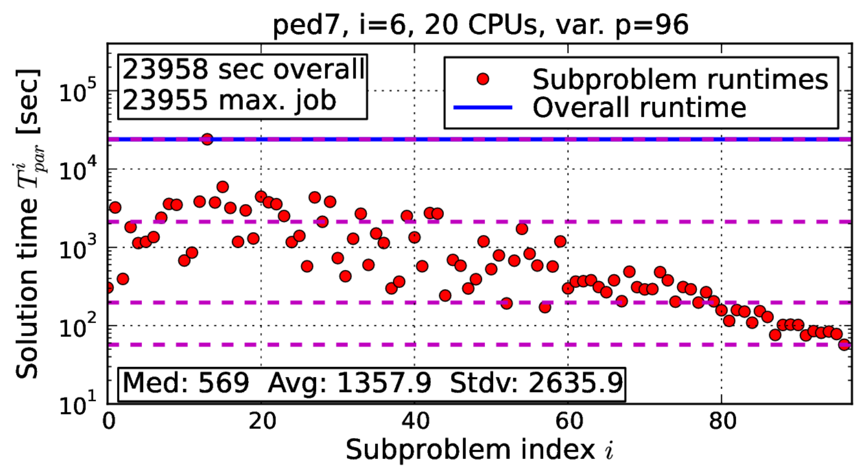
- Sequential runtime T_{seq} = 118,383 sec.



(Fixed-depth)

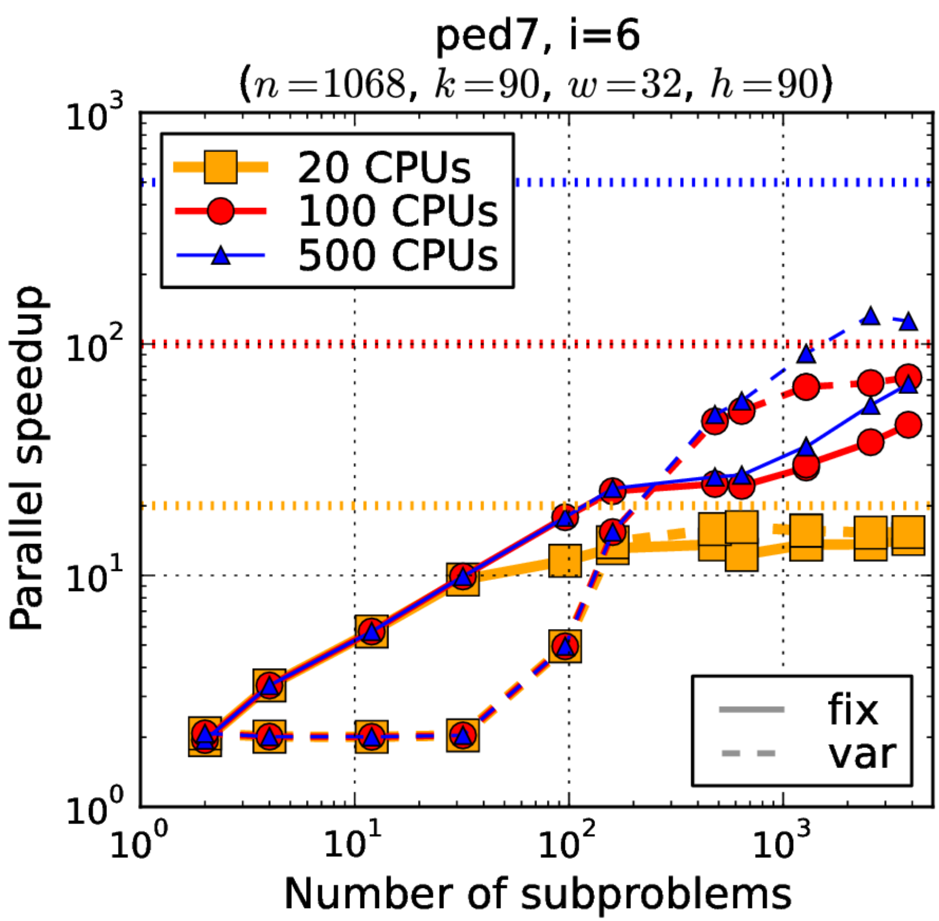


(Variable-depth)

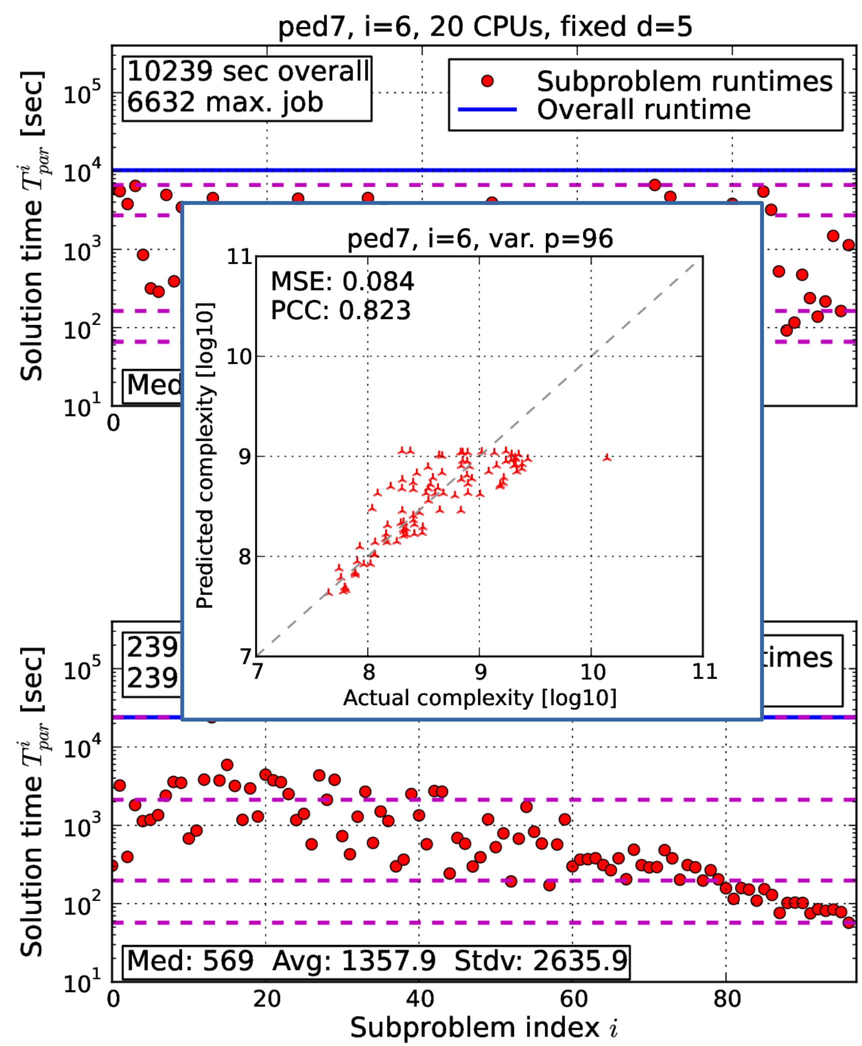


Example: Pedigree7, *i*=6

- Sequential runtime
  T_{seq} = 118,383 sec.

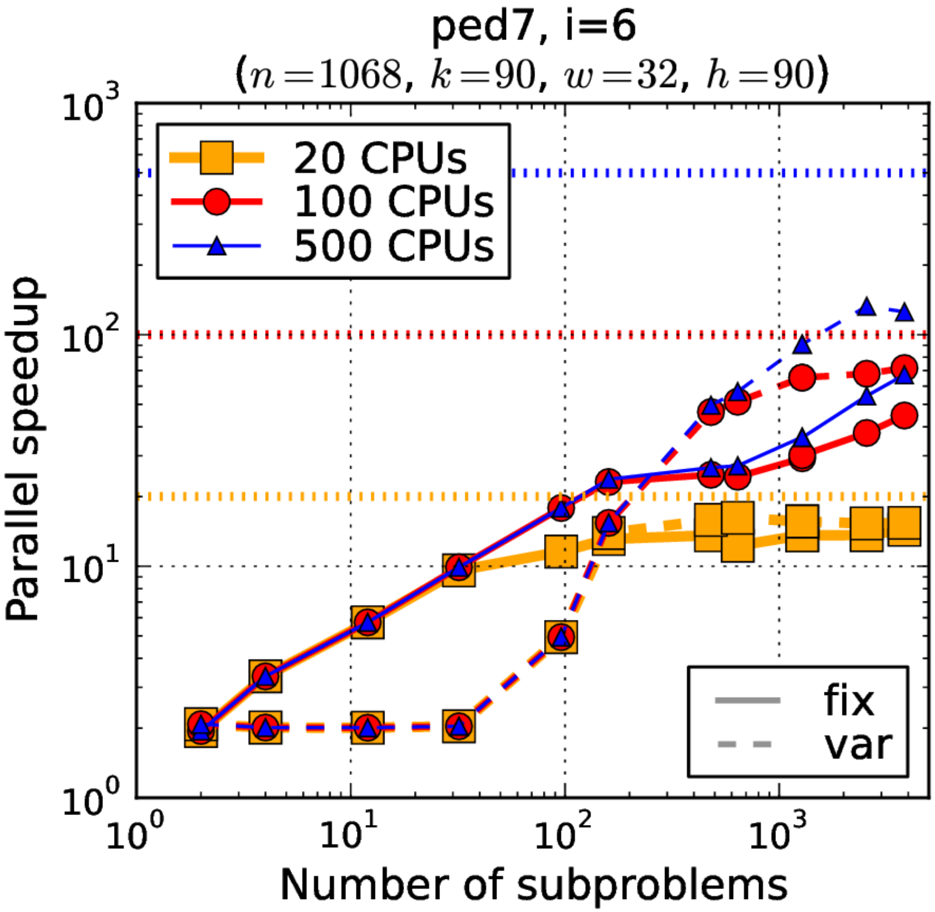


(Fixed-depth)



Example: Pedigree7, *i*=6

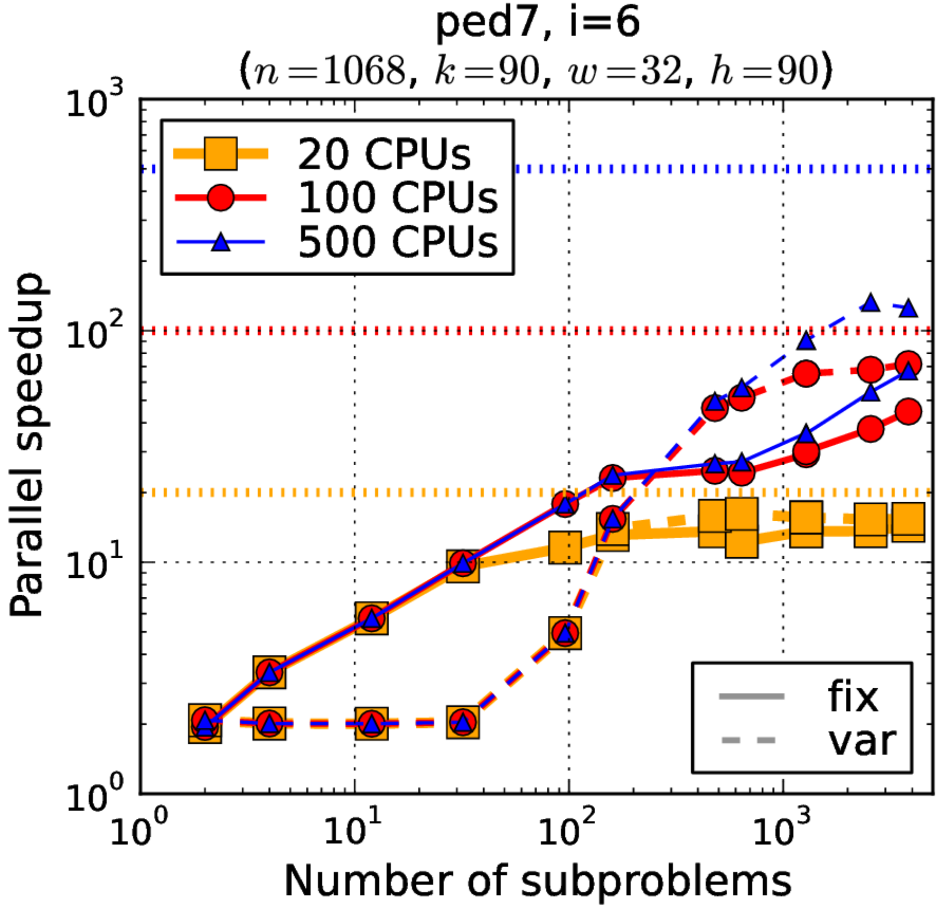
- Sequential runtime
  T_{seq} = 118,383 sec.



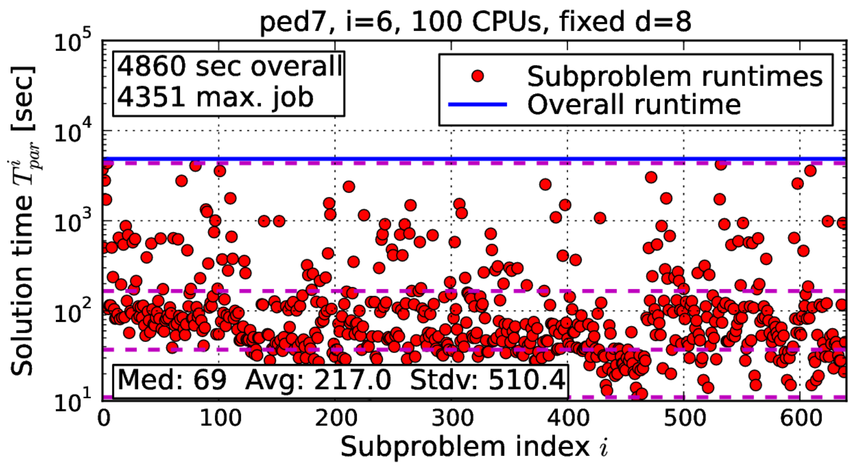
)15

Example: Pedigree7, *i*=6

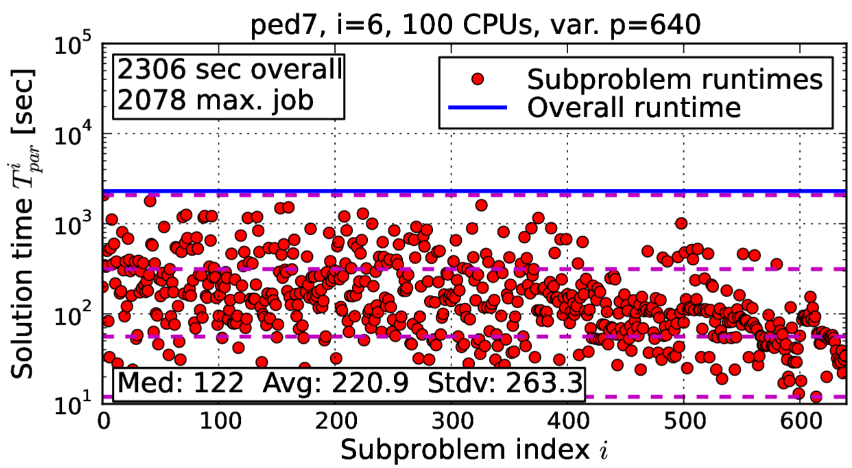
- Sequential runtime
  T_{seq} = 118,383 sec.



(Fixed-depth)

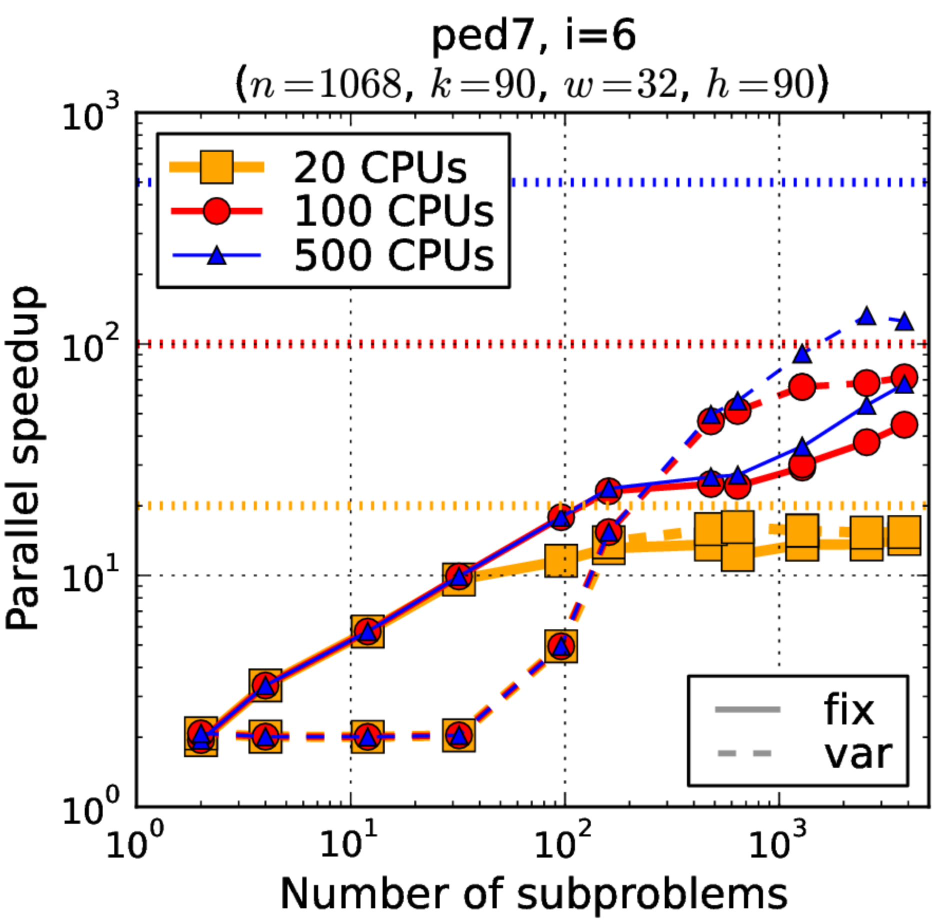


(Variable-depth)

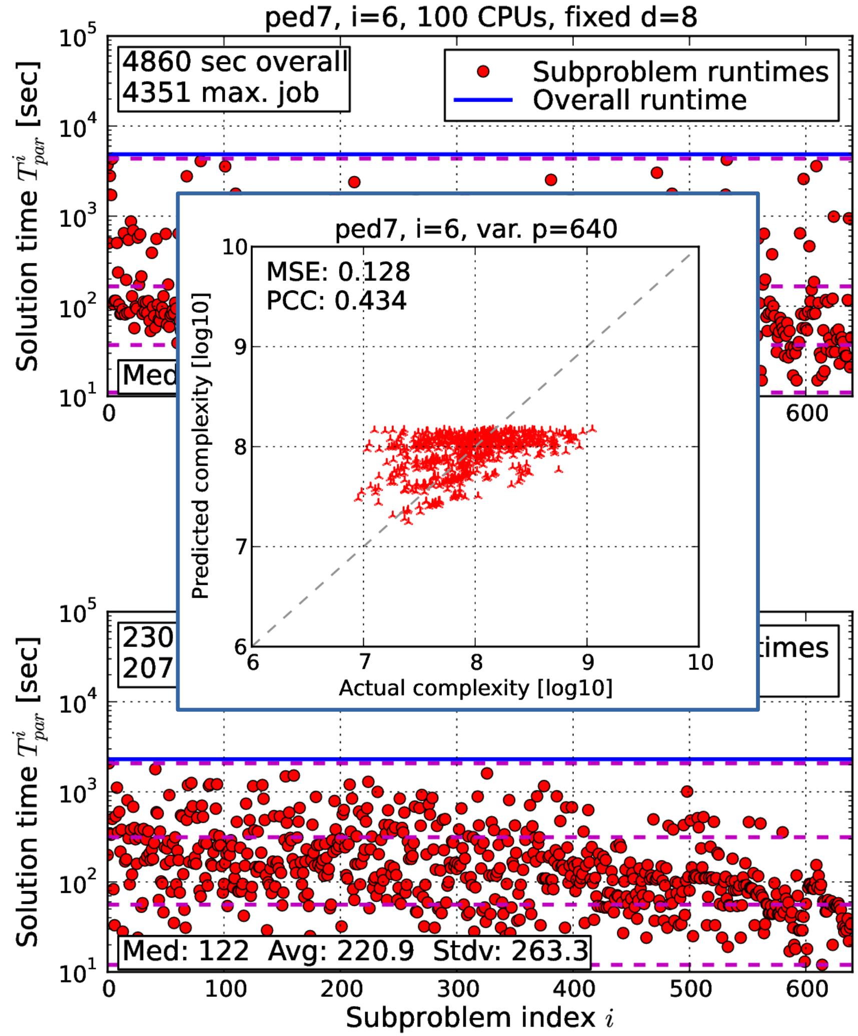


Example: Pedigree7, *i*=6

- Sequential runtime
  T_{seq} = 118,383 sec.

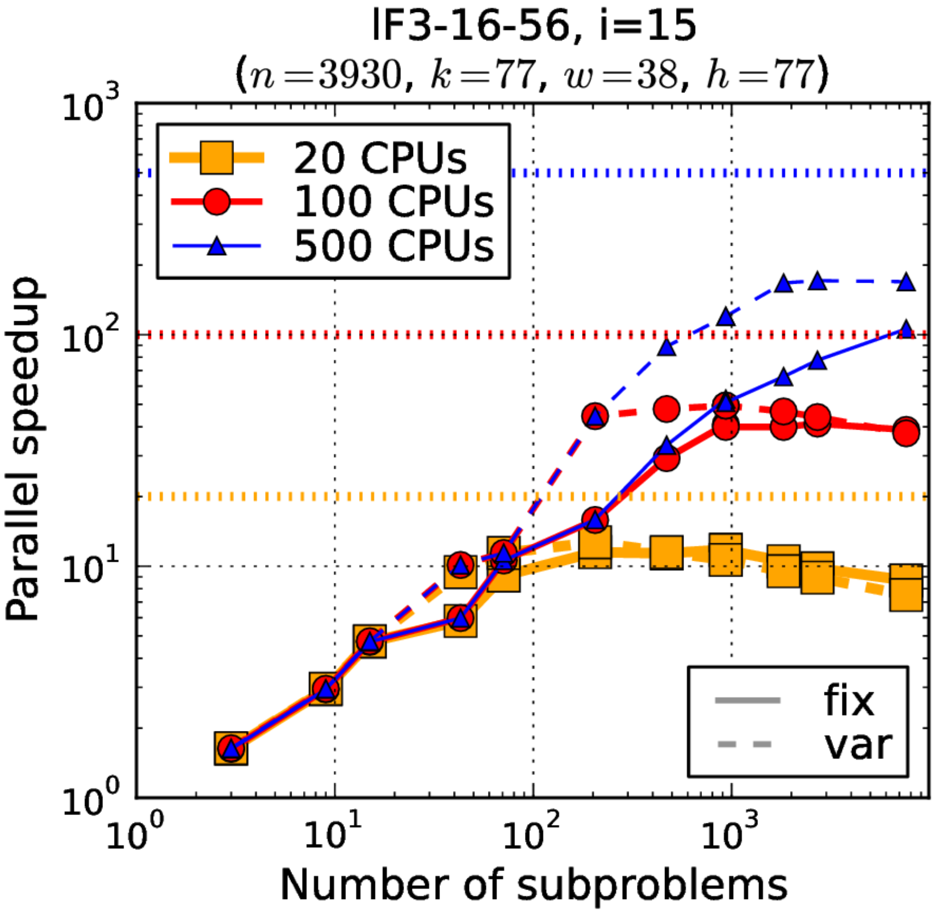


(Fixed-depth)



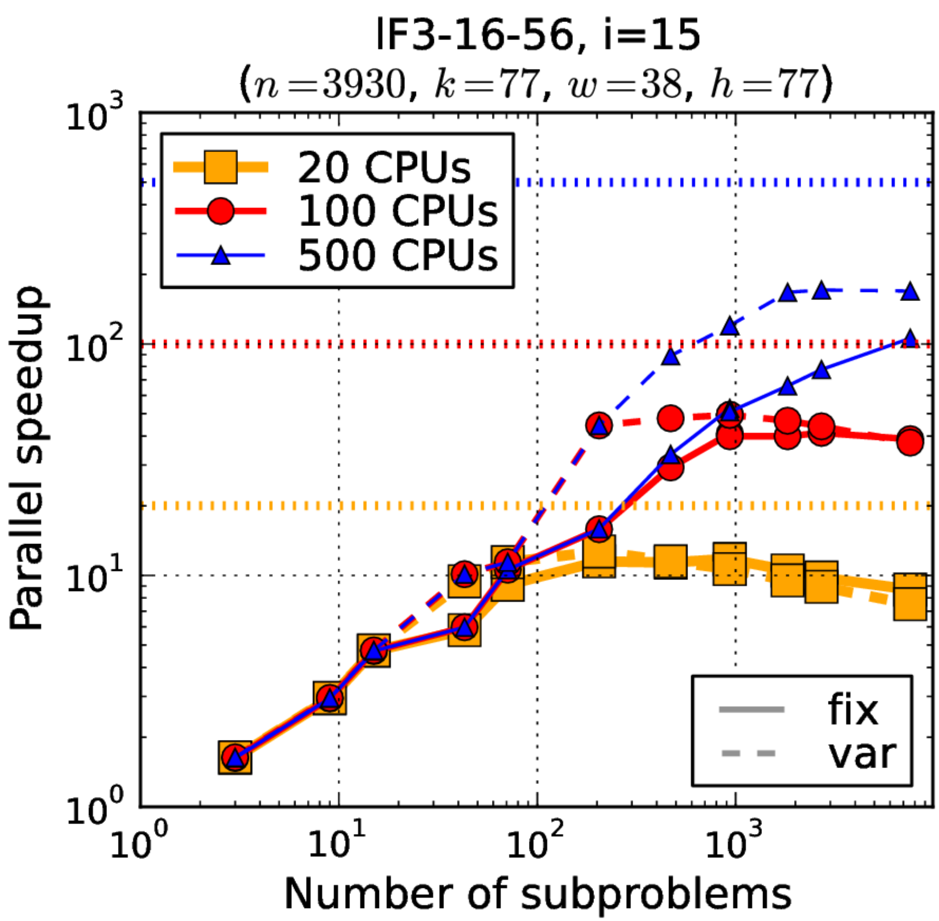
Example: LargeFam3-16-56, *i*=15

- Sequential runtime
  T_{seq} = 1,891,710 sec.

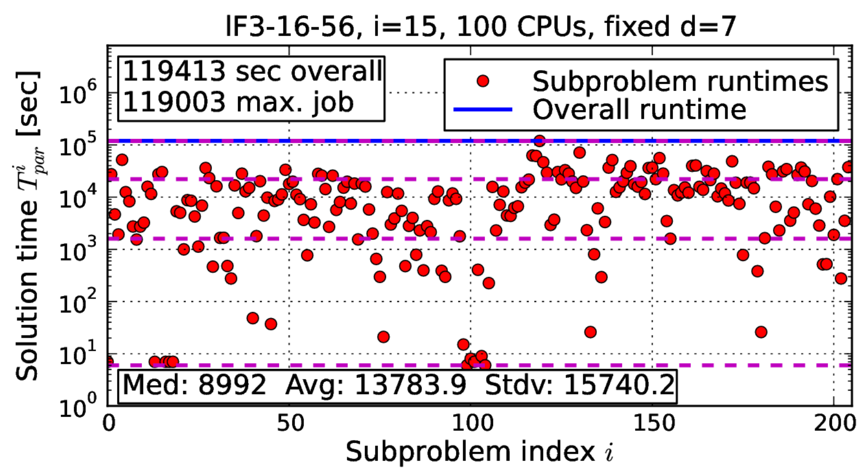


Example: LargeFam3-16-56, *i*=15

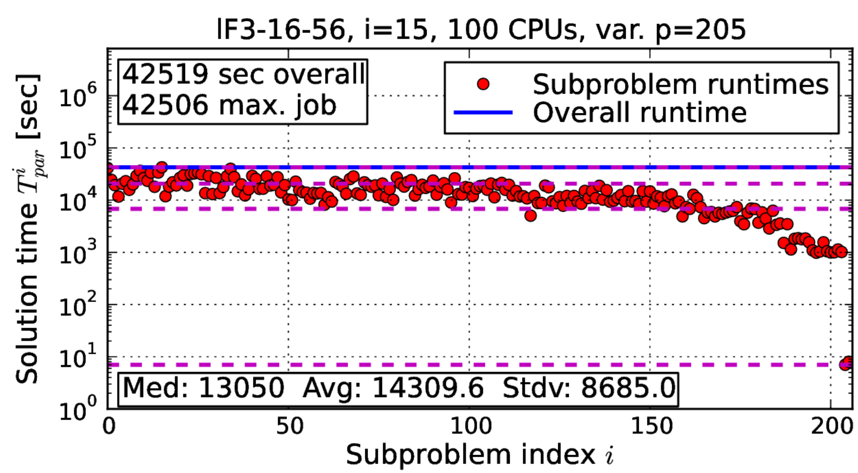
- Sequential runtime
  T_{seq} = 1,891,710 sec.



(Fixed-depth)

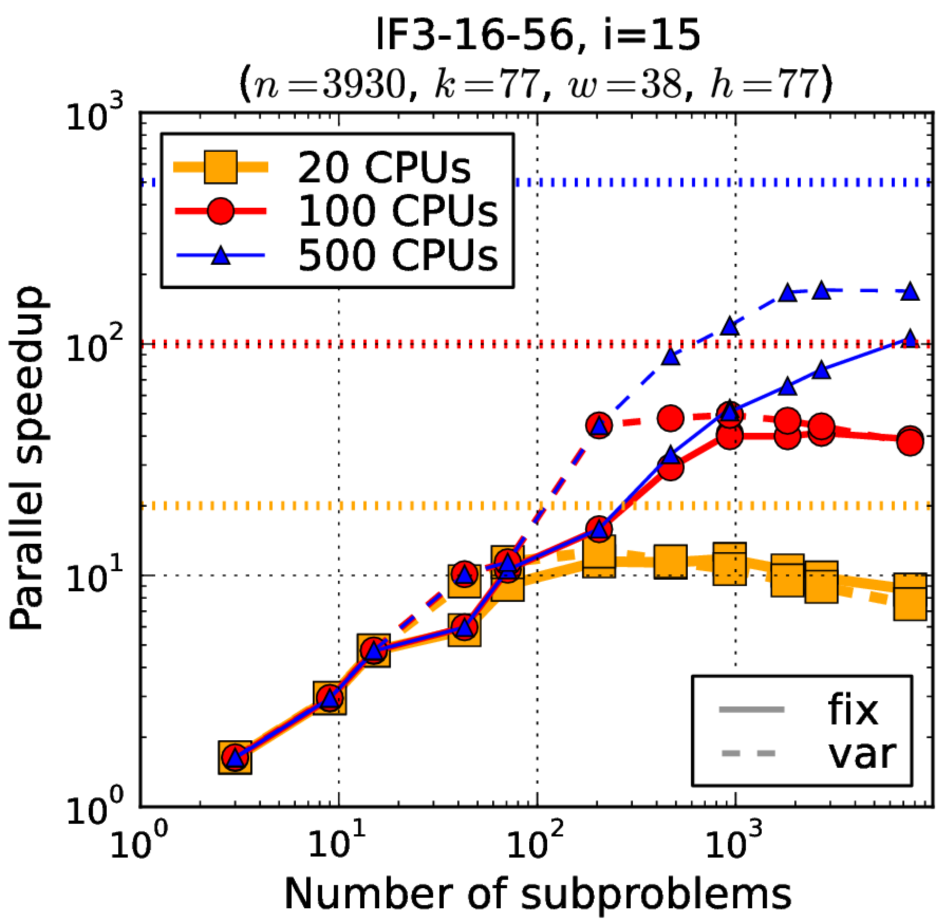


(Variable-depth)

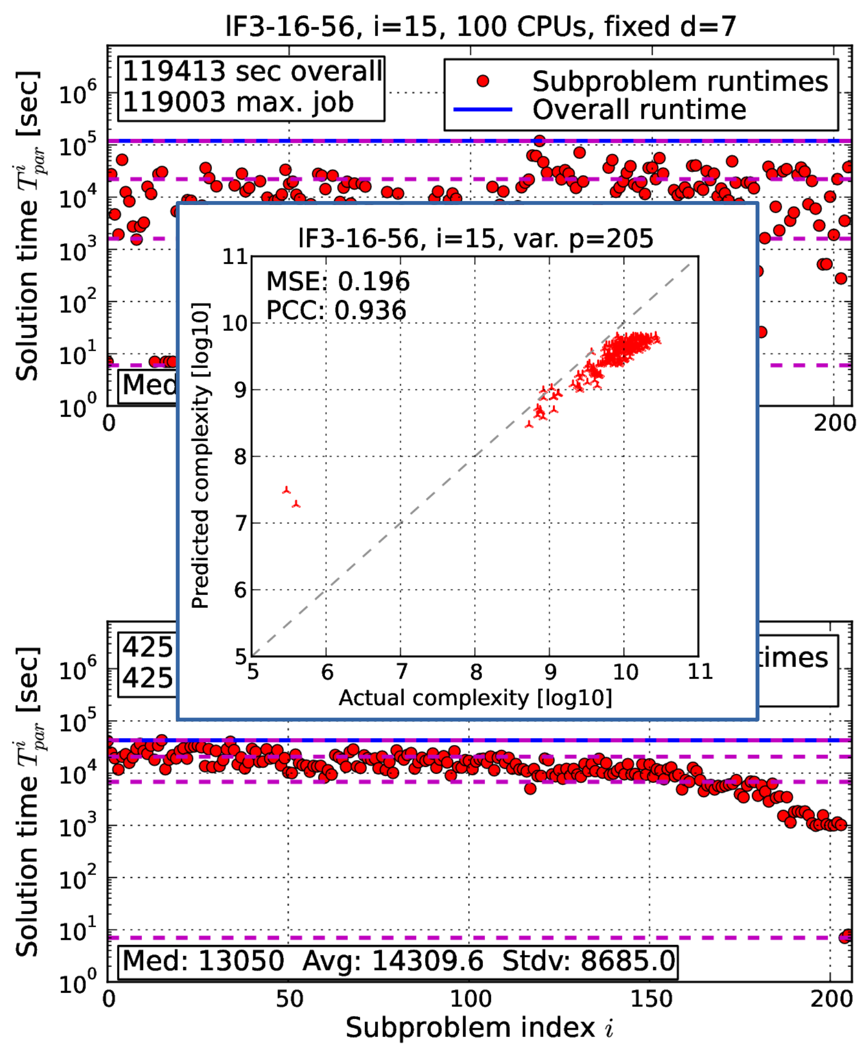


Example: LargeFam3-16-56, *i*=15

- Sequential runtime
  T_{seq} = 1,891,710 sec.

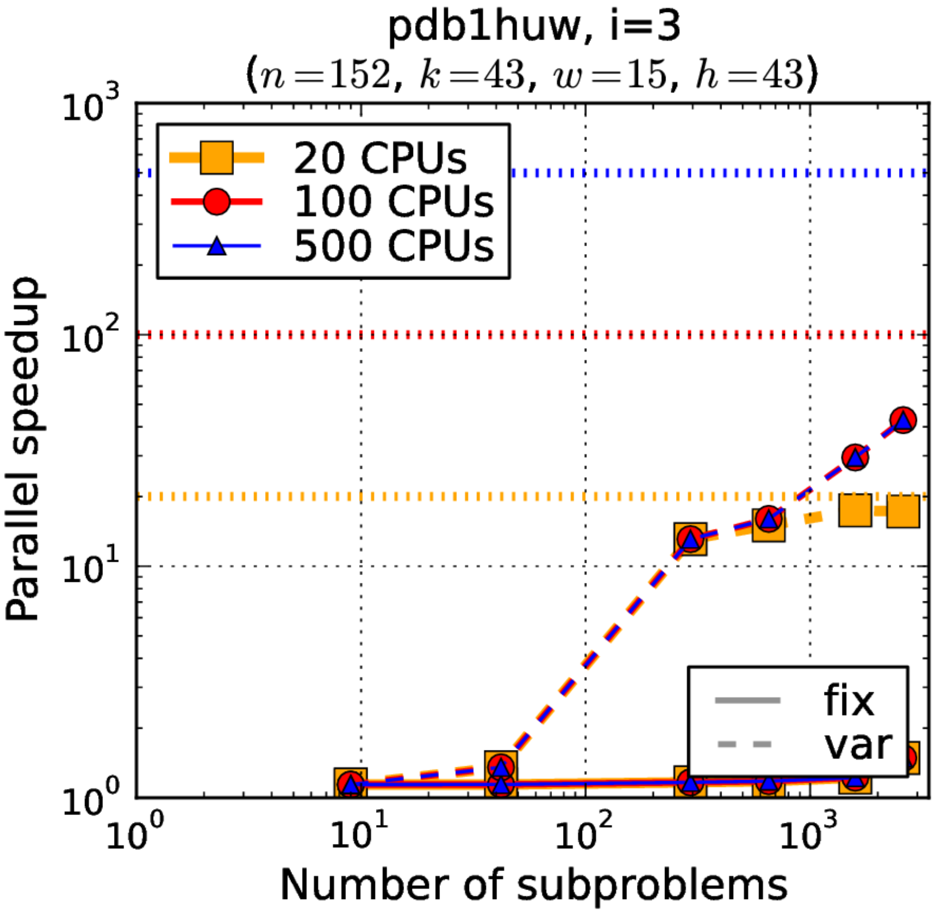


(Fixed-depth)



Example: Pdb1huw, *i*=3

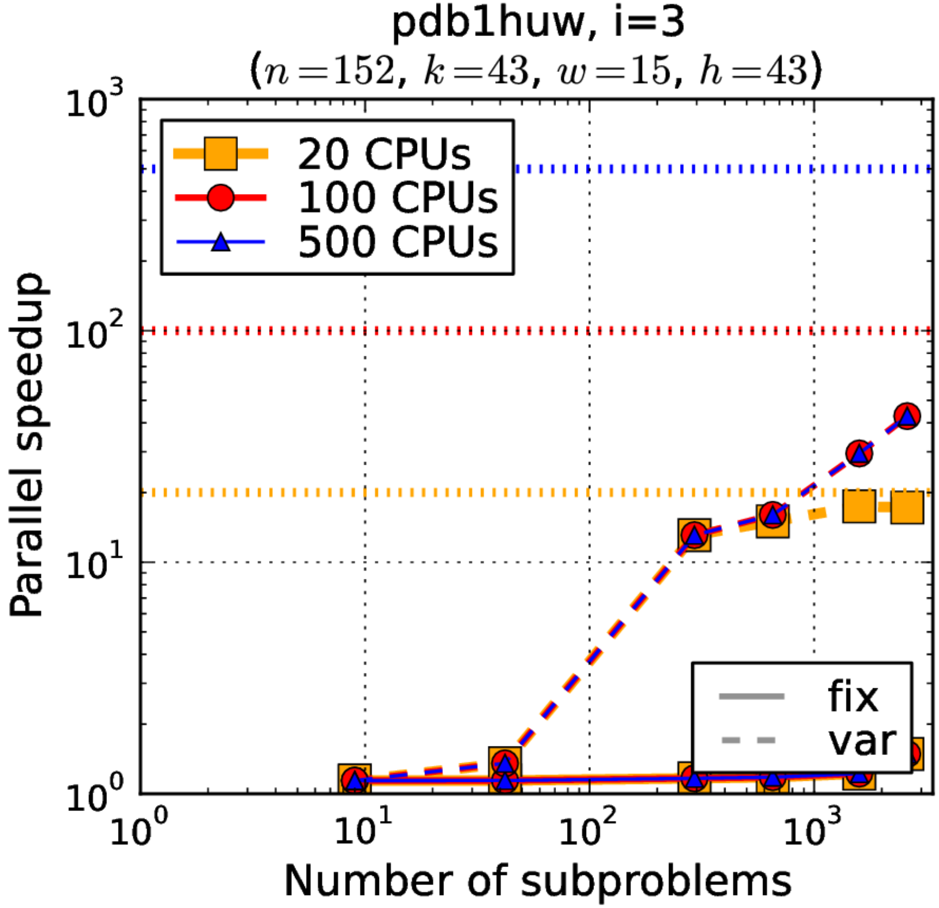
- Sequential runtime
  T_{seq} = 545,249 sec.



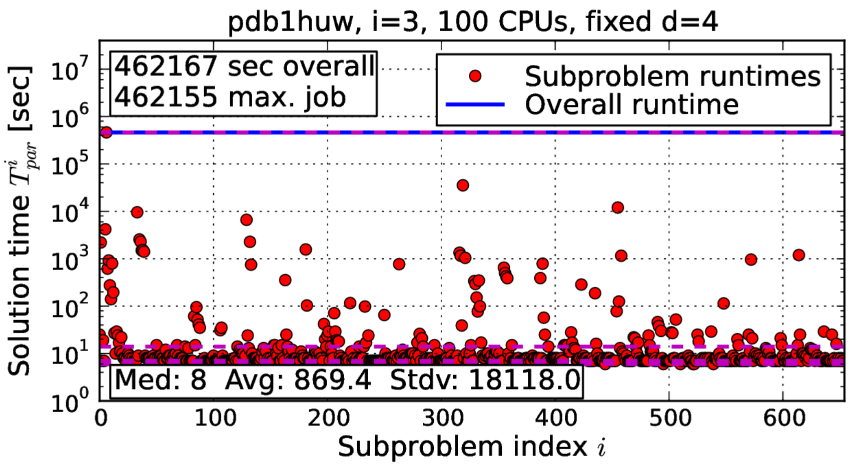
)15

Example: Pdb1huw, *i*=3

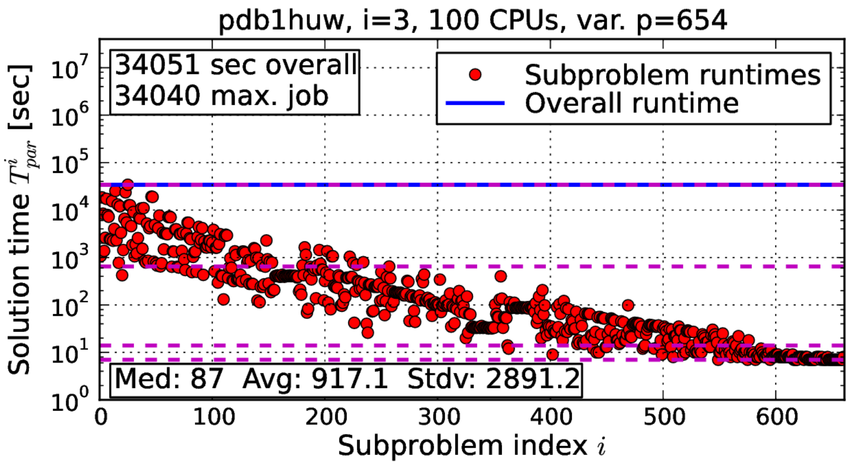
- Sequential runtime
  T_{seq} = 545,249 sec.



(Fixed-depth)

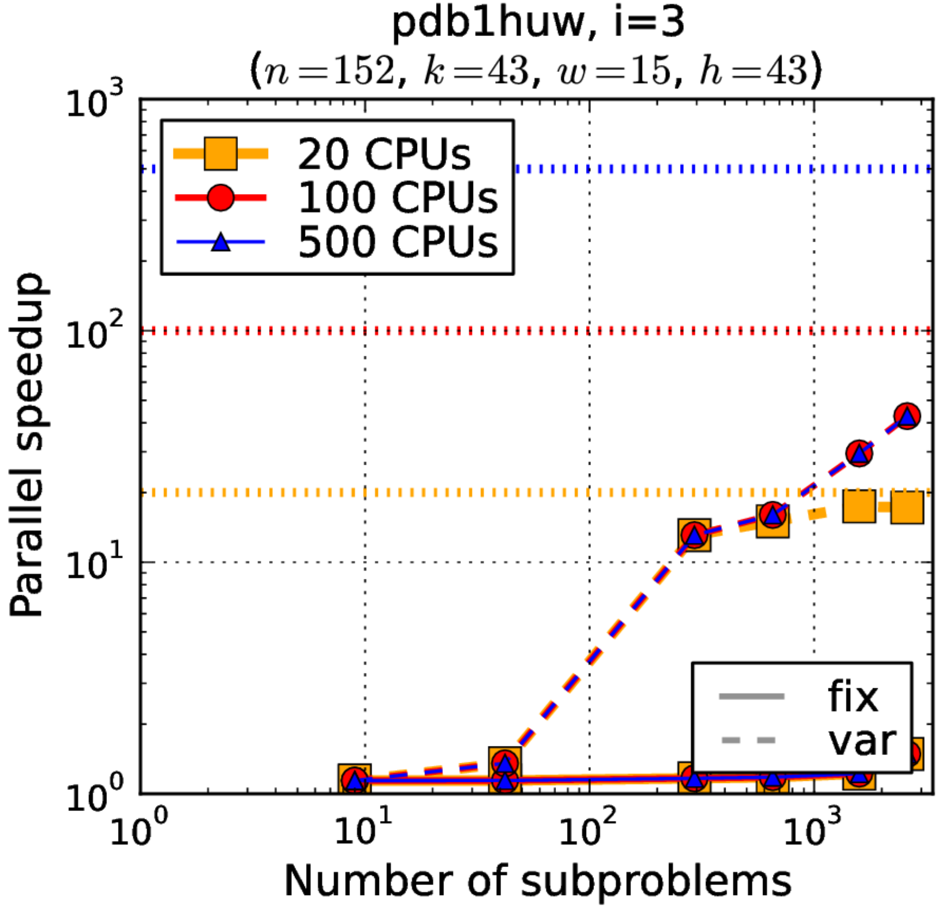


(Variable-depth)

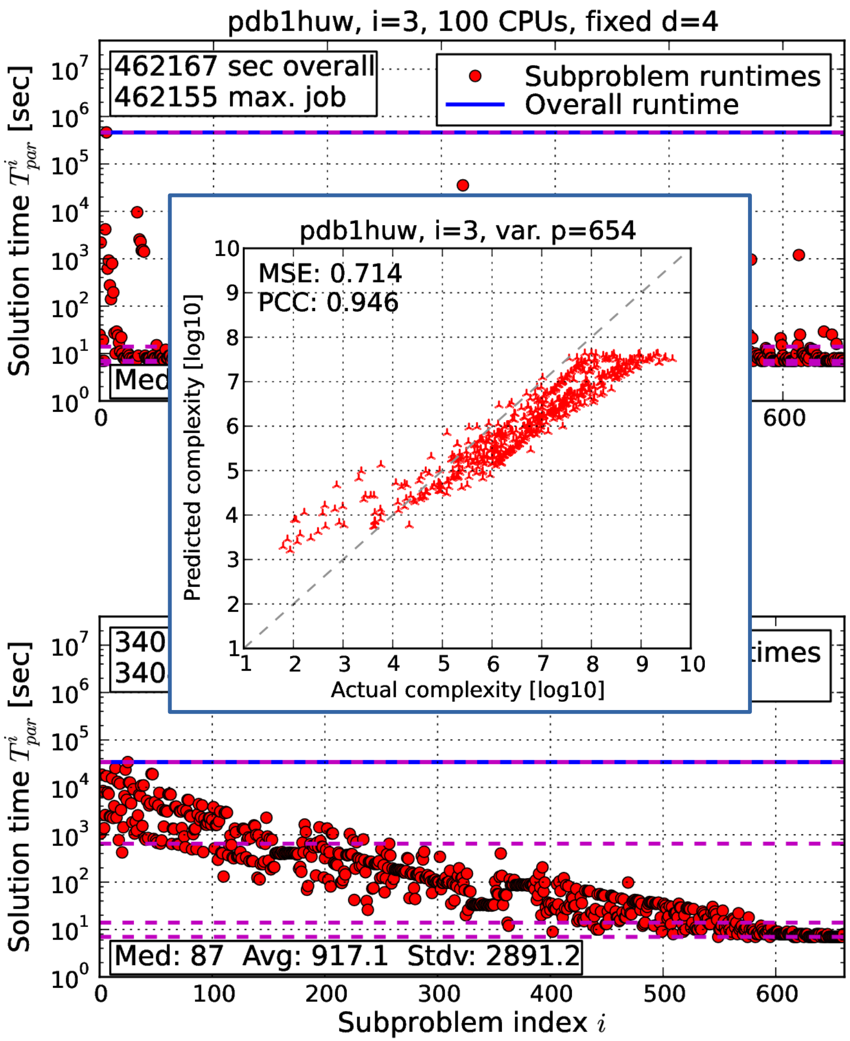


Example: Pdb1huw, *i*=3

- Sequential runtime
  T_{seq} = 545,249 sec.

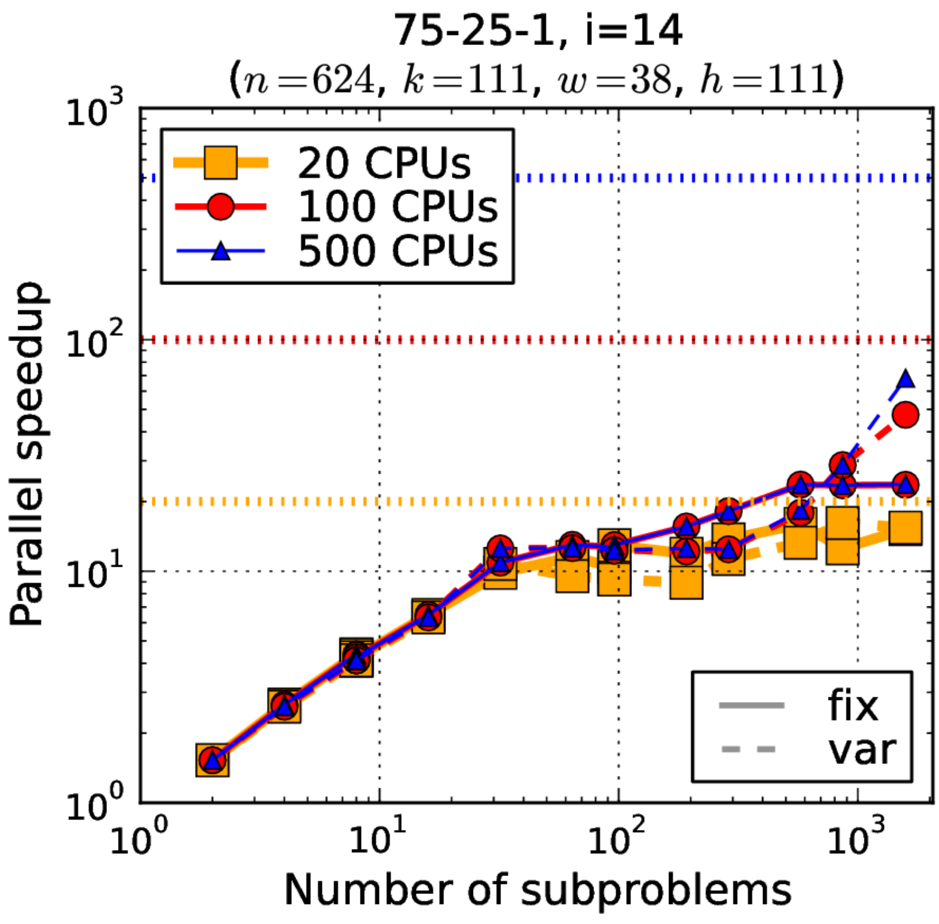


(Fixed-depth)



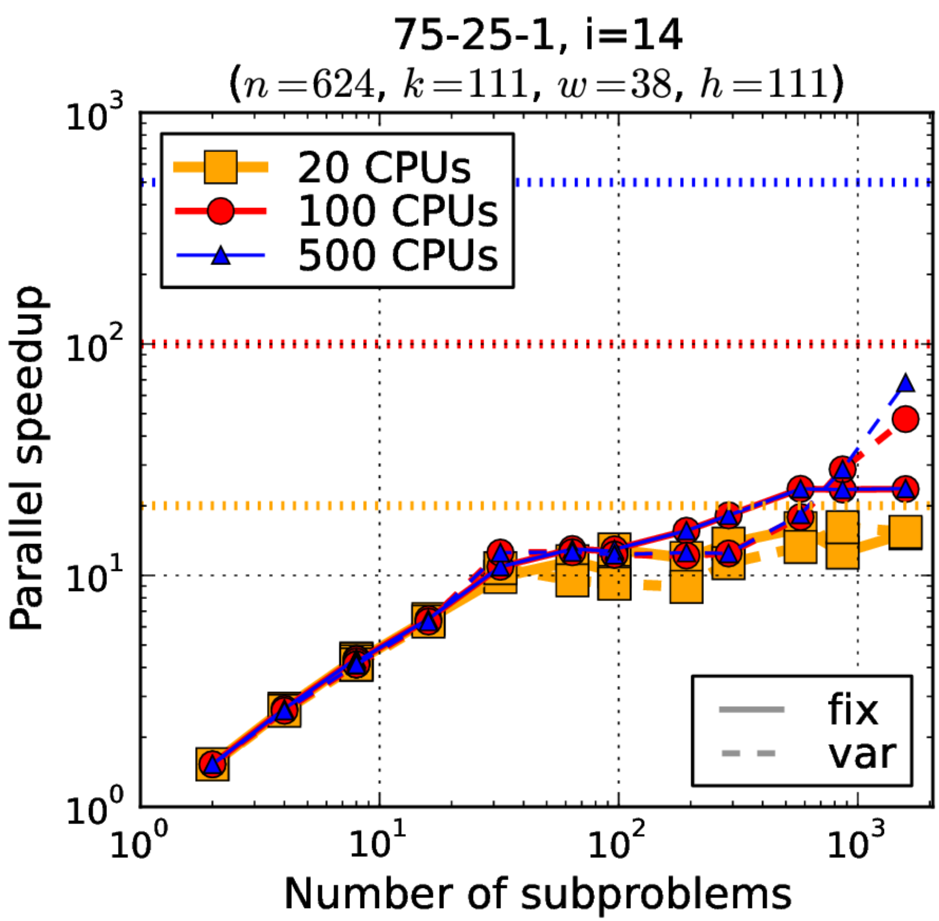
Example: 75-25-1, *i*=14

- Sequential runtime
  T_{seq} = 15,402 sec.

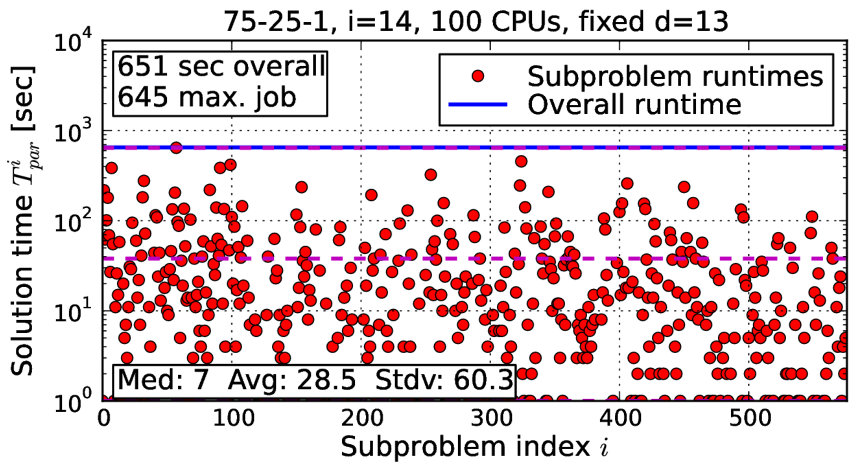


Example: 75-25-1, *i*=14

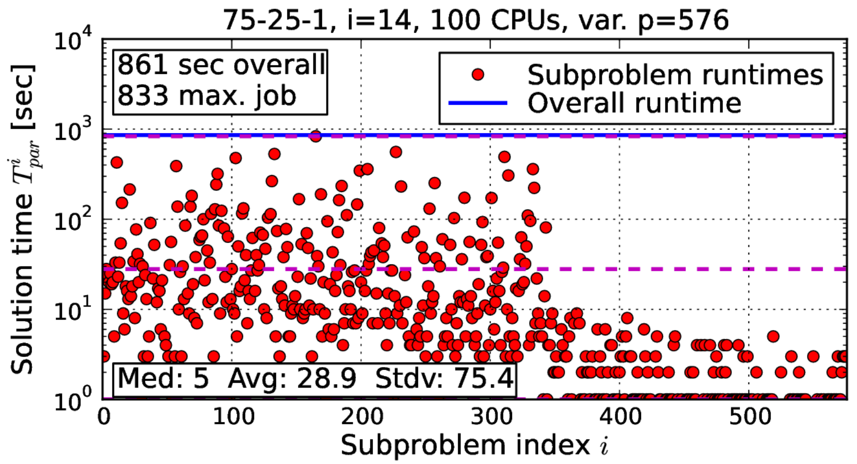
- Sequential runtime
  T_{seq} = 15,402 sec.



(Fixed-depth)

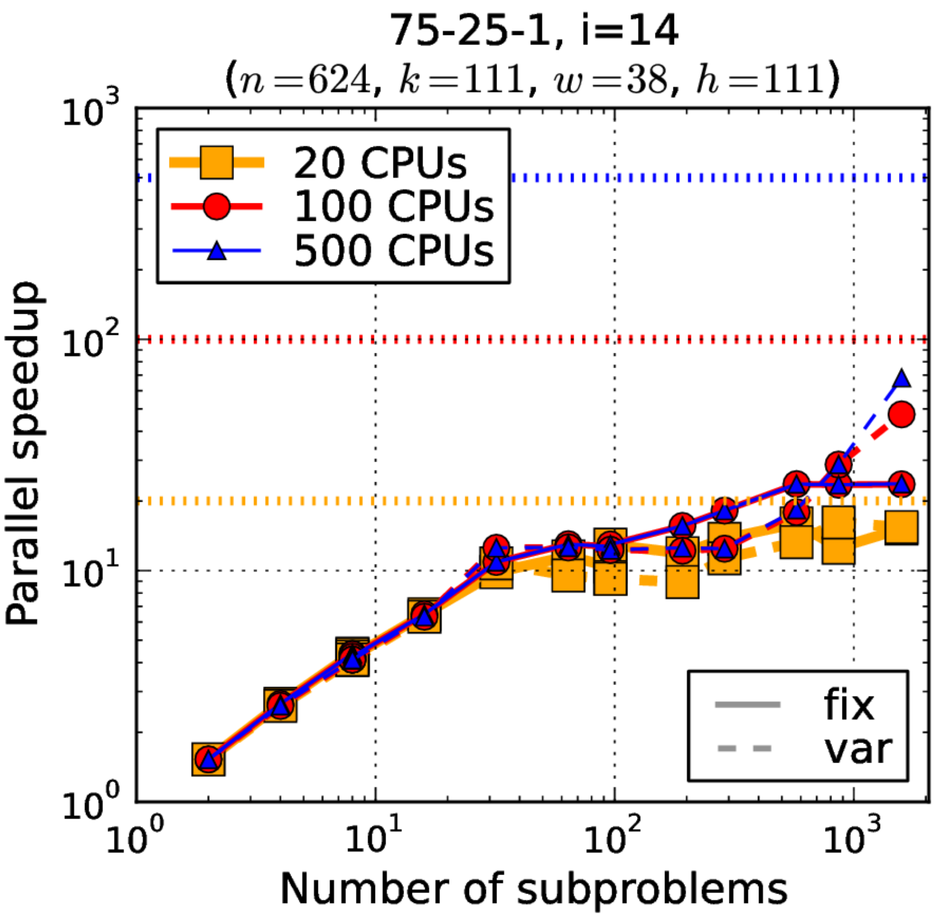


(Variable-depth)

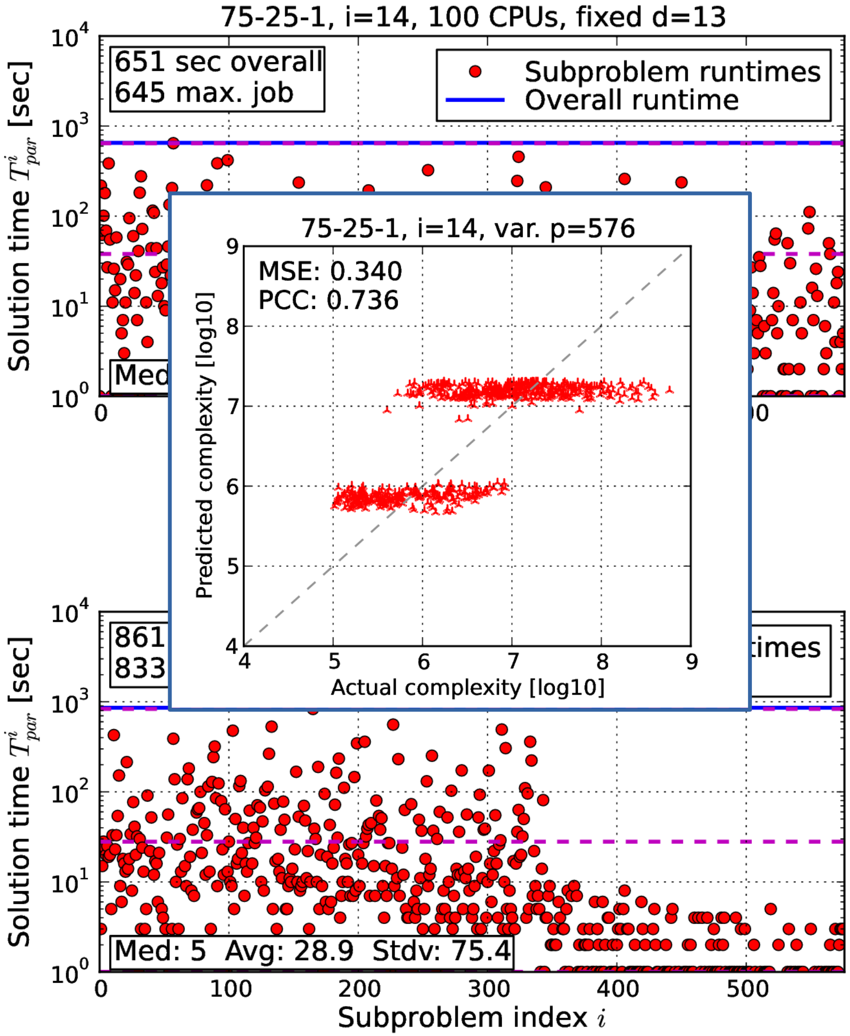


Example: 75-25-1, *i*=14

- Sequential runtime
  T_{seq} = 15,402 sec.



(Fixed-depth)

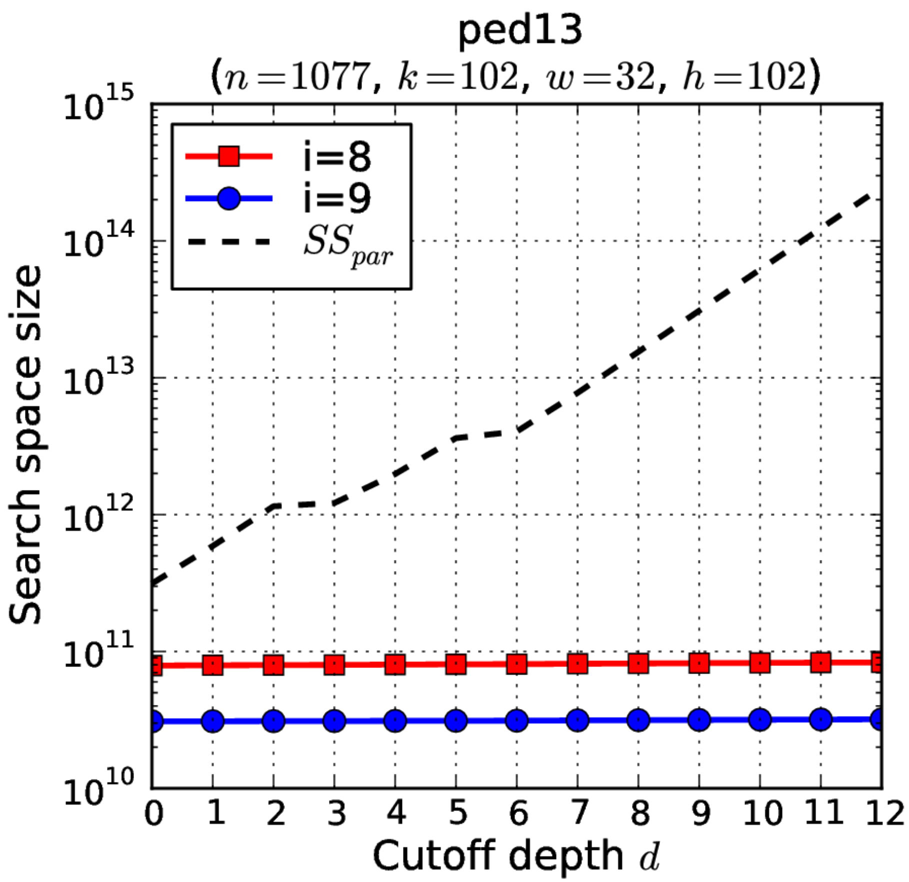


Underlying vs. Explored Search Space

- Compute SS_{par} bound (ahead of time).
  - Plot against N_{par} for different *i*-bounds.

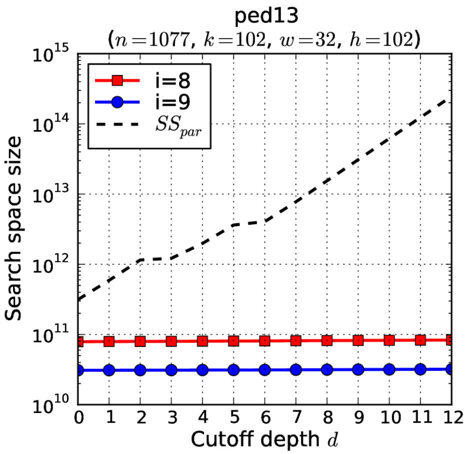
Underlying vs. Explored Search Space

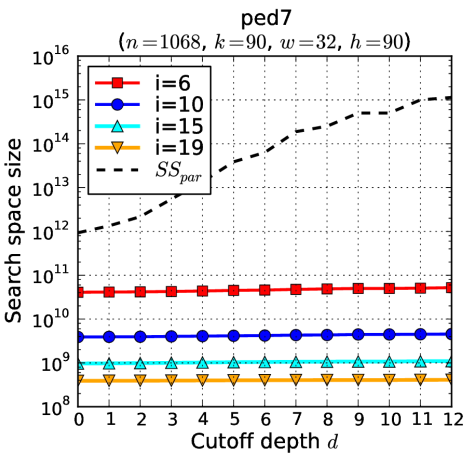
- Compute SS_{par} bound (ahead of time).
  - Plot against N_{par} for different *i*-bounds.



Underlying vs. Explored Search Space

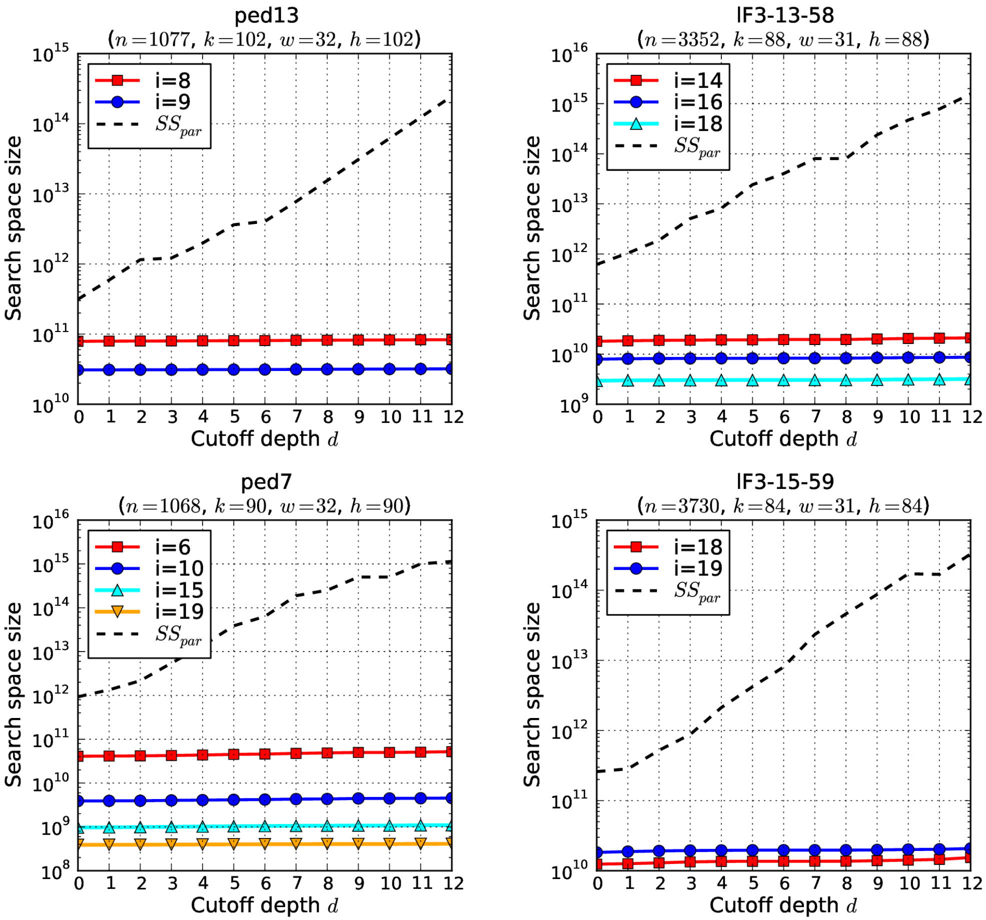
- Compute SS_{par} bound (ahead of time).
  - Plot against N_{par} for different i-bounds.





Underlying vs. Explored Search Space

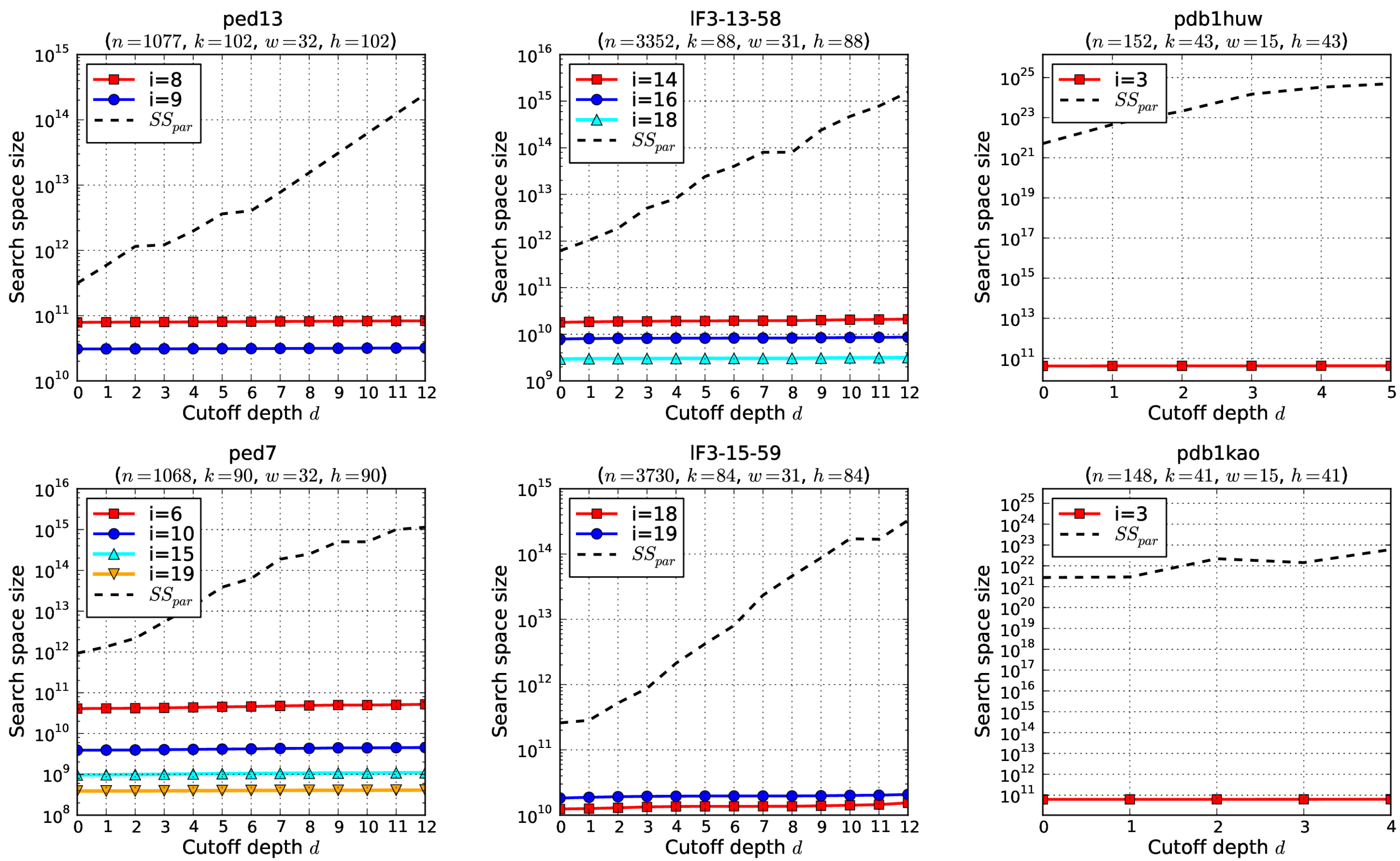
- Compute SS_{par} bound (ahead of time).
  - Plot against N_{par} for different *i*-bounds.



2015

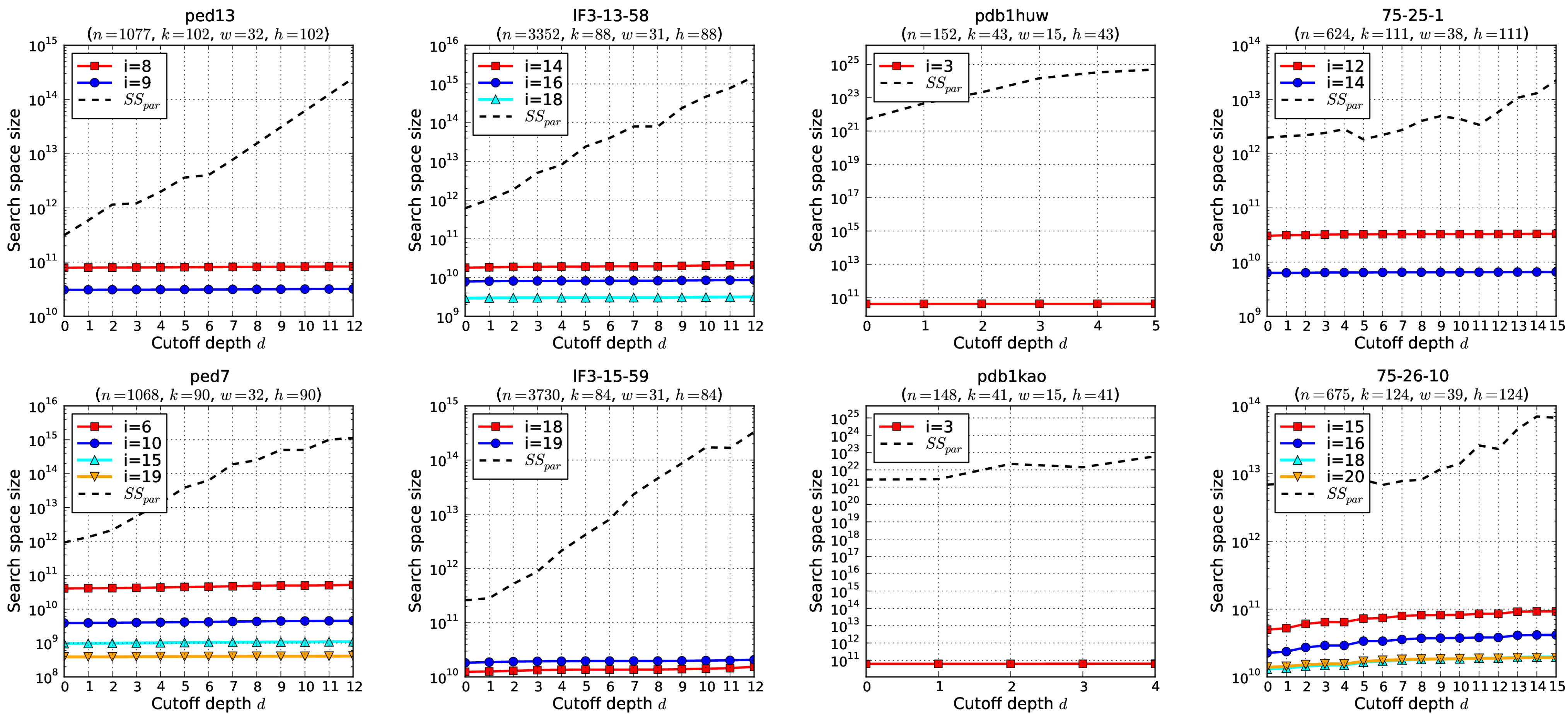
Underlying vs. Explored Search Space

- Compute SS_{par} bound (ahead of time).
  - Plot against N_{par} for different *i*-bounds.



Underlying vs. Explored Search Space

- Compute SS_{par} bound (ahead of time).
  - Plot against N_{par} for different i-bounds.

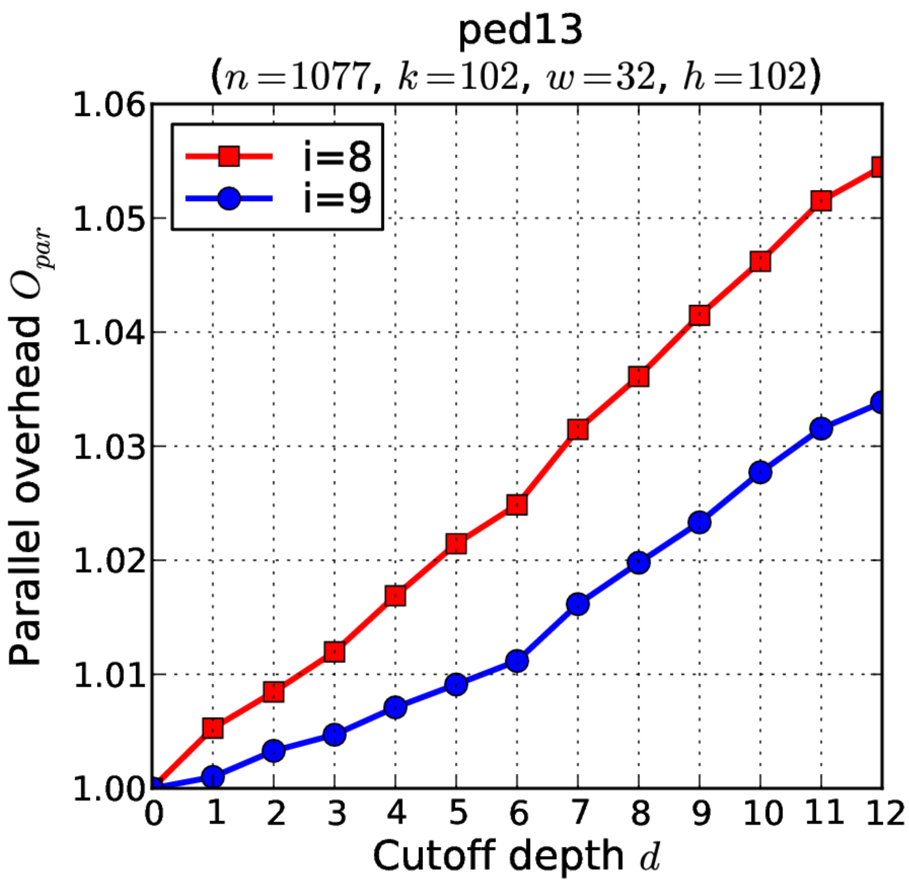


Redundancies and Overhead O_{par}

- Assess parallel redundancies in practice.
  - Node expansion overhead $O_{par} = N_{par} / N_{seq}$.

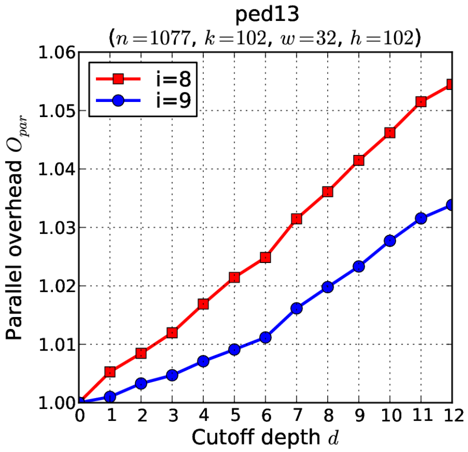
Redundancies and Overhead O_{par}

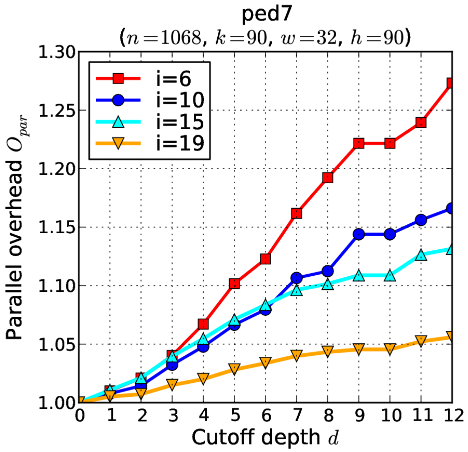
- Assess parallel redundancies in practice.
  - Node expansion overhead $O_{par} = N_{par} / N_{seq}$.



Redundancies and Overhead O_{par}

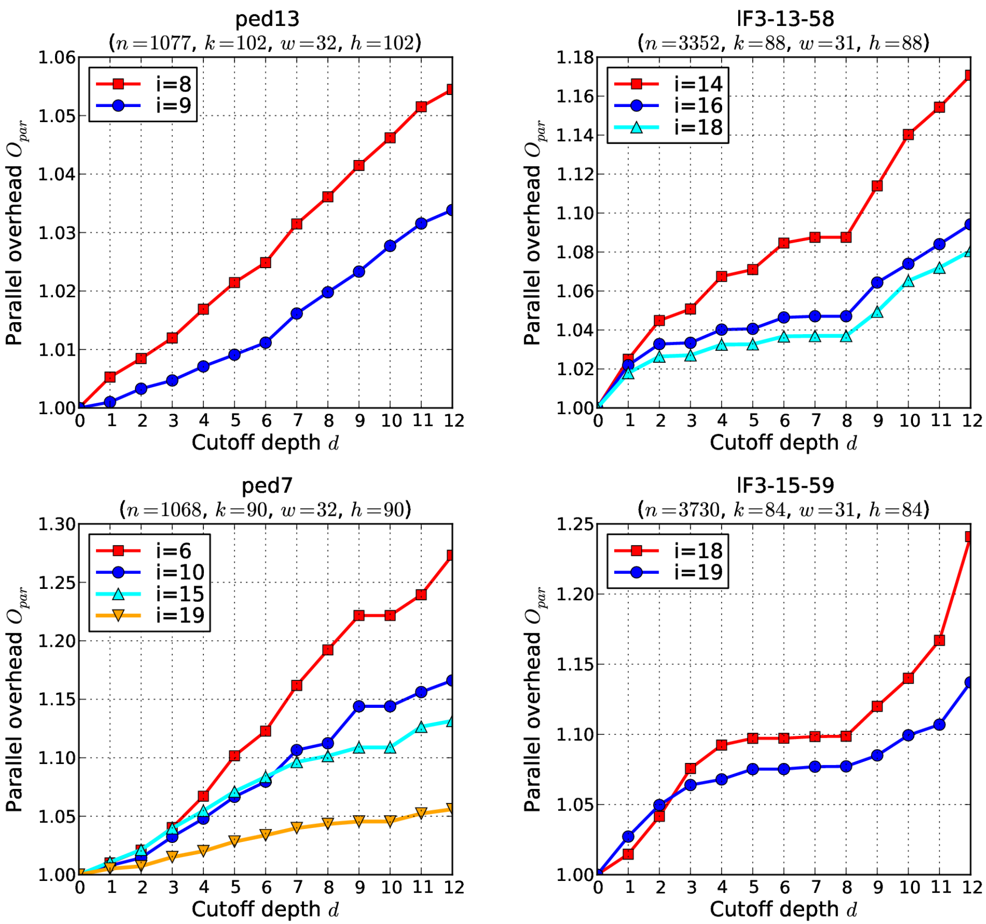
- Assess parallel redundancies in practice.
  - Node expansion overhead $O_{par} = N_{par} / N_{seq}$.





Redundancies and Overhead O_{par}

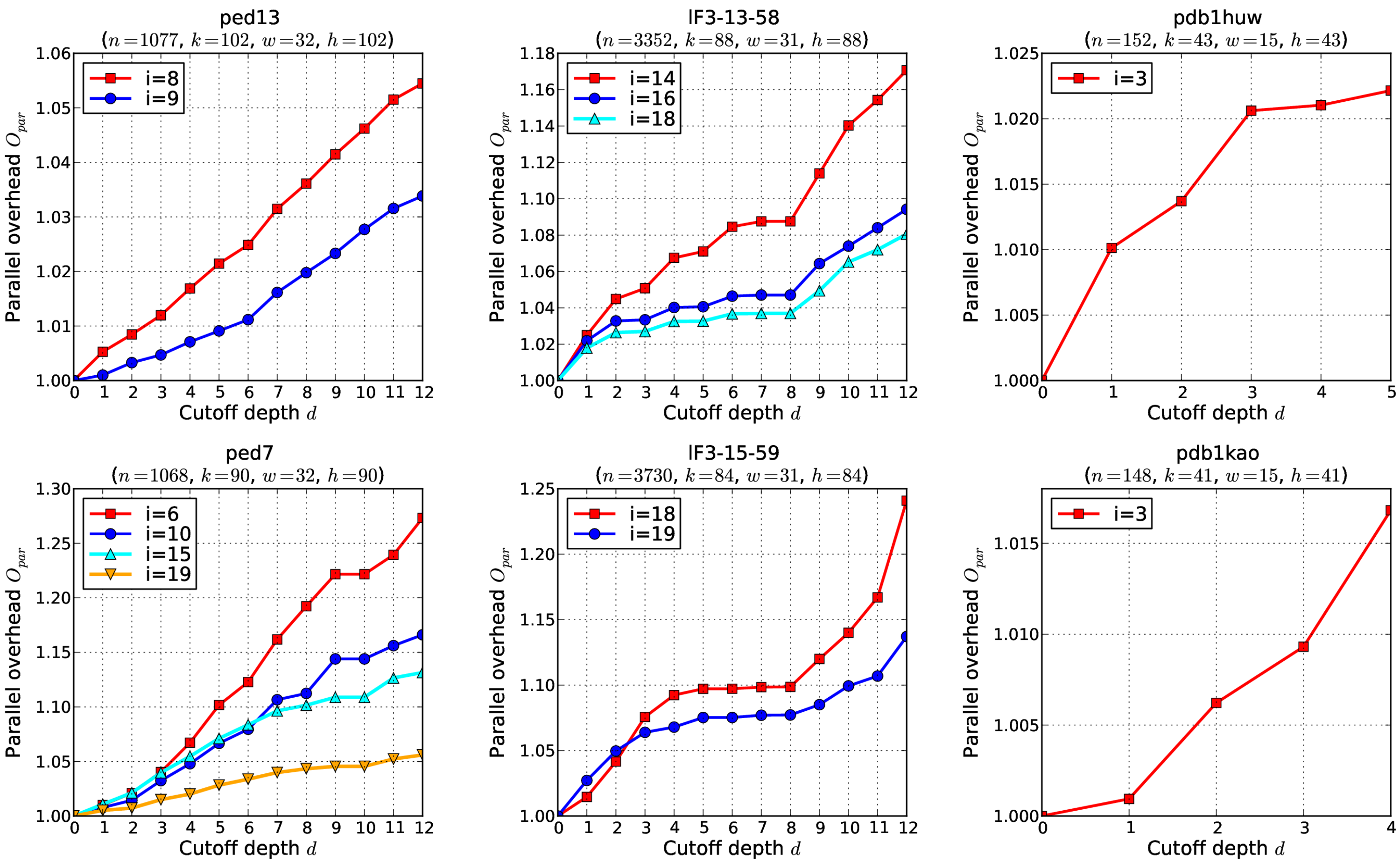
- Assess parallel redundancies in practice.
  - Node expansion overhead $O_{par} = N_{par} / N_{seq}$.



2015

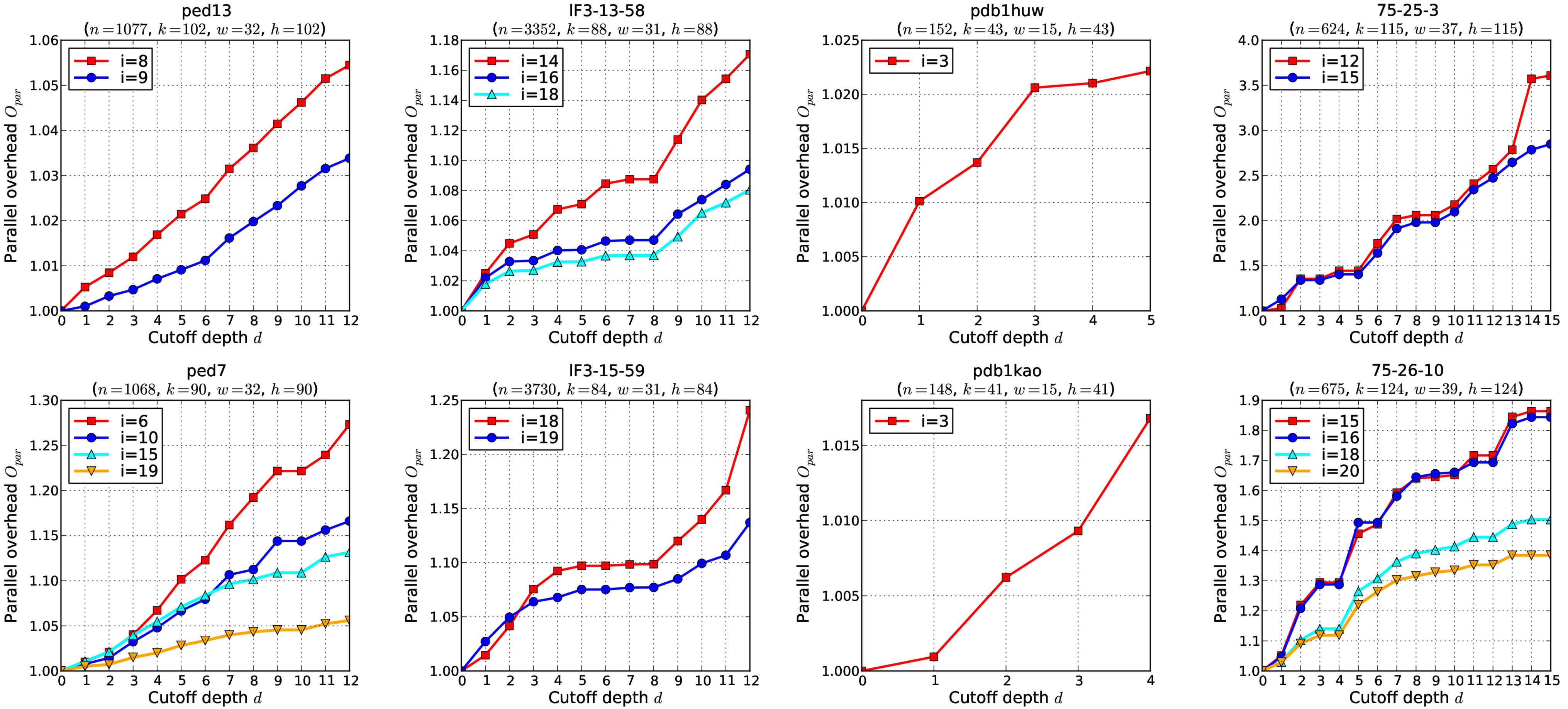
Redundancies and Overhead O_{par}

- Assess parallel redundancies in practice.
  - Node expansion overhead $O_{par} = N_{par} / N_{seq}$.



Redundancies and Overhead O_{par}

- Assess parallel redundancies in practice.
  - Node expansion overhead $O_{par} = N_{par} / N_{seq}$.

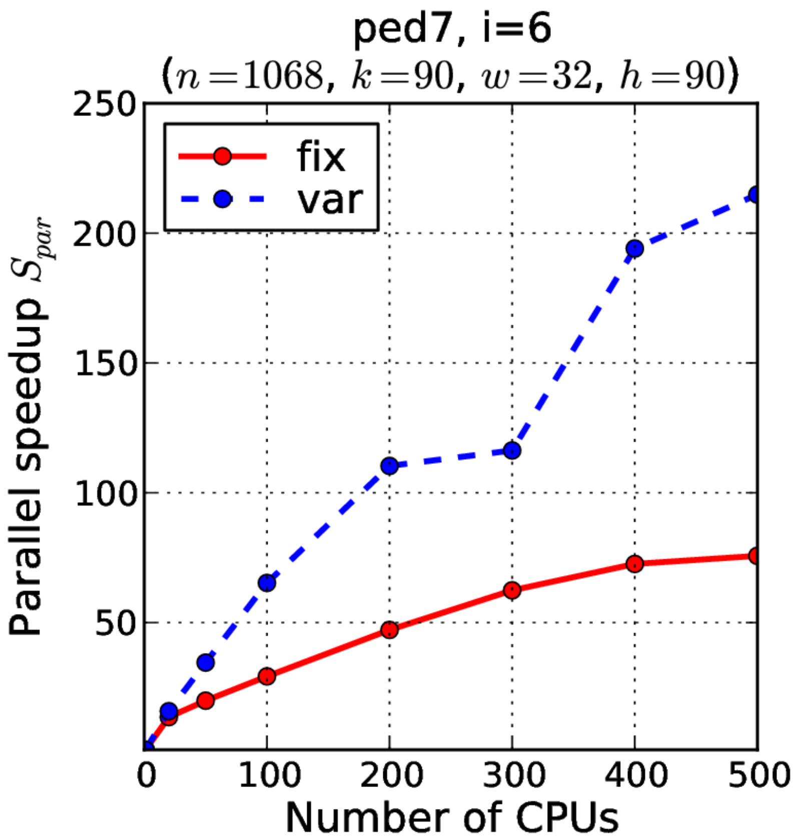


Parallel Scaling Summary

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

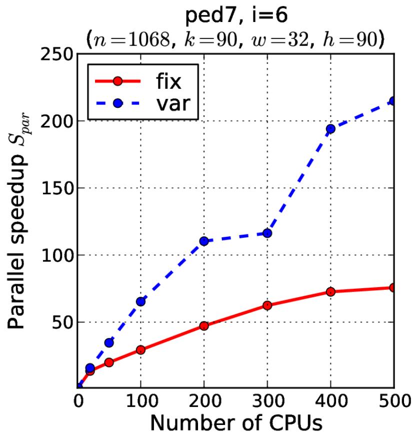
Parallel Scaling Summary

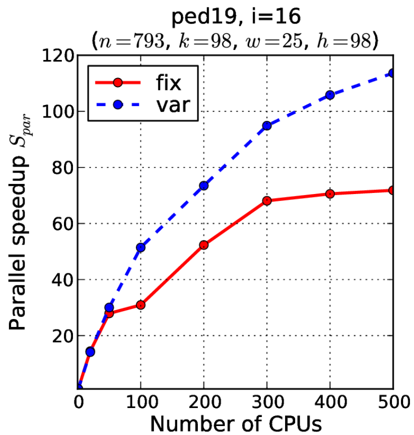
- Plot speedup against CPU count.
  - Trade off load balancing vs. overhead:
    - *#subproblems* ≈ 10 × *#CPUs*



Parallel Scaling Summary

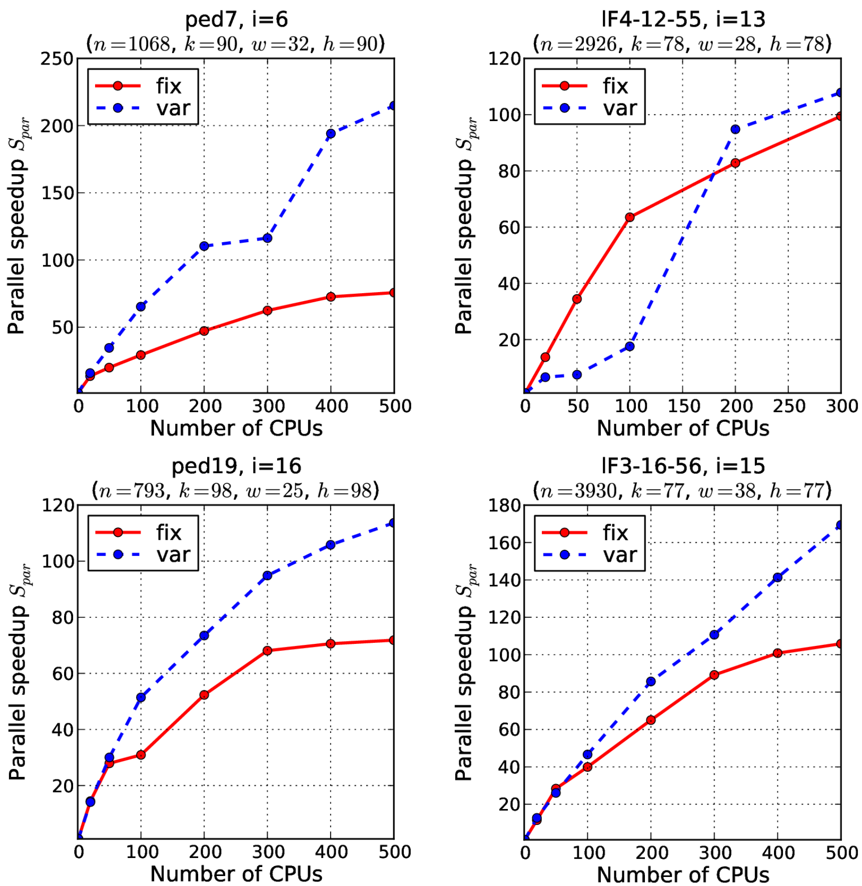
- Plot speedup against CPU count.
  - Trade off load balancing vs. overhead:
    - *#subproblems* ≈ 10 × *#CPUs*





Parallel Scaling Summary

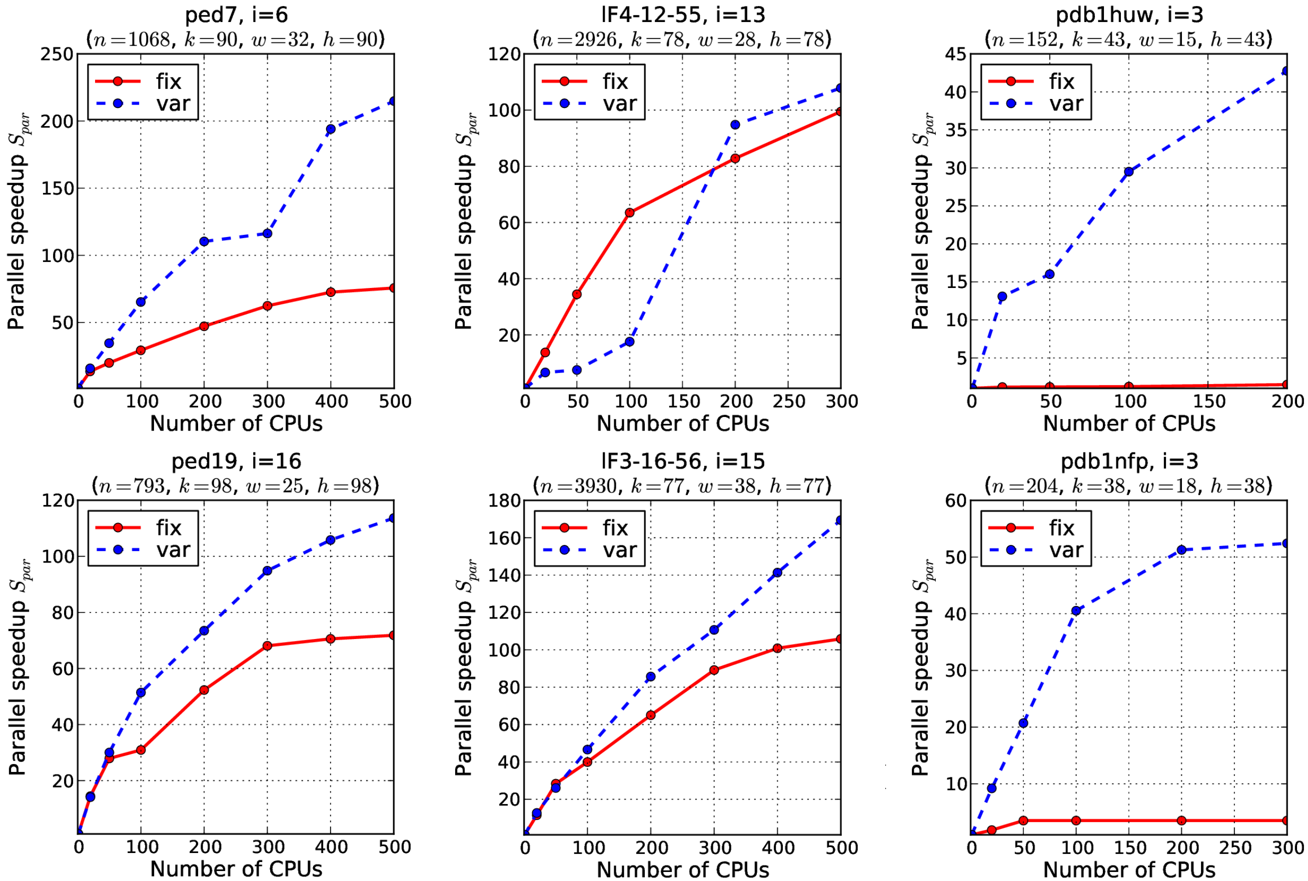
- Plot speedup against CPU count.
  - Trade off load balancing vs. overhead:
    - *#subproblems* ≈ 10 × *#CPUs*



2015

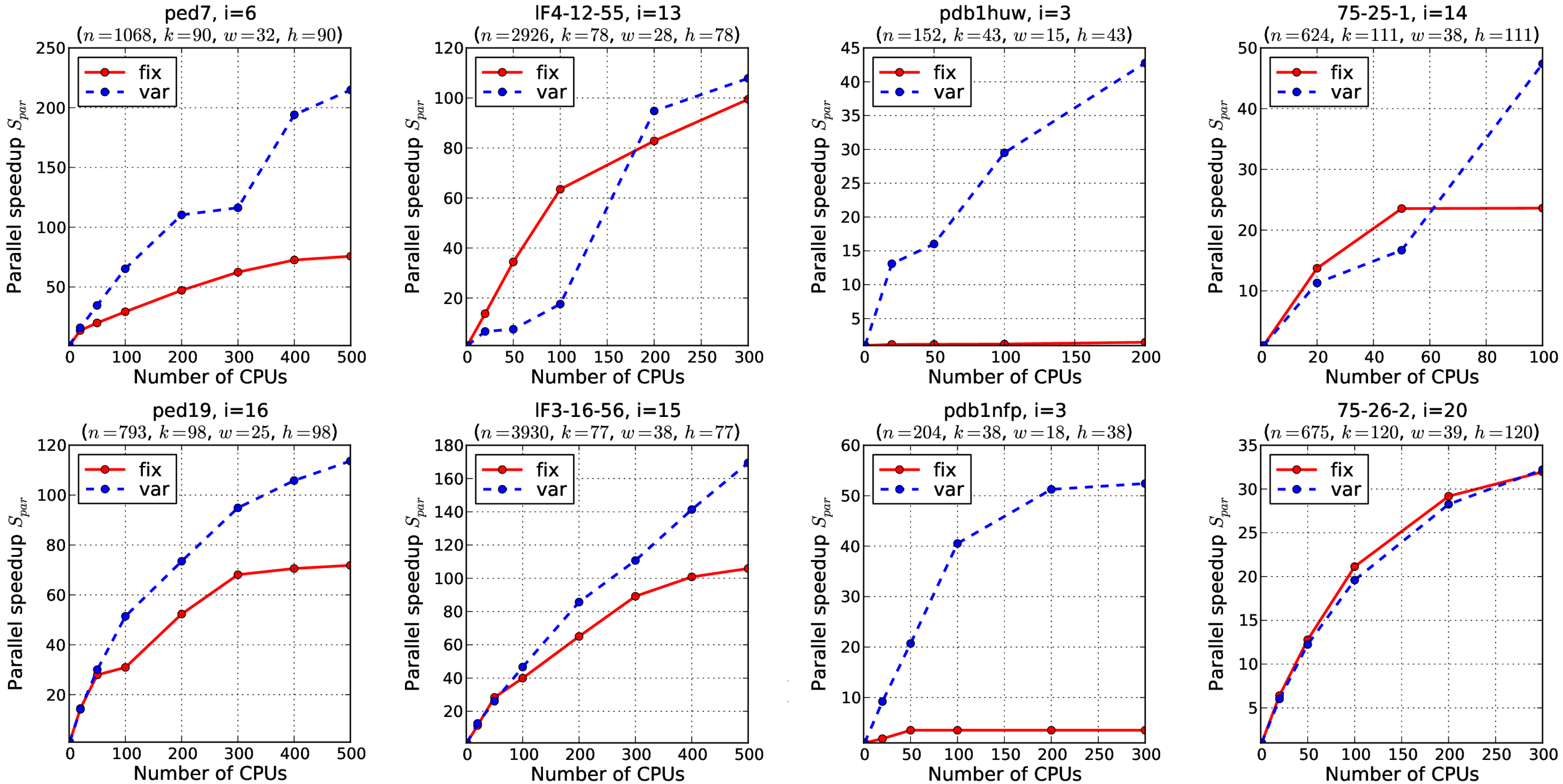
Parallel Scaling Summary

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



Parallel Scaling Summary

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



Fixed-depth vs. Variable-depth

- Compare speedup of the two parallel schemes.
  - Count cases that are 10% and 50% better.

	margin	fix	var	fix	var	fix	var	fix	var	fix	var	fix	var
		$d=2$		$d=4$		$d=6$		$d=8$		$d=10$		$d=12$	
Pedigree	10%	**20**	16	**40**	30	38	**41**	24	**49**	28	**40**	12	**28**
	50%	**20**	12	**27**	15	**30**	15	16	**22**	**20**	9	4	**13**
		(116 total)		*(116 total)*		*(116 total)*		*(116 total)*		*(108 total)*		*(88 total)*	
		$d=2$		$d=4$		$d=6$		$d=8$		$d=10$		$d=12$	
LargeFam	10%	**32**	12	21	**30**	11	**51**	7	**52**	5	**47**	8	**32**
	50%	**12**	0	**15**	10	8	**28**	3	**36**	1	**30**	5	**22**
		(84 total)		*(84 total)*		*(84 total)*		*(76 total)*		*(76 total)*		*(76 total)*	
		$d=1$		$d=2$		$d=3$		$d=4$		$d=5$		$d=6$	
Pdb	10%	0	0	5	**39**	0	**33**	0	**20**	0	**4**	0	**4**
	50%	0	0	4	**30**	0	**28**	0	**16**	0	**4**	0	**4**
		(44 total)		*(44 total)*		*(36 total)*		*(20 total)*		*(4 total)*		*(4 total)*	
		$d=2$		$d=4$		$d=6$		$d=8$		$d=10$		$d=12$	
Grid	10%	**20**	13	**12**	9	**17**	8	**27**	10	**20**	9	12	**19**
	50%	**12**	4	**10**	0	**9**	3	**11**	3	5	**6**	5	**8**
		(60 total)		*(60 total)*		*(60 total)*		*(60 total)*		*(60 total)*		*(60 total)*	

Outline

- Introduction
- Inference
- Bounds and heuristics
- AND/OR Search
- Exploiting parallelism
- **Software**
  - UAI Probabilistic Inference Competition

Software

- **aolib**
  - http://graphmod.ics.uci.edu/group/Software
    (standalone AOBB, AOBF solvers)
- **daoopt**
  - https://github.com/lotten/daoopt
    (distributed and standalone AOBB solver)

UAI Probabilistic Inference Competitions

- **2006** 

(aolib)
- **2008** 

(aolib)
- **2011** 

(daoopt)
- **2014** 

(daoopt) 

(daoopt) 

(merlin)

MPE/MAP **MMAP**

Conclusion

- 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 a distributed environment is available