Principles and Methods for Automated Inference

Rina Dechter and **Irina Rish**
Information and Computer Science
University of California, Irvine
{*dechter,irinar*}@*ics.uci.edu*

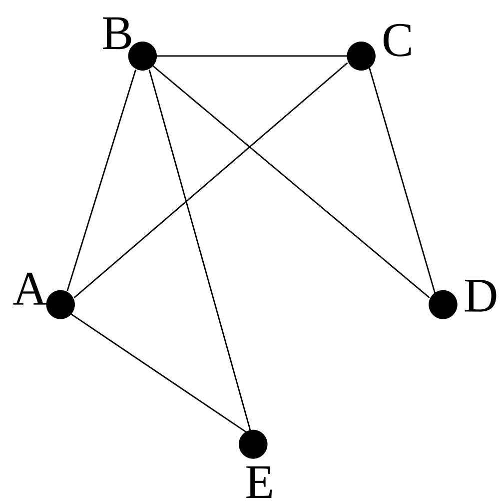
Introduction

1. Most Artificial Intelligence tasks are NP-hard.

2. Elimination and conditioning: reasoning principles common to many NP-hard tasks.

3. Problems described by assigning values to variables subject to a given set of dependencies (constraints, clauses, probabilistic relations, utility functions).

4. The dependency structure can be described by a graph: variables - nodes, dependencies - edges (constraint networks, belief networks, influence diagrams).



Artificial Intelligence Tasks

Areas:

1. Automated theorem proving
2. Planning and Scheduling
3. Machine Learning
4. Robotics
5. Diagnosis
6. Explanation

Frameworks:

1. Propositional Logic
2. Constraint Networks
3. Belief Networks
4. Markov Decision Processes

Our Focus: Tasks

- CSP and SAT:
  - Deciding if there is a solution (satisfiability).
  - Finding one or all solution.
  - Counting solutions.

- Belief Networks:
  - Belief Updating (BEL)
  - Most Probable Explanation (MPE)
  - Maximum Aposteriory hypothesis (MAP).

- Influence Diagrams and MDPs:
  - Finding Maximum Expected Utility (MEU) decision.
  - Finding optimal (MEU) policy.

Propositional SAT

Party Problem

If Alex goes, then Beki goes:

$$A \rightarrow B$$

If Chris goes, then Alex goes:

$$C \rightarrow A$$

Query:

Is it possible that Chris goes (C), but Beki is not ($\neg B$) ?

$$\Downarrow$$

Is $\varphi = \{\neg A \vee B, \neg C \vee A, \neg B, C\}$

satisfiable?

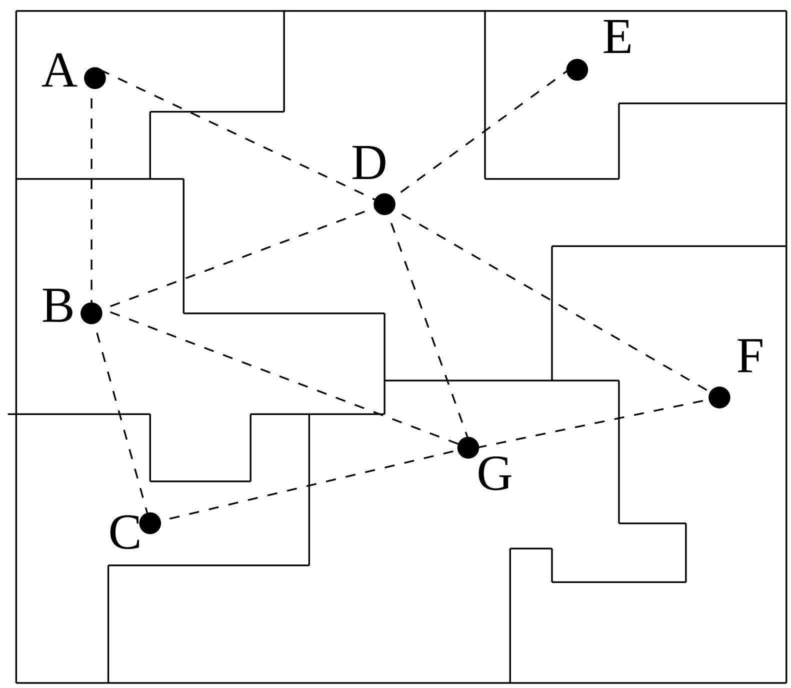
Constraint Satisfaction

Map Coloring

Variables $= \{A, B, C, D, E, F, G\}$

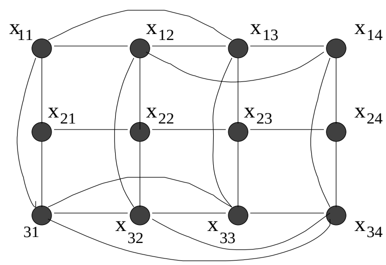
Domain $= \{red, green, blue\}$

Constraints: $A \neq B$, $A \neq D$, etc.



Constrained Optimization

Power Plant Scheduling



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

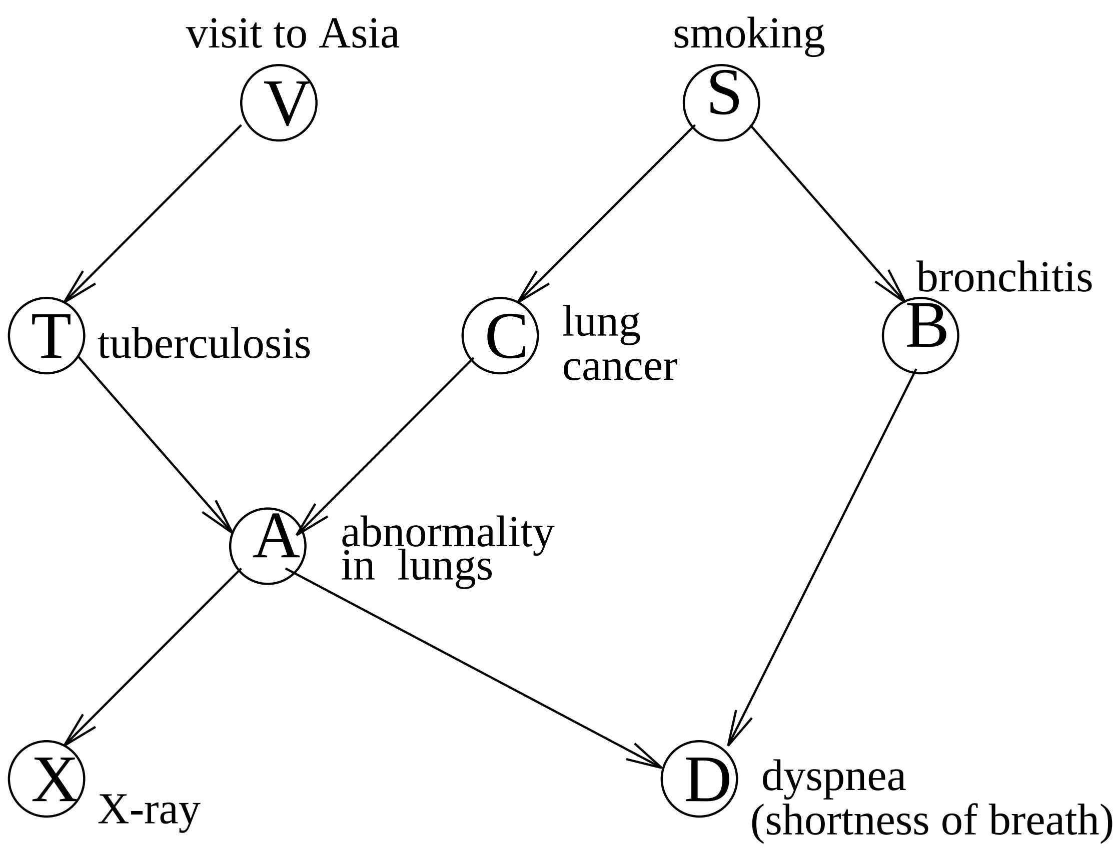
Variables = $\{X_1, ..., X_N\}$
Domain = $\{ON, OFF\}$
Constraints: $X_1 \vee X_2, \neg X_3 \vee X_4$,
minimum-on and minimum-off time, etc.

$$\sum_i P(X_i) \geq Demand$$

Objective :
minimize $Total\_Fuel\_Cost(X_1, ..., X_N)$

Belief Networks

Medical diagnosis



Query:

$$P(T = yes|S = no, D = yes) = ?$$

Decision-Theoretic Planning

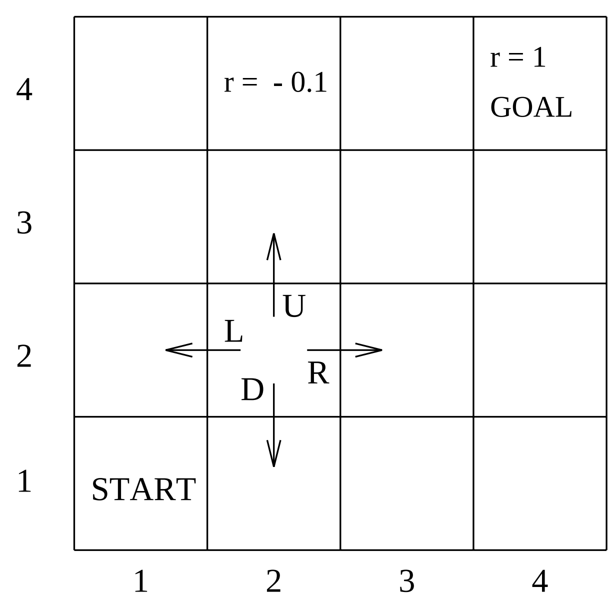
Example: Robot Navigation

State = $\{X, Y, Battery\_Level\}$

Actions = $\{North, South, West, East\}$

Probability of Success = P

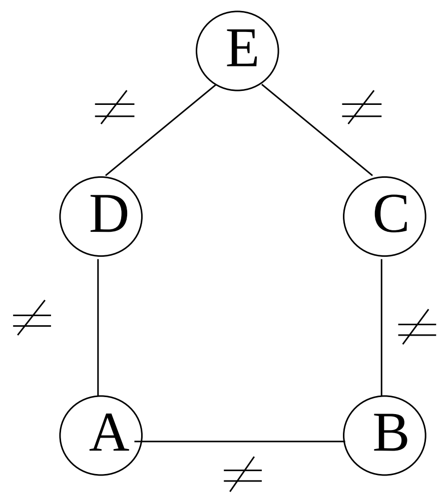
Task: reach the goal ASAP



Two Reasoning Principles: Elimination and Conditioning

Inference vs. Search
Thinking vs. Guessing

Graph coloring
Elimination



Adaptive consistency:
Bucket elimination

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

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

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

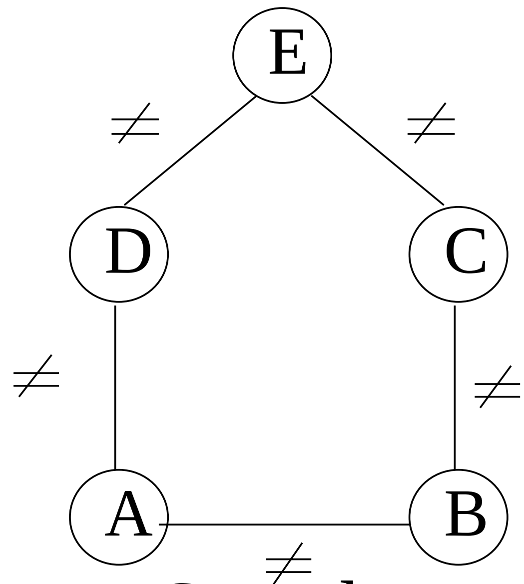
$Bucket(B)$: $B \neq A$,

$Bucket(A)$:

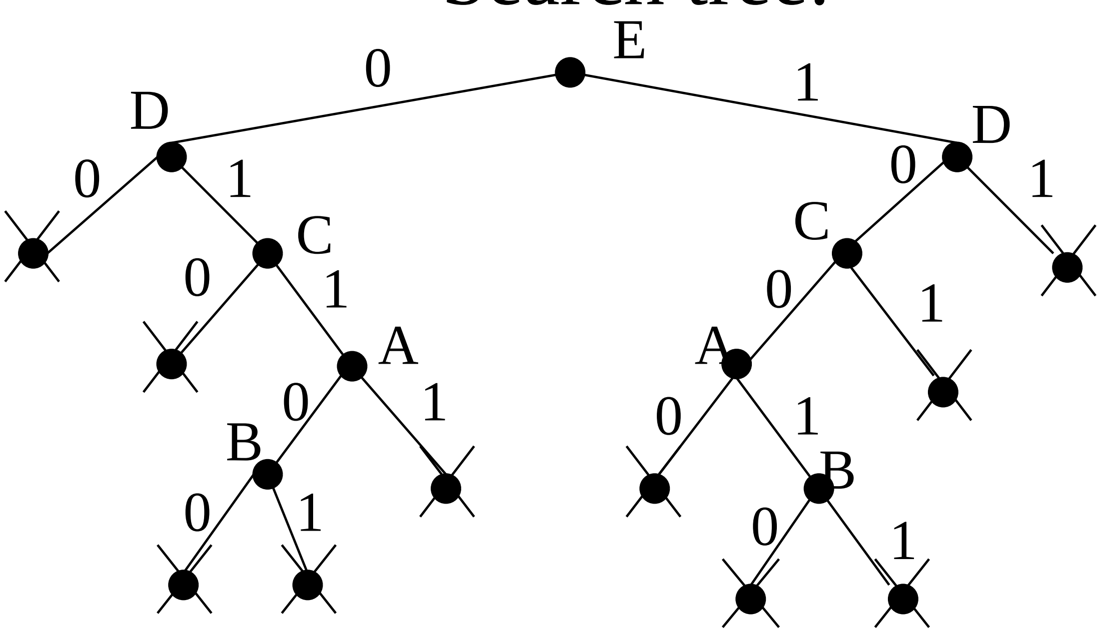
Basic step: deduction, constraint recording.

Graph Coloring

Conditioning



Search tree:



Algorithmic Principles

- *Elimination:*
  Basic operation: eliminating variables.
  Reduction to equivalent subproblems, propagating constraints, probabilities.
  Inference, deduction, “thinking”.

- *Conditioning*:
  Basic operation: value assignment, conditioning.
  “Guessing”, generating subproblems, search.

- *Paradigm:*
  Most reasoning algorithms employ one or both of those principles.

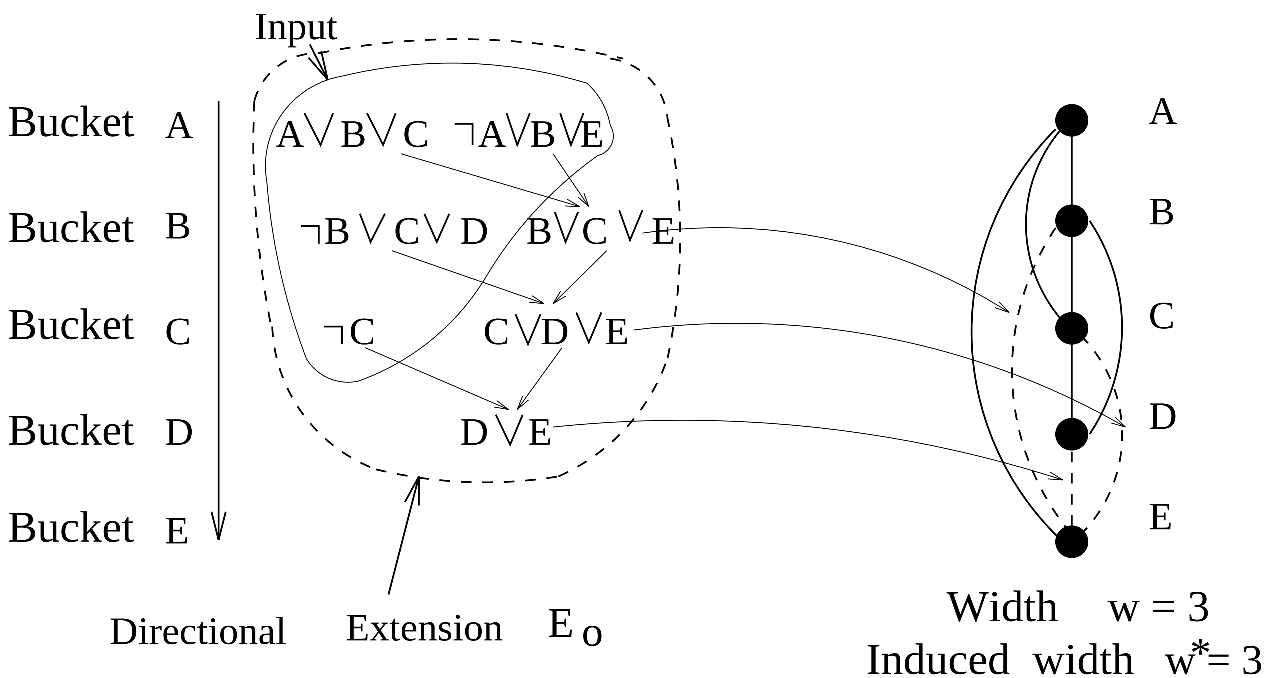
Satisfiability

Elimination

$$\varphi = (\neg A \vee B) \wedge (\neg A \vee E) \wedge (\neg B \vee C \vee D) \wedge \neg C$$

Directional resolution (DR)

Bucket elimination



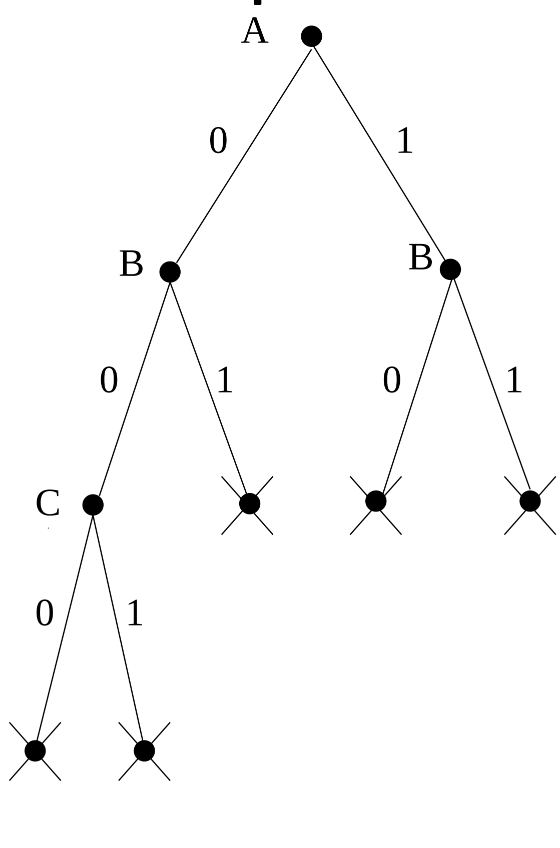
Induced width $w_d^*(A)$ = number of A's parents in the **induced graph** along ordering d

Satisfiability:
Conditioning

Guessing: conditioning on variables, search:

$$\varphi = (\neg A \vee B) \wedge (\neg C \vee A) \wedge \neg B \wedge C$$

Conditioning:
The Davis-Putnam procedure



Complexity

	Backtracking	Elimination
Worst-case time	$O(\exp(n))$	$O(n \exp(w^*))$ $w^* \leqslant n$
Average time	better than worst-case	same
Space	$O(n)$	$O(n \exp(w^*))$ $w^* \leqslant n$
Output	one solution	knowledge compilation

Known examples

Elimination examples:

- Dynamic programming (optimization)
- Davis-Putnam, directional resolution (SAT)
- Fourier elimination, Gausian elimination
- Adaptive Consistency (CSP)
- Join-tree for belief updating and CSPs

Conditioning examples:

- Branch and Bound (optimization)
- Davis-Putnam backtracking
- Backtracking (CSP)
- Cycle-cutset scheme (CSPs, Belief networks)

Bucket elimination and conditioning:
a uniform framework

- Understanding: commonality and differences.
- Ease of implementation
- Uniformity
- Technology transfer
- Allows uniform extensions to hybrids of conditioning+elimination, and to approximations.

Outline; Road Map

Methods \ Tasks	CSP	SAT	Optimization	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency, join-tree	directional resolution	dynamic programming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset, forward checking	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

Constraint Satisfaction

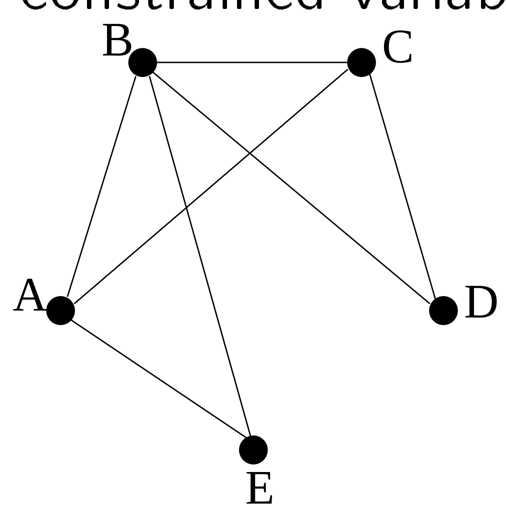
Applications:

- Configuration and design problems
- Temporal reasoning
- Scheduling
- Circuit diagnosis
- Scene labeling
- Natural language parsing

Constraint Networks

Constraint Network = $\{X, D, C\}$

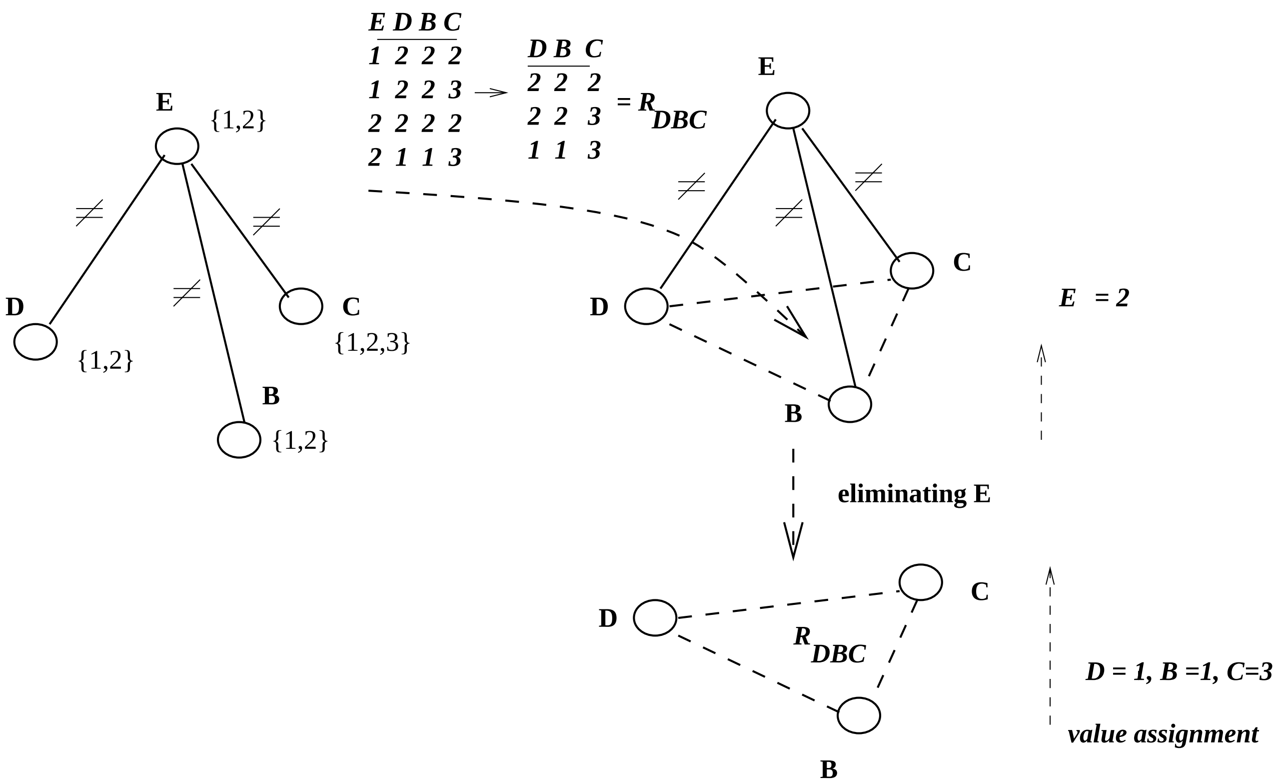
Variables: $X = \{X_1, ..., X_n\}$
Domains: $D = \{D_1, ..., D_n\}$, $D_i = \{v_1, ..., v_k\}$
Constraints: $C = \{C_1, ...C_l\}$
A constraint graph: A node per variables, an edge between constrained variables.



A solution: an assignment of a value to each variable that does not violate any constraint.

The Idea of Elimination

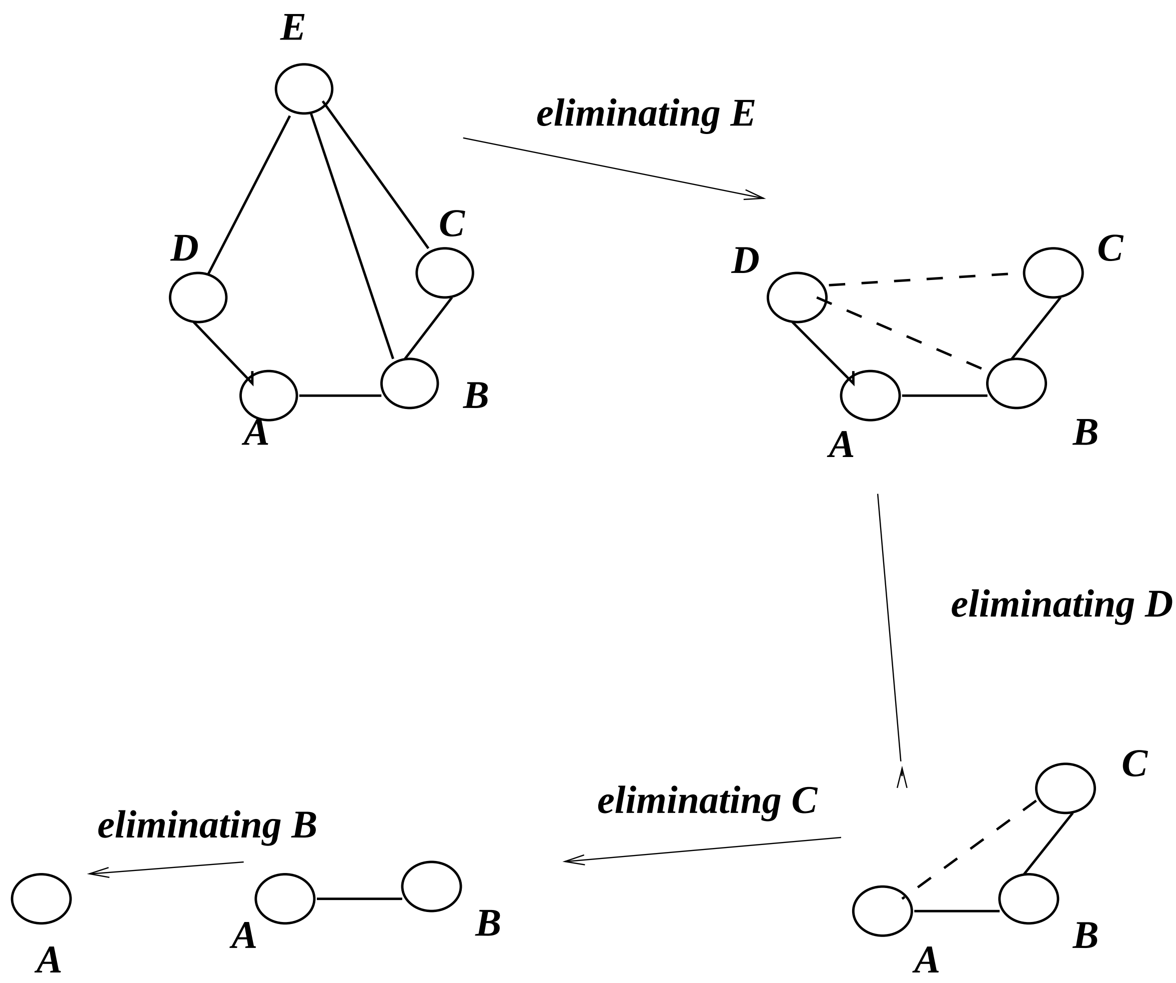
Eliminate variables one by one:



$$R_{DBC} = \Pi_{(-E)} R_{ED} \bowtie R_{EB} \bowtie R_{EC}$$

The Idea of Elimination

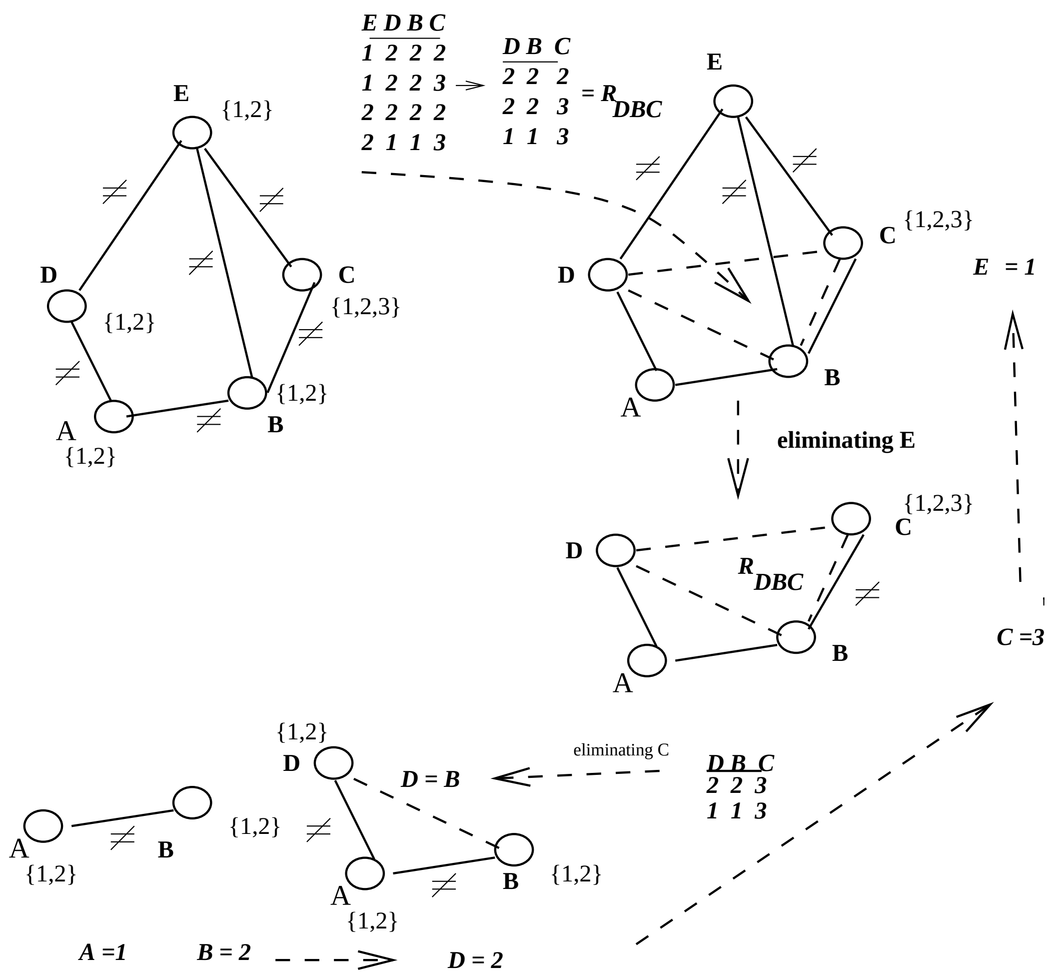
Eliminate variables one by one:



Solution generation process is backtrack-free

The Idea of Elimination

Eliminate variables one by one:



Solution genration process is backtrack-free

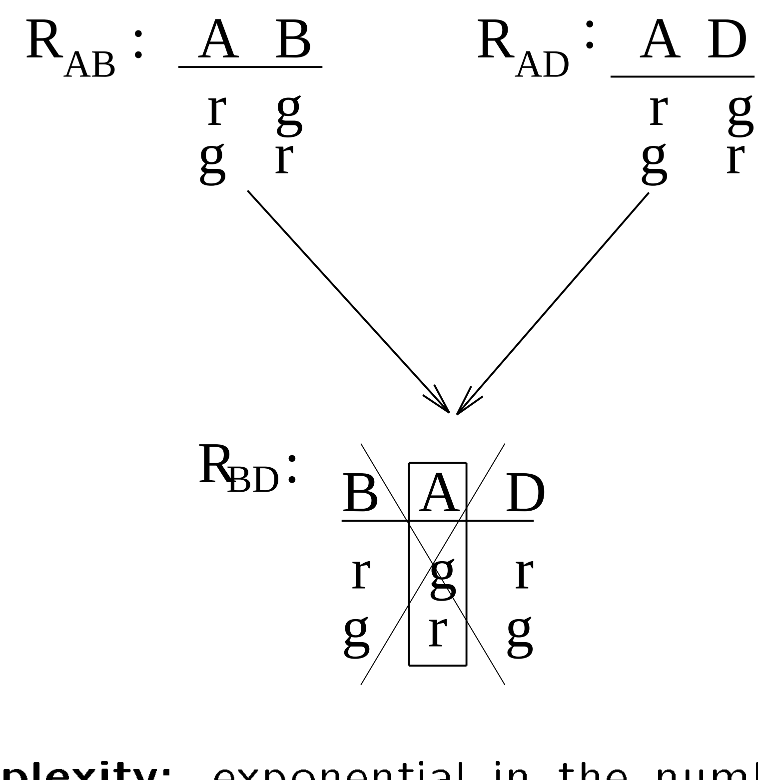
Bucket Operation:

Join followed by projection

Finding all solutions of constraints $R_1, ..., R_n$ using join:

$$Solutions \ = \ R_1 \bowtie R_2 \bowtie, ..., \bowtie R_m$$

The operation in bucket E:
Join: $R_{EBCD} \leftarrow R_{EB} \bowtie R_{ED} \bowtie R_{EC}$
Project: $R_{BCD} = \Pi_{BCD}(R_{BCDE})$

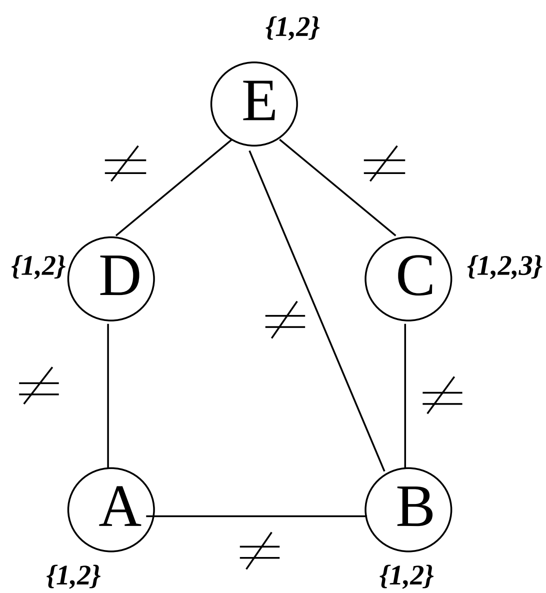


Join complexity: exponential in the number of variables.

Adaptive Consistency

Bucket elimination

(Dechter and Pearl 1987, Seidel, 1981)

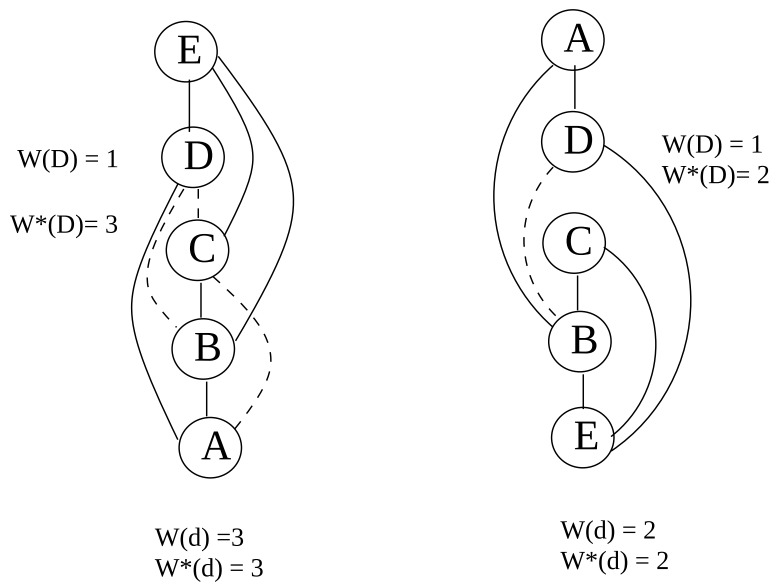


$Bucket(E)$: $E \neq D$, $E \neq C$ $E \neq B$
$Bucket(D)$: $D \neq A$, $||$, R_{DCB}
$Bucket(C)$: $C \neq B$ $||$ R_{ACB}
$Bucket(B)$: $B \neq A$, $||$ R_{AB}
$Bucket(A)$: $||$ R_A

$Bucket(A)$: $A \neq D$, $A \neq B$
$Bucket(D)$: $D \neq E$, $||$ R_{DB}
$Bucket(C)$: $C \neq B$ $C \neq E$,
$Bucket(B)$: $B \neq E$, $||$ R_{BE}, R_{BE}
$Bucket(E)$: $||$, R_E

Width and Induced Width

- **Width** of an ordered graph: $w(d)$
  The maximum number of earlier neighbors.

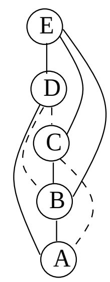


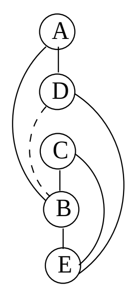
- **Induced width:** $w^*(d)$.
  The width in the *ordered induced graph*, generated by recursively connecting parents.

Width and Induced Width

- **Width** of an ordered graph: $w(d)$

  The maximum number of earlier neighbors.



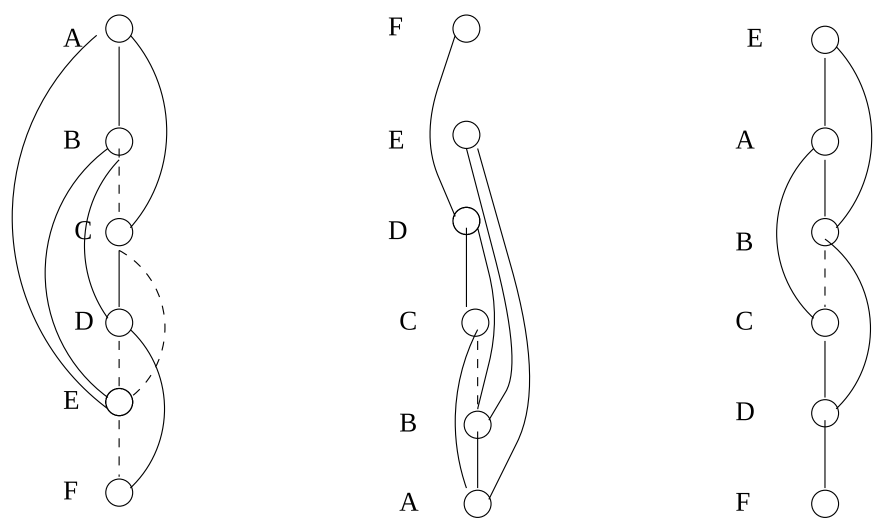


- **Induced width:** $w^*(d)$.

  The width in the *ordered induced graph*, generated by recursively connecting parents.

Width and induced width

- **Width** of an ordered graph: $w(d)$
  The maximum number of earlier neighbors.



(a) (b) (c)

- **Induced width:** $w^*(d)$.
  The width in the *ordered induced graph*, generated by recursively connecting parents.

More on Induced-width (tree-width)

- Finding minimum w* is NP-complete (Arnborg, 1985).
- Greedy ordering algorithms : min-width ordering, min induced-width (Bertele, Briochi 1972, Freuder 1982).
- Approximation orderings.
- The induced width of a given ordering is easy to compute.
- $n \times n$ grids have width of 2 but induced-width of n.
- Trees have induced-width of 1.
- Tree-width equals induced-width $+1$.

Adaptive Consistency

Initialize: Partition constraints into $bucket_1, ...bucket_n$.
For $p = n$ *downto* 1, process $bucket_p$

  for all relations $R_1, ...R_m \in bucket_p$ do
  $R_{new} \leftarrow$ Find solutions to $bucket_p$ and project out X_p.
    If R_{new} is not empty, then add
    to appropriate lower bucket.

Return $\cup_j bucket_j$.

$$R_{new} \leftarrow \Pi_{(-X_p)}(\bowtie_{j=1}^{m-1} R_j)$$

Properties of Elimination: Tractable classes

Theorem:
Adaptive-consistency generates a problem that can be solved without deadends (backtrack-free).

Theorem:
The time and space complexity of Adaptive-consistency along d is $O(exp(w*(d)))$.

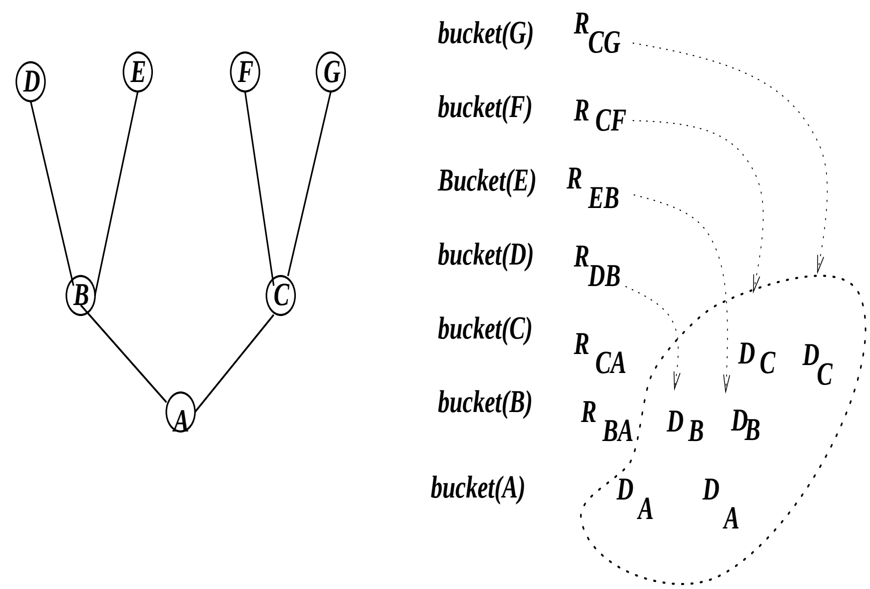
Conclusion:
Problems having bounded induced-width ($w^* \leq b$) can be solved in polynomial time.

Special cases:
Trees and series-parallel networks.

Solving Trees

(Mackworth and Freuder 1985)

Adaptive-consistency is linear for trees.



Only domain (unary) constraints are recorded. This is known as **arc-consistency**.

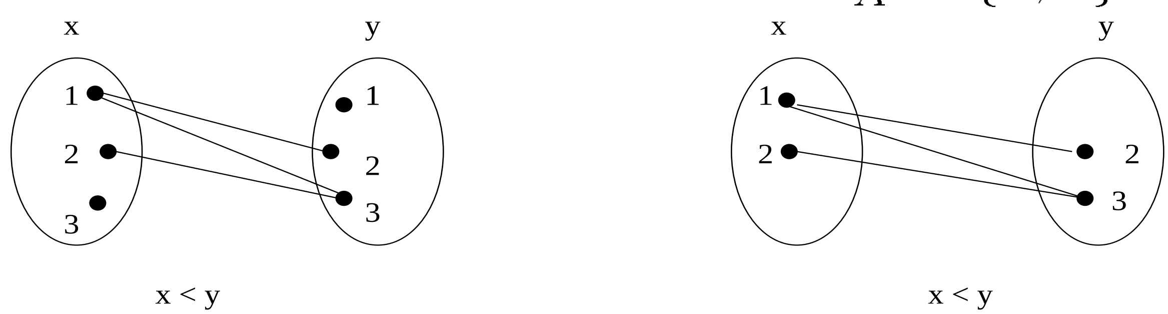
Adaptive consistency is equivalent to enforcing directional arc-consistency for trees.

Arc-Consistency

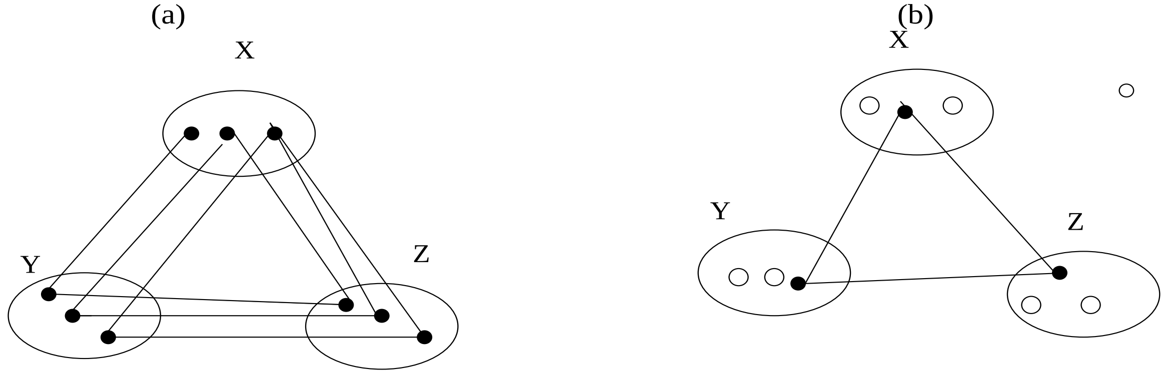
When only domain (unary) constraints are recorded, the operation is called **arc-consistency**.

$$R_A \leftarrow \Pi_A R_{AB} \bowtie D_B$$

Example: $R_A = \{1, 2, 3\}$, $R_B = \{1, 2, 3\}$,
$A < B$ reduces domain of A to $R_A = \{1, 2\}$.



(a) (b)

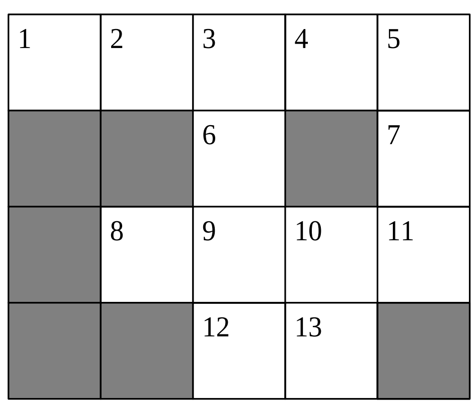


(a) (b)

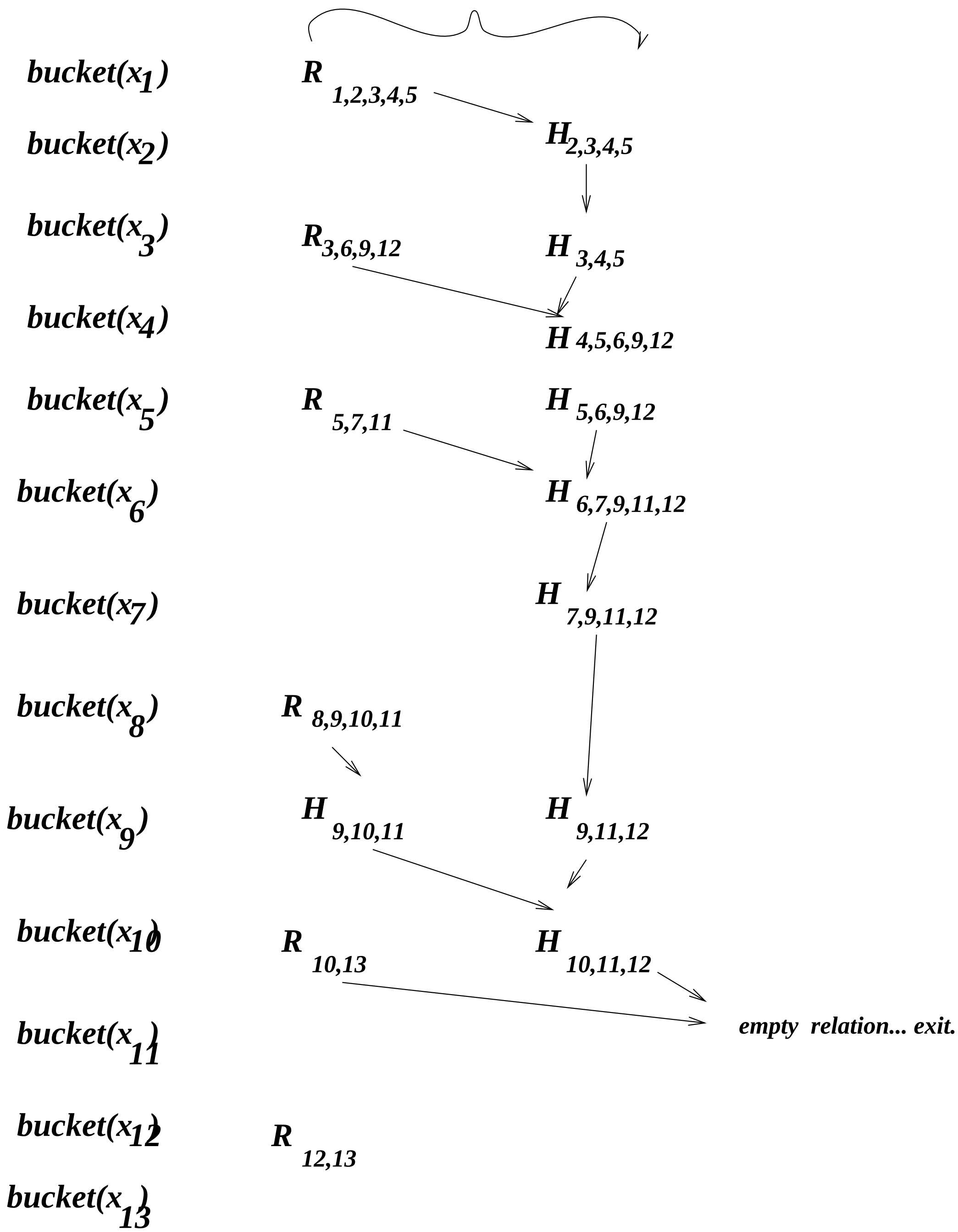
Allows distributed message passing.

Crossword Puzzle

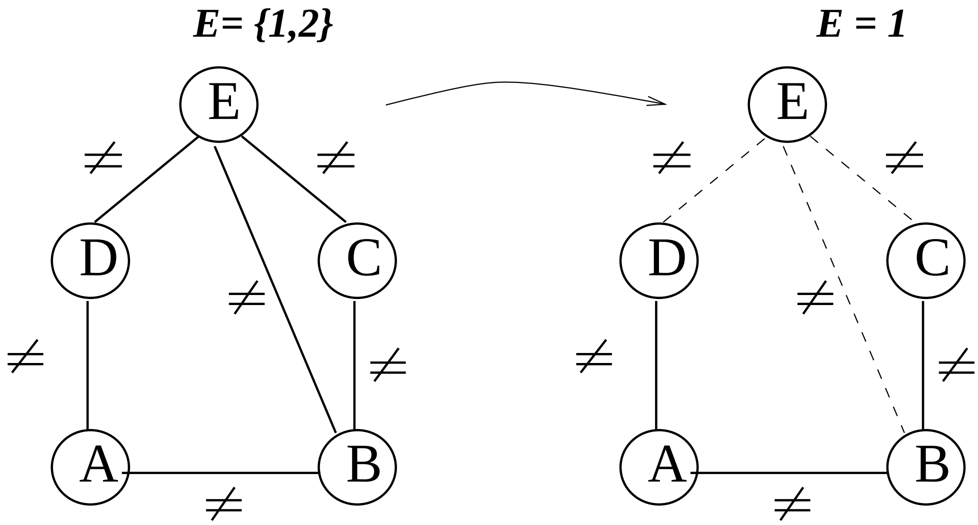
$$
\begin{aligned}
R_{1,2,3,4,5} &= \{(\mathrm{H,O,S,E,S}),\ (\mathrm{L,A,S,E,R}),\ (\mathrm{S,H,E,E,T}),\\
&\quad (\mathrm{S,N,A,I,L}),\ (\mathrm{S,T,E,E,R})\}\\
R_{3,6,9,12} &= \{(\mathrm{H,I,K,E}),\ (\mathrm{A,R,O,N}),\ (\mathrm{K,E,E,T}),\ (\mathrm{E,A,R,N}),\\
&\quad (\mathrm{S,A,M,E})\}\\
R_{5,7,11} &= \{(\mathrm{R,U,N}),\ (\mathrm{S,U,N}),\ (\mathrm{L,E,T}),\ (\mathrm{Y,E,S}),\\
&\quad (\mathrm{E,A,T}),\ (\mathrm{T,E,N})\}\\
R_{8,9,10,11} &= R_{3,6,9,12}\\
R_{10,13} &= \{(\mathrm{N,O}),\ (\mathrm{B,E}),\ (\mathrm{U,S}),\ (\mathrm{I,T})\}\\
R_{12,13} &= R_{10,13}
\end{aligned}
$$



Crossword Puzzle



The Power of Assignments

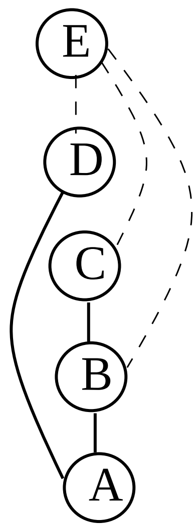


$E = 1$ is an assignment. An observation.

$Bucket(E)$: $E \neq D$, $E \neq C$ $E \neq B$, $\mathbf{E = 1}$
$Bucket(D)$: $D \neq A$ $||$ $R_D = \{2\}$
$Bucket(C)$: $C \neq B$, $||$ $R_C = \{2, 3\}$
$Bucket(B)$: $B \neq A$, $||$ $R_B = \{2\}$
$Bucket(A)$:

Case of observed buckets:
Assign value to each relation separately
Graph effect:
Delete all arcs incident to observation.
Reduced complexity: based on $w*$ of modified graph.

The power of assignments



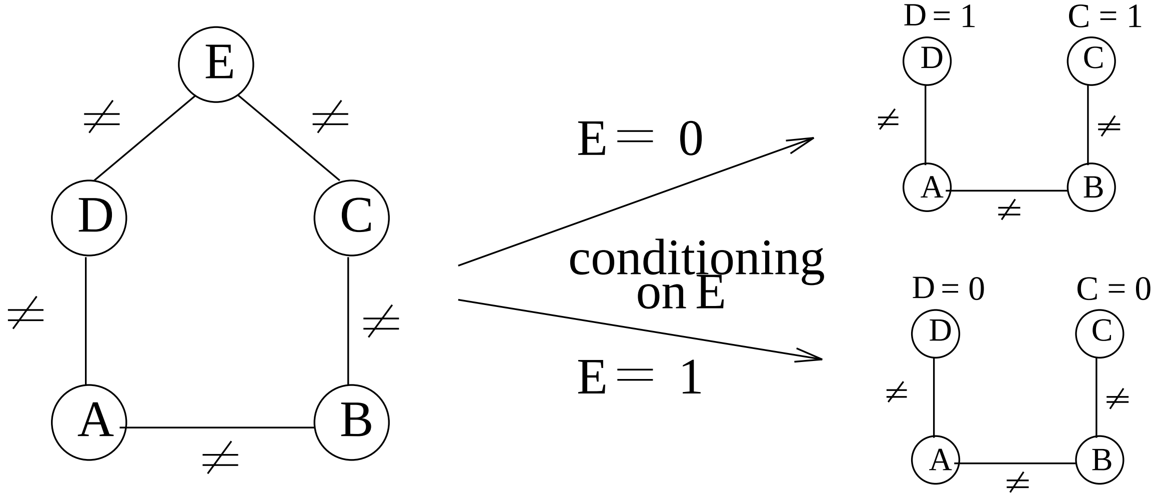
$E = 1$ is an assignment. An observation.

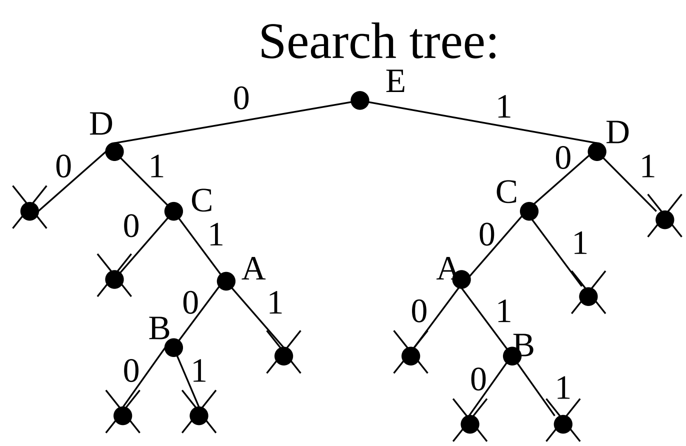
$Bucket(E)$: $E \neq D$, $E \neq C$ $E \neq B$, $\mathbf{E = 1}$
$Bucket(D)$: $D \neq A$ || $D_D = \{2\}$
$Bucket(C)$: $C \neq B$, || $D_C = \{2, 3\}$
$Bucket(B)$: $B \neq A$, || $D_B = \{2\}$
$Bucket(A)$:

Case of observed buckets:
Assign value to each relation seprately
Graph effect:
Delete all arcs incident to observation.
Reduced complexity: based on $w*$ of modified graph.

The idea of Conditioning:

Conditioning exploit the power of assignment:





Basic step: guessing, conditioning.

Leads to backtracking search.

Complexity: exponential time, linear space.

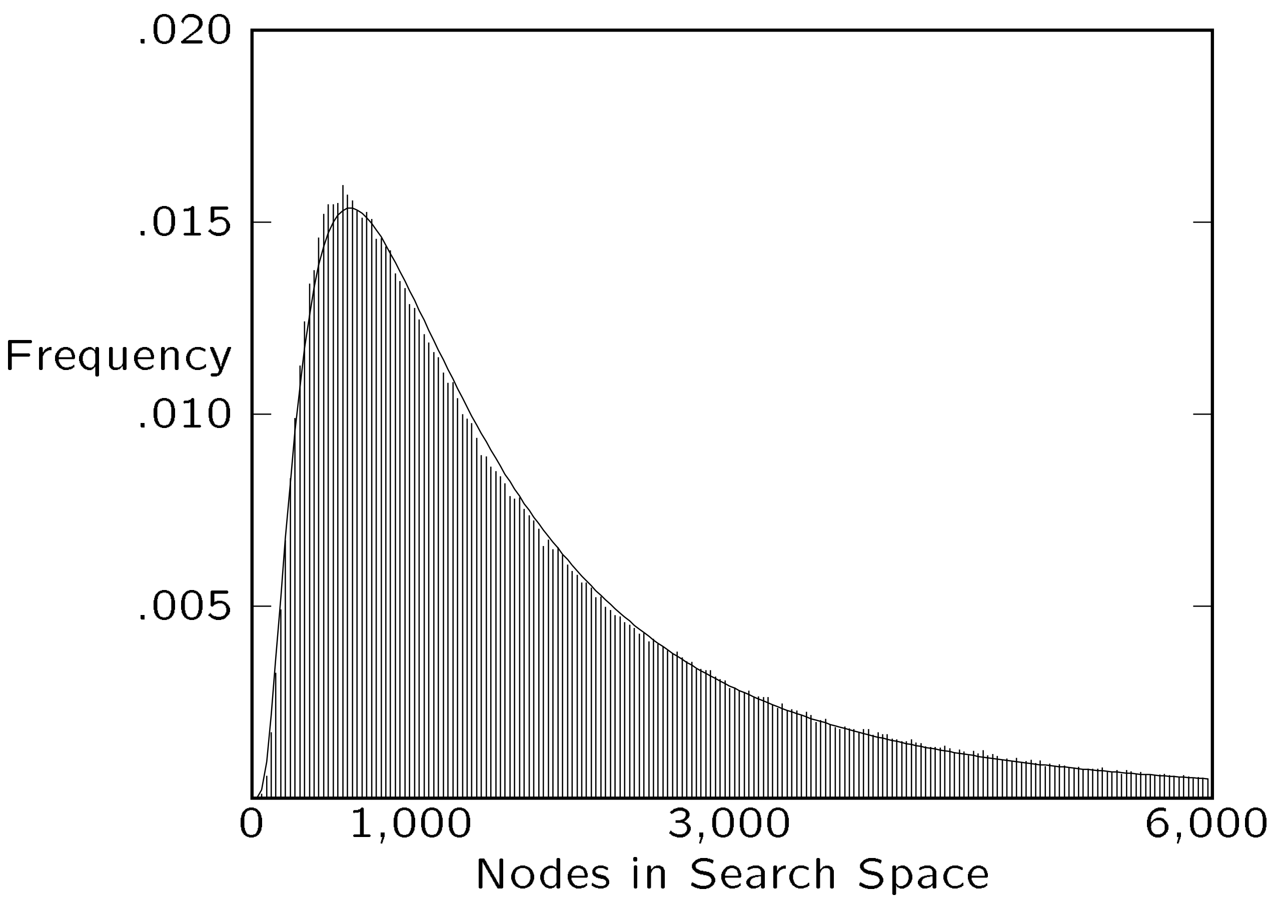
Variety of Backtracking Algorithms

Simple Backtracking +
variable/value ordering heuristics +
constraint propagation + smart backjumping
+ learning no-goods+ ...

- Forward Checking [Haralick & Elliot, 1980]
- Backjumping [Gaschnig 1977, Dechter 1990, Prosser, 1993]
- Backmarking [Gaschnig 1977]
- BJ+DVO [Frost & Dechter 1994]
- Constraint learning [Dechter 1990] [Frost & Dechter 1994] [Bayardo & Miranker, 1996]

Search Complexity Distributions

Complexity histograms (deadends, time) $\Rightarrow$ continuous distributions [Frost, Rish, Vila, 1997]:



BJ-DVO on unsolvable binary CSPs

Complexity Comparison

	Backtracking	Elimination
Worst-case time	O(exp(n))	O(n exp(w*)) w*$\leqslant$ n
Average time	better than worst-case	same
Space	O(n)	O(n exp(w*)) w*$\leqslant$ n
Output	one solution	knowledge compilation

Pair-wise Elimination

(Dechter and van Beek, 1997)

In certain problem pair-wise elimination suffices.

Simultaneous Join-project elimination

$$\begin{aligned} Bucket(E) = \quad & \{R_{ED}, R_{EC}, R_{EAB}\} \\ \rightarrow \quad & R_{ABCD} \end{aligned}$$

Pair-wise elimination:

$$Bucket(E) = \quad \{R_{ED}, R_{EC}, R_{EAB}\}$$

$$\rightarrow \quad R_{DC}, \quad R_{ADB}, \quad R_{ACB}$$

Pair-wise elimination is complete for:

- Linear inequalities
- propositional variables
- Crossword puzzles

Bucket elimination for linear inequalities

$Bucket(x)$:

$$\{x - y \leq 17, \quad 5x + 2.5y + z \leq 84, \quad t - x \leq 2\} \rightarrow$$

$$5t + 2.5y + 5z \leq 94, \qquad t - y \leq 19$$

Linear elimination:
$\sum_{i=1}^{(r-1)} a_i x_i + a_r x_r \leq c$,
$\sum_{i=1}^{(r-1)} b_i x_i + b_r x_r \leq d$.

$$\sum_{i=1}^{r-1} (-a_i \frac{b_r}{a_r} + b_i) x_i \leq -\frac{b_r}{a_r} c + d.$$

(If a_r, b_r opposite signs)

Fourier elimination:

Bucket elimination algorithm for linear inequalities. Complexity is **not** bounded by the induced-width.

Temporal constraint networks:

A tractable case when inequalities are $x - y \leq 16$, $x \leq 5$.

Fourier Elimination: Bucket-elim for Linear Inequalities

Input: Linear inequalities set, 0

Output: A back-track free set

Intialize: partition into buckets

$$
\begin{array}{ll}
B(x): & x - y \leq 17, 5x + 2.5y + Z = 84, t - x \leq 2 \\
B(t): & t - x \leq 19 \qquad\qquad\qquad 5t + 2.5y + 5Z \leq 94 \\
B(y): & \\
B(z): &
\end{array}
$$

$$
\begin{array}{ll}
B(z): & 5x + 2.5y + z \leq 84 \\
B(y): & x - y \leq 17 \\
B(x): & t - x \leq 2 \\
B(t): &
\end{array}
$$

$$
\begin{array}{ll}
B(t): & t - x \leq 2 \\
B(x): & x - y \leq 17, 5x + 2.5y + Z \leq 84 \\
B(y): & \\
B(z): &
\end{array}
$$

Temporal Constraint Networks

(Dechter, Meiri and Pearl 1990)

Variables:	$X_1, \ldots, X_n$
Domains:	Real numbers
Constraints:	$X_i \leq b, X_i - X_j \leq c$
	binary difference inequalities

Algorithm for STP is Bucket elimination

$B(x) : x - y \leq 5, \quad x > \ 3, \quad t - x \leq 10$
$B(y) : y \leq 10 \ \ || \quad - y \leq 2, \ \ t - y \leq 15$
$B(z) :$
$B(t) : \ || \ t \leq 25$

Algorithm records only Binary constraints of same type

$$\text{Complexity} \Rightarrow 0(n^3)$$
$$\Rightarrow 0(w^* n^2)$$

Summary

1. Bucket elimination for CSPs = Adaptive consistency

2. Performance characterized by induced width of ordered graph. Time and space $O(exp(w_d^*))$.

3. The bucket operation: join-project.

4. Value assignments reduce induced width and reduce complexity.

5. Conditioning: backtracking search
   Worst case time $O(exp(n))$, but much better on average. Linear space.

6. Bucket elimination for linear inequalities = Fourier elimination.

Fourier Elimination

Initialize: partition inequalities into $bucket_1, \ldots, bucket_n$.

For $p \leftarrow n$ **downto** 1

  for each pair $\{\alpha, \beta\} \subseteq bucket_i$,

    compute $\gamma = elim_p(\alpha, \beta)$.

    If γ has no solutions,

    return inconsistency.

    else add γ to

    the appropriate bucket.

return $E_o(\varphi) \leftarrow \bigcup_i bucket_i$.

"Road Map":

Tasks and Methods

Tasks / Methods	CSP	SAT	Optimi-zation	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency, join-tree	directional resolution	dynamic program-ming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset, forward checking	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

Propositional Satisfiability

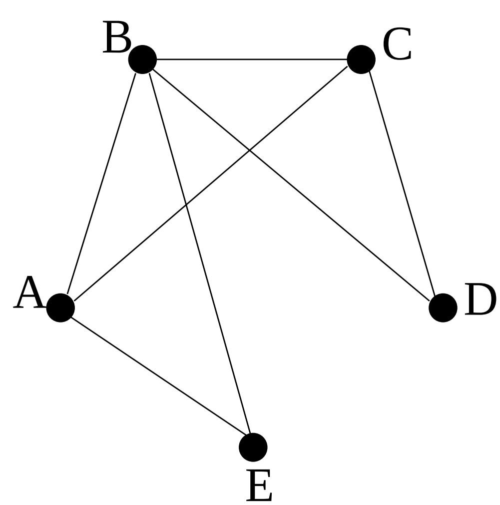
Conjunctive normal form (CNF)

$\varphi = (A \vee B \vee C) \wedge (\neg A \vee B \vee E) \wedge (\neg B \vee C \vee D)$

Is φ satisfiable? If it is, find a solution (*model*).

CNF: conjunction of **clauses**
clause: disjunction of **literals**
literal: A or $\neg A$

Interaction graph:

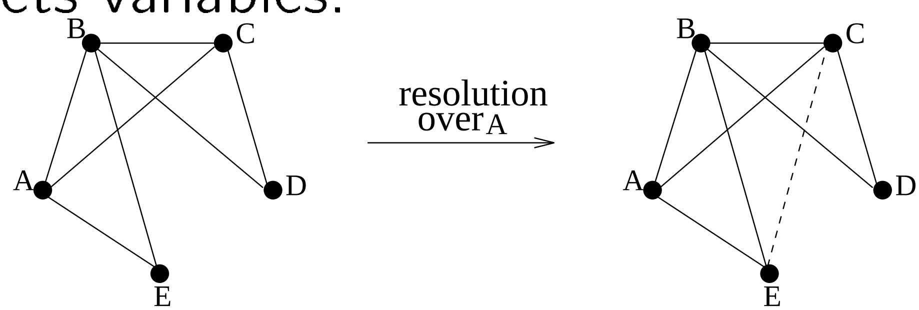


Variables (*propositions*) $\Rightarrow$ nodes
Constraints (*clauses*) $\Rightarrow$ cliques

Elimination: Resolution

The operation in a bucket: pair-wise resolution
$(A \vee B) \wedge (\neg A \vee E) \wedge (A \vee \neg C)$:
$(A \vee B) \wedge (\neg A \vee E) \Rightarrow (B \vee E)$,
$(\neg A \vee E) \wedge (A \vee \neg C) \Rightarrow (E \vee \neg C)$.

Resolution creates clauses $\Rightarrow$
connects variables:



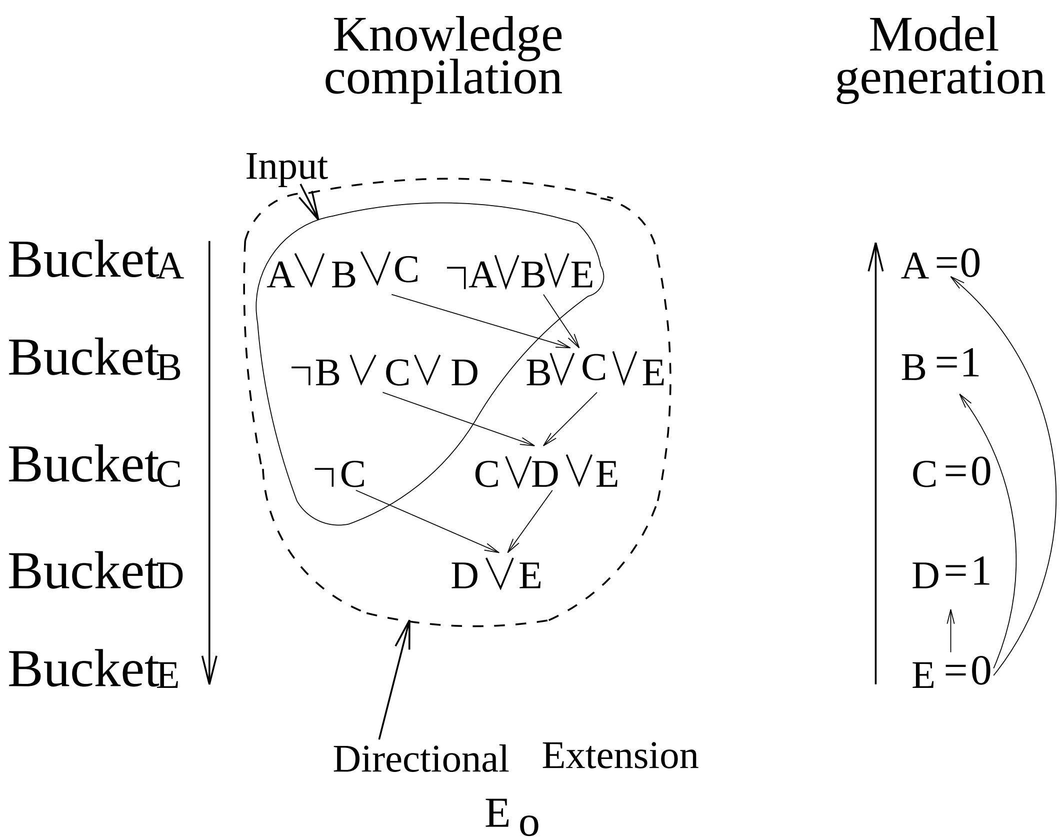
Special case:
Unit resolution - resolution with **unit clauses**:
$\neg A \wedge (A \vee B \vee C) \Rightarrow (B \vee C)$

Unit propagation - unit resolution until no unit clause is left.

Directional Resolution

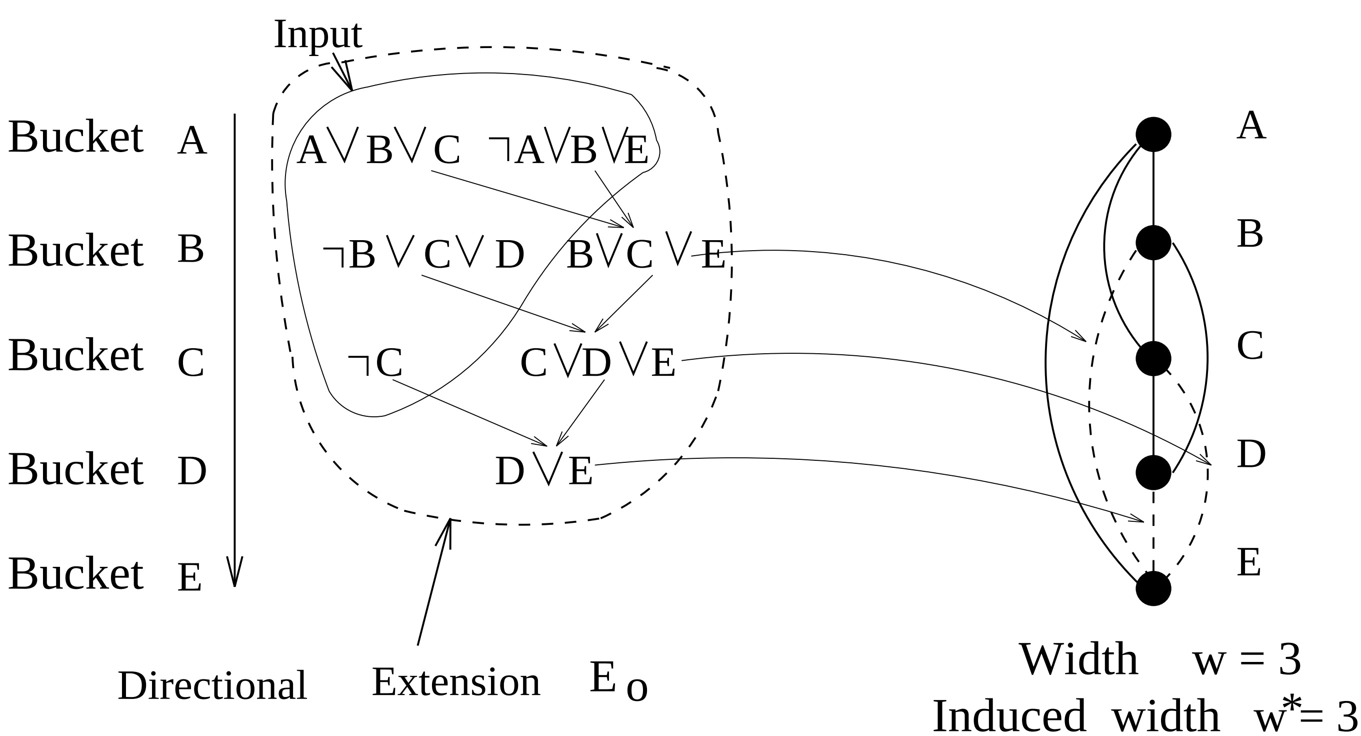
Bucket Elimination

$$\varphi = \neg C \wedge (A \vee B \vee C) \wedge (\neg A \vee B \vee E) \wedge (\neg B \vee C \vee D)$$



Resolution: logical inference (**"thinking"**)

DR Complexity



$$|bucket_i| = O(exp(w^*)) \Rightarrow |E_o| = O(nexp(w^*))$$

$$\Downarrow$$

Time(DR) and Space(DR) $= O(nexp(w^*))$

Directional Resolution (DR)
[Davis,Putnam, 1960] [Dechter, Rish, 1994]

Input: A cnf theory φ, $d = Q_1, ..., Q_n$.
Output: A *directional extension* $E_d(\varphi)$,
equivalent to φ; $E_d(\varphi) = \emptyset$ iff φ is unsatisfiable.
1. **Initialize:** generate a partition of clauses,
$bucket_1, ..., bucket_n$, where $bucket_i$ contains
all the clauses whose highest literal is Q_i.
2. **For** $i = n$ to 1 do:
    Resolve each pair
    $\{(\alpha \vee Q_i), (\beta \vee \neg Q_i)\} \subseteq bucket_i$.
    If $\gamma = \alpha \vee \beta$ is empty,
    return $E_d(\varphi) = \emptyset$,
    else add γ to the appropriate bucket.
3.Return $E_d(\varphi) \leftarrow \bigcup_i bucket_i$.

Conditioning: Assignment

Conditioning adds a literal to φ

$A = 0 \Rightarrow \neg A \wedge \varphi$

$A = 1 \Rightarrow A \wedge \varphi$

Conditioning implies:

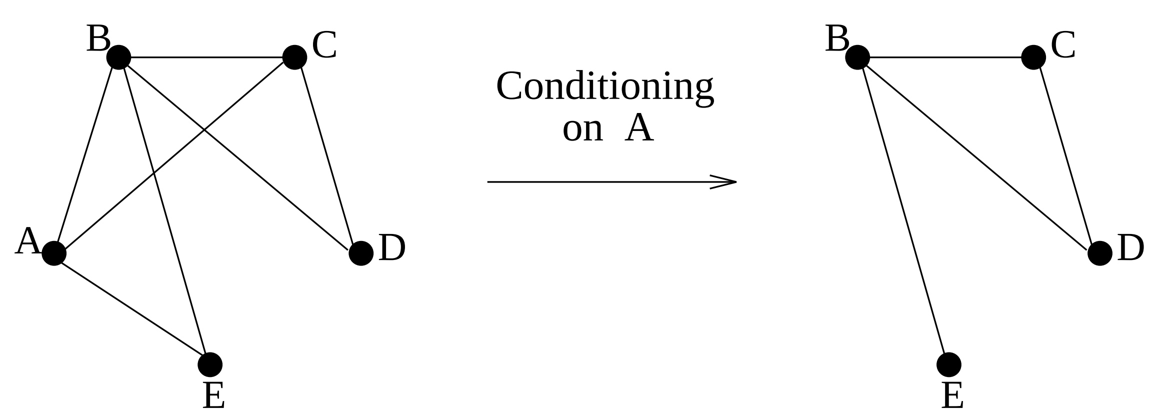
- *unit resolution*:

$A = 0 \Rightarrow \neg A \wedge (A \vee B \vee C) \Rightarrow (B \vee C)$

- deleting *tautologies*:

$A = 0 \Rightarrow \neg A \wedge (\neg A \vee B \vee E) \Rightarrow$ clause $(\neg A \vee B \vee E)$ is deleted from φ.

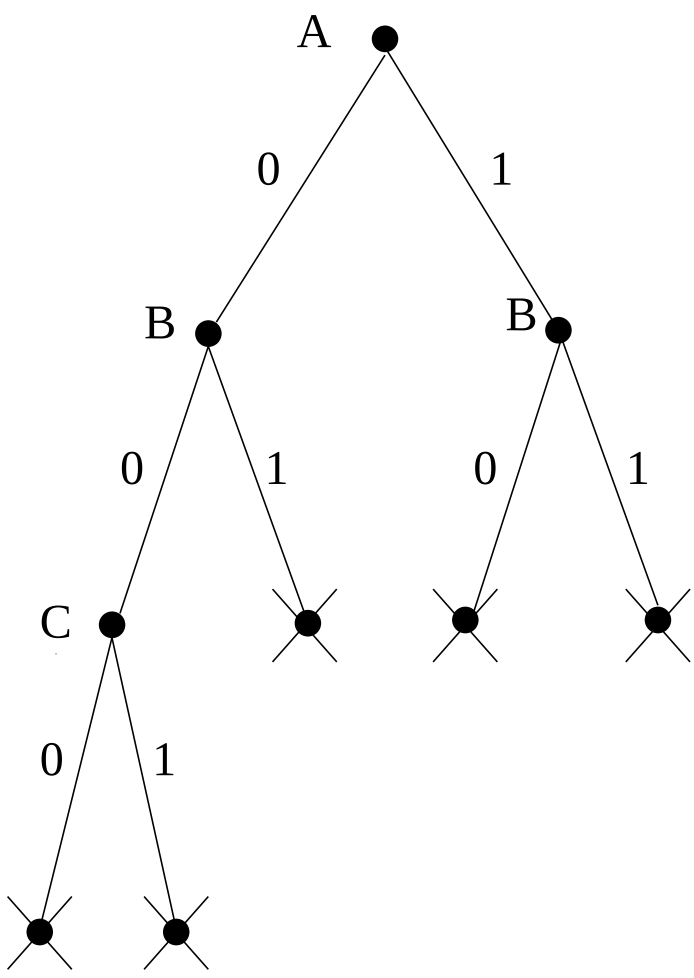
- deleting a variable from the graph



Backtracking Search

Conditioning

$$\varphi = (\neg A \vee B) \wedge (\neg C \vee A) \wedge \neg B \wedge C$$



Search: **"guessing"** (partial) solutions

The Davis-Putnam Procedure

[Davis, Logemann, Loveland, 1962]

DP(φ)
Input: A cnf theory φ.
Output: A decision of whether φ is satisfiable.
1. Unit_propagate(φ);
2. If the empty clause generated return(*false*);
3. else if all variables are assigned return(*true*);
4. else
5. $\quad$ $Q =$ some unassigned variable;
6. $\quad$ return(DP($\varphi \wedge Q$) $\vee$
7. $\quad\quad$ DP($\varphi \wedge \neg Q$))

Historical Perspective

- 1960 - resolution-based Davis-Putnam algorithm.

- 1962 - original Davis-Putnam was replaced by conditioning procedure [Davis, Logemann and Loveland, 1962] due to memory explosion, resulting in a backtrack search known as the **Davis-Putnam(-Logemann-Loveland) procedure**.

- The dependency on a graph parameter called **induced width** was not known in 1960.

- 1994 - *Directional Resolution*, a rediscovery of the original Davis-Putnam [Dechter and Rish, 1994]. Identification of tractable classes.

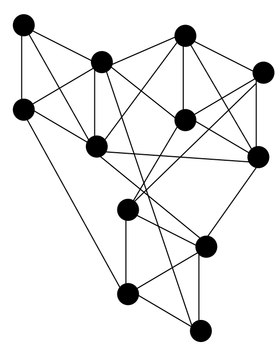
Experimental Results:

DP vs DR on k-CNFs

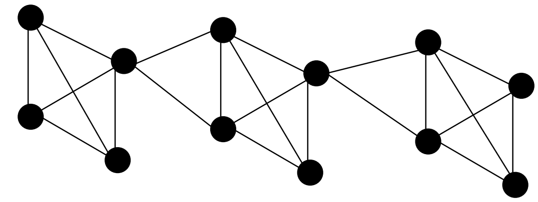
[Dechter and Rish, 1994

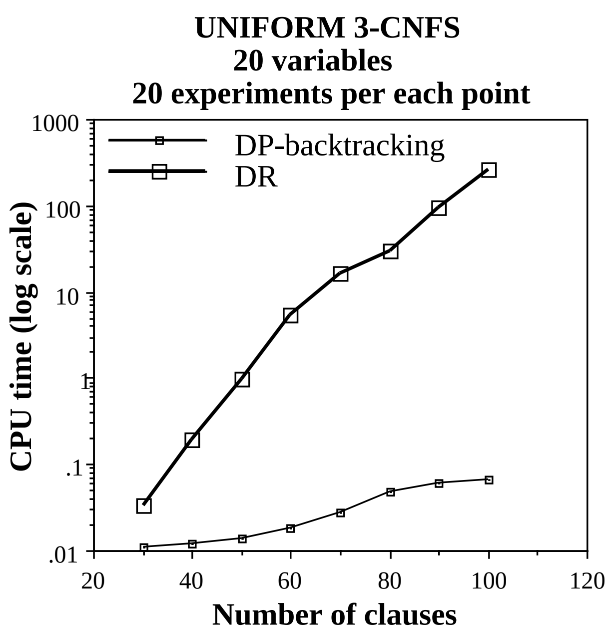
1. **Uniform random 3-CNF**: N variables, C clauses
2. **Random (k,m)-tree**: a tree of $k+m$-node cliques with k-node intersections (clique separators)

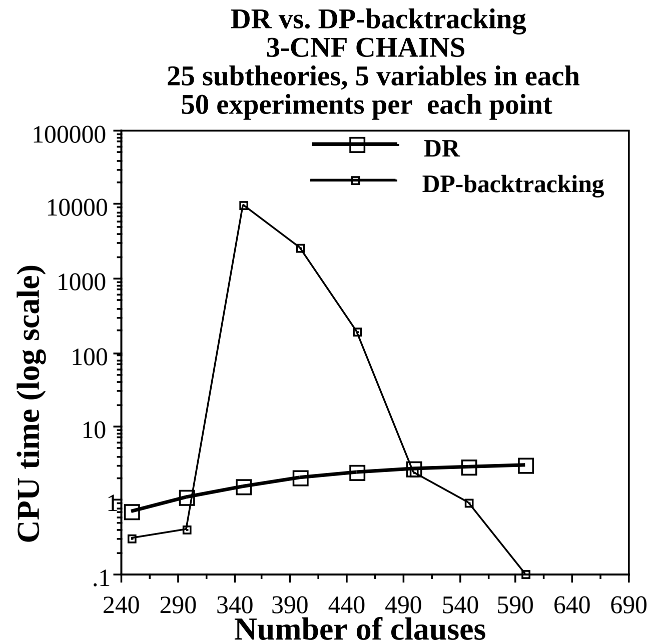
Uniform random 3-CNFs:



(k,m)-tree CNFs:







Why Hybrids?

	Backtracking	Elimination
Worst-case time	O(exp(n))	O(n exp(w*)) $w^* \leqslant n$
Average time	better than worst-case	same
Space	O(n)	O(n exp(w*)) $w^* \leqslant n$
Output	one solution	knowledge compilation

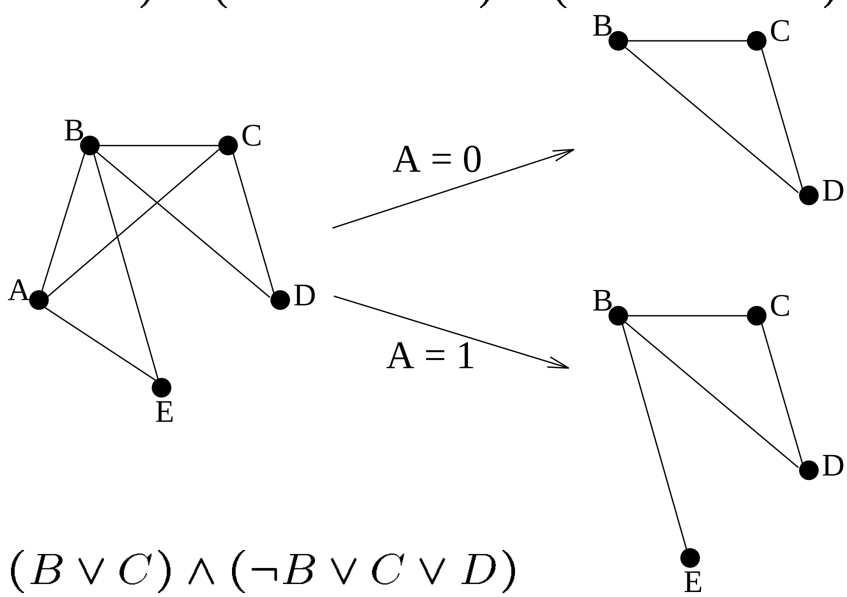
Backtracking + Resolution = Hybrids

Conditioning (backtracking)

+ Elimination (resolution)

[Rish and Dechter, 1996]

$\varphi = (A \vee B \vee C) \wedge (\neg A \vee B \vee E) \wedge (\neg B \vee C \vee D)$



$A = 0 \Rightarrow (B \vee C) \wedge (\neg B \vee C \vee D)$
$A = 1 \Rightarrow (B \vee E) \wedge (\neg B \vee C \vee D)$

Idea:

conditioning reduces w^*

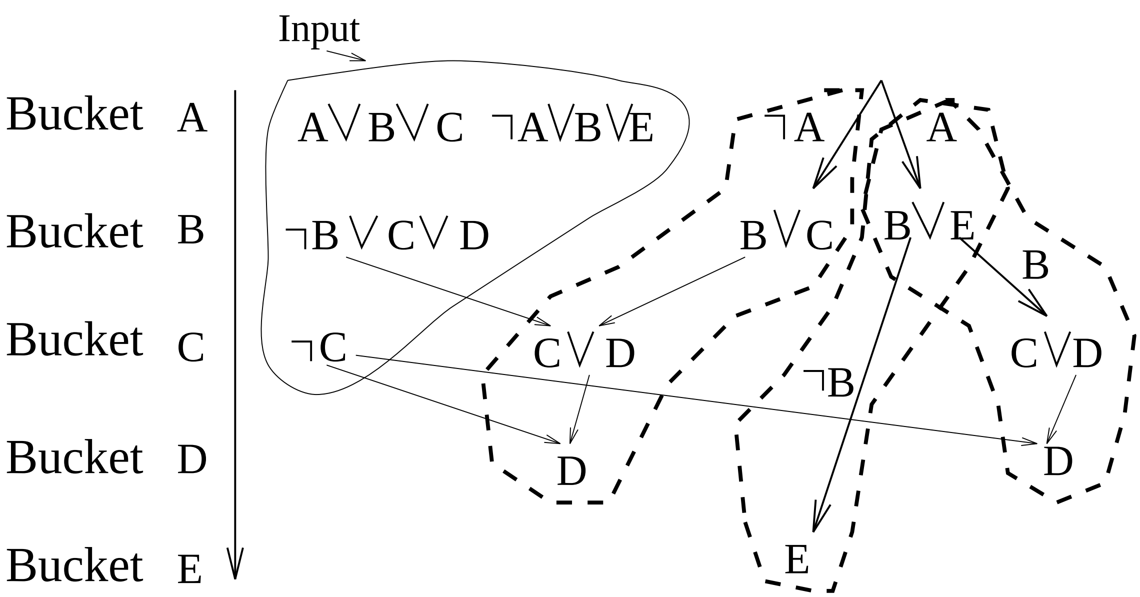
$+$

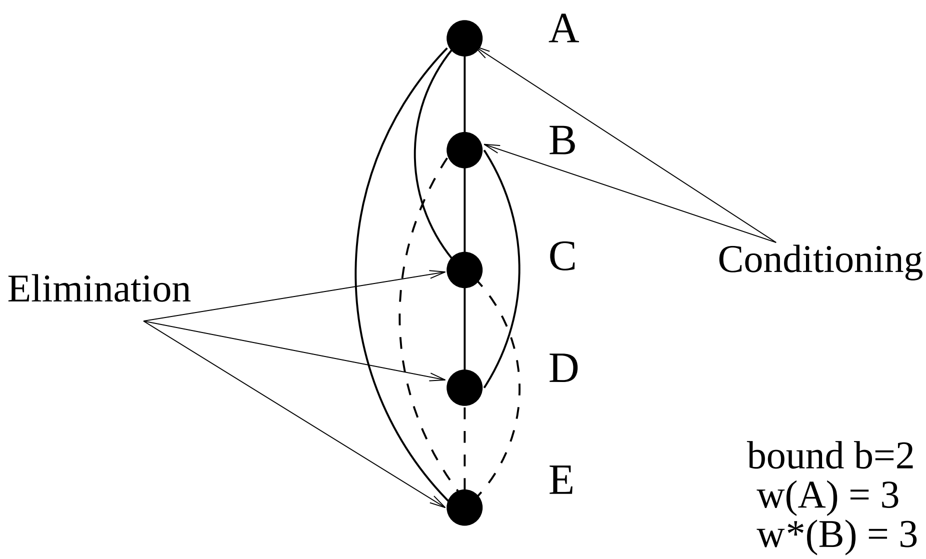
elimination *guarantees* $O(exp(w^*)), w^* < n$

Conditioning+DR:

Algorithm $DCDR(b)$

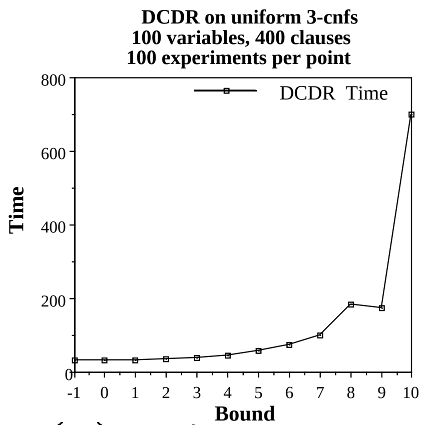
Resolve if $w^*(X_i) < b$, otherwise condition.



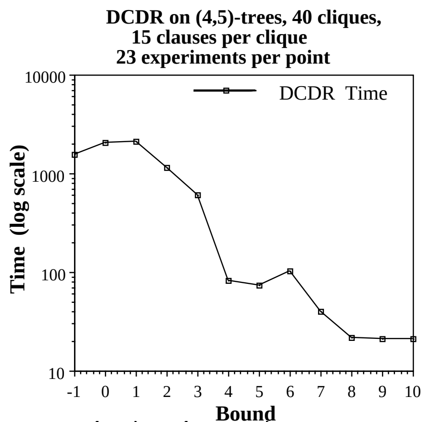


DCDR(b):

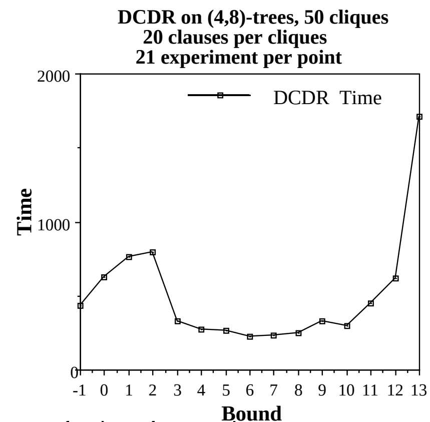
Experimental Results



(a) uniform CNFs

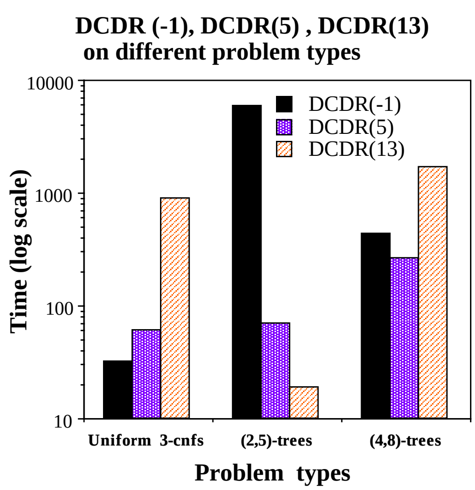


(b) (4,5)-trees



(c) (4,8)-trees

Summary:



$b < 0$: pure DP

$b \geq w^*$: pure DR

$0 \leq b < w^*$: pure DR

Time $exp(b + |cond(b)|)$, space $exp(b)$

Summary

1. Bucket elimination: Directional Resolution (resolution-based Davis-Putnam).
   Time and space $O(exp(w_o^*))$.

2. Conditioning: backtracking search (backtracking-based Davis-Putnam Procedure).
   Time $O(exp(n))$, better on average; space $O(n)$.

3. Conditioning (Backtracking) $+$
   Elimination (Resolution):
   Conditioning when $w^* \geq b$, resolution otherwise.
   Time $exp(b + |cond(b)|)$, space $exp(b)$.

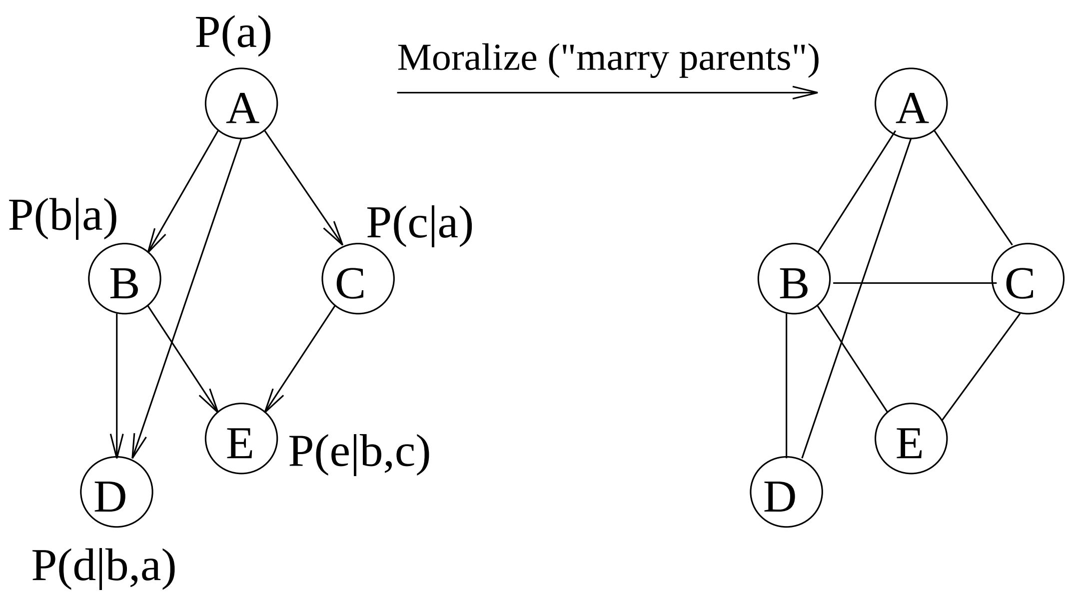
"Road Map":

Tasks and Methods

Tasks / Methods	CSP	SAT	Optimi-zation	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency, join-tree	directional resolution	dynamic program-ming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset, forward checking	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

Belief Networks

• Belief networks are acyclic directed graphs annotated with conditional probability tables.



Tasks (NP-hard):

• belief-updating (BEL)

• Finding most probable explanation (MPE)

• Finding maximum aposteriori hypothesis (MAP)

• Finding maximum expected utility (MEU)

Common Queries

1. **Belief assessment:**
   Find $bel(x_i) = P(X_i = x_i|e)$.

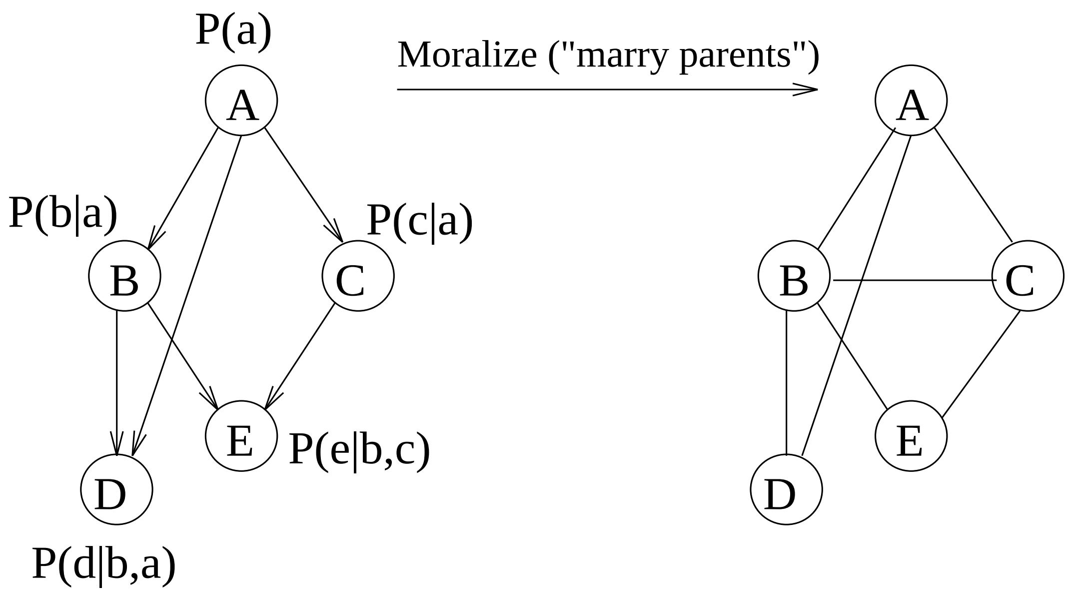
2. **Most probable explanation** (MPE):
   Find x^o s.t. $p(x^o) = \max_{\bar{x}n} \Pi_{i=1}^n P(x_i|x_{pa_i}, e)$.

3. **Maximum aposteriori hypothesis (**MAP**):**
   Given $A = \{A_1, ...A_k\} \subseteq X$, find $a^o = ({a^o}_1, ...{a^o}_k)$ s.t.
   $p(a^o) = \max_{\bar{a}_k} \sum_{x_{X-A}} \Pi_{i=1}^n P(x_i|x_{pa_i}, e)$.

4. **Maximum expected utility (**MEU**):**
   Given $u(x) = \sum_{Q_j \in Q} f_j(x_{Q_j})$, find decisions $d^o = ({d^o}_1, ..., {d^o}_k)$
   $\max_d \sum_{x_{k+1},...,x_n} \Pi_{i=1}^n P(x_i|x_{pa_i,d})u(x)$.

Belief Updating

$P(a|e=0) = \alpha P(a, e=0).$



Ordering: $a, \; b, \; c, \; d, \; e$

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

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

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

Ordering: $a, \; e, \; d, \; c, \; e$

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

$$P(a, e=0) = P(a) \sum_e \sum_d \sum_c P(c|a) \sum_b P(b|a) P(d|a,b)$$

$$P(e|b,c)$$

Backwards Computation = Elimination

Ordering: a, b, c, d, e

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

$= P(a)\sum_b P(b|a)\sum_c P(c|a)P(e=0|b,c)\sum_d P(d|b,a)$

$= P(a)\sum_b P(b|a)\lambda_D(a,b)\sum_c P(c|a)P(e=0|b,c)$
$= P(a)\sum_b P(b|a)\lambda_D(a,b)\lambda_C(a,b)$
$= P(a)\lambda_B(a)$

The Bucket elimination process:

$bucket(E) = \quad P(e|b,c),\ e=0$
$bucket(D) = \quad P(d|a,b)$
$bucket(C) = \quad P(c|a)$
$bucket(B) = \quad P(b|a)$
$bucket(A) = \quad P(a)$

Backwards Computation, Different Ordering

Ordering: a, e, d, c, b

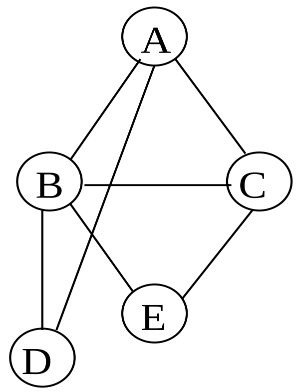
$P(a, e=0) = P(a) \sum_{e=0} \sum_d \sum_c P(c|a) \sum_b P(b|a)$
$P(d|a, b) P(e|b, c)$

$P(a) \sum_{e=0} \sum_d \sum_c P(c|a) \lambda_B(a, d, c, e)$
$P(a) \sum_{e=0} \sum_d \lambda_C(a, d, e)$
$P(a) \sum_{e=0} \lambda_D(a, e)$
$P(a) \lambda_D(a, e=0)$

The bucket elimination Process:

$$
\begin{aligned}
bucket(B) = &\quad P(e|b, c), P(d|a, b), P(b|a) \\
bucket(C) = &\quad P(c|a) \;\; || \; \lambda_B(a, d, c, e) \\
bucket(D) = &\quad \qquad\quad || \;\; \lambda_C(a, d, e) \\
bucket(E) = &\quad e = 0 \;\; || \;\; \lambda_D(a, e) \\
bucket(A) = &\quad P(a) \;\; || \;\; \lambda_D(a, e=0)
\end{aligned}
$$

Bucket Elimination and Induced Width



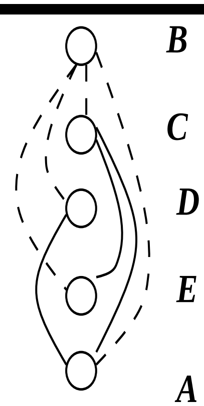
Ordering: a, b, c, d, e

$bucket(E) = \quad P(e|b,c),\ e = 0$
$bucket(D) = \quad P(d|a,b)$
$bucket(C) = \quad P(c|a) \ \ || \ P(e = 0|b,c)$
$bucket(B) = \quad P(b|a) \ \ || \ \ \lambda_D(a,b), \lambda_C(b,c)$
$bucket(A) = \quad P(a) \ \ || \ \ \lambda_B(a)$

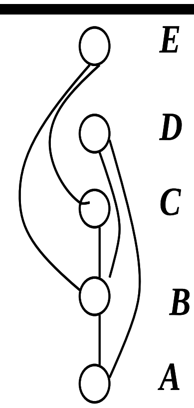
Ordering: a, e, d, c, b

$bucket(B) = \quad P(e|b,c), P(d|a,b), P(b|a)$
$bucket(C) = \quad P(c|a) \ \ || \ \ \lambda_B(a,c,d,e)$
$bucket(D) = \quad || \ \lambda_C(a,d,e)$
$bucket(E) = \quad e = 0 \ \ || \ \lambda_D(a,c)$
$bucket(A) = \quad P(a) \ \ || \ \ \lambda_E(a)$

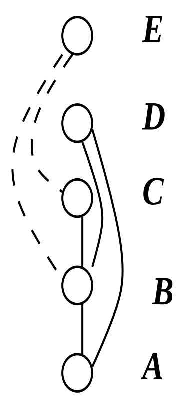
Bucket Elimination and Induced Width



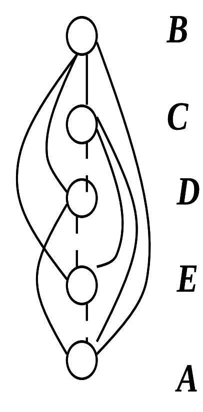
$w^* = 2$



$w^* = 2$

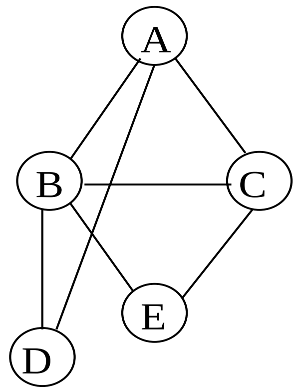


$w^* = 2$



$w^* = 4$

Handling Observations



Observing $b = 1$

Ordering: a, e, d, c, b

$$\begin{aligned}
bucket(B) = &\quad P(e|b,c), P(d|a,b), P(b|a), b = 1 \\
bucket(C) = &\quad P(c|a), \;\; || \;\; P(e|b=1,c) \\
bucket(D) = &\quad \;\; || \;\; P(d|a,b=1) \\
bucket(E) = &\quad e = 0 \;\; || \;\; \lambda_C(e,a) \\
bucket(A) = &\quad P(a), \;\; || \;\; P(b=1|a) \; \lambda_D(a), \lambda_E(e,a)
\end{aligned}$$

Ordering: a, b, c, d, e

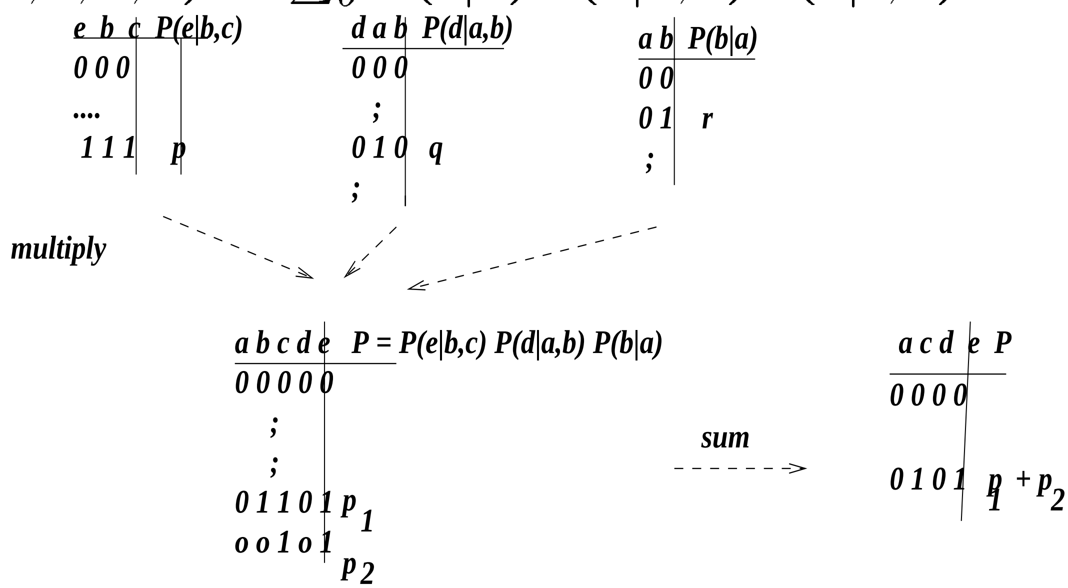
$$\begin{aligned}
bucket(E) = &\quad P(e|b,c), \; e = 0 \\
bucket(D) = &\quad P(d|a,b) \\
bucket(C) = &\quad P(c|a) \;\; || \;\; \lambda_E(b,c) \\
bucket(B) = &\quad P(b|a), b = 1 \;\; || \;\; \lambda_D(a,b), \lambda_C(a,b) \\
bucket(A) = &\quad P(a) \;\; || \;\; \lambda_B(a)
\end{aligned}$$

The Bucket Operation

Elimination: multiply and sum

$bucket(B) = \{P(e|b,c), P(d|a,b), P(b|a)\} \rightarrow$

$\lambda_B(a,c,d,e) = \sum_b P(b|a)P(d|a,b)P(e|b,c)$



Observed bucket:

$bucket(B) = \{P(e|b,c), P(d|a,b), P(b|a), b = 1\} \rightarrow$

$\lambda_B(a) = P(b = 1|a)$

$\lambda_B(a,d) = P(d|a, b = 1)$

$\lambda_B(e,c) = P(e|b = 1, c)$.

Elim-bel

Input: A belief network $\{P_1, ..., P_n\}$, d,e.

Output: belief of X_1 given e.

1. **Initialize:**
2. **Process buckets** from $p = n$ to 1
   for matrices $\lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$ do
   - **If** (observed variable) $X_p = x_p$ assign $X_p = x_p$ to each λ_i.
   - **Else**, (multiply and sum)
     $\lambda_p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i$.
     Add λ_p to its bucket.
3. **Return** $Bel(x_1) = \alpha P(x_1) \cdot \Pi_i \lambda_i(x_1)$

Irrelevant buckets for elim-bel

Buckets that sum to 1 are **irrelevant**.
Identification: no evidence, no new functions.

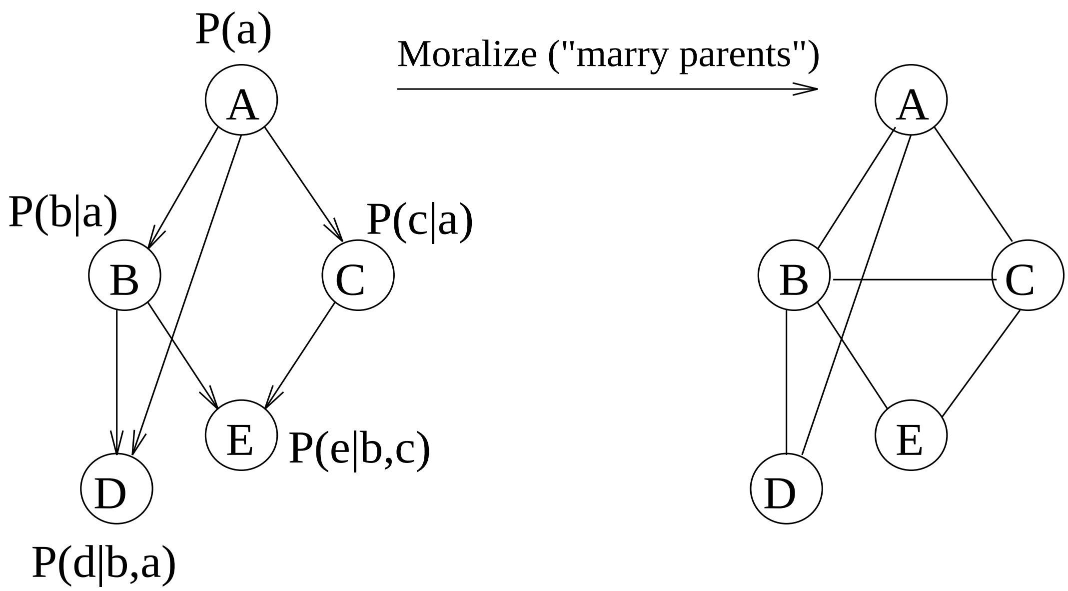
Recursive recognition : ($bel(a|e)$)

$bucket(E) =$ $P(e|b,c),\ e = 0$
$bucket(D) =$ $P(d|a,b)$,...skipable bucket
$bucket(C) =$ $P(c|a)$
$bucket(B) =$ $P(b|a)$
$bucket(A) =$ $P(a)$

Complexity: Use induced width in moral graph without irrelevant nodes, then update for evidence arcs.

Finding the MPE

(An optimization task)



Ordering: $a,\ b,\ c,\ d,\ e$

$m = \max_{a,b,c,d,e=0} P(a,b,c,d,e) =$

$= \max_a P(a) \max_b P(b|a) \max_c P(c|a) \max_d P(d|b,a)$

$\max_{e=0} P(e|b,c)$

Ordering: $a,\ e,\ d,\ c,\ b$

$m = \max_{a,e=0,d,c,b} P(a,b,c,d,e)$

$m = \max_a P(a) \max_e \max_d \cdot$

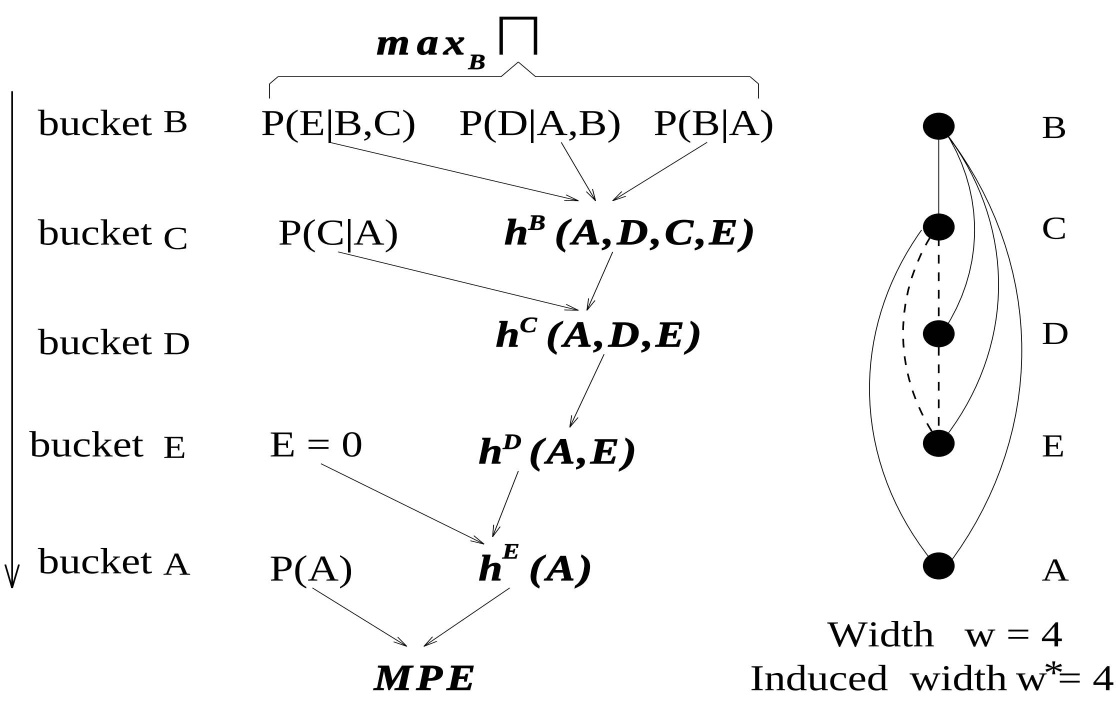
$$\mathsf{max}_c\, P(c|a)\, \mathsf{max}_b\, P(b|a)P(d|a,b)P(e|b,c)$$

Algorithm Elim-mpe

Input: A Belief network $P = \{P_1, ..., P_n\}$

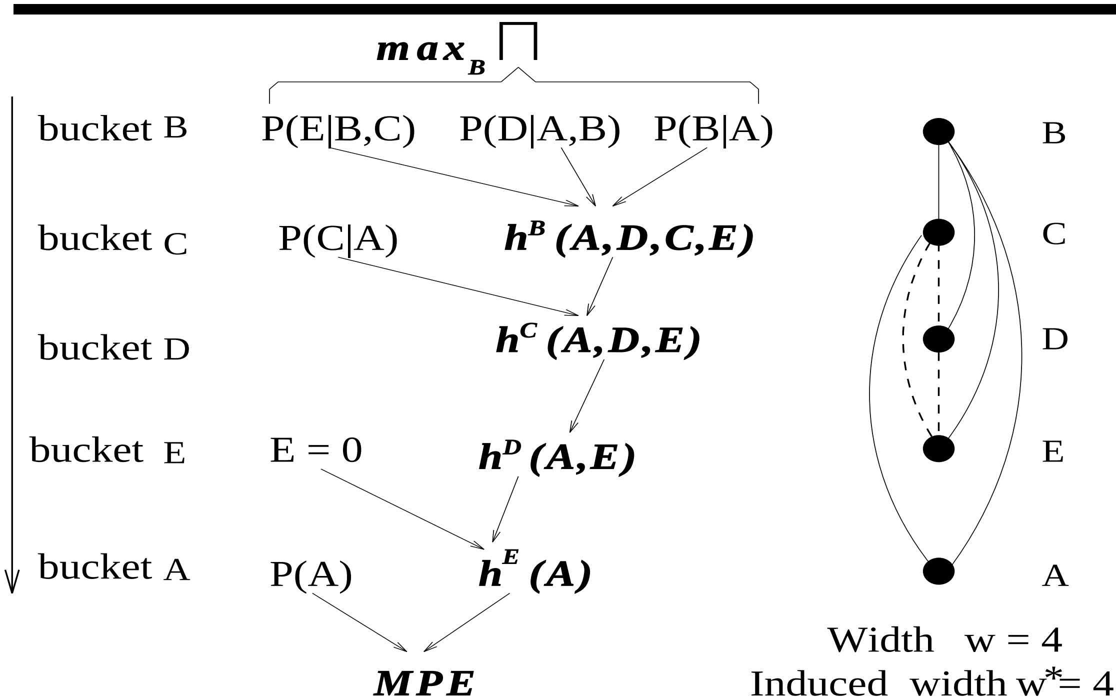
Output: MPE

1. **Initialize:** Partition into buckets.
2. **Process buckets from last to first:**



3. **Forward:** Assign values in ordering d

Generating the MPE Tuple



Step 3:

$a_0 = argmax_a P(a) \cdot h(a)$

$e_0 = E = 0$

$d_0 = argmax_d h(a_0, d, e_0)$

$c_0 = argmax_c P(c|a_0) \cdot h(a_0, d_0, c, e_0)$

$b_0 = argmax_b P(e_0|b, c_0) \cdot P(d_0|a_0, b) \cdot P(b|a_0)$

Return a_0, e_0, d_0, c_0, b_0

Elim-mpe

Input: A belief network $\{P_1, ..., P_n\}$; d; e.
Output: mpe

1. **Initialize:**

2. **Process buckets:** for $p = n$ to 1 do
   for matrices $h_1, h_2, ..., h_j$ in $bucket_p$ do
   - **If** (observed variable) assign $X_p = x_p$ to each h_i and put in buckets.
   - **Else,** (multiply and maximize)
     $h_p = \max_{X_p} \Pi_{i=1}^{j} h_i$.
     $x_p^{opt} = argmax_{X_p} h_p$.
     Add h_p to its bucket.
3. **Forward:** Assign values in ordering d

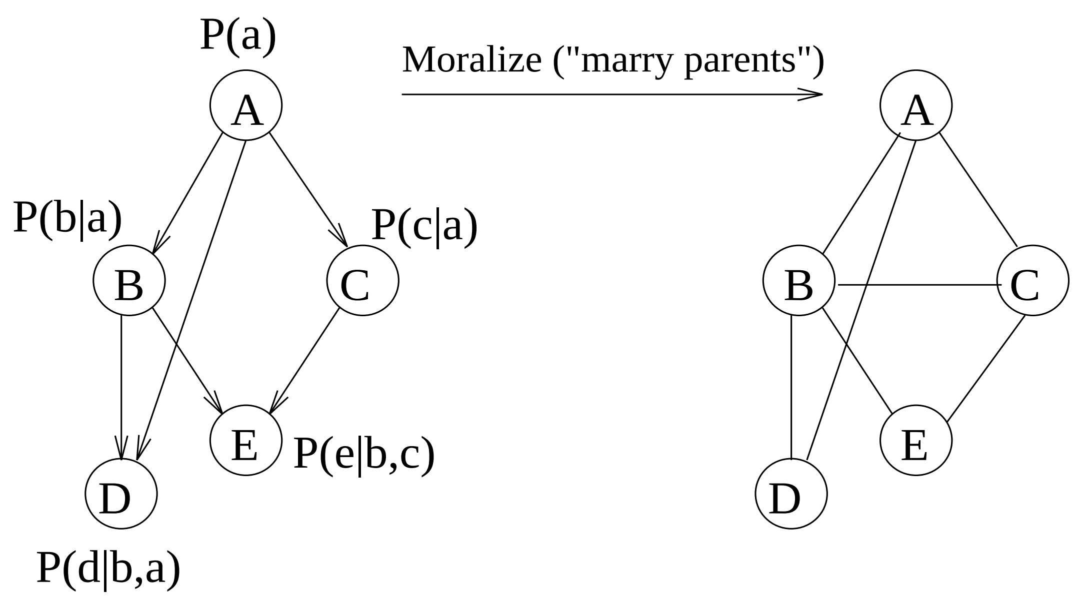
Theorem: Elim-mpe finds the value of the most probable tuple and a corresponding tuple.

Cost Networks and Dynamic Programming

Belief networks and cost networks

$P(a,b,c,d,e) = P(a)P(b|a)P(c|a)P(e|b,c)P(d|a,b)$

$C(a,b,c,d,e) = -logP = C(a)+C(b,a)+C(c,a)+ C(e,b,c)+C(d,a,b)$



- Minimize sum-of-costs.

Elim-opt, Dynamic Programming

(Bertele and Briochi, 1972)

Algorithm elim-opt
Input: A cost network (X, D, C), $C = \{C_1, ..., C_l\}$; ordering o; e.
Output: The minimal cost assignment.
1. **Initialize:** Partition the cost components into buckets.
2. **Process buckets** from $p \leftarrow n$ downto 1
For costs $h_1, h_2, ..., h_j$ in $bucket_p$, do:

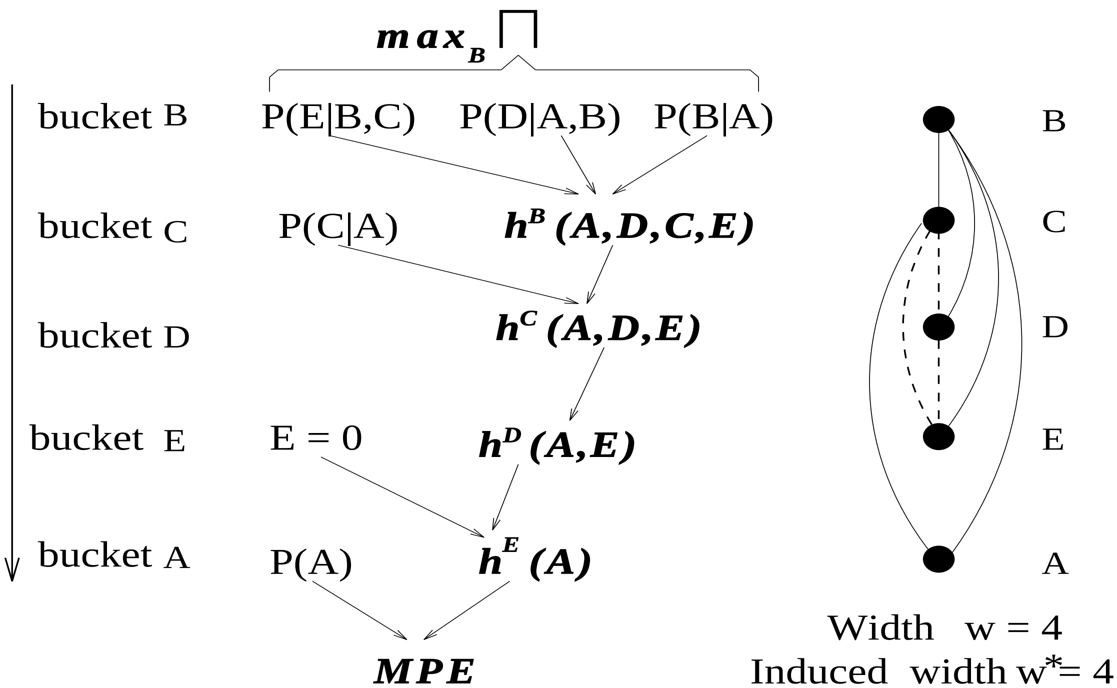
- **If** (observed variable) $X_p = x_p$, assign $X_p = x_p$ to each h_i and put in buckets.
- **Else,** (sum and minimize)
  $h^p = min_{X_p} \sum_{i=1}^{j} h_i$.
  $x_p^{opt} = argmin_{X_p} h^p$.
  Add h^p to its bucket.
  3. **Forward:** Assign minimizing values in ordering o

Algorithm Elim-Opt

(Dechter, Ijcai97)

$min_{a,d,c,b,e=0} C(a,b,c,d,e) = min_{a,d,c,b}$
$C(a,c) + C(a,b,d) + C(b,e) + C(b,c) + C(c,e)$

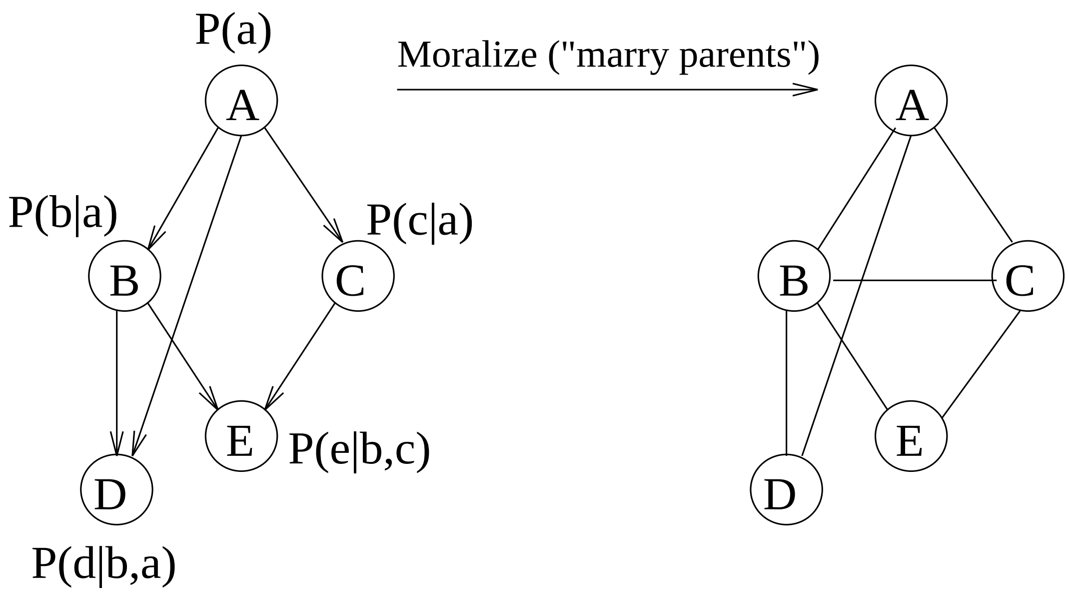
1. **Partition** $C = \{C_1, ..., C_r\}$ into buckets
2. **Process buckets from last to first:**



3. **Forward:** Assign values in ordering d

Finding the MAP

(An optimization task)



Variables A and B are the hypothesis variables.

Ordering: $a,\ b,\ c,\ d,\ e$

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

$= \max_a P(a) \max_b P(b|a) \sum_c P(c|a) \sum_d P(d|b,a)$
$\sum_{e=0} P(e|b,c)$

Ordering: $a,\ e,\ d,\ c,\ b$ .... illegal ordering

$\max_{a,b} P(a,e,e=0) = \max_{a,b} \sum_P (a,b,c,d,e)$

$$\max_{a,b} P(a,b,e=0) = \max_a P(a) \max_b P(b|a) \sum_d \cdot$$

$$\max_c P(c|a) P(d|a,b) P(e=0|b,c)$$

Elim-map

Maximum aposteriori hypothesis (MAP):
Given $A = \{A_1, ...A_k\} \subseteq X$, find $a^o = ({a^o}_1, ...{a^o}_k)$ s.t. $p(a^o) = \max_{\bar{a}_k} \sum_{x_{X-A}} \Pi_{i=1}^{n} P(x_i | x_{pa_i}, e)$.

Input: A belief network and hypothesis $A = \{A_1, ..., A_k\}$, d, e.
Output: An map.
1. **Initialize:**
2. **Process buckets :** for $p = n$ **to** 1 **do**
   for matrices $\beta_1, \beta_2, ..., \beta_j$ in $bucket_p$ do
   - **If** observed variable, assign $X_p = x_p$.
   - **Else**, (multiply and sum or max)
     $\beta_p = \sum_{X_p} \Pi_{i=1}^{j} \beta_i$,
     $(X_p \in A)$ $\beta_p = \max_{X_p} \Pi_{i=1}^{j} \beta_i$
     $a^0 = argmax_{X_p} \beta_p$.
     Add β_p to its bucket.
3. **Forward:** Assign values to A.

Variable ordering is restricted: max-buckets should preceede (processed after) summation buckets.

Complexity of bucket elimination

Theorem

Given a belief network having n variables, observations e, the complexity of elim-mpe, elim-bel, elim-map along d, is time and space

$$O(n \cdot exp(w * (d))$$

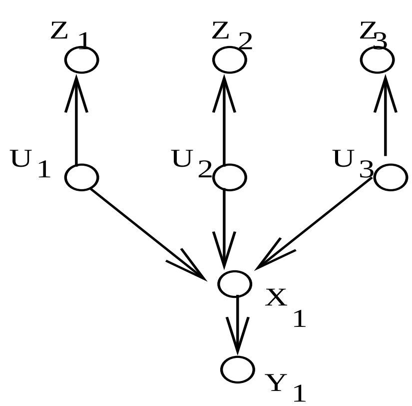
where $w * (d)$ is the induced width of the moral graph whose edges connecting evidence to earlier nodes, were deleted.

Bucket-Elimination for trees and Poly-Trees

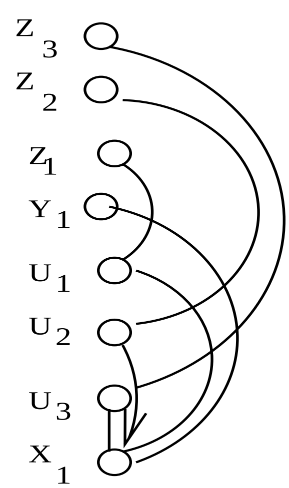
Elim-bel, elim-mpe, elim-map are linear for poly-trees.

They are similar to single root query of Pearl's propagation on poly-trees, if using topological ordering (and *super-bucket* processing of parents.)

Example:



(a)



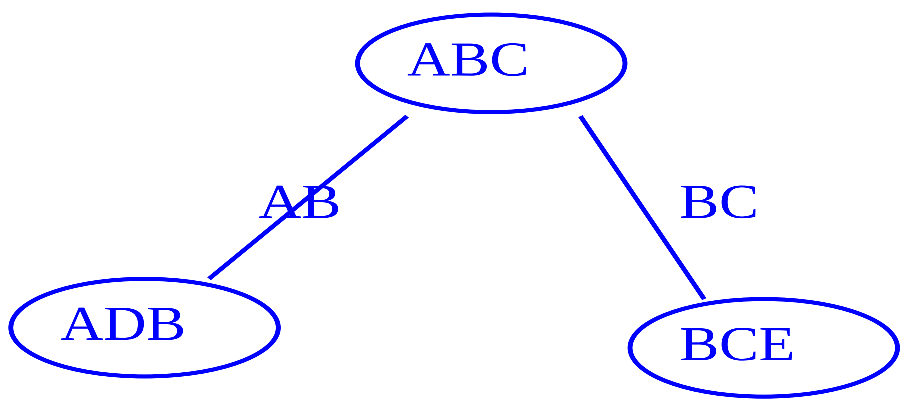
(b)

Relationship with join-tree clustering

(constraint networks and belief networks)

Ordering: a, b, c, d, e

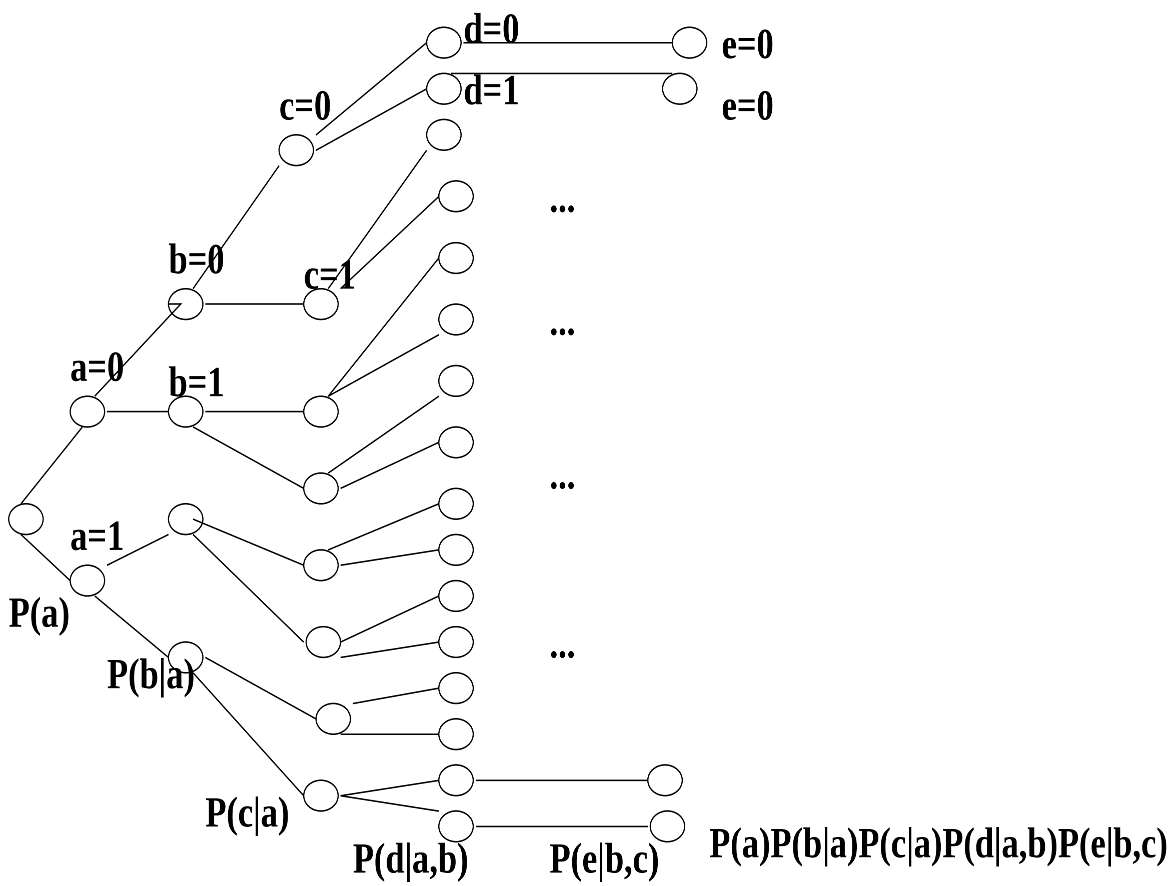
$$
\begin{aligned}
bucket(E) = &\quad P(e|b,c) \\
bucket(D) = &\quad P(d|a,b) \\
bucket(C) = &\quad P(c|a), \;\; || \;\; \lambda_E(a,b) \\
bucket(B) = &\quad P(b|a), \;\; || \;\; \lambda_C(a,b) \\
bucket(A) = &\quad P(a), \;\; || \;\; \lambda_B(a)
\end{aligned}
$$



A clique in tree-clustering can be viewed as a set of buckets.

Conditioning: Generates the Probability Tree

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



Complexity of conditioning:

Time: exponential

Space: linear.

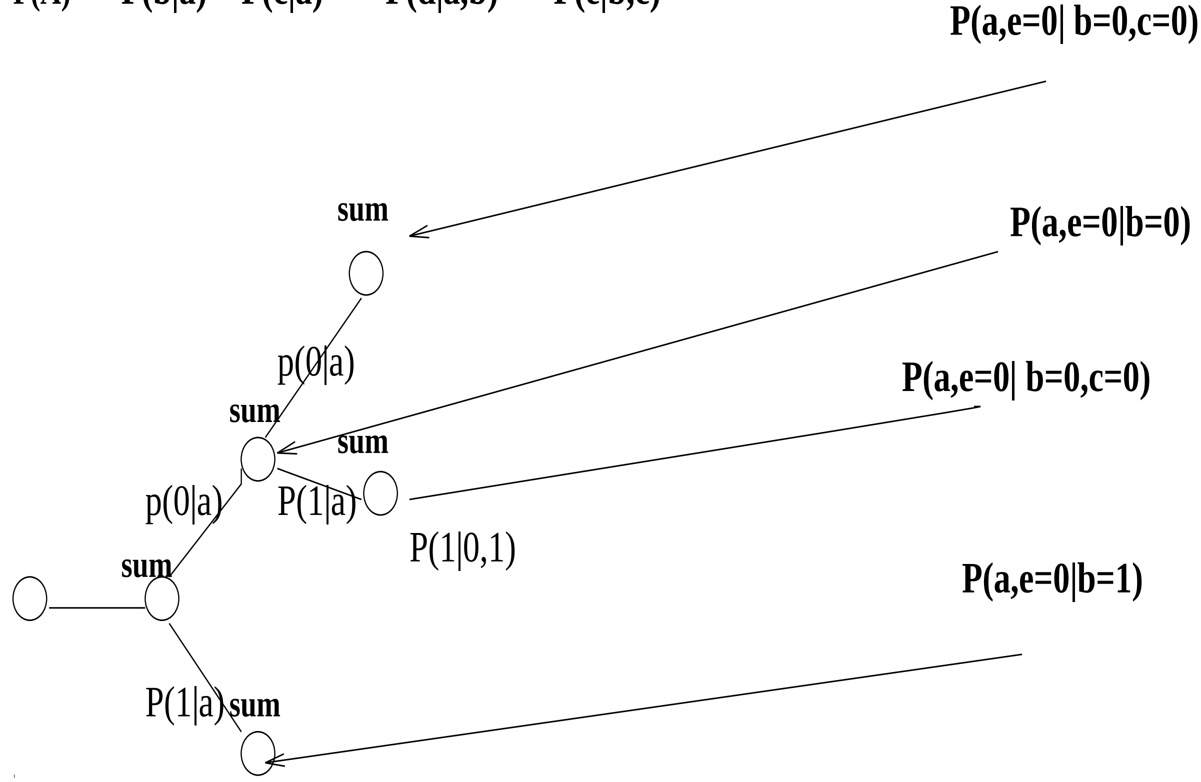
Conditioning+ Elimination

$P(a,e=0) =$

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

A B C D E

P(A) P(b|a) P(c|a) P(d|a,b) P(e|b,c)



Method: Search until a problem having a small w^* is created.

Conditioning + Elimination

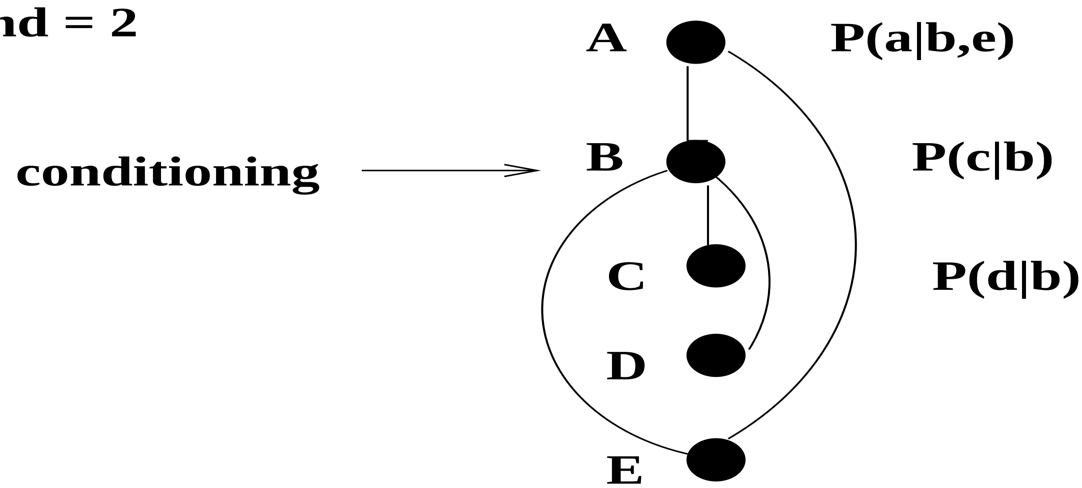
Trading space for time

- Algorithm *elim-cond(b)*, b bounds width:

When $b > width$, apply conditioning.

- $b = 0$ is full conditioning,
- $b = w^*$ is pure bucket elimination
- $b = 1$ is the cycle-cutset method.
- Time $exp(b + |cond(b)|)$, space $exp(b)$

bound = 2

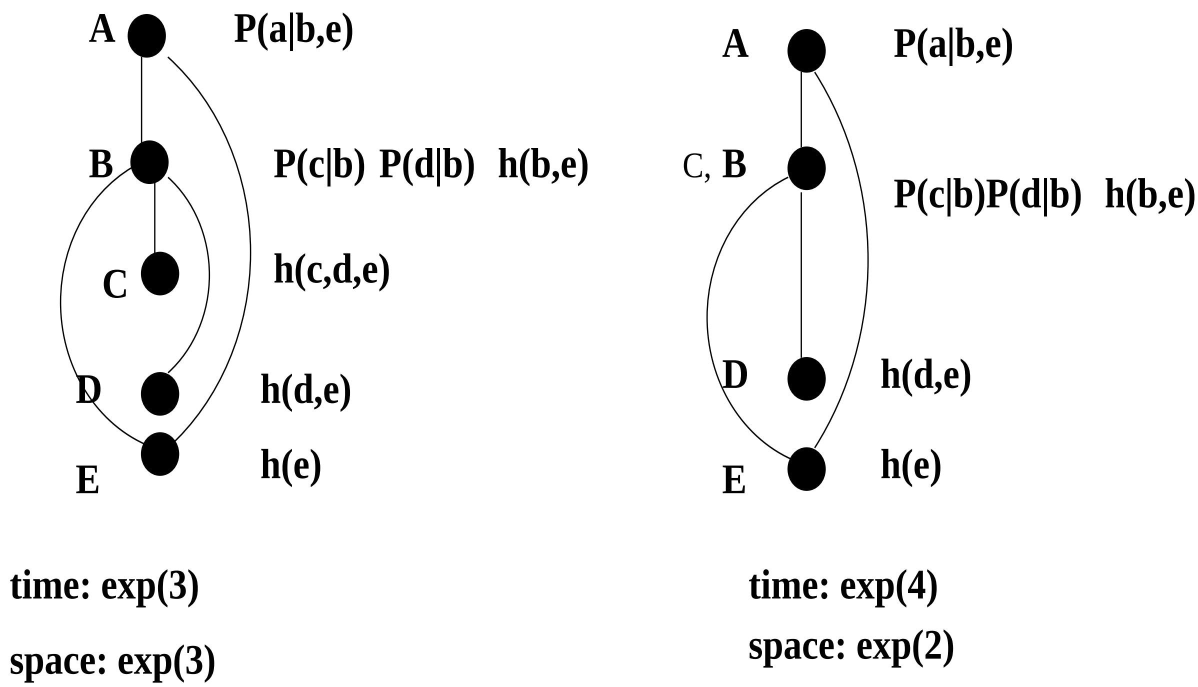


Super-Bucket Elimination

Trading space for time

(Dechter and El Fattah, UAI 1996)

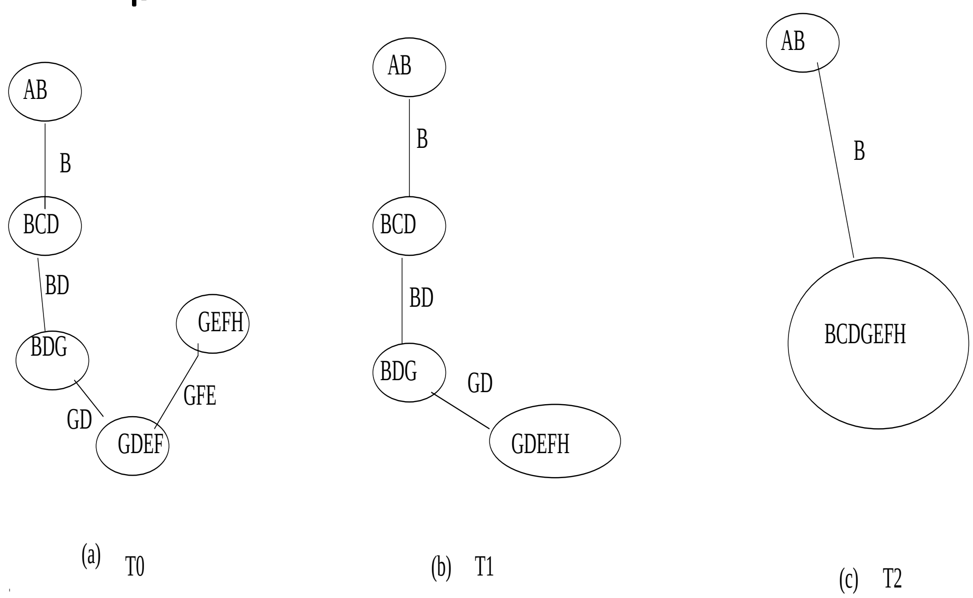
- Eliminating a few variables "at once".



- Here conditioning is local to super-buckets.

The Super-Bucket Idea

Larger super-buckets (cliques) means more time and less space:



Complexity:

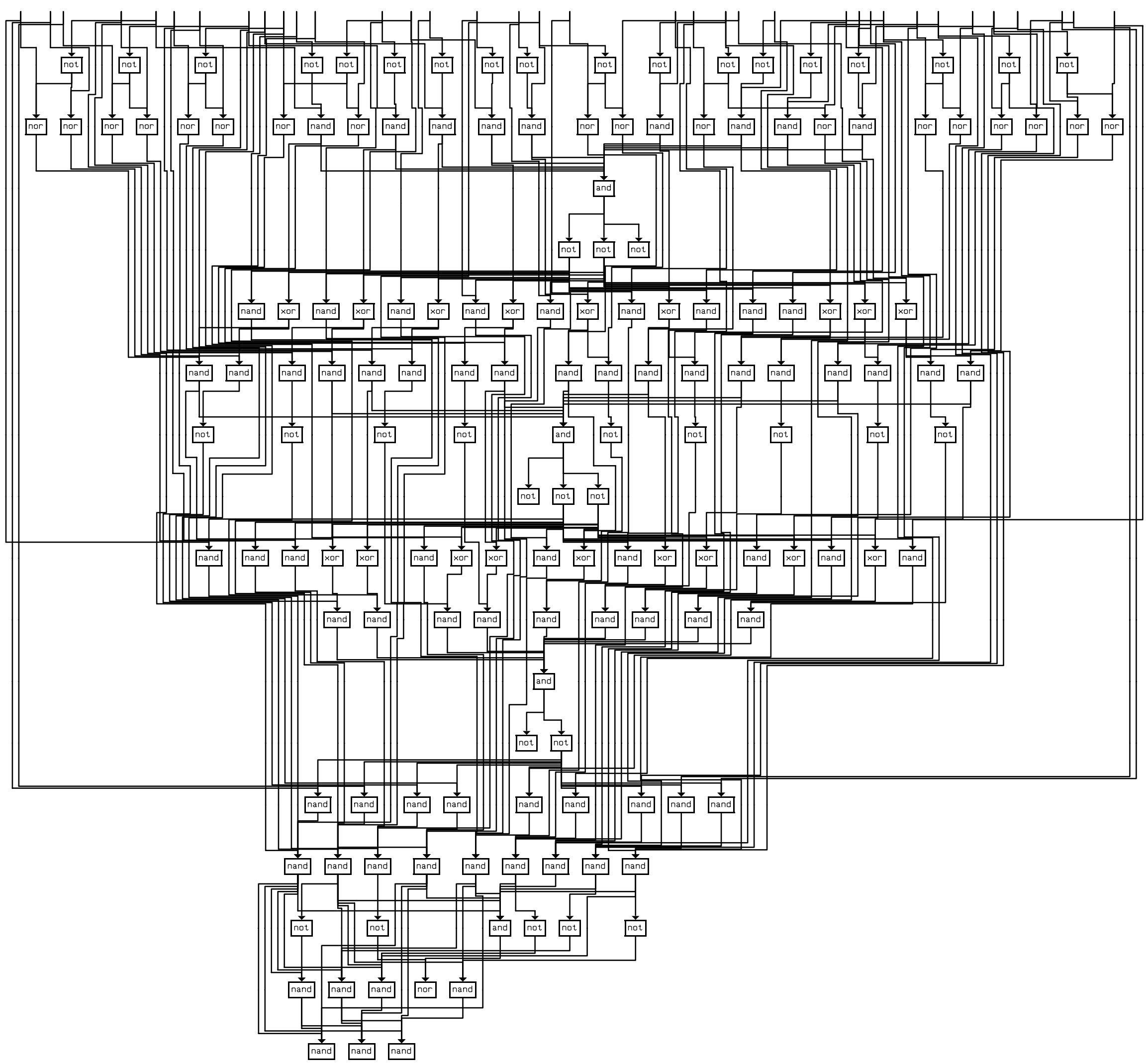
1. Time: exponential in clique and super-bucket size

2. Space: exponential in separator size.

Application: Circuit Diagnosis

Problem: Given a circuit and unexpected output, identify faulty components. The problem can be modeled as a constraint optimization problem and solved by bucket elimination.

Circuit C432

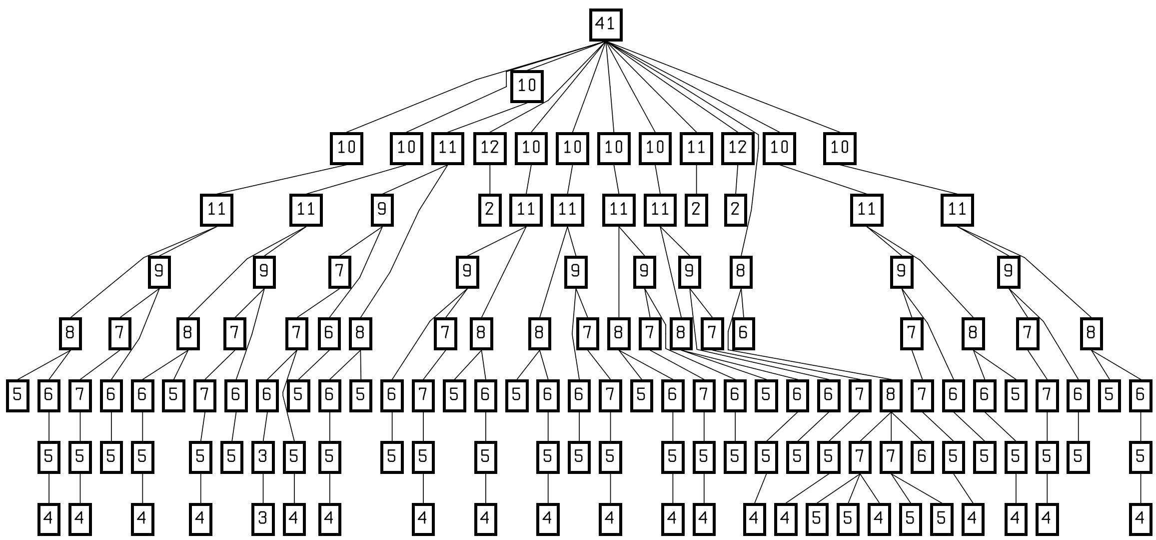


Benchmark Circuits

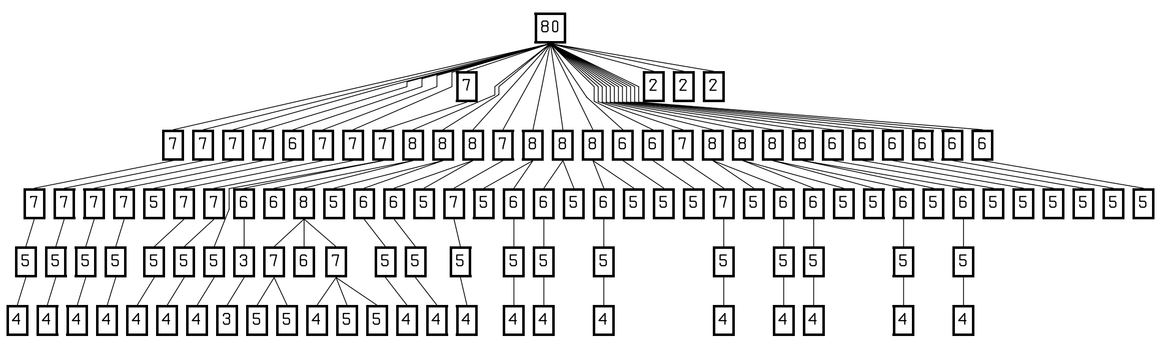
Circuit Name	Circuit Function	Total Gates	Input Lines	Output Lines
C17		6	5	2
C432	Priority Decoder	160 (18 EXOR)	36	7
C499	ECAT	202 (104 EXOR)	41	32
C880	ALU and Control	383	60	26
C1355	ECAT	546	41	32
C1908	ECAT	880	33	25
C2670	ALU and Control	1193	233	140
C3540	ALU and Control	1669	50	22
C5315	ALU and Selector	2307	178	123
C6288	16-bit Multiplier	2406	32	32
C7552	ALU and Control	3512	207	108

Secondary Trees for C432

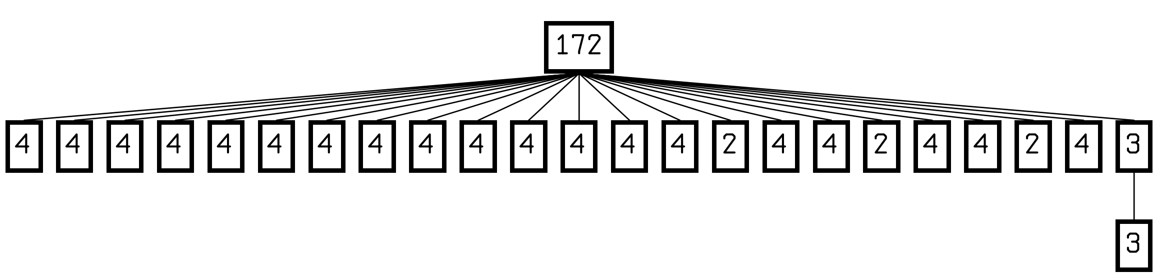
Separator size 11



Separator size 7



Separator size 3



Time-Space tradeoff for circuits

"Road Map": Tasks and Methods

Methods \ Tasks	CSP	SAT	Optimization	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency, join-tree	directional resolution	dynamic programming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset, forward checking	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

Approximation algorithms

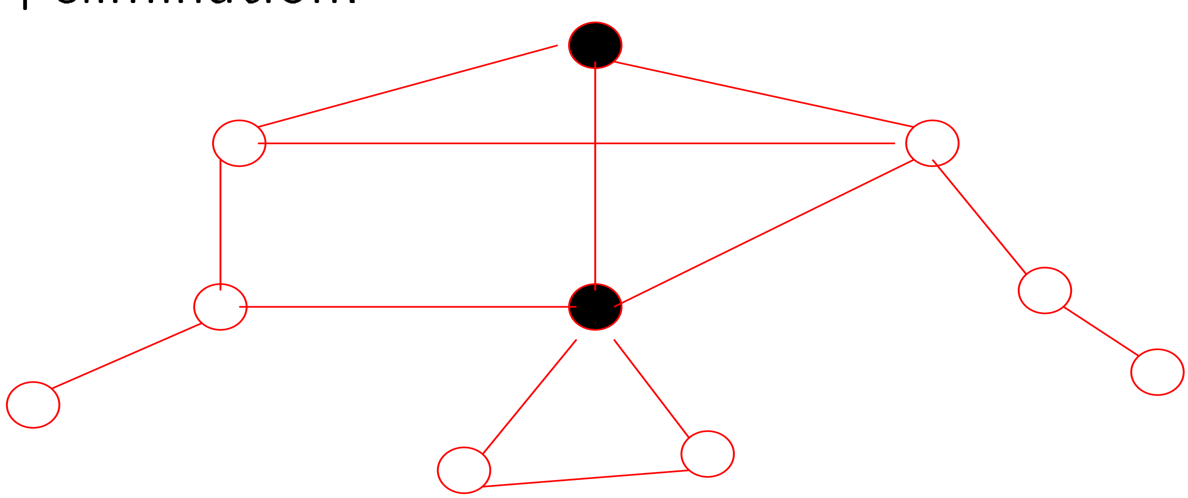
- **Approximating conditioning:**

Random search, GSAT, stochastic simulation.

- **Approximating elimination:**

Local consistency algorithms, bounded resolution, the mini-buckets approach.

- **Approximation of hybrids** of conditioning +elimination.



Approximating conditioning:

Randomized Hill-climbing search

(Hopfield 1982, kirkpatrick et. al, 1983)

(Minton et. al. 1990, Selman et. al, 1992)

For CSP and SAT:

GSAT: (one try)
1. Guess an assignment to all the variables.
2. Improve assignment by fliping a value using a guiding hill-climbing function: the number of conflicting constraints.
3. Use randomization to get out of local minimas.
4. After a fixed time stop and start a new try.

Randomized hill climbing frequently solve large and hard satisfiable problems.

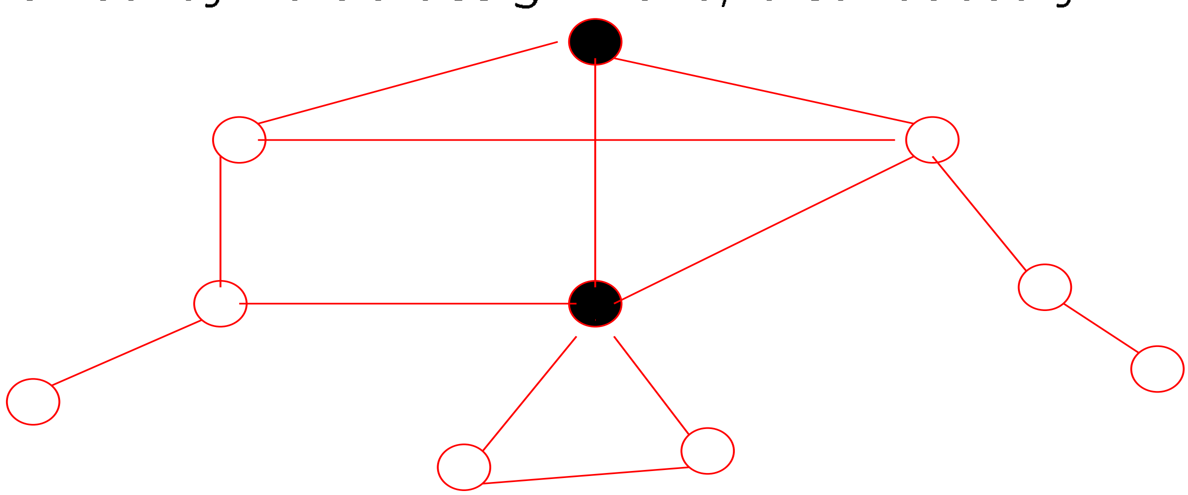
Distributed version: Energy minimization in a Hopfiled neural network (Hopfiled, 1982), Boltzman machines.

Approximating Conditioning with elimination

Energy minimization in Neural networks

(Pinkas and Dechter, JAIR 1995)

• Cutset nodes run the original greedy update function relative to neighbors. The rest of the nodes run the arc-consistency algorithm followed by value assignment, distributedly.



Approximating Conditioning in a Hybrid

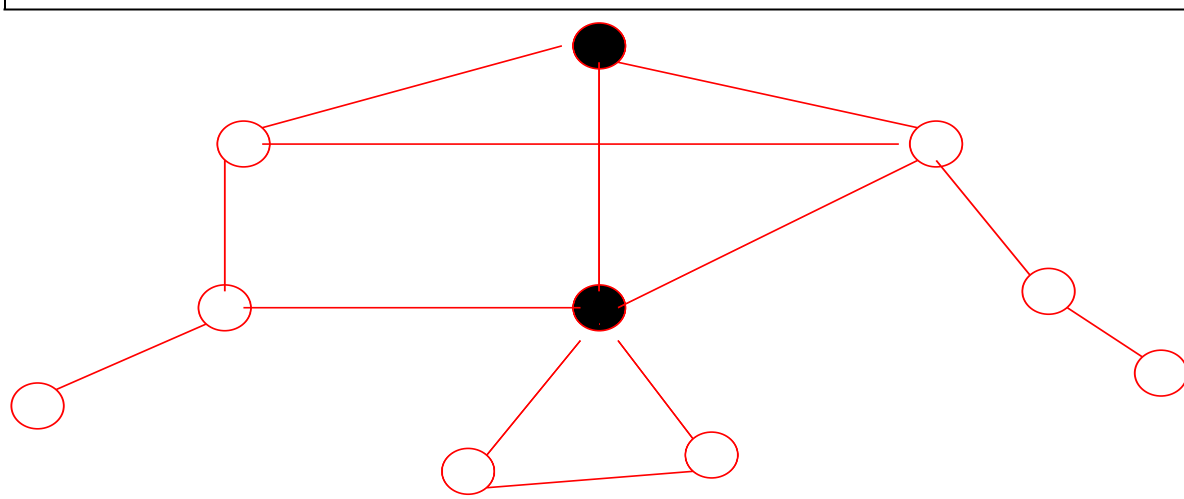
GSAT with Cycle-Cutset

(Kask and Dechter, AAAI 1996)

Algorithm (GSAT +cycle-cutset)
Input: A CSP, variables divided into cycle cutset and tree variables

Output: An assignment to all the variables.
One try:
Create a random initial assignment, and then alternatively executes these two steps:

1. Run Tree Algorithm on the problem, where the values of cycle cutset variables are fixed.
2. Run GSAT on the problem, where the values of tree variables are fixed.



GSAT with cycle-cutset

(Kask and Dechter, AAAI 1996)

Binary CSP, 100 instances per line, 100 variables, 8 values, tight					
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved
125	11 %	29 sec	46	10 sec	90
130	12 %	46 sec	29	16 sec	77
135	14 %	65 sec	13	23 sec	52

Binary CSP, 100 instances per line, 100 variables, 8 values, tight					
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved
160	20 %	52 sec	33	20 sec	90
165	21 %	60 sec	13	30 sec	80
170	22 %	70 sec	4	40 sec	54

Binary CSP, 100 instances per line, 100 variables, 8 values, tight					
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved
235	34 %	52 sec	69	14 sec	66
240	35 %	76 sec	57	22 sec	57
245	36 %	113 sec	40	43 sec	40

Binary CSP, 100 instances per line, 100 variables, 8 values, tight					
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved
290	41 %	55 sec	74	13 sec	30
294	42 %	85 sec	80	25 sec	23
300	43 %	162 sec	63	45 sec	19

GSAT with cycle-cutset

(Kask and Dechter, AAAI 1996)

"Road Map":

Tasks and Methods

Methods \ Tasks	CSP	SAT	Optimization	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency, join-tree	directional resolution	dynamic programming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset, forward checking	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

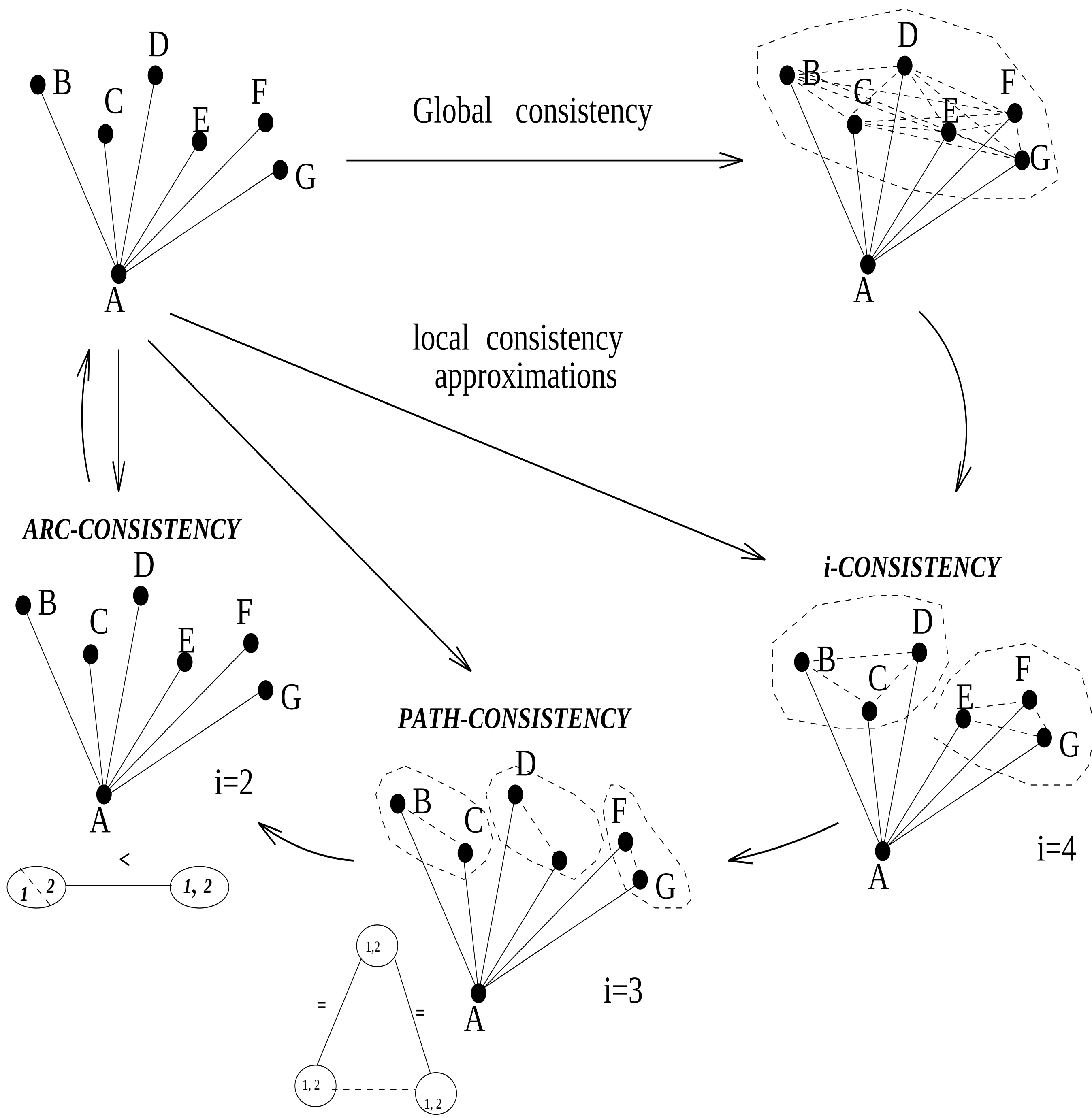
Approximating Elimination: Local Inference

- **Problem:** bucket elimination (inference) algorithms are intractable when w^* is large.

- **Approximation idea:**
  bound the arity of recorded dependencies (constraints/probabilities/utilities), i.e. perform **local inference**.

  CSPs: local consistency;
  SAT: bounded resolution;
  Belief networks, optimization:
  mini-buckets.

CSP: from Global to Local Consistency



i-consistency

- **i-consistency:**
  Any consistent assignment to any $i-1$ variables is consistent with at least one value of any i-th variable.

  Arc-consistency $\Leftrightarrow$ 2-consistency
  Path-consistency $\Leftrightarrow$ 3-consistency

- **strong i-consistency:**
  k-consistency for every $k \leq i$

- **directional i-consistency:**
  Given an ordering, X_k is i-consistent with any $i-1$ *previous* variables.

- **strong directional i-consistency:**
  Given an ordering, X_k is *strongly* i-consistent with any $i-1$ *previous* variables.

Enforcing Directional i-consistency

- Directional i-consistency bounds the size of recorded constraints by i.

- For $i > w^*$, directional i-consistency is equivalent to adaptive consistency (bucket elimination).

Consistency Algorithms

SAT: Bounded Directional Resolution (BDR(i))

- BDR(i) enforces directional i-consistency

- **Bucket Operation**: **bounded** resolution.

  Resolvents on more than i variables are not recorded:

  e.g., $(A \vee B \vee \neg C) \wedge (\neg A \vee D \vee E) \rightarrow (B \vee \neg C \vee D \vee E)$ is not recorded by BDR(3).

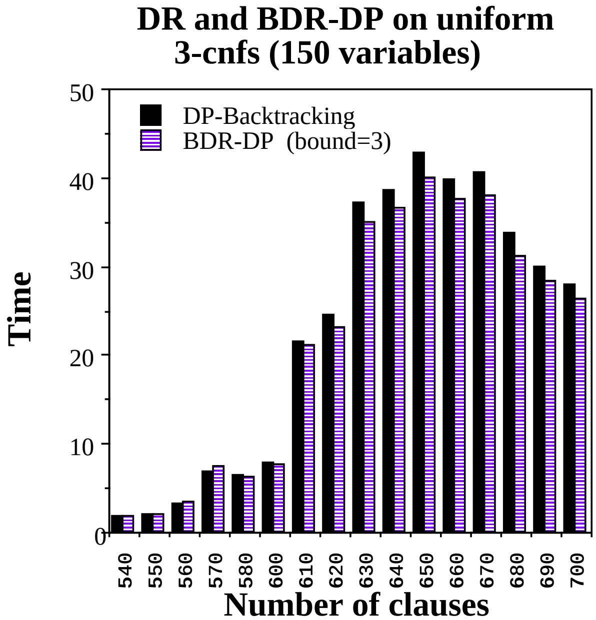
- Non-directional version: **k-closure** [van Gelder, 1996]. Enforces full k-consistency.

Preprocessing by i-consistency

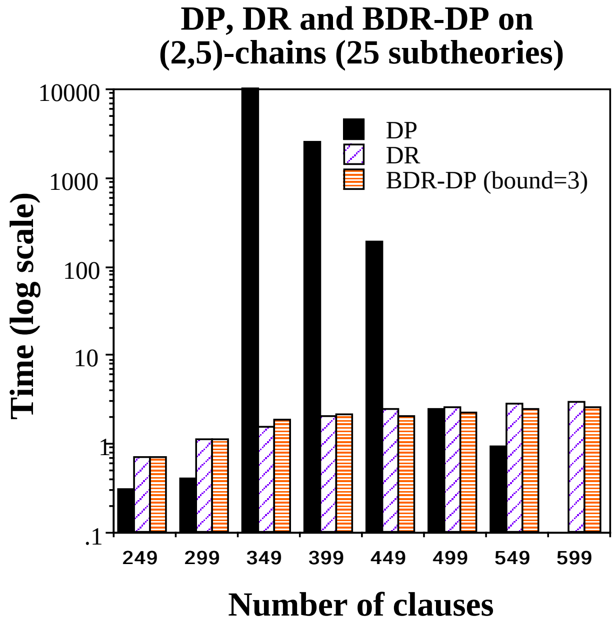
Complete algorithm **BDR-DP(i)** runs BDR(i) as a preprocessing before DP-backtracking.

Experimental Results:

Uniform random CNFs



(k,m)-tree CNFs



Probabilistic Inference: Mini-Bucket Approximation

Idea:

bound the size of probabilistic components by splitting a bucket into mini-buckets.

MPE example:

bucket (X) =

$$\underbrace{\{\, \mathbf{h_1, \ldots, h_r, h_{r+1}, \ldots, h_n} \,\}}$$

$$h^X = \max_X \prod_{i=1}^{n} h_i$$

$$\underbrace{\{\, \mathbf{h_1, \ldots, h_r} \,\}} \qquad \underbrace{\{\, \mathbf{h_{r+1}, \ldots, h_n} \,\}}$$

$$g^X = \left(\max_X \prod_{i=1}^{r} h_i\right) \cdot \left(\max_X \prod_{i=r+1}^{n} h_i\right)$$

$$\Downarrow$$

$$\mathbf{h^X \leq g^X}$$

- Complexity decrease:

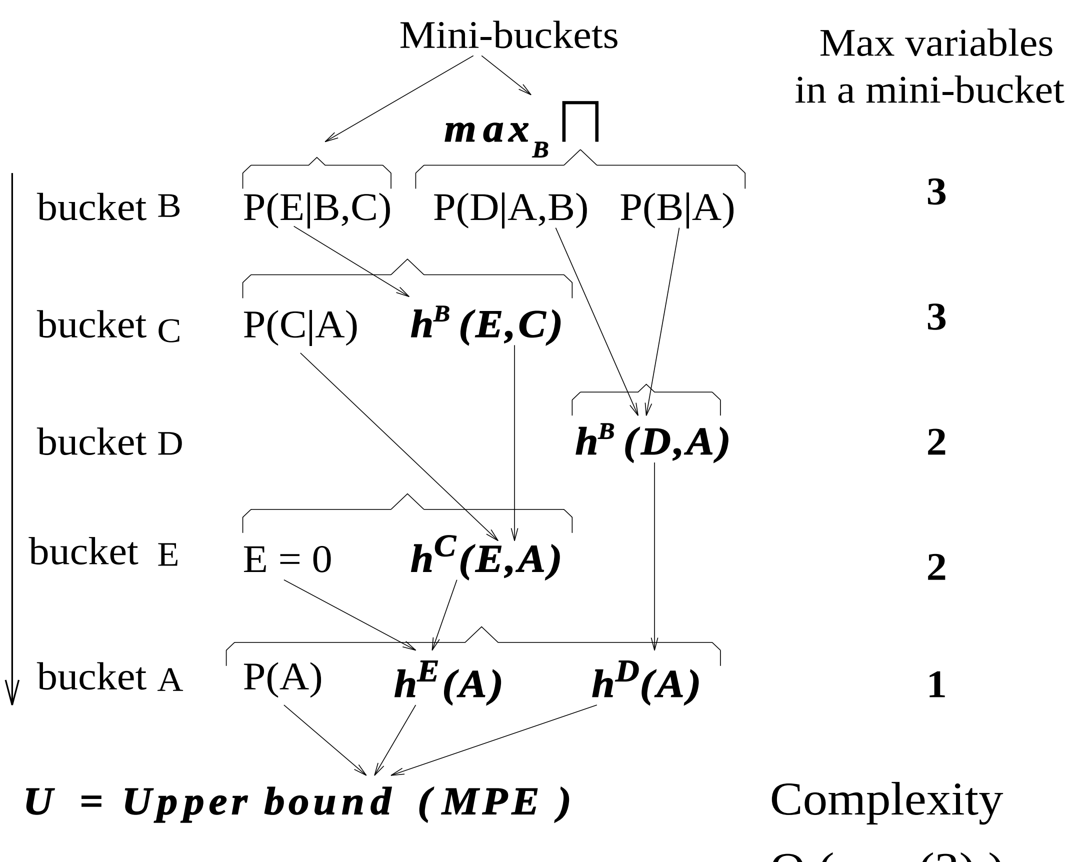
$$O(e^n) \rightarrow O(e^r) + O(e^{n-r})$$

Approx-mpe(i)

[Dechter and Rish, 1997]

i - max number of variables in a mini-bucket

Input: A Belief network $P = \{P_1, ..., P_n\}$
Output: upper and lower bounds on MPE
1. **Initialize:** Partition into buckets.
2. **Process buckets from last to first:**



3. **Forward:** Assign values in ordering d

Lower bound = P(solution).

About approx-mpe(i)

- **Complexity:**
  $O(exp(2i))$ time and $O(exp(i))$ space.

- **Accuracy:**
  determined by **Upper bound/Lower bound** ratio. As i increase, accuracy increases.

- **Applications:**
  - As an anytime algorithm.
  - As heuristics in Best-First Search.

- **Other probabilistic tasks:**
  mini-bucket idea can be used for approximate belief updating, finding MAP and MEU [Dechter and Rish,1997].

Anytime Approximations

anytime-mpe(ϵ)

1. **Initialize:** $i = 1$.
2. **While** computation resources are available
3. Increase i
4. $U \leftarrow$ upper bound of approx-mpe(i)
5. $L \leftarrow$ lower bound of approx-mpe(i)
6. Retain best solution so far
7. **If** $U/L \leq \epsilon$, return solution
8. **end-while**
9. **Return** current maximum mpe.

anytime-mpe(1) is an exact algorithm.
It can be **orders of magnitude faster than elim-mpe**.

Best-First Search

- Mini-bucket records upper-bound heuristics.

- The evaluation function over $\bar{x}_p = (x_1, ..., x_p)$:

$$f(\bar{x}_p) = g(\bar{x}_p) \cdot h(\bar{x}_p)$$

$$g(\bar{x}_p) = \Pi_{i=1}^{p-1} P(x_i | x_{pa_i})$$

$$h(\bar{x}_p) = \Pi_{h_j \in bucket_p} h_j$$

Best-First:

Expand a node with maximal evaluation function.

Properties:

- An exact algorithm.
- Better heuristics lead to more pruning.

Approximate Elimination for Belief Updating

- **elim-bel** is similar to **elim-mpe** where maximization is replaced by summation [UAI-96].

- **Approximation idea:**
sum of products $\leq$ product of sums, i.e.

$$\sum_{X_p} \Pi_{i=1}^{j} \lambda_i \leq \Pi_{i=1}^{j} \sum_{X_p} \lambda_i$$

Even better: bound by max

$$\sum_{X_p} \Pi_{i=1}^{j} \lambda_i \leq \sum_{X_p} \lambda_1 \cdot \Pi_{l=2}^{j} \max_{X_p} \lambda_l$$

We can use min or $mean$, instead of max, yielding lower bounds and a mean value.

- **approx-bel-max(i)**:

Generates an upper bound to joint belief.
Complexity: $O(exp(2i))$.

Empirical Evaluation

Test Problems:

- CPCS networks
- Uniform random networks
- Random noisy-OR networks
- Probabilistic decoding

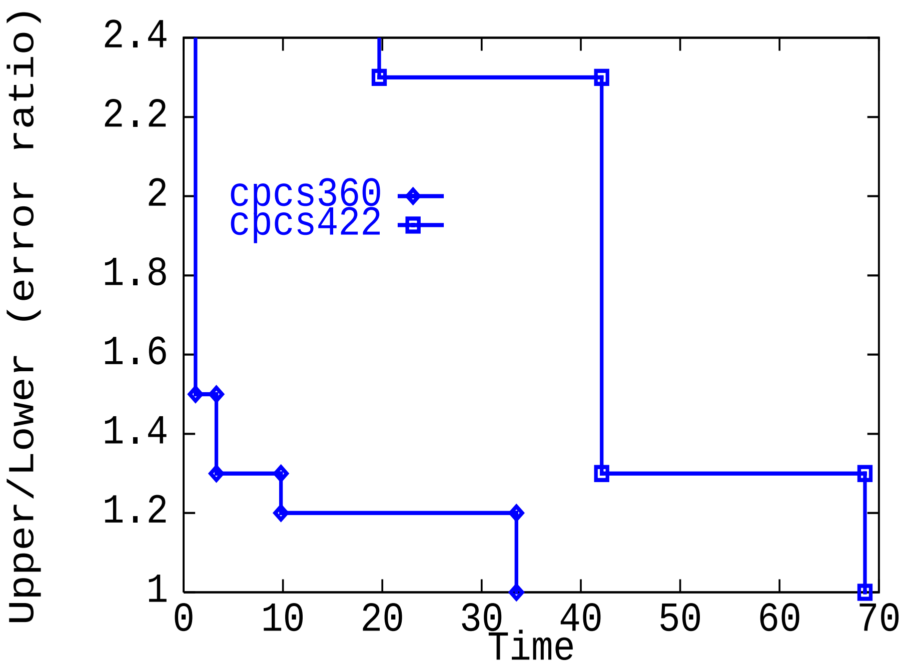
Algorithms:

- *elim-mpe*
- *approx-mpe(i)*
- *anytime-mpe(ϵ)*

CPCS Networks

cpcs360 - 360 binary nodes, 729 edges
cpcs422 - 422 binary nodes, 867 edges
Evidence (E) $=$ 0, 2, and 10 nodes

anytime-mpe(1) performance:



anytime-mpe(1) versus elim-mpe

Algorithm	Time (sec)			
	cpcs360		**cpcs422**	
	E = 0	E = 10	E = 0	E = 2
anytime-mpe(1)	33.5	108	68.6	234.8
elim-mpe	443.8	263.6	$>$ 405.6	$>$ 416.3

- **anytime-mpe(1)** is 100% accurate
- 2-3 orders of magnitude more efficient than **elim-mpe**
- exact **elim-mpe** ran out of memory on cpcs422; **anytime-mpe(1)** found exact solution in $<$ 70 sec.

Noisy-OR Networks

Random noisy-OR generator:

Random graph: n nodes, e edges.
Noisy-OR $P(x|pa(x))$ is defined by **noise** q:
link probability $P(x = 1|pa_i(x) = 1) = 1 - q$,
leak probability $P(x = 1|\forall i pa_i(x) = 0) = 0$.

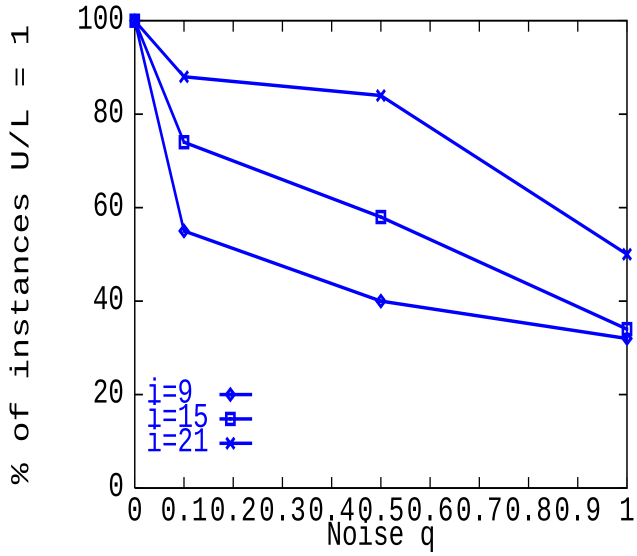
Results on (50 nodes, 150 edges)-networks

10 evidence nodes, 200 instances

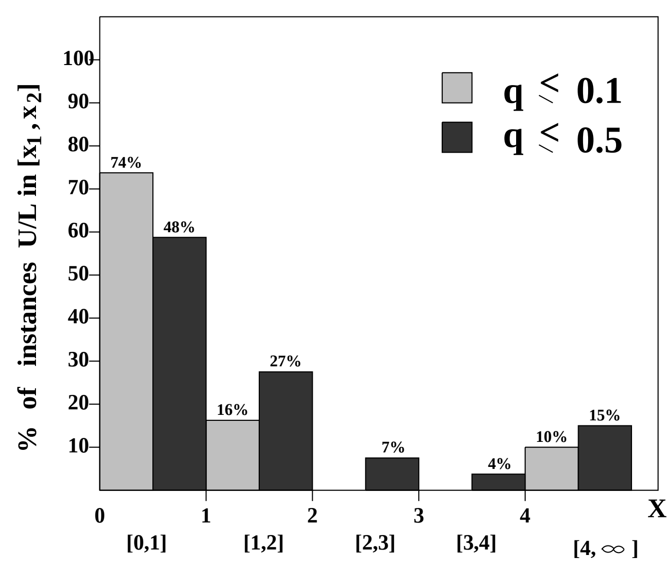
- elim-mpe ran out of memory;
approx-mpe(i) time: from 0.1 sec for $i = 9$ to 80 sec for $i = 21$.

- Accuracy increases with $q \to 0$, 100 % for $q = 0$ (Figure (a)).

- U/L is extreme: either really good (=1) or really bad (> 4);
U/L becomes less extreme with increasing noise q (Figure (b)).



(a)



(b)

Random Networks

Random graphs (n nodes, e edges) and uniform random $P(x|pa(x))$.

approx-mpe(12)

60 nodes, 90 edges, 200 instances					
$[\epsilon - 1, \epsilon]$	i	Lower bound		Upper bound	
		M/L %	Mean Te/Ta	U/M %	Mean Te/Ta
$[1, 2]$	12	85.5	24.4	81	23.5
$[2, 3]$	12	11.5	29.7	13.5	29.1
$[3, 4]$	12	0.5	11.4	5	37.3
$[4, \infty]$	12	2.5	21.1	0.5	14.0

• In $\approx$ 80% of cases, approx-mpe is more efficient by 1-2 orders of magnitude while achieving accuracy factor of at least 2.

30 nodes, 80 edges, 200 instances					
$[\epsilon - 1, \epsilon]$	i	Lower bound		Upper bound	
		M/L %	Mean Te/Ta	U/M %	Mean Te/Ta
$[1, 2]$	12	51	41.3	29	27.0
$[2, 3]$	12	15	41.3	32	50.5
$[3, 4]$	12	11	69.2	17	45.4
$[4, \infty]$	12	23	44.5	22	60.6

• approx-mpe effectiveness decreases with increasing density.

• Lower bound is usually closer to MPE than the Upper bound

Notation:
M/L% = % of instances s.t. MPE value / Lower Bound $\in [\epsilon - 1, \epsilon]$
U/M% = % of instances s.t. Upper Bound / MPE value $\in [\epsilon - 1, \epsilon]$
Mean Te/Ta = Mean value of elim-mpe time/approx-mpe time (Te/Ta) on the instances s.t. M/L (or U/M) $\in [\epsilon - 1, \epsilon]$

Probabilistic Inference: Iterative Belief Propagation (IBP)

Pearl's belief propagation (BP) algorithm records only unary dependencies. BP is exact for poly-trees.

Approximation scheme:

Iterative application of BP to a cyclic network.

Recent empirical results:

IBP is surprisingly successfull for probabilistic decoding (state-of-the art decoder).

Probabilistic Decoding

Goal:

Reliable communication over a noisy channel

Technique: Error-correcting codes

$U = (u_1, ..., u_k)$ - input *information* bits
$X = (x_1, ..., x_n)$ - additional **code** bits
Codeword (U, X) (**channel input**) is transmitted trough a **noisy channel**.
Result: real-valued **channel output** Y.

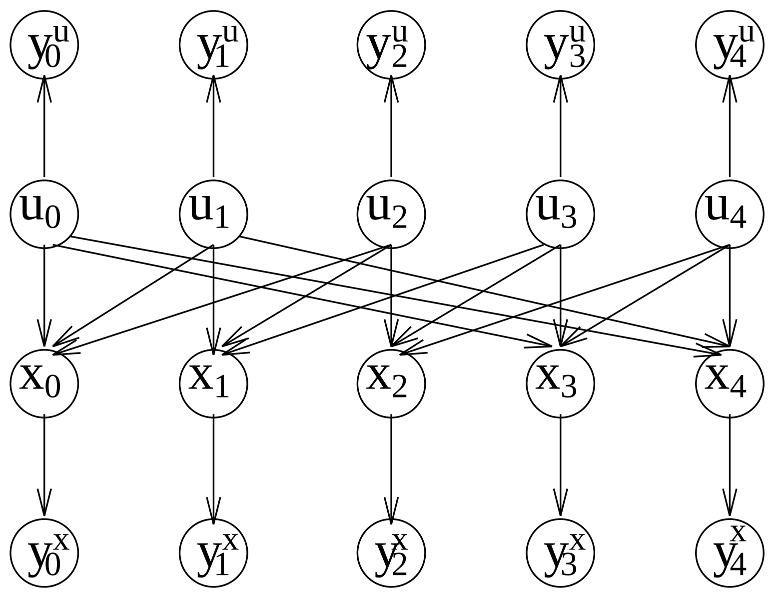
Decoding task: given Y, find U' s.t.:

1. (block-wise decoding)
$u' = \arg\max_u P(u|y)$, or

2. (bit-wise decoding)
$u_k^* = \arg\max_{u_k} P(u_k|y), 1 \leq k \leq K$.

Bayesian Network Representation

Linear block code:



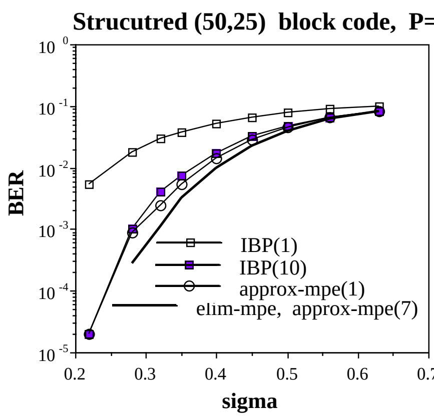
Problem parameters:
k - the number of the input information bits;
n - the number of code bits;
p - the number of parents of each code bit;
σ - the noisy channel parameter (*Gaussian noise*).

Encoding: parity check (pairwise XOR)
$x = u_1 \oplus u_2 \oplus ... \oplus u_m$, where u_i are parents of x, and $\oplus$ is summation modulo 2 (XOR).

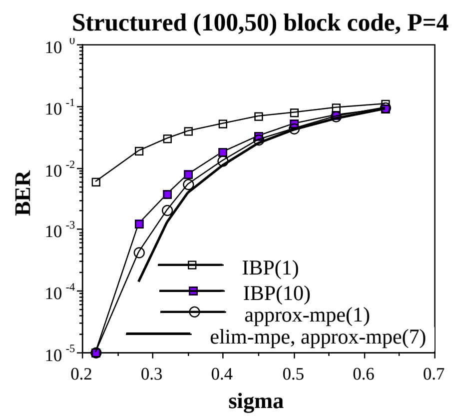
Structured Low-w^* Codes

Error measure: the **bit error rate** (BER).

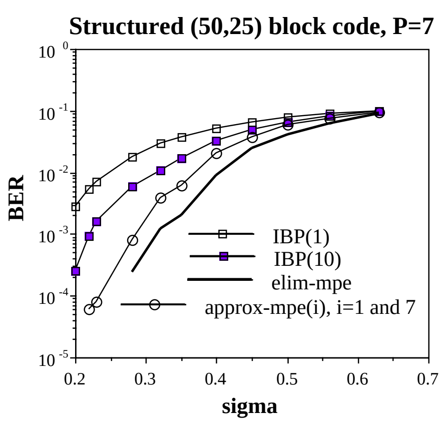
Approx-mpe(i) outperforms iterative belief propagation (IBP(I), I is the number of iterations) on **structured problems** with **small parent set size**:



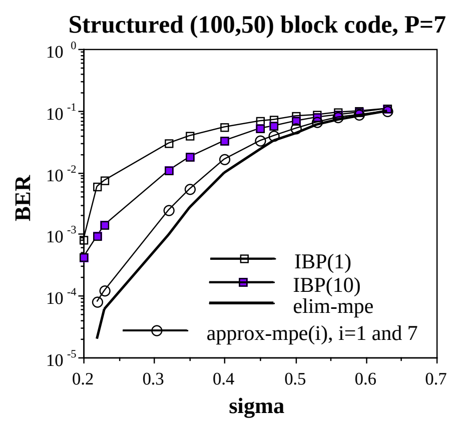
(a)



(b)



(c)

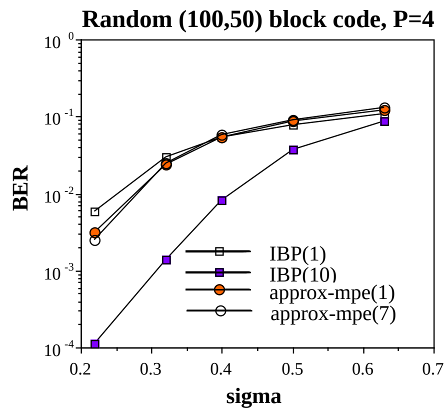


(d)

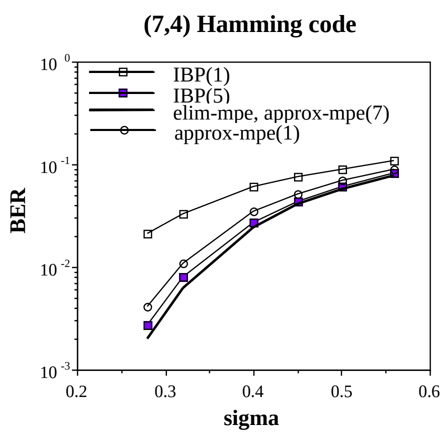
BER for exact elim-mpe and approximate IBP(1), IBP(10), approx-mpe(1) and approx-mpe(7) (1000 instances per point). Structured block codes with R=1/2 and (a) K=25, P=4, (b) K=50, P=4, (c) K=25, P=7, and (d) K=25, P=7. The induced width of the networks was 6 for (a) and (b), and 12 for (c) and (d).

Random (high-w^*) Codes and Hamming Codes

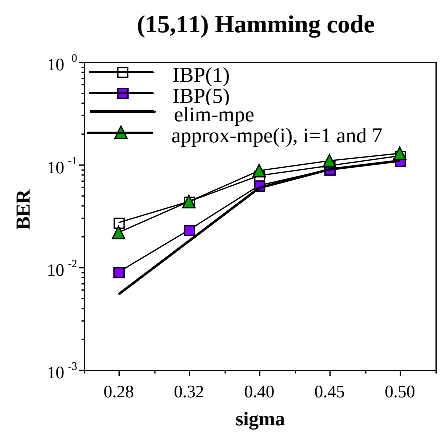
On the other hand, IBP outperforms approx-mpe(i) on **random problems** (high w^*) and on Hamming codes:



(a)



(b)



(c)

BER for exact elim-mpe and approximate IBP(1), IBP(5), approx-mpe(1) and approx-mpe(7) (10000 instances per point). Random block codes with R=1/2 and (a) K=50, P=4, and Hamming codes with (b) K=4, N=7 and (c) K=11, N=15. w^* of Hamming networks was (a) 3 and (b) 9, respectively, while w^* of the random networks was ≥ 30.

Summary

- **CPCS networks**:
approx-mpe(i) finds MPE for low $i \Rightarrow$
anytime-mpe(1) outperforms elim-mpe (often by 1-2 orders of magnitude)

- **Noisy-OR networks**:
approx-mpe(i) is more accurate than on random problems, especially for $q \rightarrow 0$

- **Random networks**:
approx-mpe(i) is not very effective, especially with increasing network density

- **Coding networks**:
approx-mpe(i) outperforms *iterative belief propagation* on low-w^* structured networks, but the opposite results are observed on high-w^* random coding networks.

"Road Map":

Tasks and Methods

Tasks / Methods	CSP	SAT	Optimization	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency, join-tree	directional resolution	dynamic programming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset, forward checking	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

Decision-Theoretic Planning

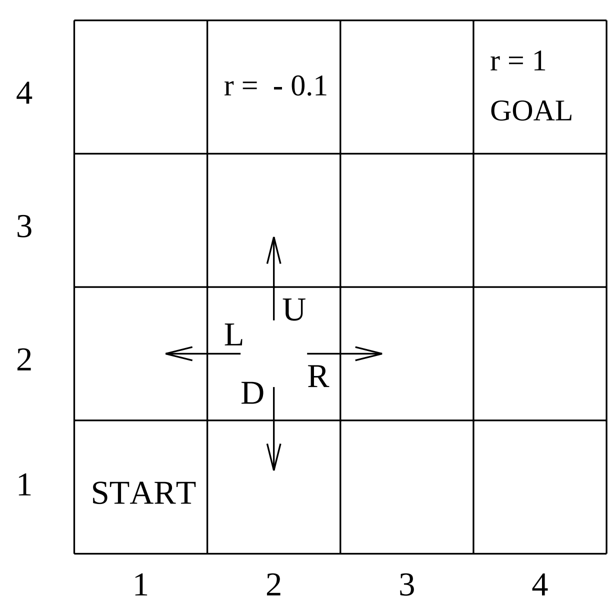
Example: Robot Navigation

State = { *Location, Cluttered, Direction, Battery*}

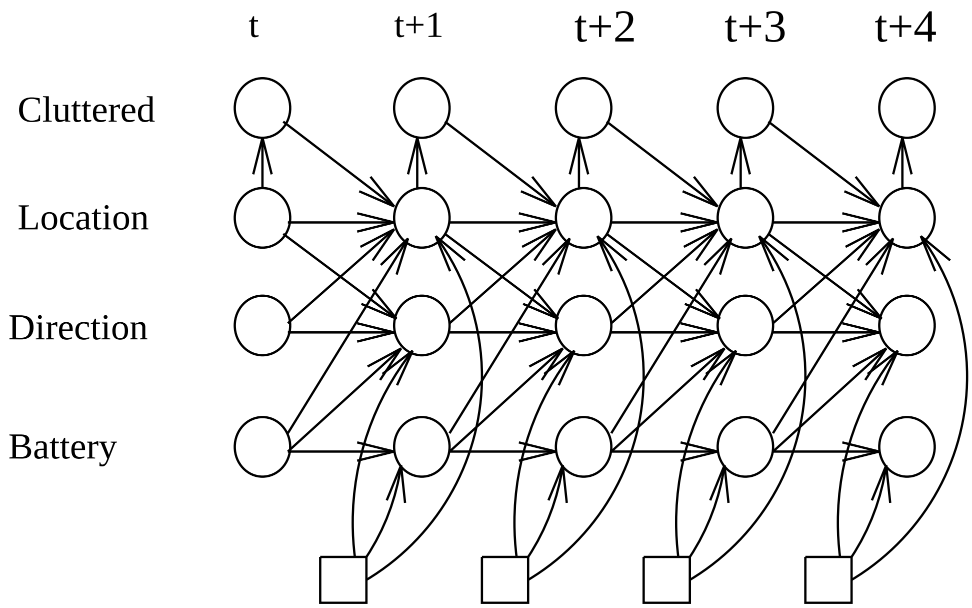
Actions = $\{North, South, West, East\}$

Probability of Success = P

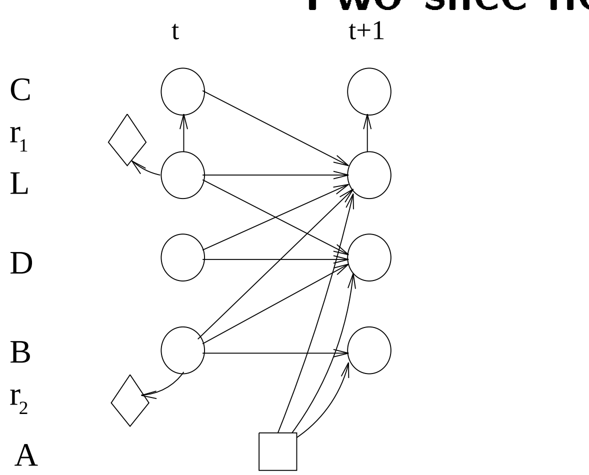
Task: reach the goal ASAP



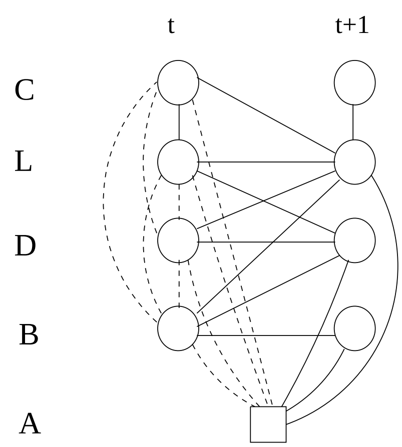
Dynamic Belief Networks



Two-slice networks



Two-stage influence diagram



interaction graph

Markov Decision Process

- $x = \{x_1, ..., x_n\}$ - state, D - domain, $\Omega_x = D^n$ - state space

- $a = \{a_1, ..., a_m\}$ - action, D_a - domain, $\Omega_a = D_a^n$ - action space

- P_{xy}^a - transition probabilities

- $r(x, a)$ - reward of taking action a in state x

- N - number of time slices

Problem: Find optimal *policy*

1. **Finite**-horizon MDP ($N < \infty$)

$$\pi = (d^1, ..., d^N), d^t : \Omega_x \to \Omega_a$$

2. **Infinite**-horizon MDP ($N = \infty$)

$$\pi : \Omega_x \to \Omega_a$$

Criterion:
maximum expected total (discounted) reward

$$\max_{\pi} V_\pi(x) = r(x, \pi(x)) + \lambda \sum_{y \in \Omega_X} P(y|x, \pi(x)) V_\pi(y).$$

Dynamic Programming: Elimination

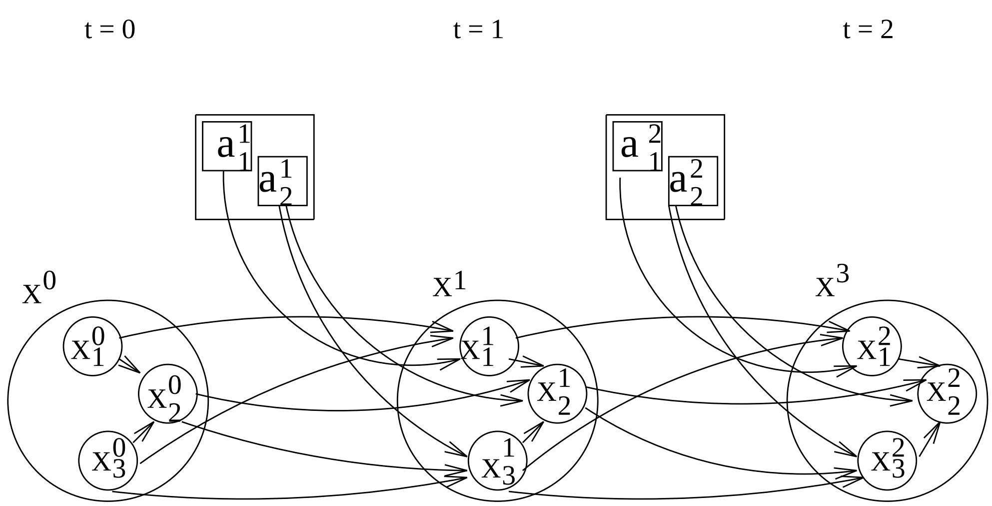
Optimality Equation:

$$V(x^t) = \max_{a^t}[r(x^t, a^t) + \sum_{x^{t+1}} P(x^{t+1}|x^t, a^t)]V^{t+1},$$

$$V^N = r^N(x^N).$$

Complexity:

$$O(N|\Omega_a||\Omega_X|^2) = O(N|D_a|^m|D|^{2n}).$$

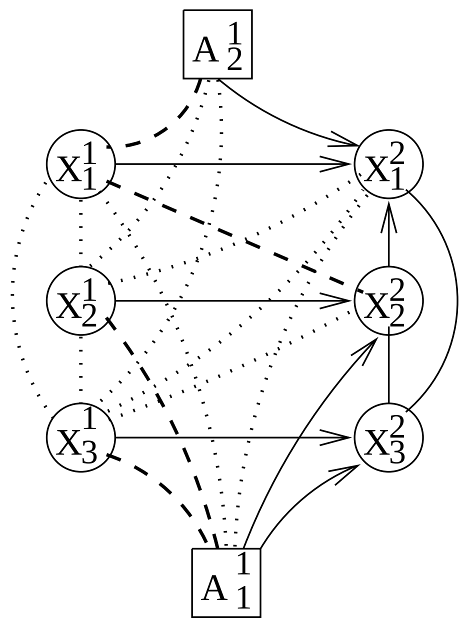


Decomposability :

$$r(x^t, a^t) = \sum_{i=1}^{n} r_i(x_i^t, a_i^t)$$
$$P(x^t|x^{t-1}, a^{t-1}) = \prod_{i=1}^{n} P(x_i^t|pa(x_i^t))$$

Bucket Elimination

$o = (X^1_1 \;\; X^1_2 \;\; X^1_3 \;\; A^1_1 \; A^1_2 \; X^2_1 \;\; X^2_2 \;\; X^2_3)$



X^2_3 $\quad P(X^2_3 \mid X^1_3, A^1_1) \qquad r_1(X^2_3, X^2_2) \qquad r_2(X^2_1, X^2_3)$

X^2_2 $\quad P(X^2_2 \mid X^1_2, A^1_1) \quad P(X^2_1 \mid X^2_2, X^1_1 \; A^1_2) \quad f(X^2_2, X^1_3, A$

X^2_1 $\quad \ldots \qquad g(X^2_1, X^1_2, A^1_1, X^1_1, A$

A^2_2 $\quad \ldots$

Complexity: $O(exp(w^*))$

Elim-meu

Input: A belief network $\{P_1, ..., P_n\}$; decision variables $D_1, ..., D_k$.
Output: $d_1, ..., d_k$, maximizing expected utility.

1. **Initialize:** Partition probability and utility matrices $\lambda_1, ..., \lambda_j$, $\theta_1, ..., \theta_l$.
2. **Backward:** For $p = n$ to 1 do
   for $\lambda_1, ..., \lambda_j, \theta_1, ..., \theta_l$ in $bucket_p$ do
   - **If** (observed variable), assign $X_p = x_p$.
   - **Else,**
     $\lambda_p = \sum_{X_p} \Pi_i \lambda_i$
     $\theta_p = \frac{1}{\lambda_p} \sum_{X_p} \Pi_{i=1}^{j} \lambda_i \sum_{j=1}^{l} \theta_j$,
     Add θ_p and λ_p to their buckets.
3. **Forward:** Assign values in ordering o using information in buckets.

Elimination and Conditioning

1. Finite-horizon MDPs:
   Dynamic Programming = elimination along temporal ordering (N slices).

2. Infinite-horizon MDPs:
   Value Iteration = elimination along temporal ordering (iterative)
   Policy Iteration = conditioning on A_i, elimnation on X_j (iterative).

3. Bucket elimination: "non-temporal" orderings.

Complexity $O(exp(w^*)), n \leq w^* \leq 2n$

$$\Downarrow$$

Further research: conditioning; approximations.