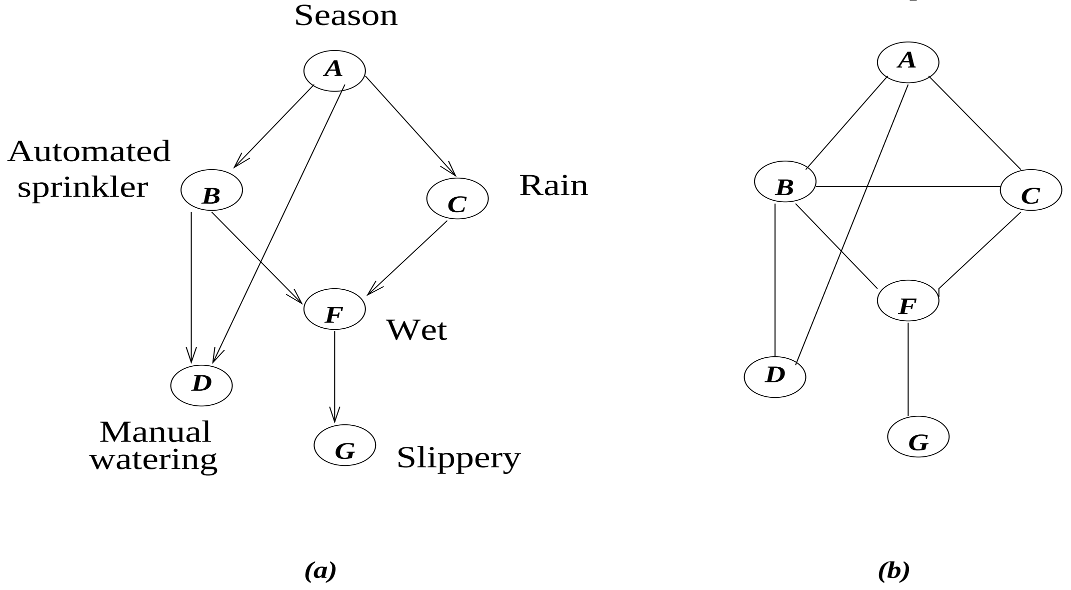
“Road Map”:

Tasks and Methods

Tasks / Methods	CSP	SAT	Optimi-zation	Belief updating	MPE, MAP, MEU	Solvir linea equali inequa
elimination	adaptive consistency join-tree	directional resolution	dynamic program-ming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gauss Fourie elimin
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 $bel(x_i) = P(X_i = x_i|e)$.

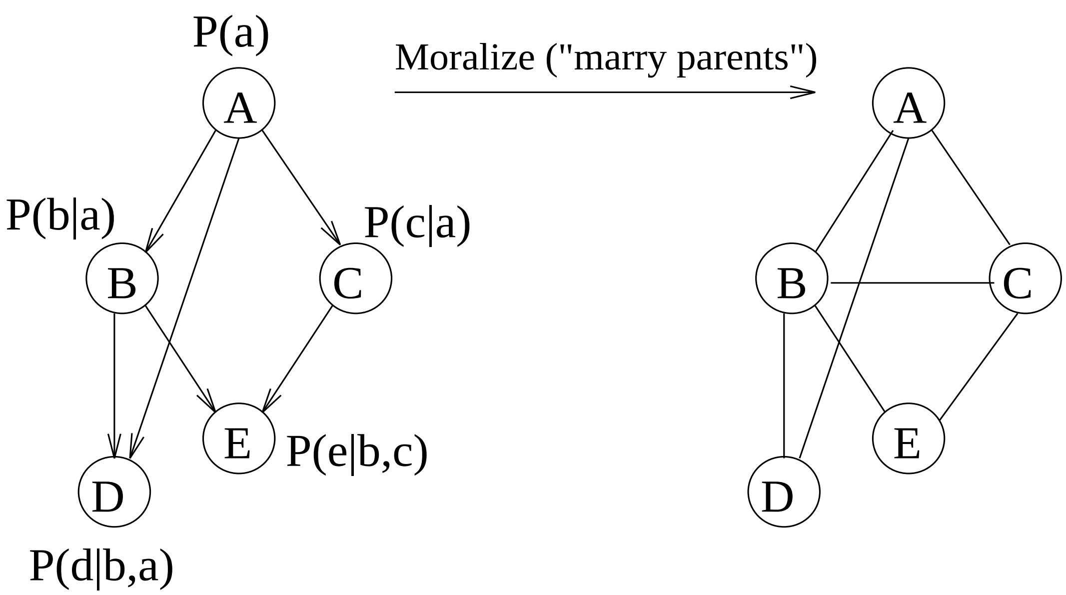
2. **Most probable explanation** (MPE):
   Find x^o s.t. $p(x^o) = \max_{\bar{x}_n} \Pi_{i=1}^n P(x_i|x_{pa_i}, e)$.

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

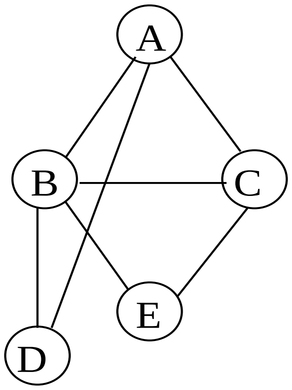
$P(a,e=0) = P(a)\sum_{e=0}\sum_d\sum_c P(c|a)\sum_b P(b|a)$
$P(d|a,b)P(e|b,c)$

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

The bucket elimination Process:

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

Bucket Elimination and Induced Width



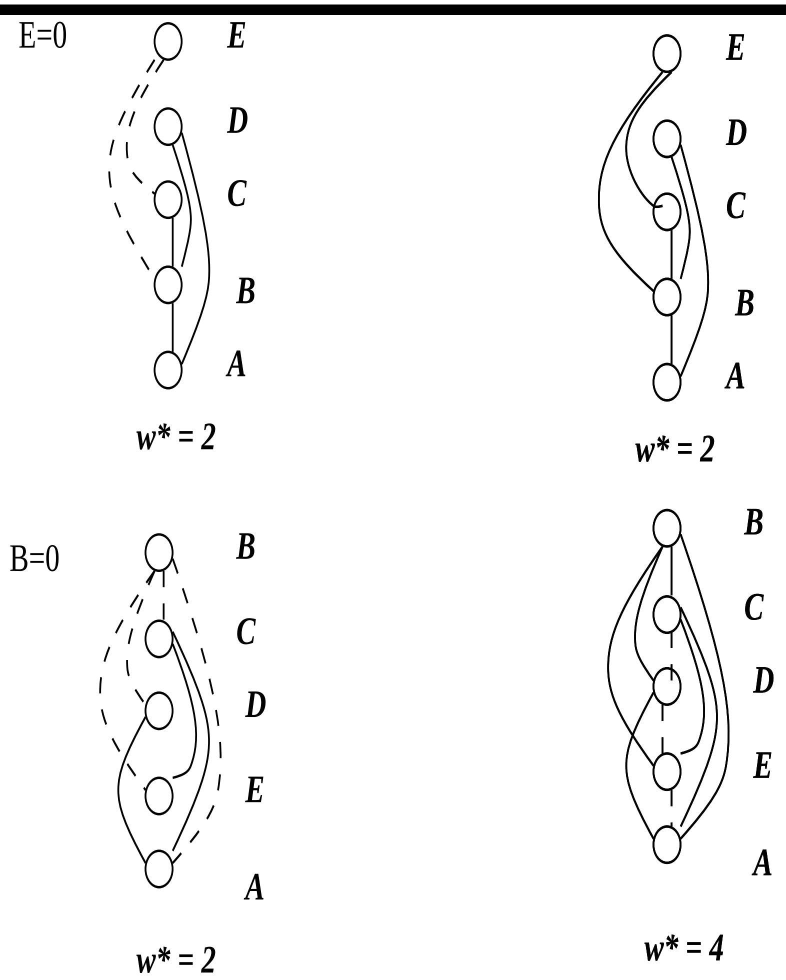
Ordering: a, b, c, d, e

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

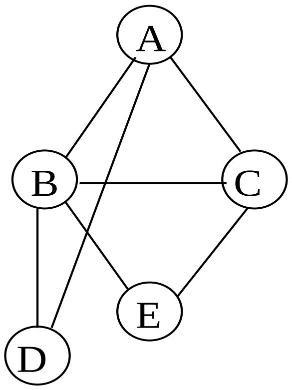
Ordering: a, e, d, c, b

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

Bucket Elimination and Induced Width



Handling Observations



Observing $b = 1$

Ordering: a, e, d, c, b

$bucket(B) = \quad P(e|b,c), P(d|a,b), P(b|a), b = 1$
$bucket(C) = \quad P(c|a), \quad || \quad P(e|b = 1, c)$
$bucket(D) = \quad || \quad P(d|a, b = 1)$
$bucket(E) = \quad e = 0 \quad || \quad \lambda_C(e,a)$
$bucket(A) = \quad P(a), \quad || \quad P(b = 1|a) \ \lambda_D(a), \lambda_E(e,a)$

Ordering: a, b, c, d, e

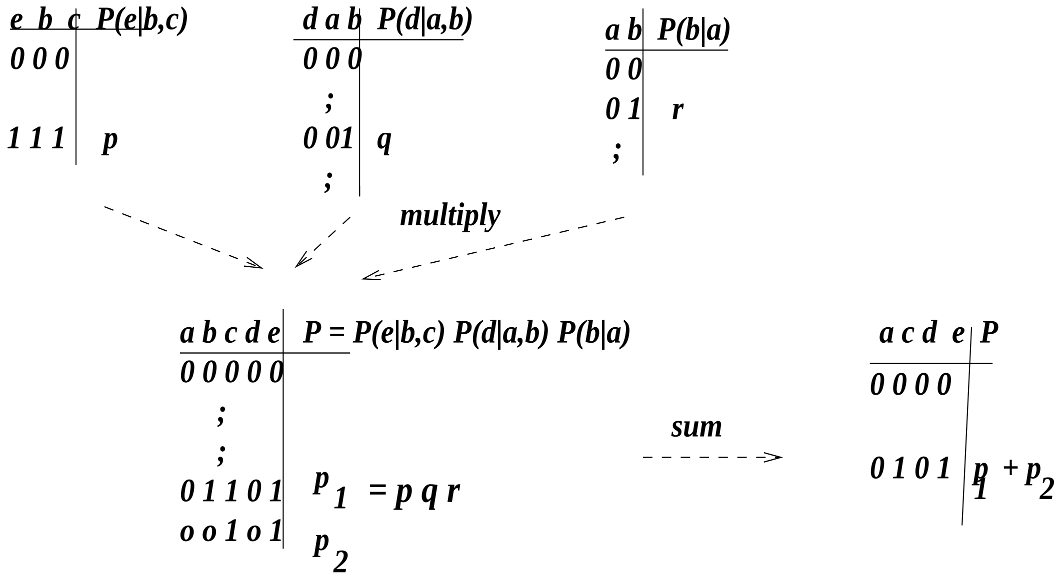
$bucket(E) = \quad P(e|b,c), \ e = 0$
$bucket(D) = \quad P(d|a,b)$
$bucket(C) = \quad P(c|a) \quad || \quad \lambda_E(b,c)$
$bucket(B) = \quad P(b|a), b = 1 \quad || \quad \lambda_D(a,b), \lambda_C(a,b)$
$bucket(A) = \quad P(a) \quad || \quad \lambda_B(a)$

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.

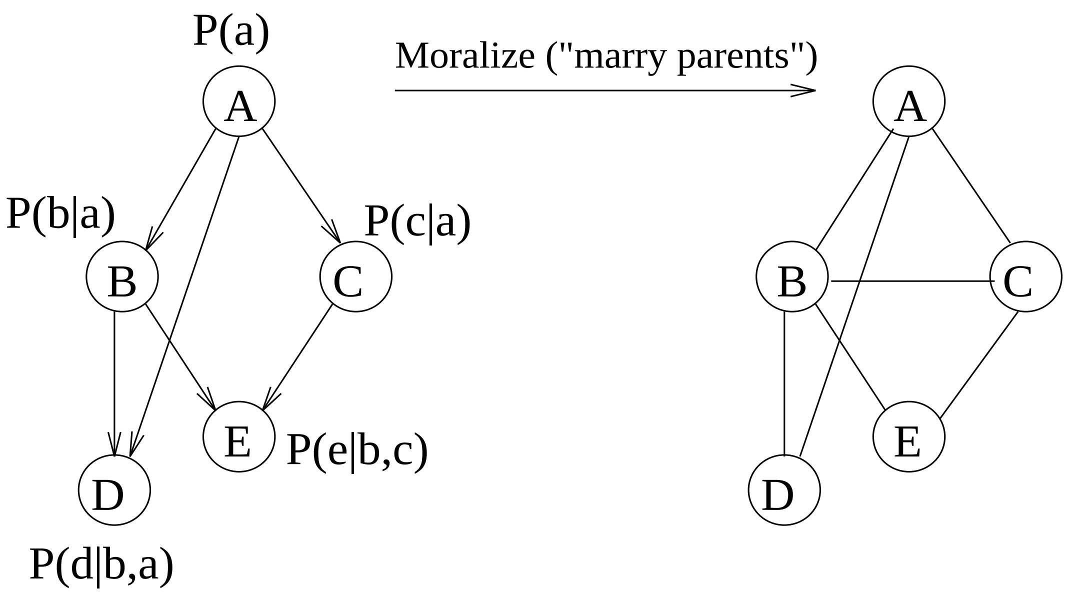
Recursive recognition : ($bel(a|e)$)

$bucket(E) =$ $P(e|b,c),\ e = 0$
$bucket(D) =$ $P(d|a,b)$,...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: $a, \; e, \; d, \; c, \; b$

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

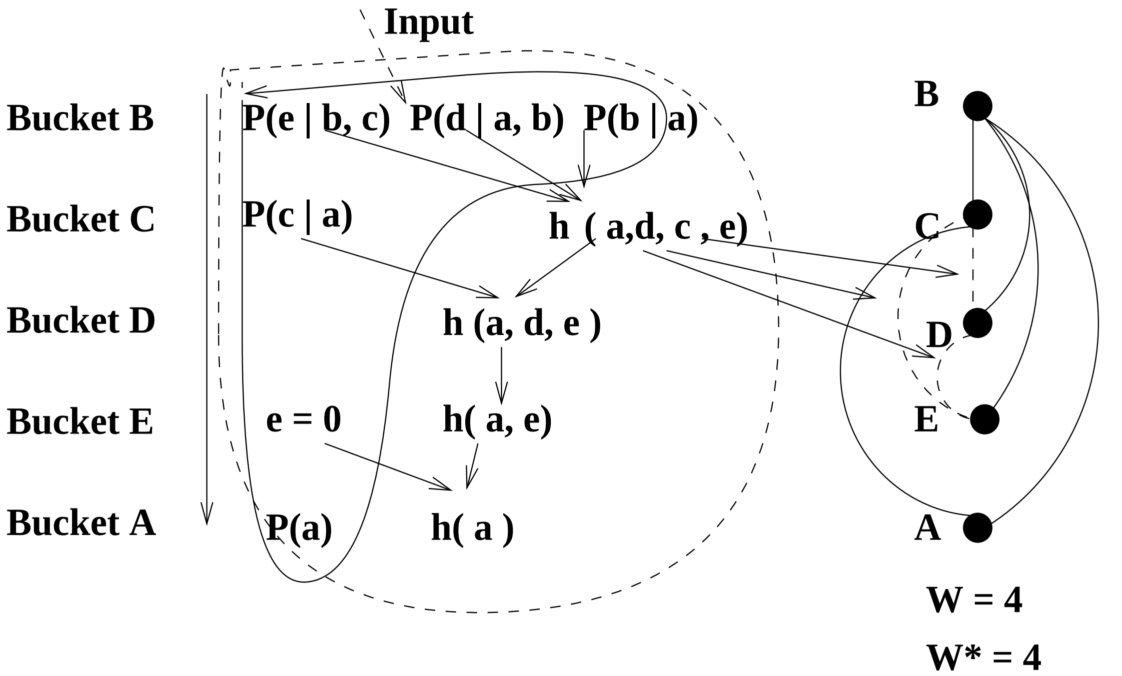
$m = \max_a P(a) \max_e \max_d \cdot$

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

Algorithm Elim-mpe

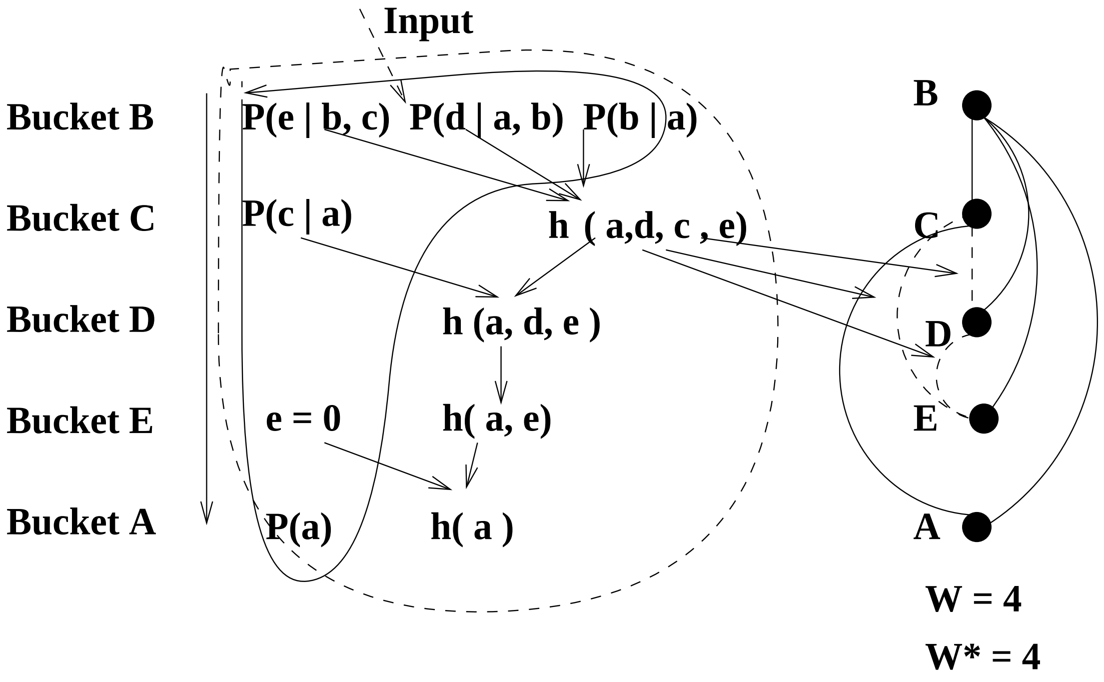
Input: A Belief network $P = \{P_1, ..., P_n\}$
Output: MPE

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:**

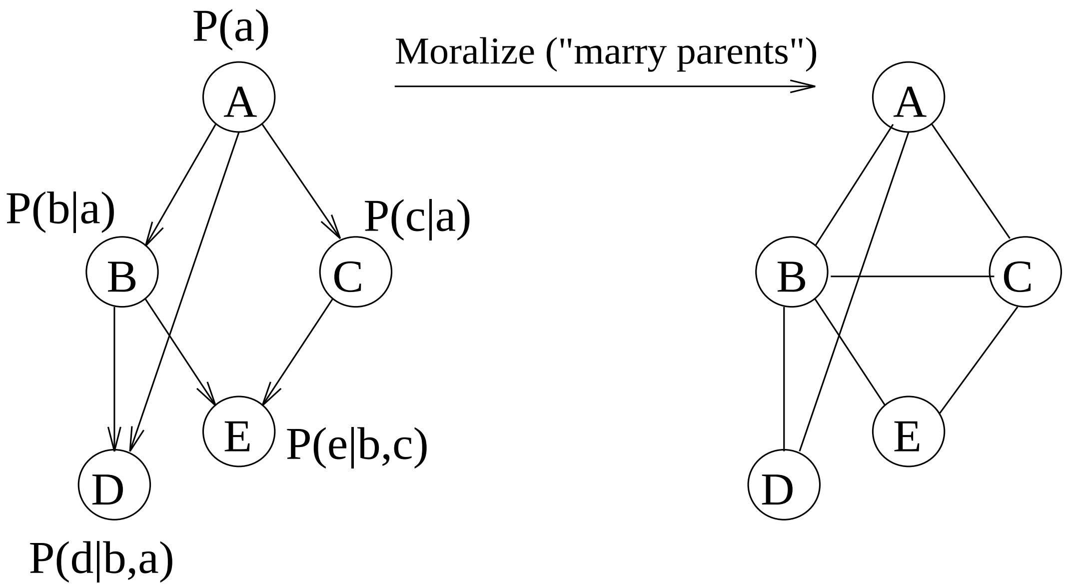
2. **Process buckets:** for $p = n$ to 1 do
   for matrices $h_1, h_2, ..., h_j$ in $bucket_p$ do
   - **If** (observed variable) assign $X_p = x_p$ to each h_i and put in buckets.
   - **Else,** (multiply and maximize)
     $h_p = \max_{X_p} \Pi_{i=1}^{j} h_i$.
     $x_p^{opt} = argmax_{X_p} h_p$.
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

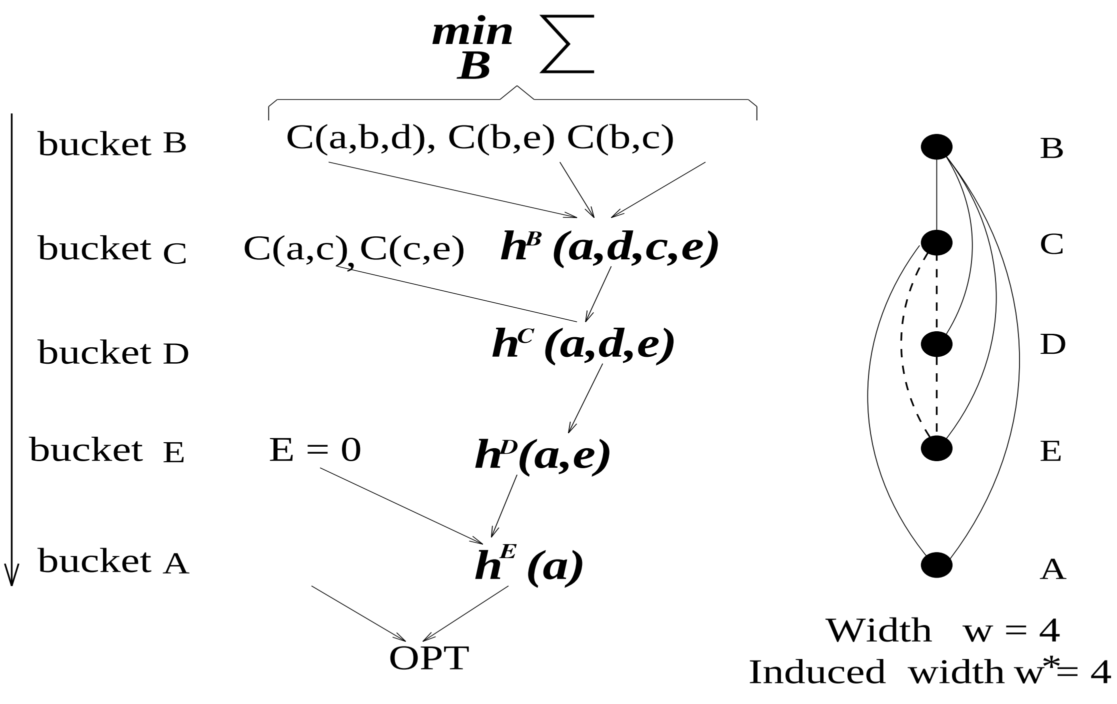
- **If** (observed variable) $X_p = x_p$, assign $X_p = x_p$ to each h_i and put in buckets.
- **Else,** (sum and minimize)
  $h^p = min_{X_p} \sum_{i=1}^{j} h_i$.
  $x_p^{opt} = argmin_{X_p} h^p$.
  Add h^p to its bucket.
  3. **Forward:** Assign minimizing values in ordering o

Algorithm Elim-Opt

(Dechter, Ijcai97)

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

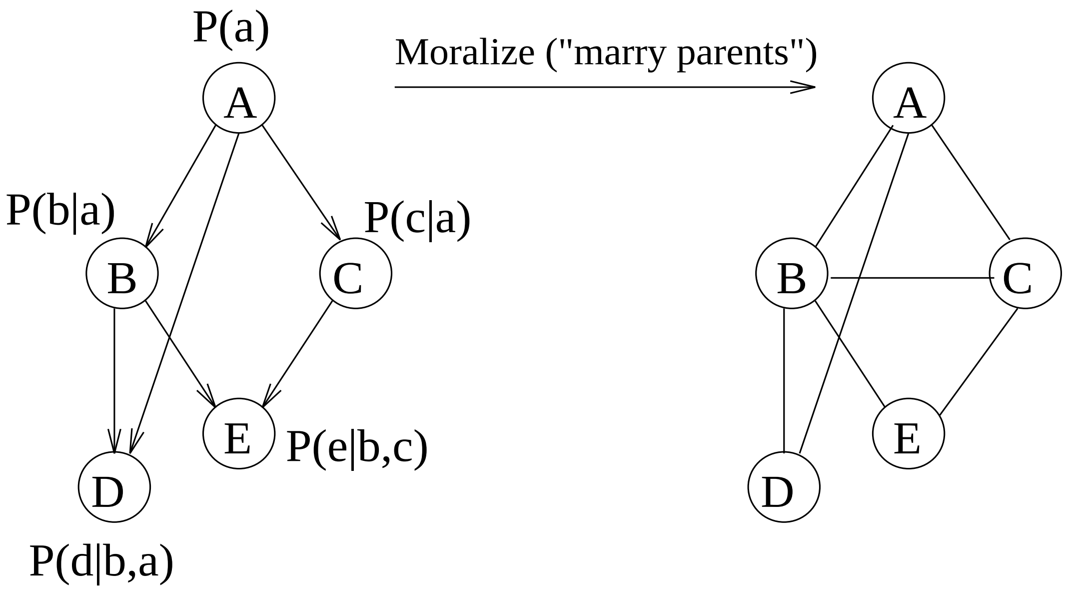
1. **Partition** $C = \{C_1, ..., C_r\}$ into buckets
2. **Process buckets from last to first:**



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. $p(a^o) = \max_{\bar{a}_k} \sum_{x_{X-A}} \prod_{i=1}^{n} P(x_i | x_{pa_i}, e)$.

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)
     $\beta_p = \sum_{X_p} \prod_{i=1}^{j} \beta_i$,
     $(X_p \in A)$ $\beta_p = \max_{X_p} \prod_{i=1}^{j} \beta_i$
     $a^0 = argmax_{X_p} \beta_p$.
     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

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

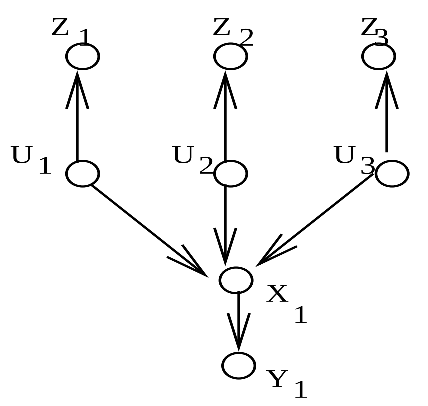
where $w*(d)$ is the induced width of the moral graph whose edges connecting evidence to earlier nodes, were deleted.

Bucket-Elimination for trees and Poly-Trees

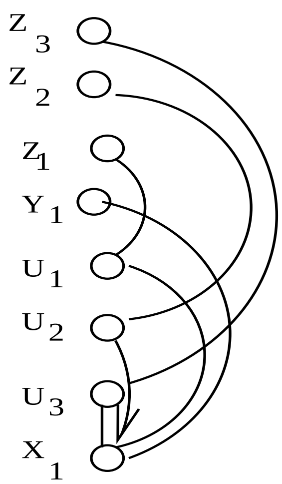
Elim-bel, elim-mpe, elim-map are linear for poly-trees.

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

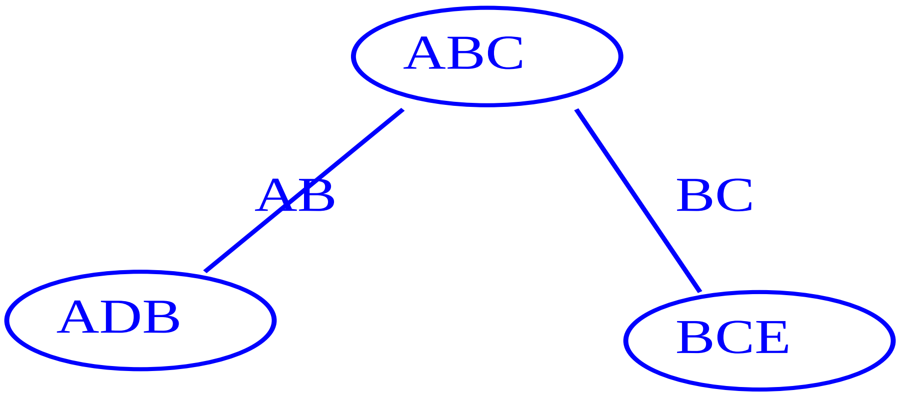
$bucket(E) = \quad P(e|b,c)$

$bucket(D) = \quad P(d|a,b)$

$bucket(C) = \quad P(c|a), \ || \ \lambda_E(a,b)$

$bucket(B) = \quad P(b|a), \ || \ \lambda_C(a,b)$

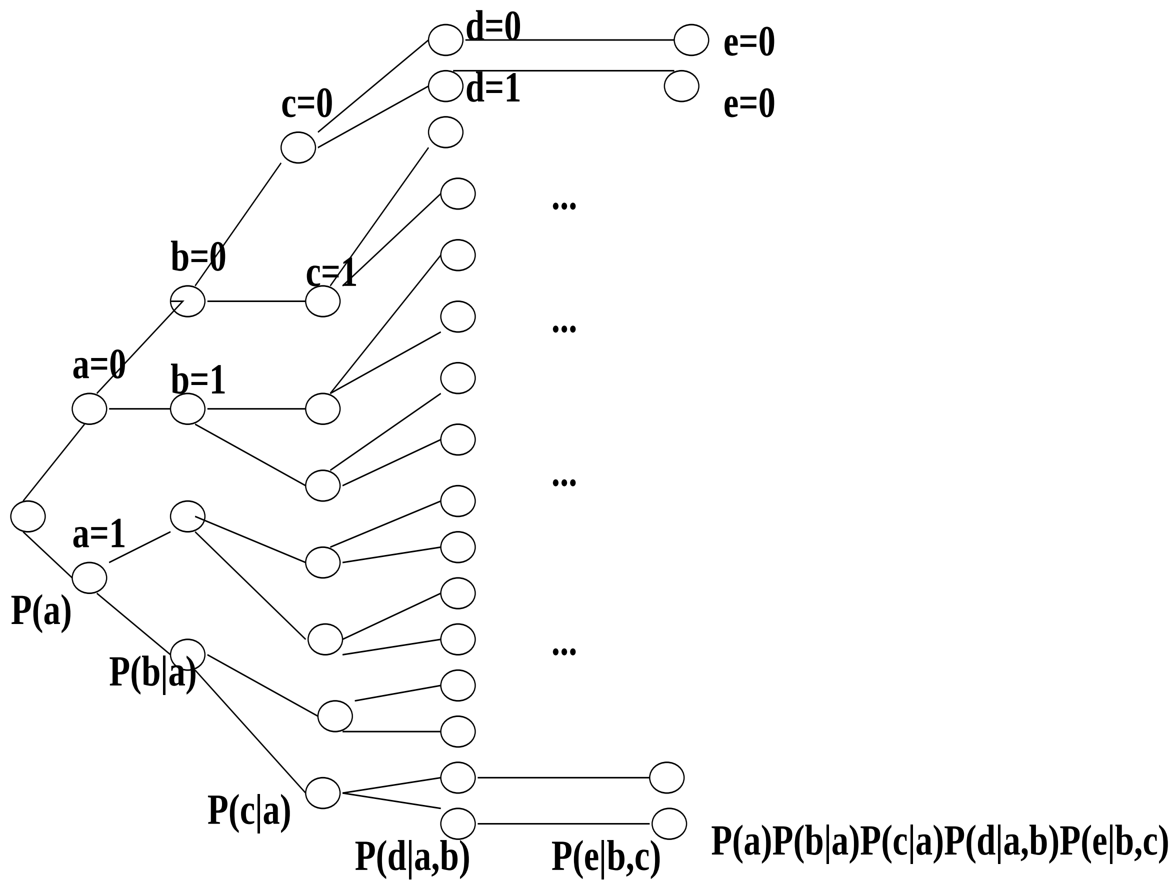
$bucket(A) = \quad P(a), \ || \ \lambda_B(a)$



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:

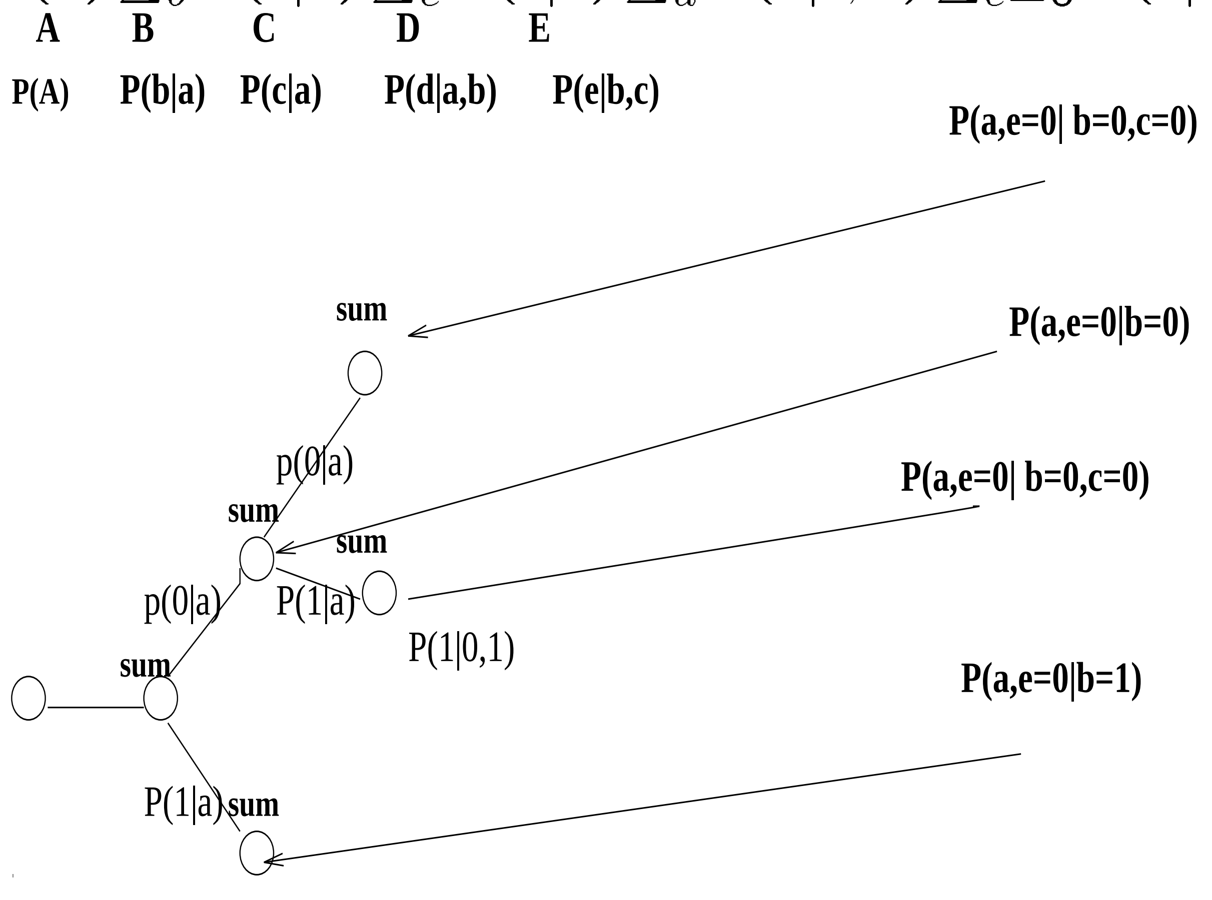
Time: exponential

Space: linear.

Conditioning+ Elimination

$P(a, e = 0) =$

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



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

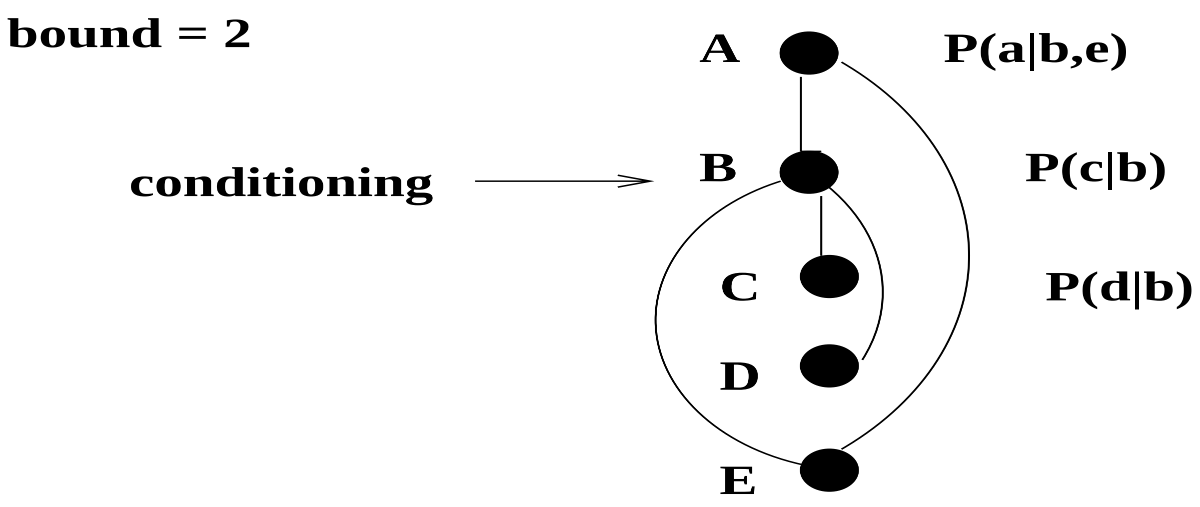
Conditioning + Elimination

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 $exp(b)$

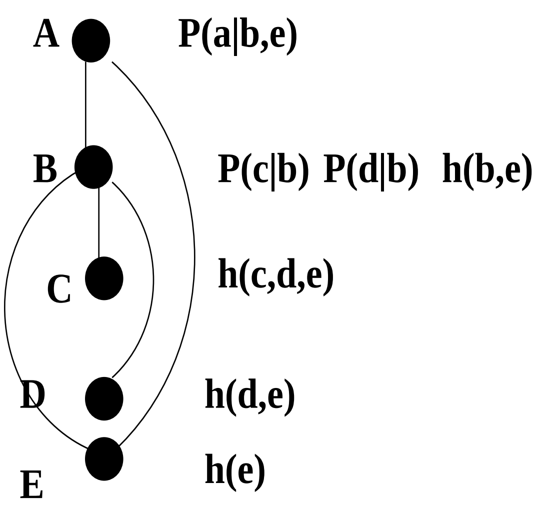


Super-Bucket Elimination

Trading space for time

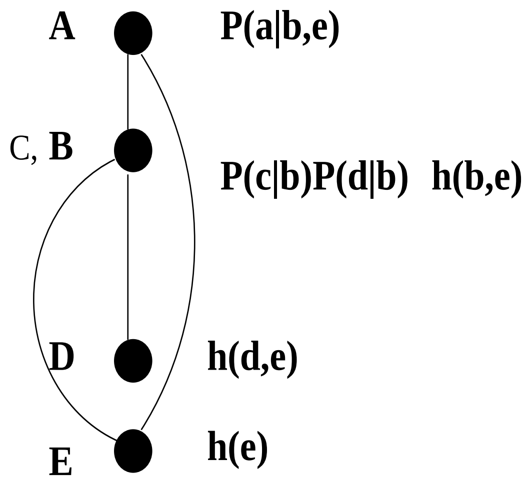
(Dechter and El Fattah, UAI 1996)

- Eliminating a few variables "at once".



time: exp(3)

space: exp(3)



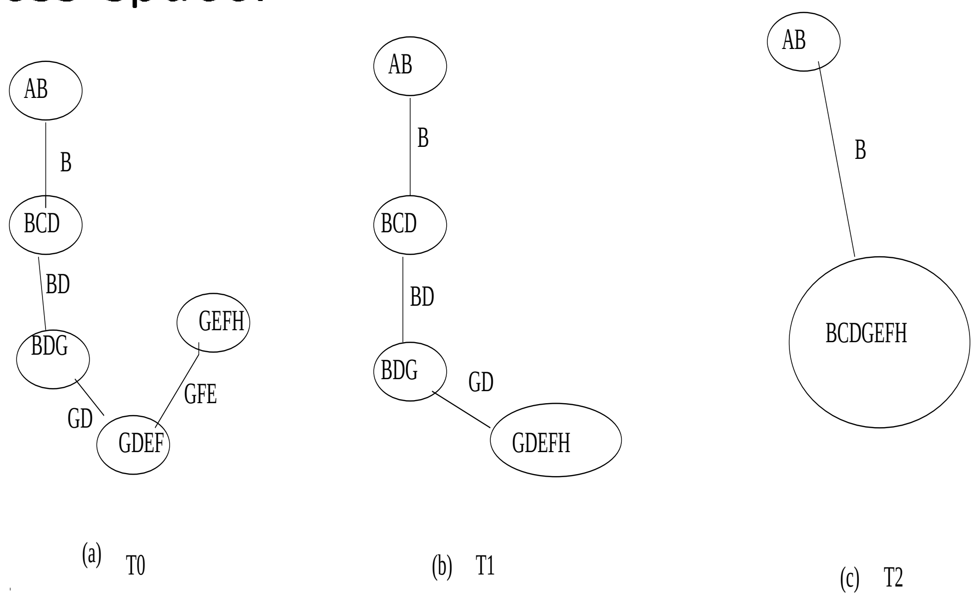
time: exp(4)

space: exp(2)

- Here conditioning is local to super-buckets.

The Super-Bucket Idea

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



Complexity:

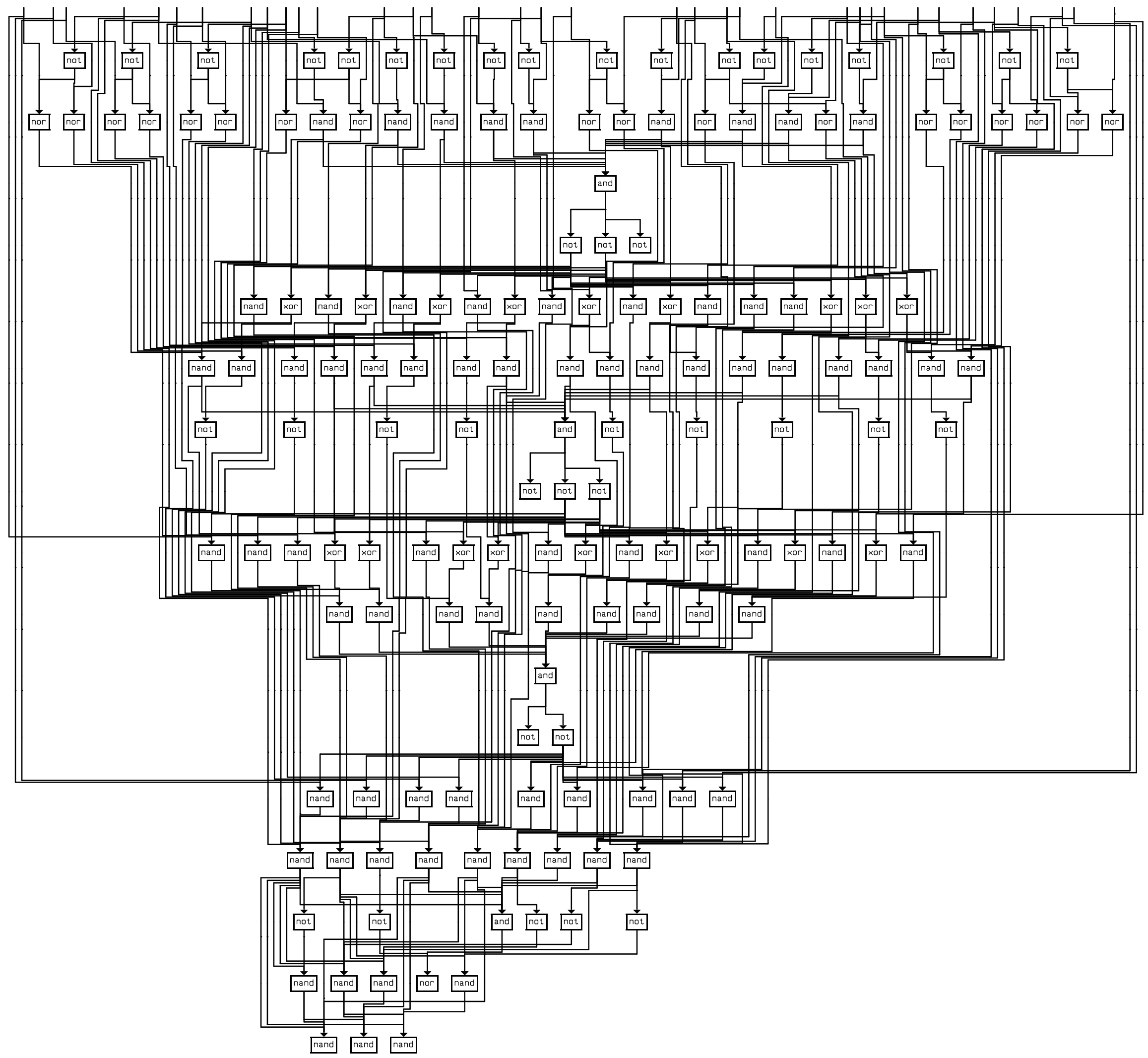
1. Time: exponential in clique and super-bucket size

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

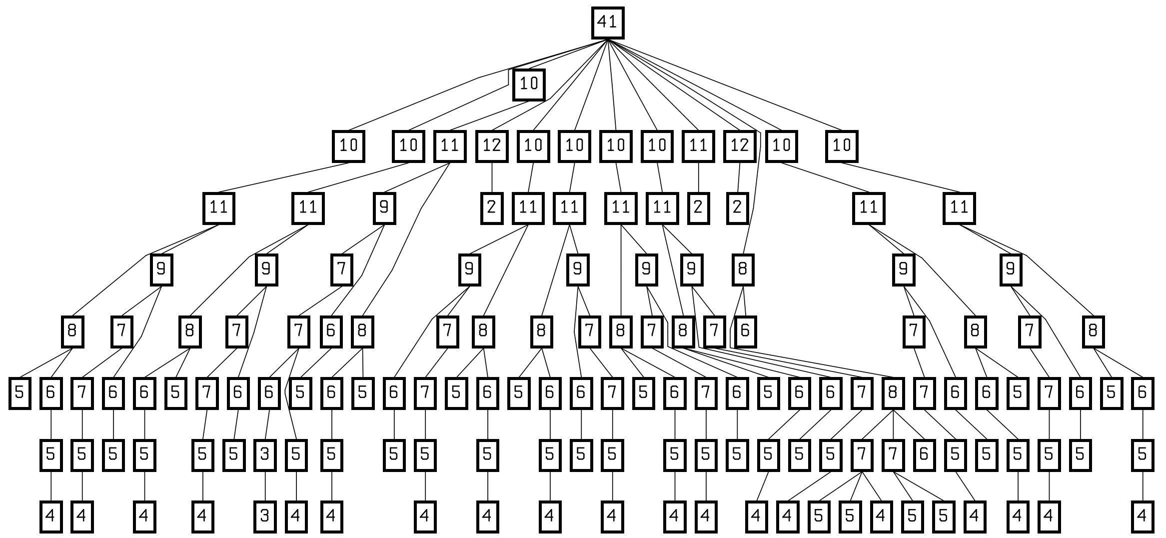


Benchmark Circuits

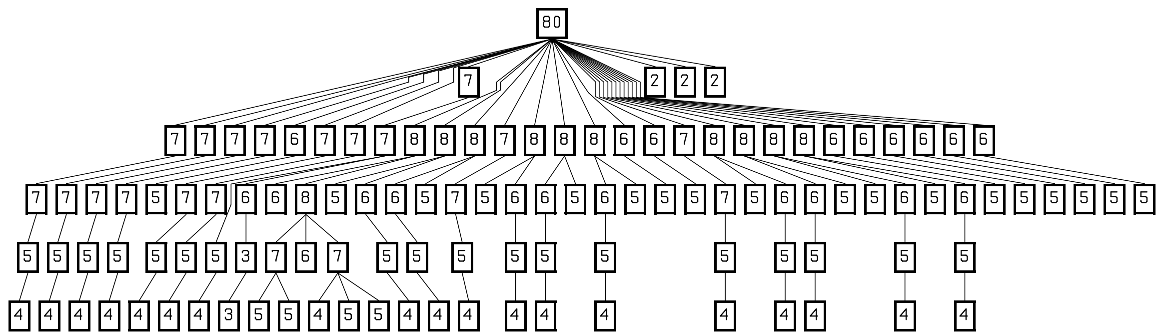
Circuit Name	Circuit Function	Total Gates	Input Lines	Output Lines
C17		6	5	2
C432	Priority Decoder	160 (18 EXOR)	36	7
C499	ECAT	202 (104 EXOR)	41	32
C880	ALU and Control	383	60	26
C1355	ECAT	546	41	32
C1908	ECAT	880	33	25
C2670	ALU and Control	1193	233	140
C3540	ALU and Control	1669	50	22
C5315	ALU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

Secondary Trees for C432

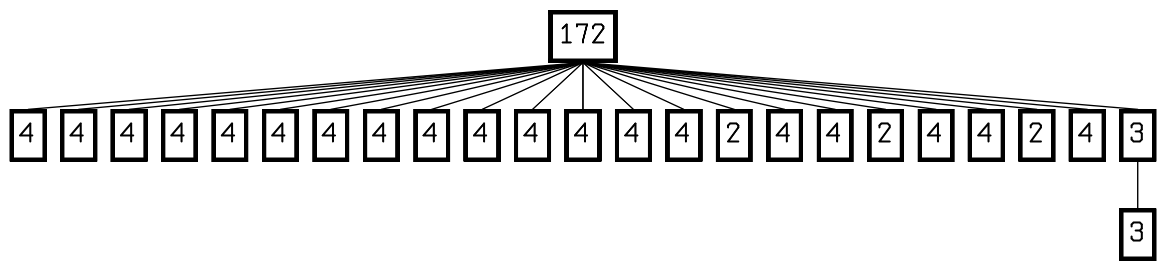
Separator size 11



Separator size 7



Separator size 3



Time-Space tradeoff for circuits

"Road Map":

Tasks and Methods

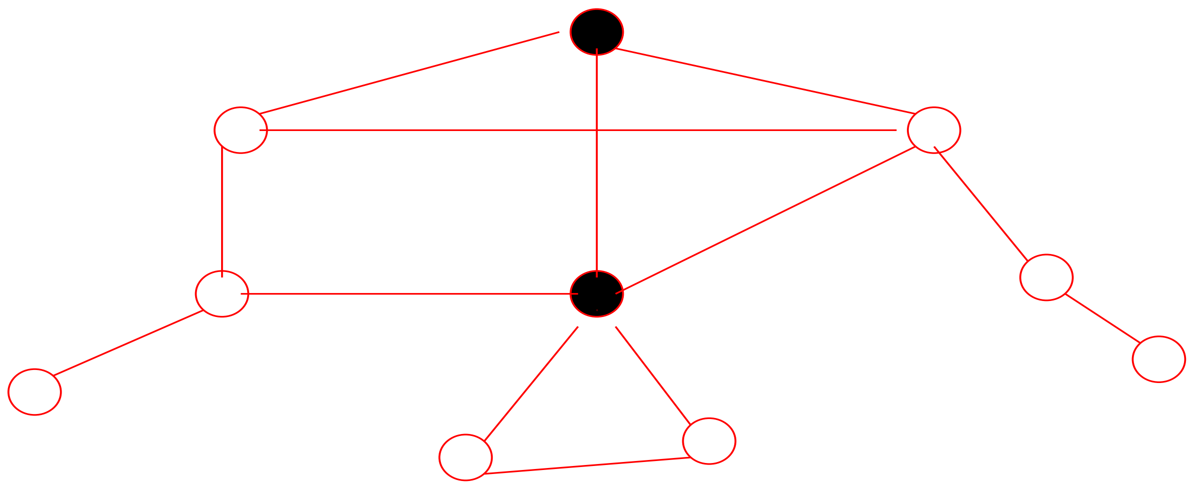
Methods \ Tasks	CSP	SAT	Optimi-zation	Belief updating	MPE, MAP, MEU	Solvin linea equali inequa
elimination	adaptive consistency, join-tree	directional resolution	dynamic program-ming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gauss Fourie elimin
conditioning	backtracking search	backtracking (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.

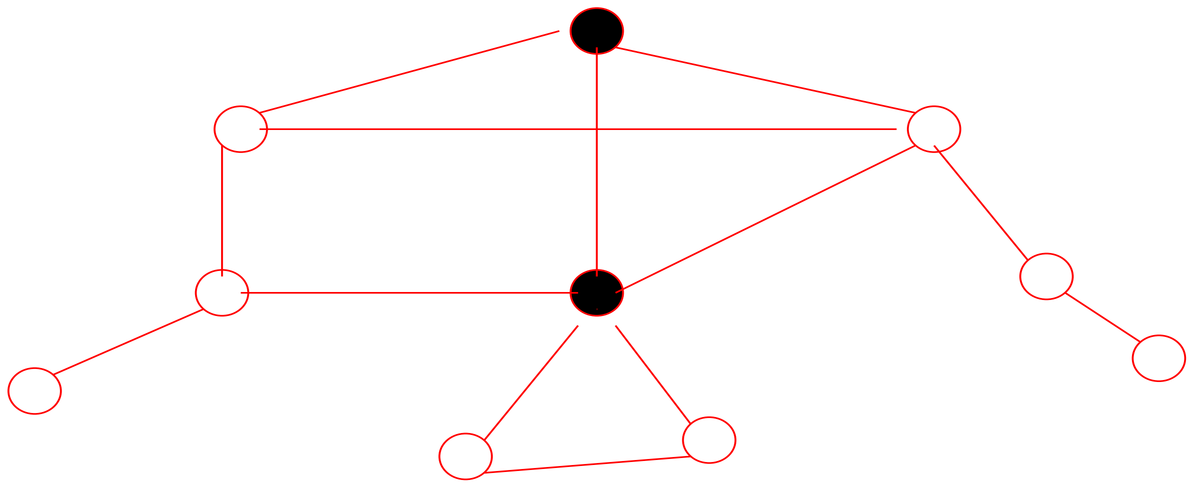
Distributed version: Energy minimization in a Hopfiled neural network (Hopfiled, 1982), Boltzman machines.

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

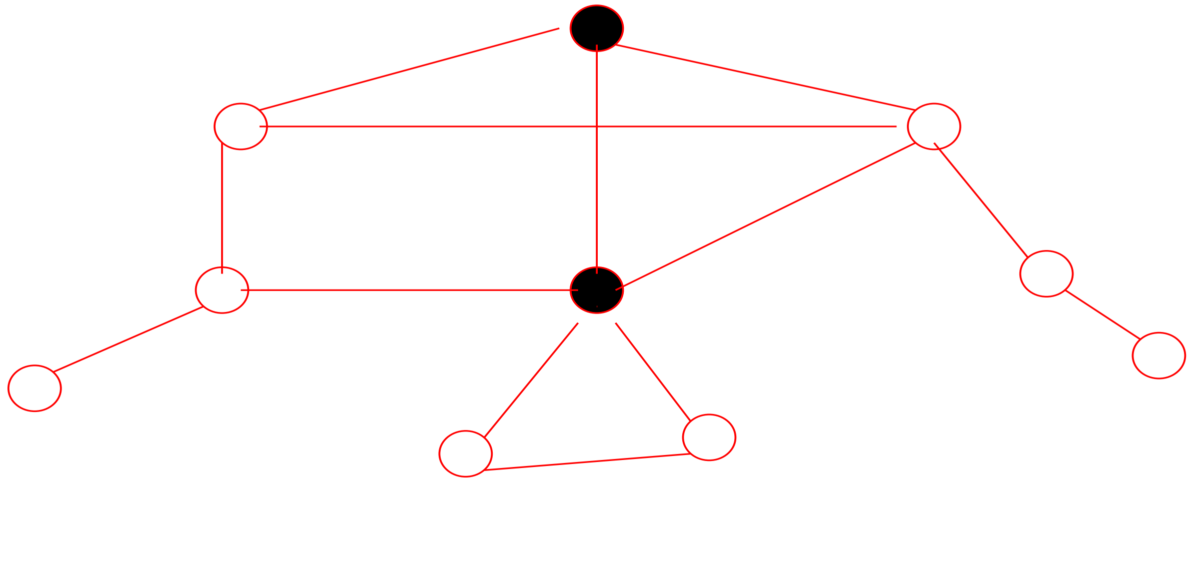
GSAT with Cycle-Cutset

(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, tightness 44/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
125	11 %	29 sec	46	10 sec	90	2 sec
130	12 %	46 sec	29	16 sec	77	6 sec
135	14 %	65 sec	13	23 sec	52	10 sec
Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 40/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
160	20 %	52 sec	33	20 sec	90	7 sec
165	21 %	60 sec	13	30 sec	80	17 sec
170	22 %	70 sec	4	40 sec	54	22 sec
Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 32/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
235	34 %	52 sec	69	14 sec	66	18 sec
240	35 %	76 sec	57	22 sec	57	29 sec
245	36 %	113 sec	40	43 sec	40	43 sec
Binary CSP, 100 instances per line, 100 variables, 8 values, tightness 28/64						
number of constraints	average cutset size	Time Bound	GSAT solved	GSAT time per solvable	GSAT+CC solved	GSAT+CC time per solvable
290	41 %	55 sec	74	13 sec	30	25 sec
294	42 %	85 sec	80	25 sec	23	41 sec
300	43 %	162 sec	63	45 sec	19	82 sec

GSAT with cycle-cutset

(Kask and Dechter, AAAI 1996)

"Road Map":

Tasks and Methods

Tasks / Methods	CSP	SAT	Optimization	Belief updating	MPE, MAP, MEU	Solvir linea equali inequa
elimination	adaptive consistency join-tree	directional resolution	dynamic programming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gauss Fourie elimin
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					

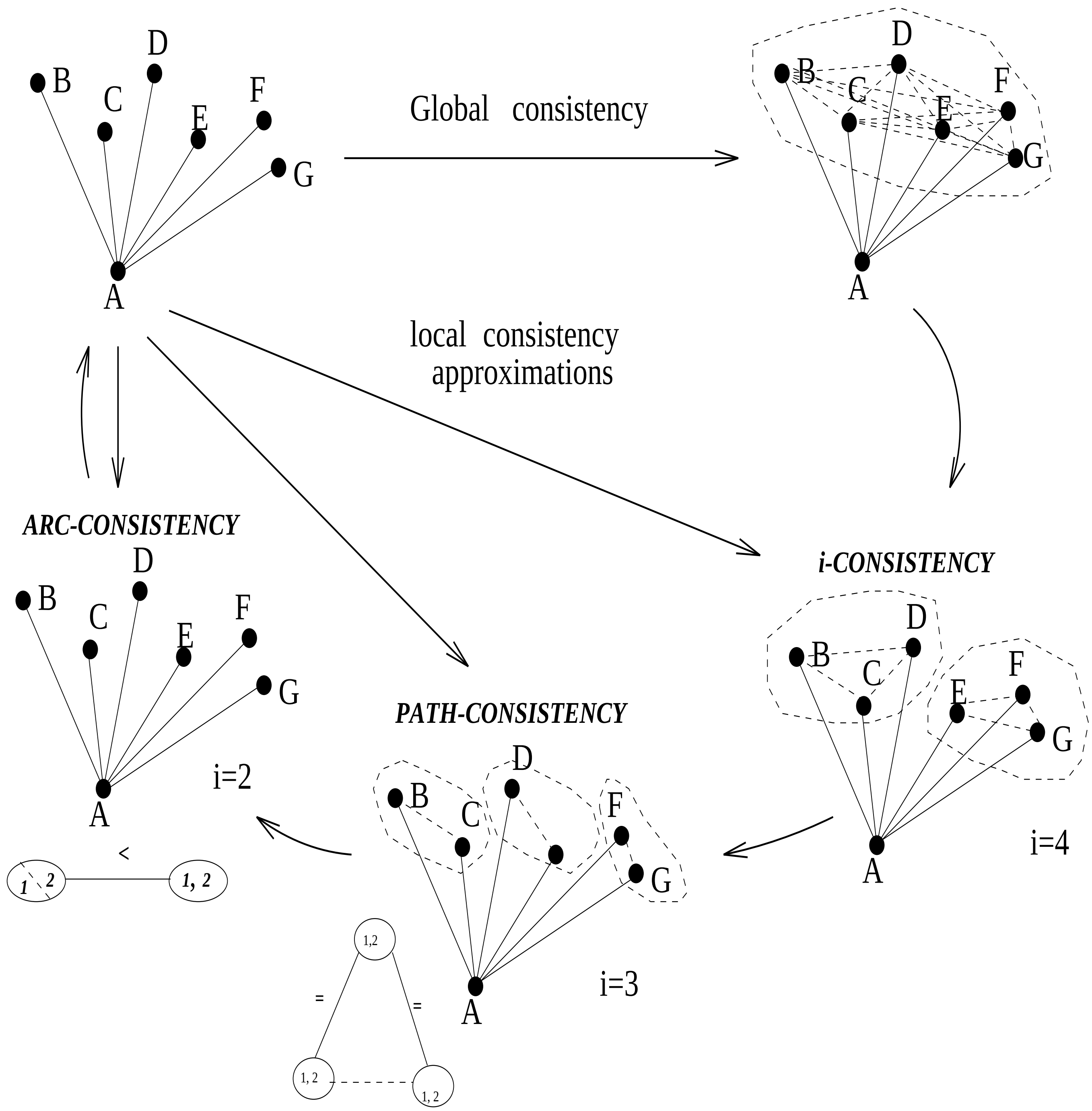
Approximating Elimination: Local Inference

- **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.

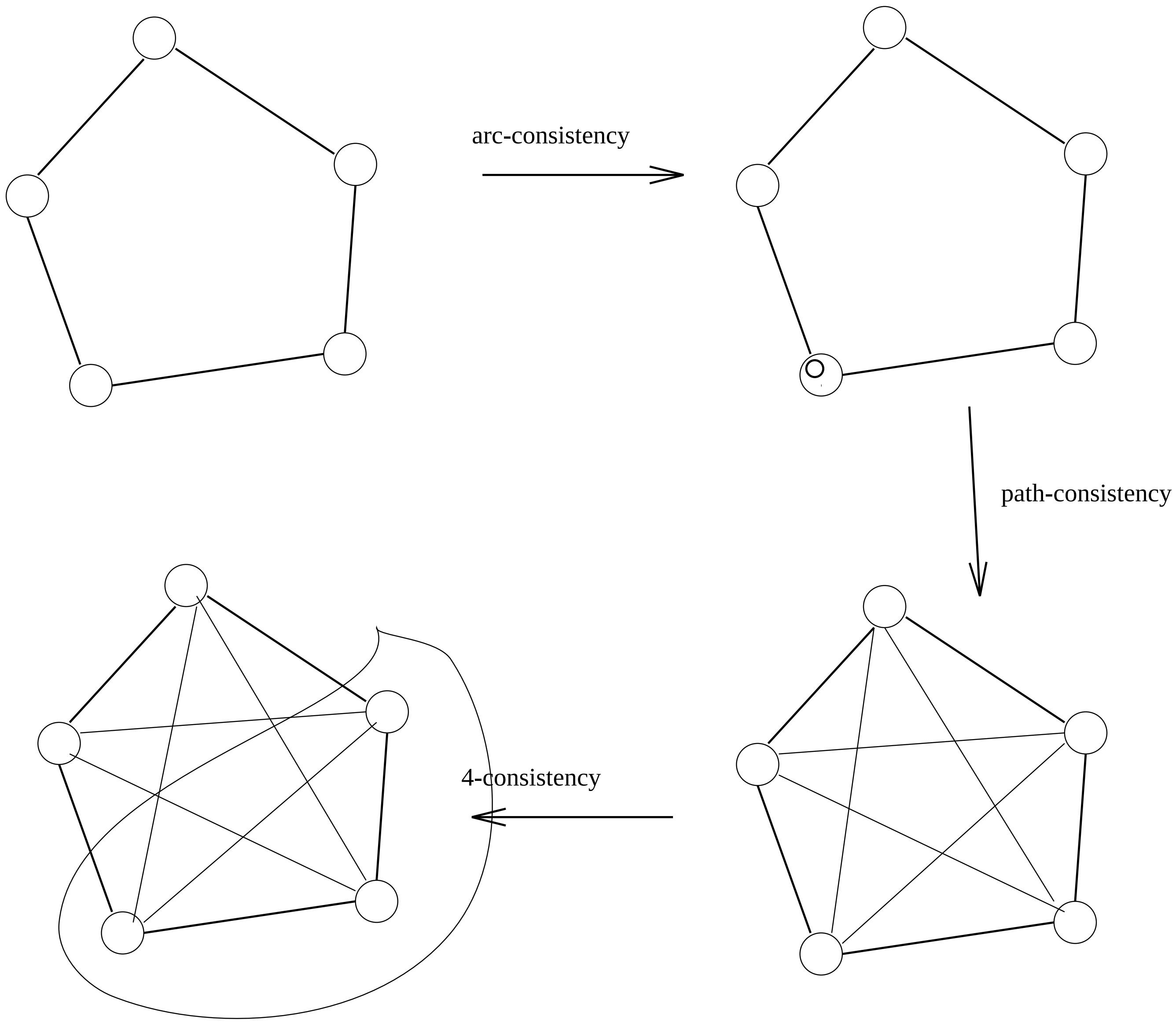
Enforcing Directional i-consistency

- 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

[Montanari,1974][Macworth1977][Freuder, 1978]



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).

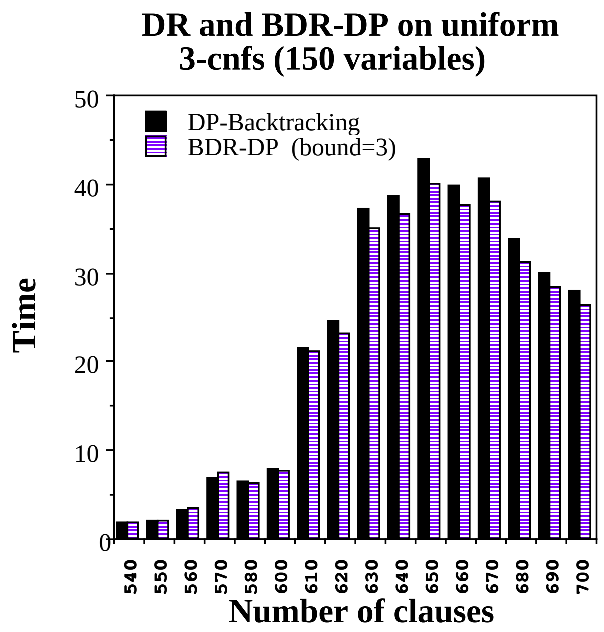
- Non-directional version: **k-closure** [van Gelder, 1996]. Enforces full k-consistency.

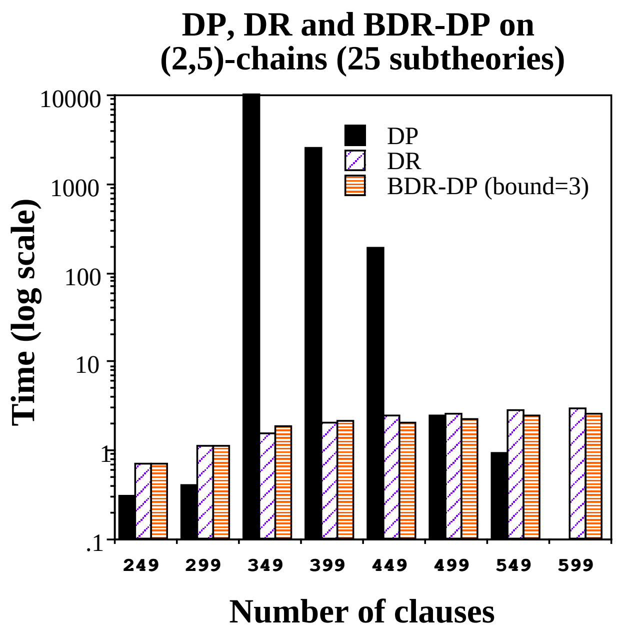
Preprocessing by i-consistency
(Dechter and Rish, KR 1994)

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) =

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

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

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

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

$$\Big\downarrow$$

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

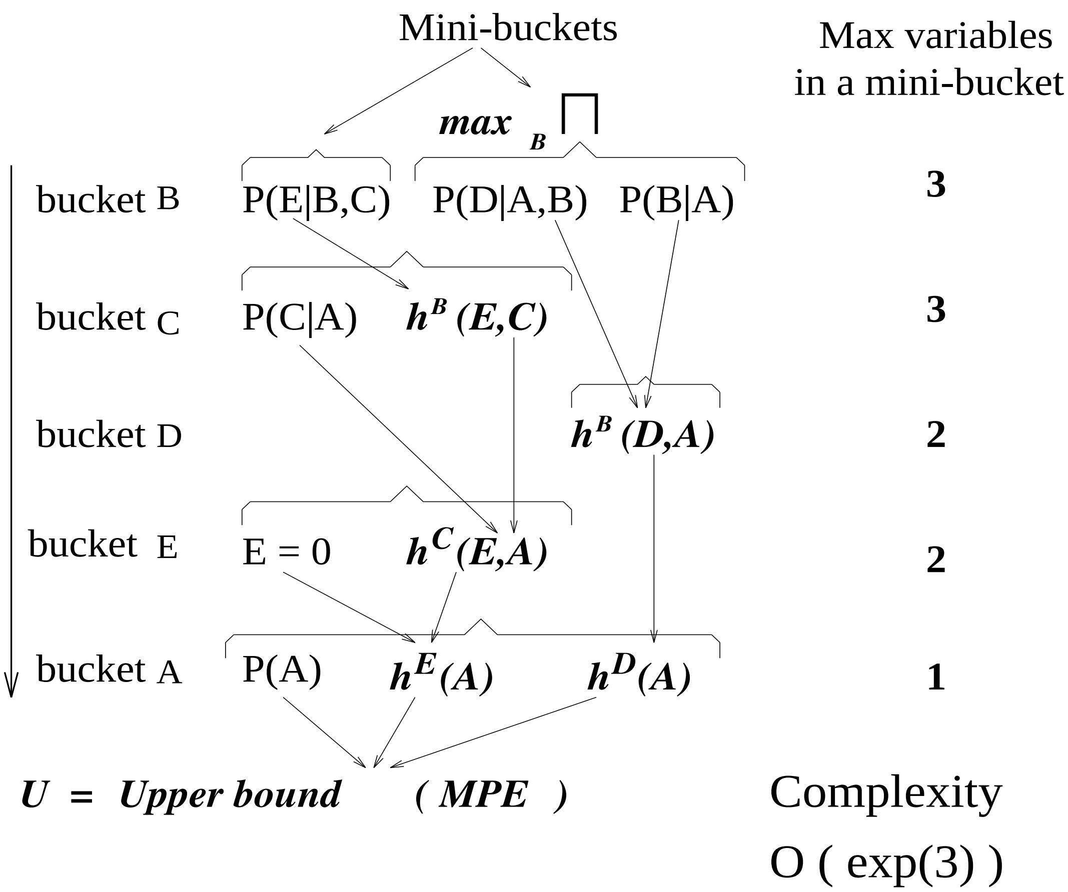
- Complexity decrease:

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

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

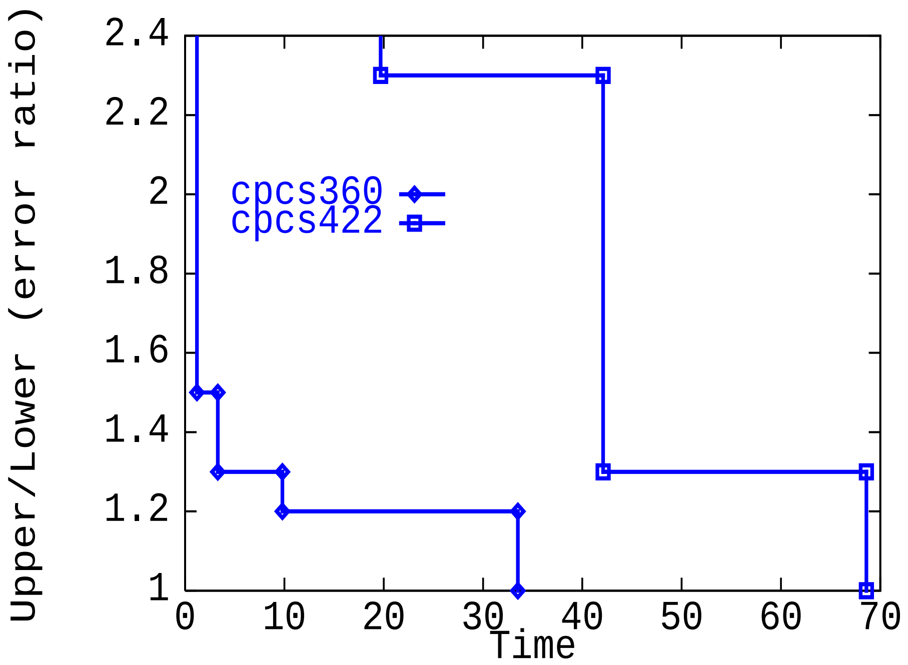
Algorithms:

- *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 $P(x = 1|\forall i pa_i(x) = 0) = 0$.

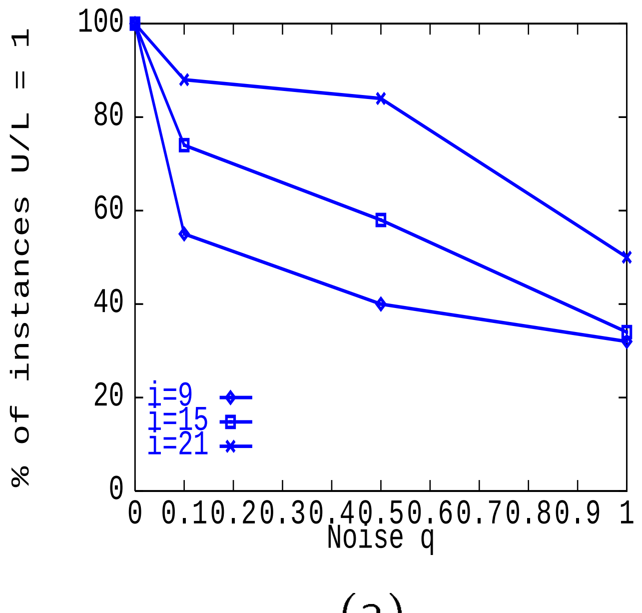
Results on (50 nodes, 150 edges)-networks

10 evidence nodes, 200 instances

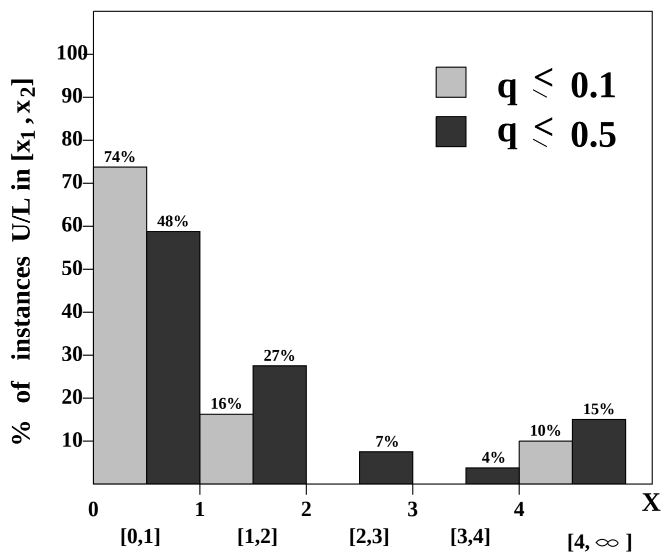
• elim-mpe ran out of memory;
approx-mpe(i) time: from 0.1 sec for $i = 9$ to 80 sec for $i = 21$.

• 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 T_e/T_a	U/M %	Mean T_e/T_a
$[1, 2]$	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 ≈ 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 T_e/T_a	U/M %	Mean T_e/T_a
$[1, 2]$	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 T_e/T_a = Mean value of elim-mpe time/approx-mpe time (T_e/T_a) 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.

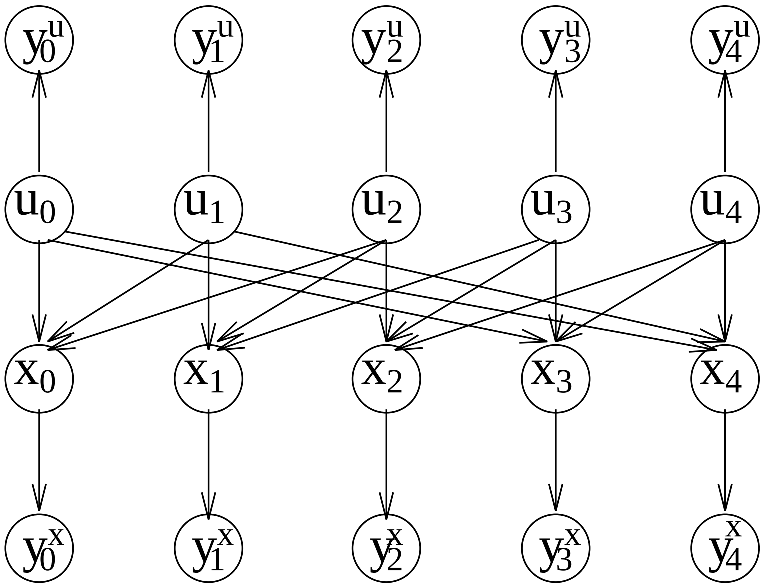
Decoding task: given Y, find U' s.t.:

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:



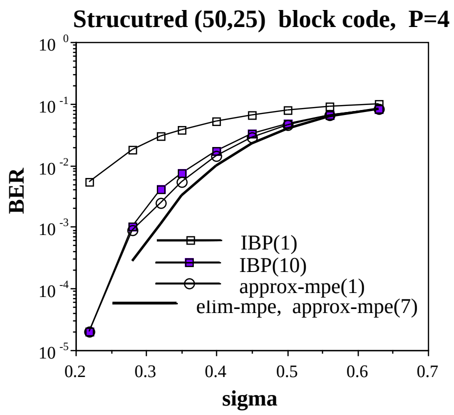
Problem parameters:
k - the number of the input information bits;
n - the number of code bits;
p - the number of parents of each code bit;
σ - the noisy channel parameter (*Gaussian noise*).

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).

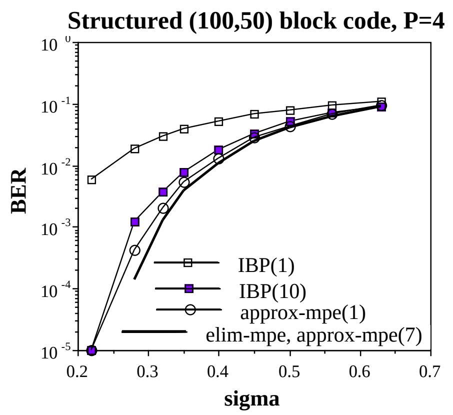
Structured Low-w^* Codes

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

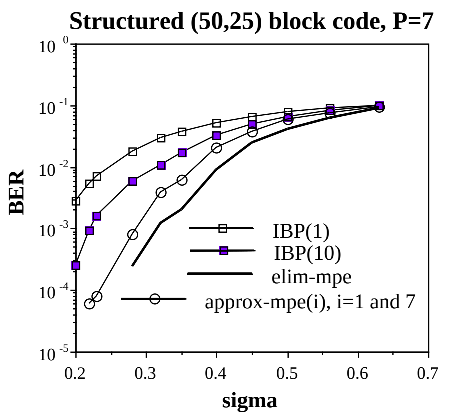
Approx-mpe(i) outperforms iterative belief propagation (IBP(I), I is the number of iterations) on **structured problems** with **small parent set size**:



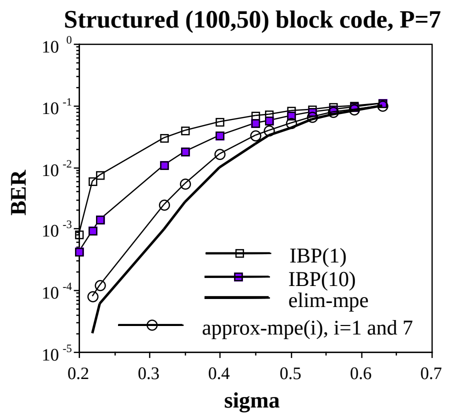
(a)



(b)



(c)

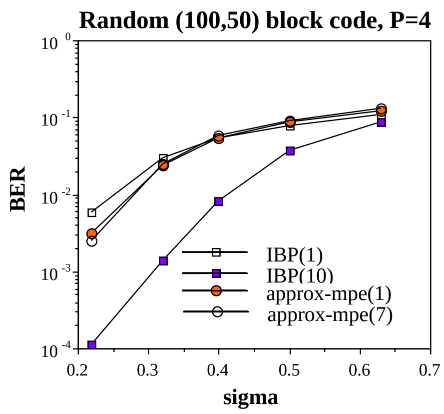


(d)

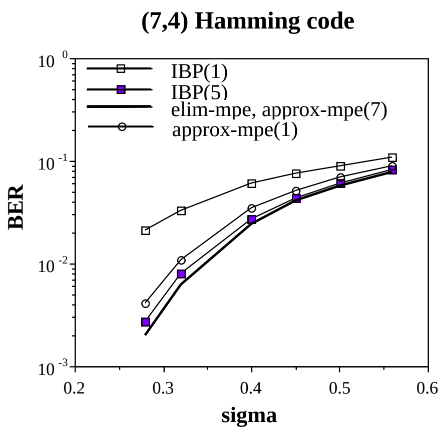
BER for exact elim-mpe and approximate IBP(1), IBP(10), approx-mpe(1) and approx-mpe(7) (1000 instances per point). Structured block codes with R=1/2 and (a) K=25, P=4, (b) K=50, P=4, (c) K=25, P=7, and (d) K=25, P=7. The induced width of the networks was 6 for (a) and (b), and 12 for (c) and (d).

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

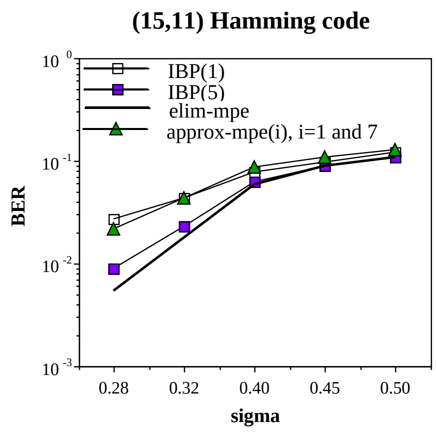
On the other hand, IBP outperforms approx-mpe(i) on **random problems** (high w^*) and on Hamming codes:



(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 $\geq$ 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 $q \to 0$

- **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.