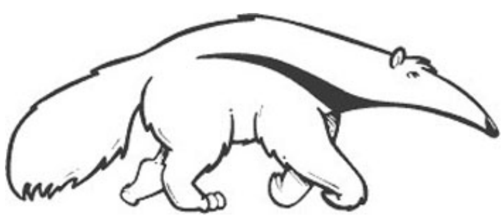
Variable Elimination

Introduction to Graphical Models

Prof. Alexander Ihler



BREN:ICS
INFORMATION AND COMPUTER SCIENCES

UNIVERSITY of CALIFORNIA IRVINE

Inference tasks in CSPs

Consider a simple coloring CSP:

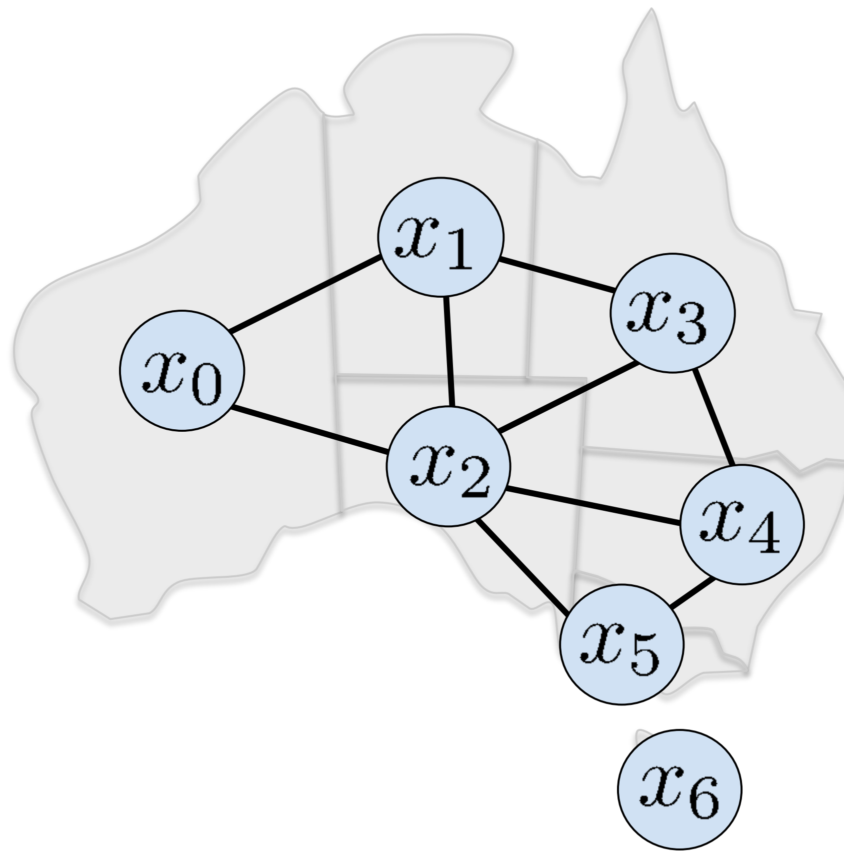
- Check for a solution:

$$F^* = \max_x \prod_\alpha f_\alpha(x_\alpha)$$

$$\hat{x} = \arg\max_x \prod_\alpha f_\alpha(x_\alpha)$$

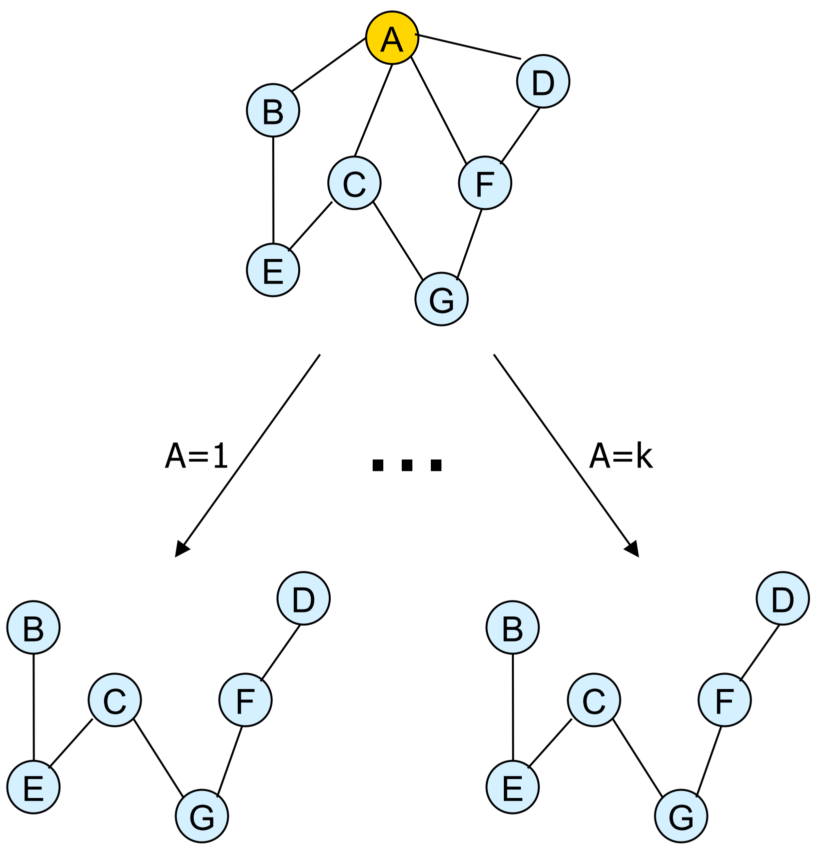
- Or, count solutions:

$$Z = \sum_x \prod_\alpha f_\alpha(x_\alpha)$$



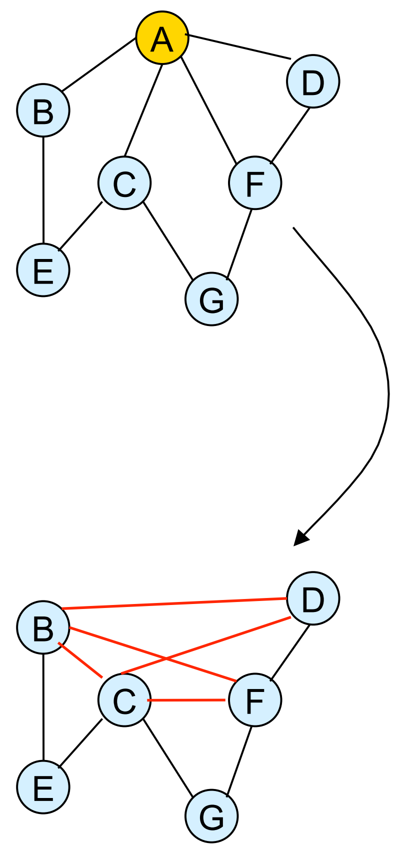
Two strategies: conditioning vs. elimination

Conditioning (search)



k "sparser" problems

Elimination (inference)



1 "denser" problem

Conditioning a cost function

A	B	f(A,B)
b	b	4
b	g	5
b	r	1
g	b	2
g	g	6
g	r	3
r	b	1
r	g	1
r	r	6

Assign **A=b**

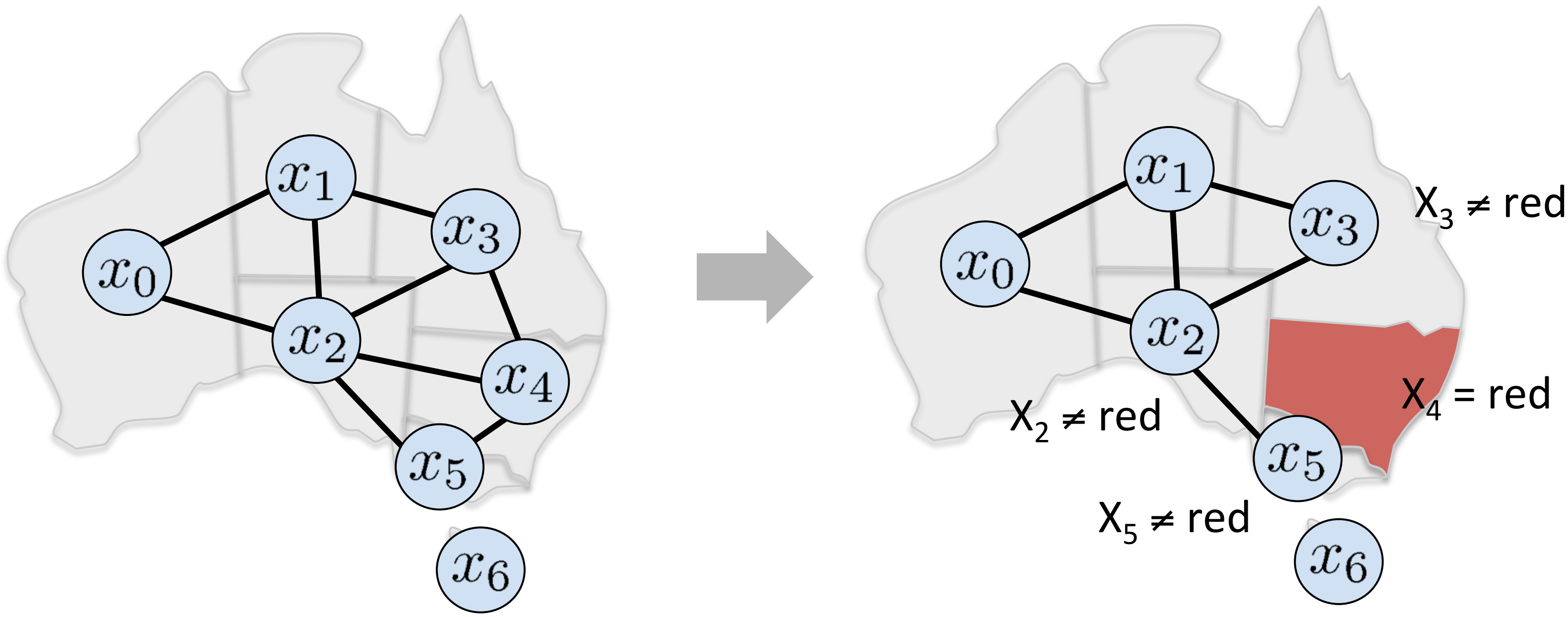
B	g(B)
b	4
g	5
r	1

Assign **B=r**

$h_{\emptyset}$
1

CSP Example

- Condition (assign) X_4:



Combination of cost functions

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

+

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

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

= 0 + 6

Elimination in a cost function

A	B	f(A,B)
b	b	4
b	g	5
b	r	1
g	b	2
g	g	6
g	r	3
r	b	1
r	g	1
r	r	4

max

Elim(**f**,**B**) →

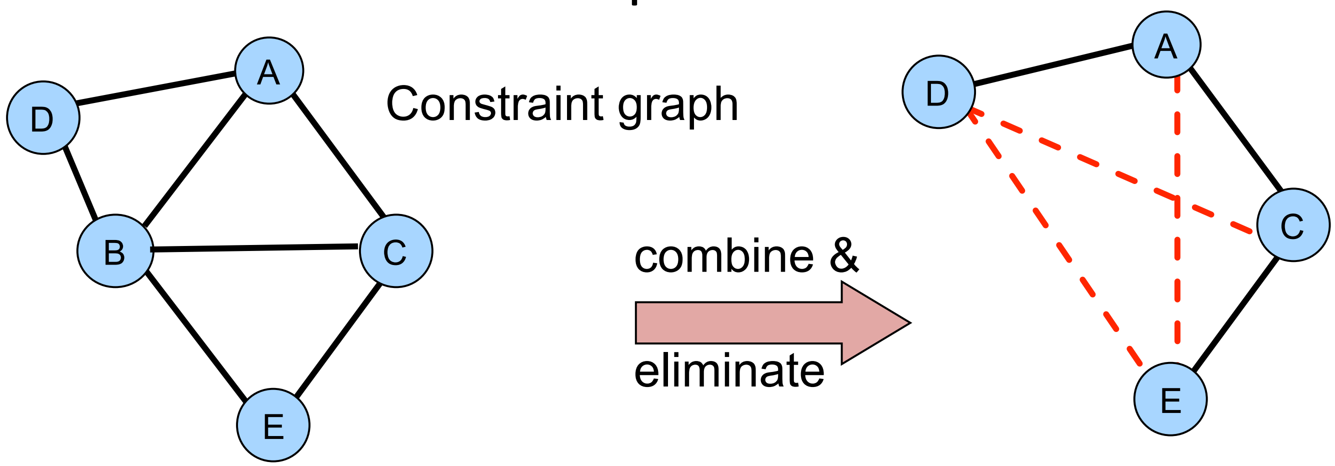
A	g(A)
b	5
g	6
r	4

Elim(**g**,**A**) →

$h_{\varnothing}$
6

Variable Elimination

- Eliminate variables in sequence:



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

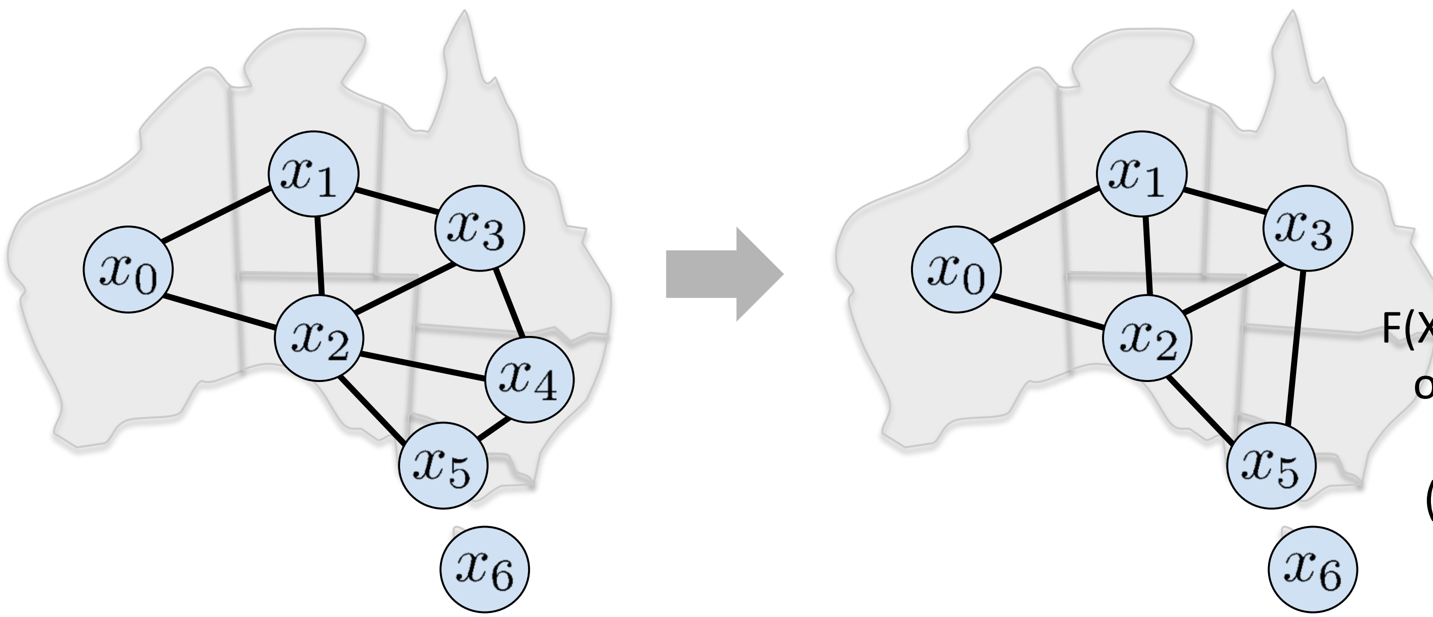
Collect & combine

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

Variable Elimination

CSP Example

- Eliminate (maximize over) X_4:



$F(X_2,X_3,X_5) = 1$ if only 2 colors used

(so, one is available for X_4)

Variable Elimination

- Bucket elimination [Dechter 1996]

  "Non-serial Dynamic Programming" [Bertele & Briochi 1973]

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

$\max_b \quad \sum$ ← Elimination/Combination operators

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

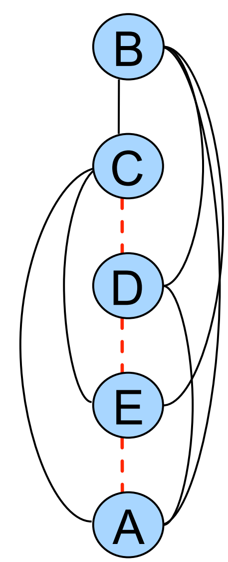
bucket C: $f(c,a) \; f(c,e) \; \lambda_{B \to C}(a,d,c,e)$

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

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

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

OPT



Generating the optimal assignment

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

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

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

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

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

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

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

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

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

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

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

Complexity of variable elimination

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

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

$\max_b \quad \sum$ ← Elimination/Combination operators

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

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

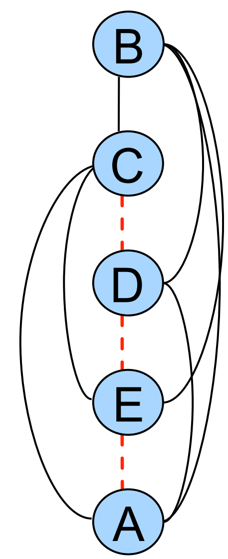
bucket D: $f(a,d) \; \lambda_{C \to D}(a,d,e)$

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

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

OPT

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



Complexity of Bucket Elimination

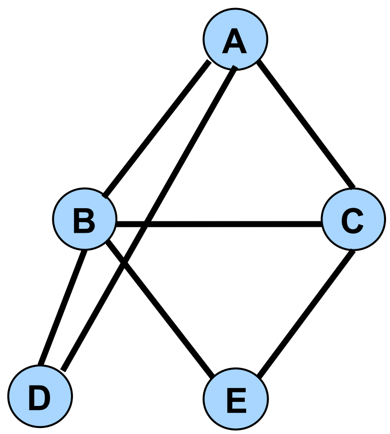
Bucket Elimination is **time** and **space**

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

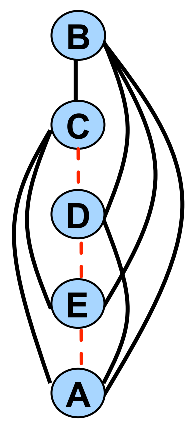
$w^*(d)$: the induced width of the primal graph along ordering *d*

r = number of functions

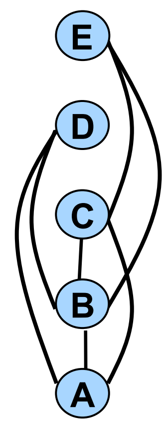
The effect of the ordering:



constraint graph



$w^*(d_1) = 4$

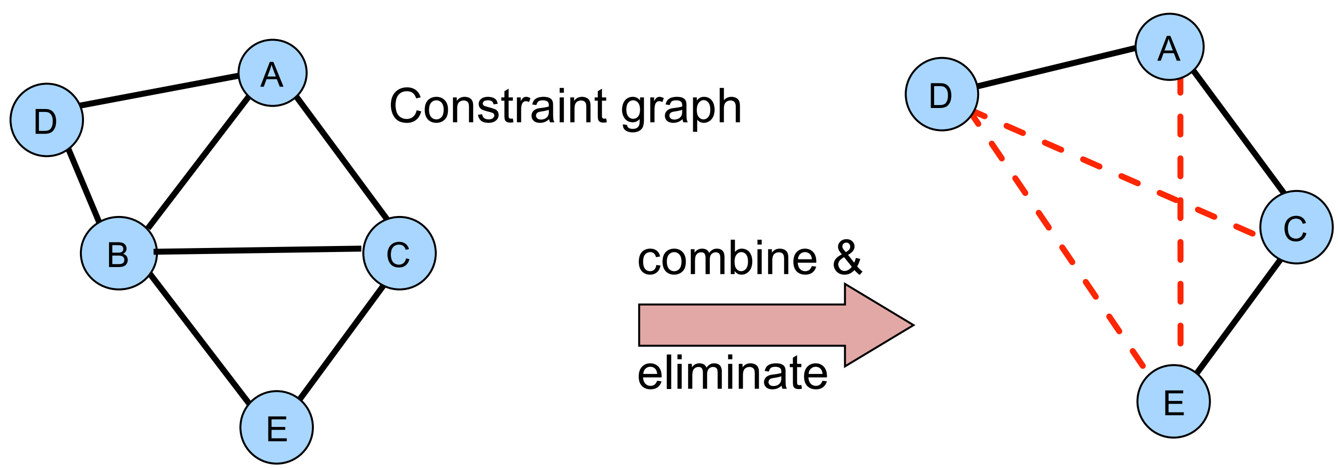


$w^*(d_2) = 2$

Finding the smallest induced width is hard!

Variable ordering heuristics

- What makes a good order?
  - Low induced width
  - Elimination creates a function over neighbors
- Finding the best order is hard
  - But we can do well with simple heuristics

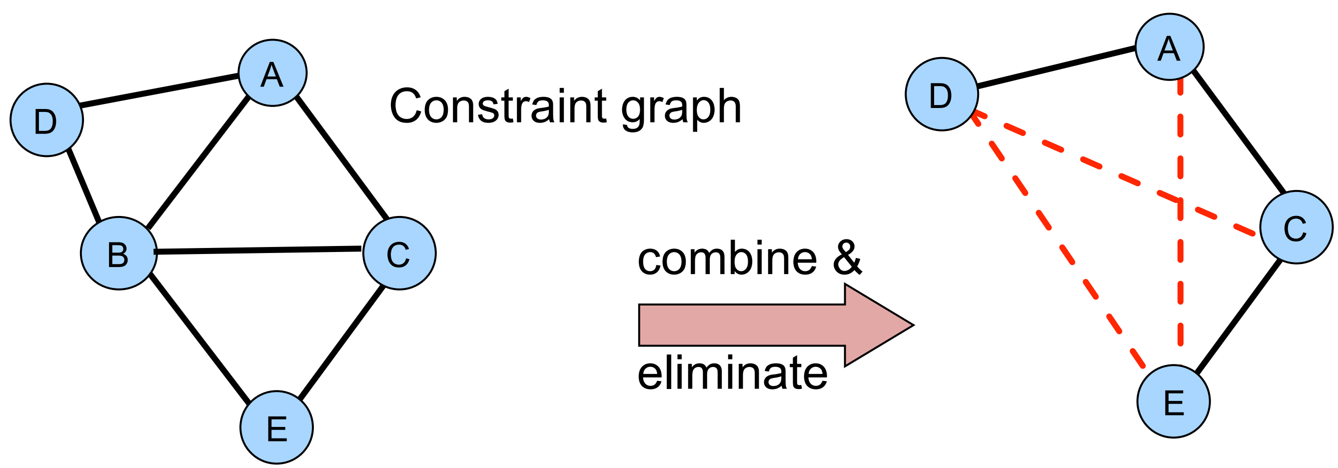


Variable ordering heuristics

- Min (induced) width heuristic

1. for i=1 to n (# of variables)
2. Select a node Xi with smallest degree as next eliminated
3. Connect Xi's neighbors:
4. E = E + { (Xj, Xk) : (Xi,Xj) and (Xi,Xk) in E }
5. Remove Xi from the graph: V = V – {Xi}
6. end

("Weighted" version: weight edges by domain size)

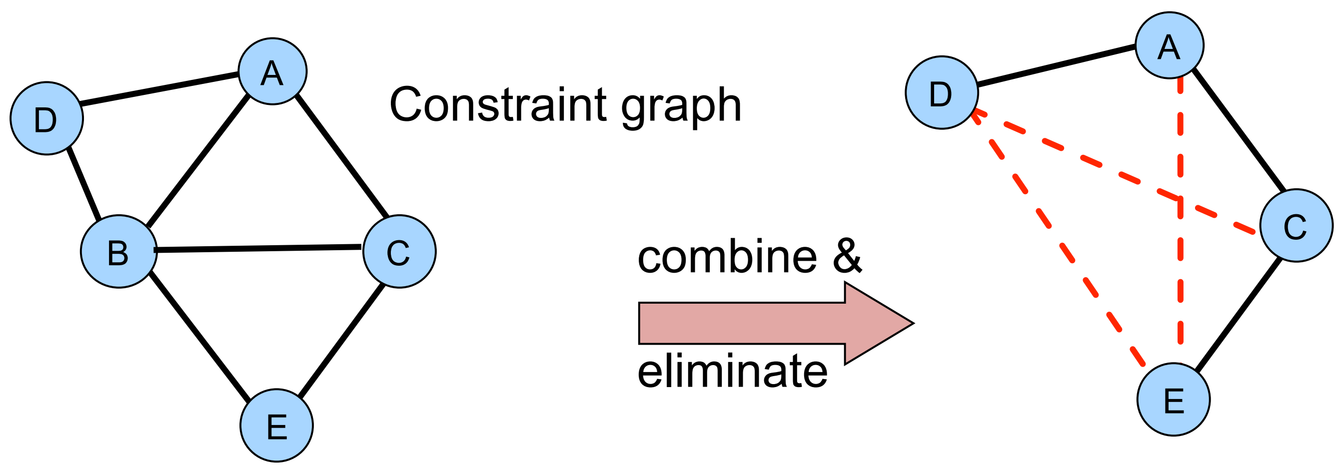


Variable ordering heuristics

- Min fill heuristic

1. for i=1 to n   (# of variables)
2. Select a node Xi with smallest "fill edges" as next eliminated
3. Connect Xi's neighbors:
4. E = E + { (Xj, Xk) : (Xi,Xj) and (Xi,Xk) in E }
5. Remove Xi from the graph: V = V – {Xi}
6. end

("Weighted" version: weight edges by domain size)



Tree-structured graphs

- If the graph is a tree, the best ordering is easy:
  - B, E have only one neighbor; no “fill”
  - Select one to eliminate; remove it
  - Now D or E have only one neighbor; no “fill”…
- Order
  - leaves to root
  - never increases the size of the factors